Unit-6: Model-checking ω-regular properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Module 1:
Overview

Does **Transition system** satisfy ω-regular property ?

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

$\downarrow$

NBA $\mathscr{A}_\phi$

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

$\downarrow$ $\qquad\qquad$ $\downarrow$

NBA $\mathscr{A}_{T.S}$ $\qquad\qquad$ NBA $\mathscr{A}_{\phi}$

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

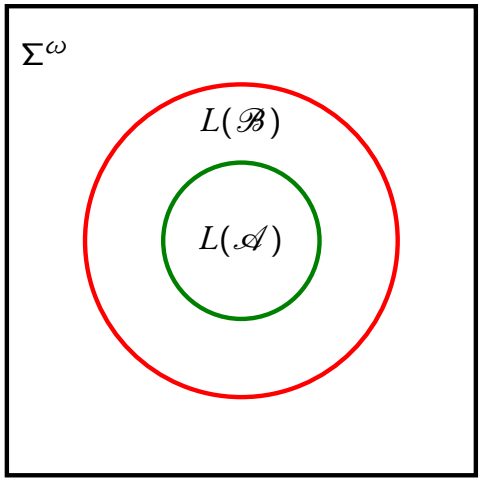
NBA $\mathscr{A}_{T.S}$

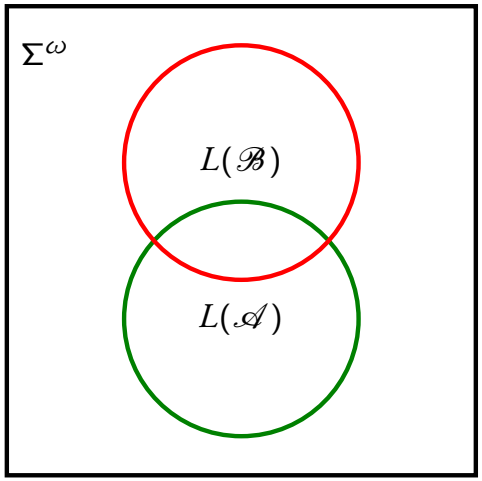
NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

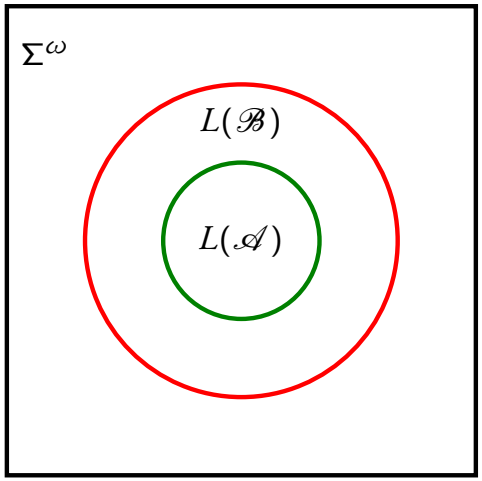
$$L(\mathscr{A}) \subseteq L(\mathscr{B})\,?$$

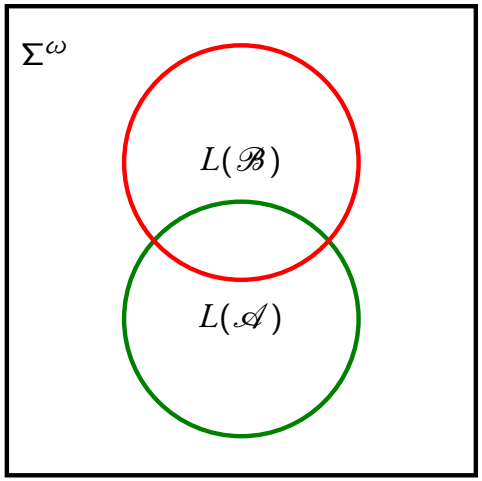
$$L(\mathscr{A}) \subseteq L(\mathscr{B})\,?$$





$$L(\mathscr{A}) \subseteq L(\mathscr{B})\,?$$





$$L(\mathscr{A}) \cap \overline{L(\mathscr{B})} \text{ is empty ?}$$

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

NBA $\mathscr{A}_{T.S}$

NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

NBA $\mathscr{A}_{T.S}$

NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

Is $L(\mathscr{A}_{T.S.}) \cap \overline{L(\mathscr{A}_{\phi})}$ empty ?

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

NBA $\mathscr{A}_{T.S}$

NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

Is $L(\mathscr{A}_{T.S.}) \cap \overline{L(\mathscr{A}_{\phi})}$ empty ?

Is $L(\mathscr{A}_{T.S.}) \cap L(\overline{\mathscr{A}_{\phi}})$ empty ?

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

NBA $\mathscr{A}_{T.S}$

NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

Is $L(\mathscr{A}_{T.S.}) \cap \overline{L(\mathscr{A}_{\phi})}$ empty ?

Is $L(\mathscr{A}_{T.S.}) \cap L(\overline{\mathscr{A}_{\phi}})$ empty ?

Is $L(\mathscr{A}_{T.S.} \times \overline{\mathscr{A}_{\phi}})$ empty ?

To be seen...

- **Converting** ω-regular expression to NBA (Module 2)

- Checking **language emptiness** of NBA (Module 3 and 4)

Unit-6: Model-checking ω-regular properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Module 2:
ω-regular expressions to NBA

$$\Sigma = \{ a, b \}$$

Example 1: Infinite word consisting only of a $\quad a^\omega$

$$\{ aaaaaaaaaaaaaaa \ldots \}$$

Example 2: Infinite words containing only a or only b $\quad a^\omega + b^\omega$

$$\{ aaaaaaaaaaaaaa \ldots , \; bbbbbbbbbbbb \ldots \}$$

Example 3: a word in $aa\Sigma^* aa$ followed by only b-s $\quad aa\Sigma^* aa \cdot b^\omega$

$$\{ aaaabbbbbbb \ldots , \; aababaabbbbbb \ldots , \; aabbbbaabbbbbbb \ldots , \; \ldots \}$$

Example 4: Infinite words where b occurs **only finitely often** $(a+b)^* \cdot a^\omega$

$$\{ aaaaaaaaaaaaaaa \ldots , \; baaaaaaaaaa \ldots , \; babbaaaaaaaaaaa \ldots , \; \ldots \}$$

Example 5: Infinite words where b occurs **infinitely often** $\quad (a^* b)^\omega$

$$\{ abababababab \ldots , \; bbbabbbabbbabbba \ldots , \; bbbbbbbbbbbbb \ldots , \; \ldots \}$$

ω-regular expressions

$$G = E_1 \cdot F_1^\omega + E_2 \cdot F_2^\omega + \cdots + E_n \cdot F_n^\omega$$

$E_1, \ldots, E_n, F_1, \ldots, F_n$ are **regular expressions**

and $\epsilon \notin L(F_i)$ for all $1 \leq i \leq n$

$$L(F^\omega) = \{ w_1 w_2 w_3 \ldots \mid \textbf{each } w_i \in L(F) \}$$

More examples

- $(a+b)^\omega$ set of **all infinite words**
- $a(a+b)^\omega$ infinite words **starting with** an a
- $(a+bc+c)^\omega$ words where every b is **immediately followed** by c
- $(a+b)^*c(a+b)^\omega$ words with a **single occurrence** of c
- $((a+b)^*c)^\omega$ words where c **occurs infinitely often**

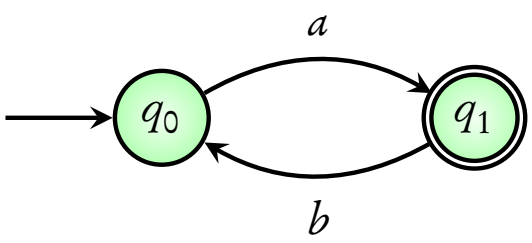
ω-regular expressions

$$G = E_1 \cdot F_1^\omega + E_2 \cdot F_2^\omega + \cdots + E_n \cdot F_n^\omega$$

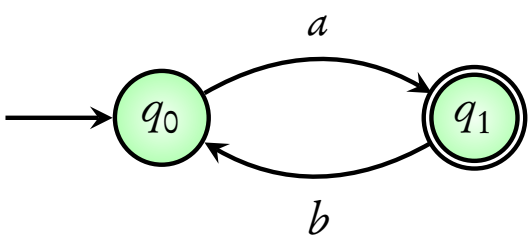
Goal: Convert ω-regular expression to NBA

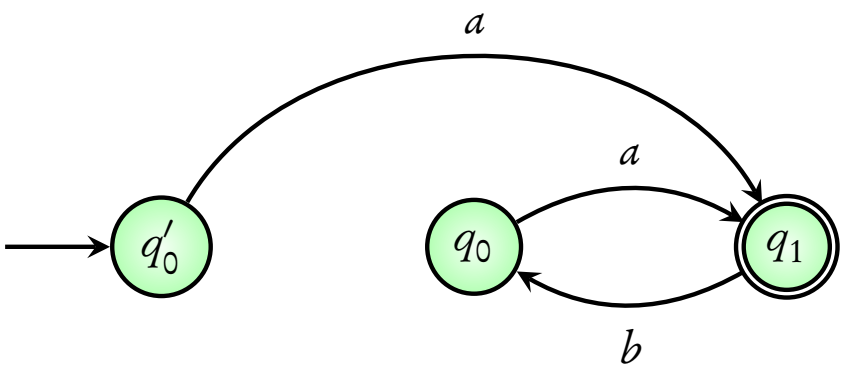
Part 1: Given regular expression U, find NBA for U^ω

NFA for U

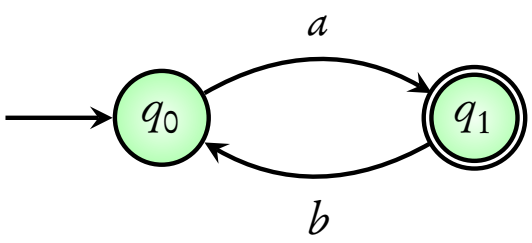


NFA for U

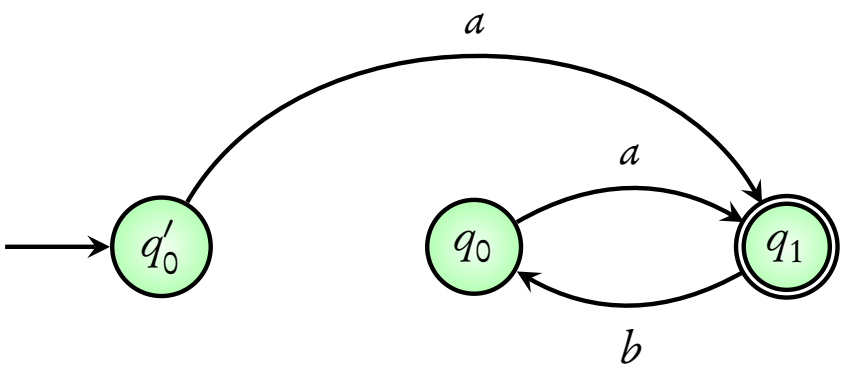




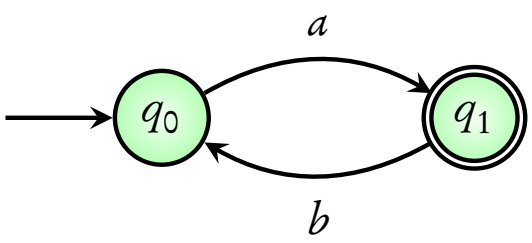
NFA for U



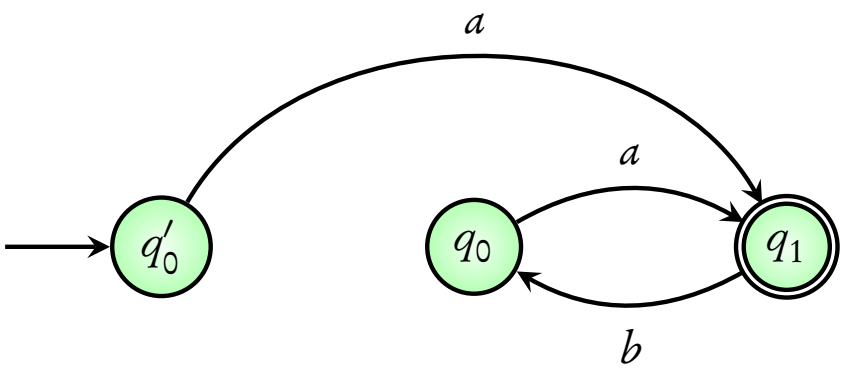
Standardized NFA

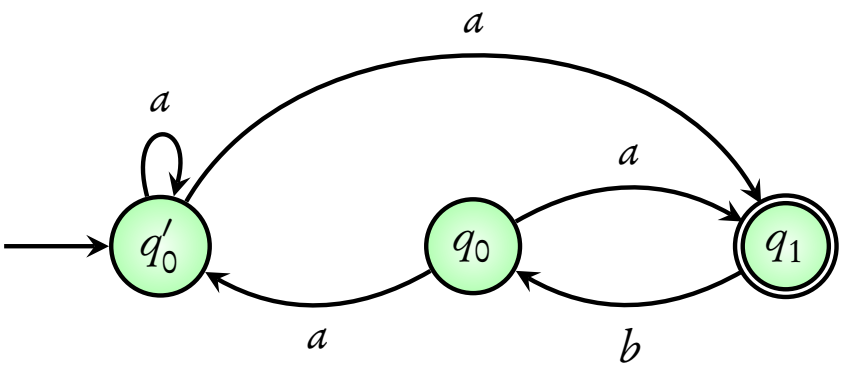


NFA for U

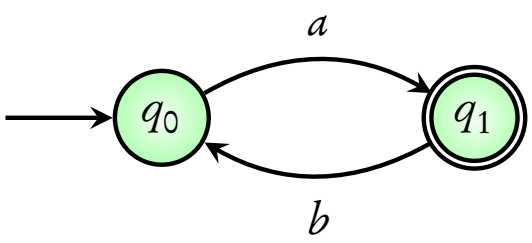


Standardized NFA

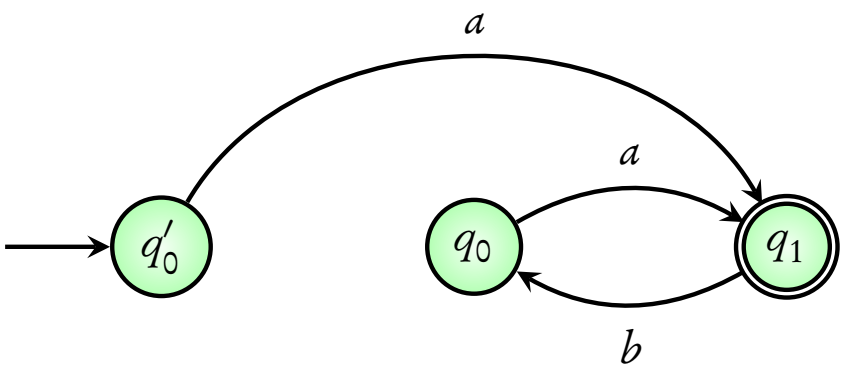


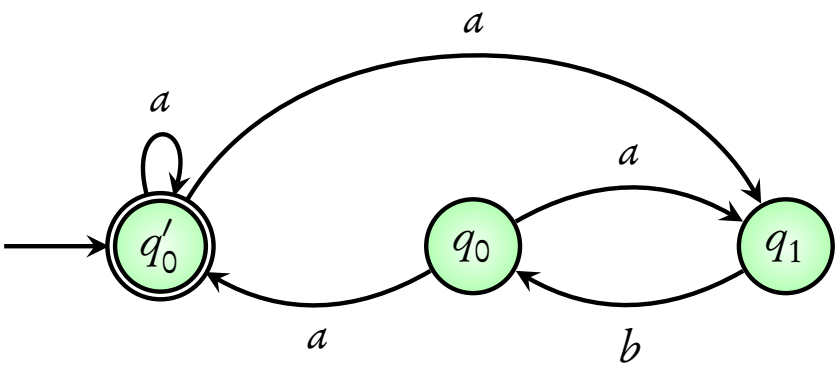


NFA for U

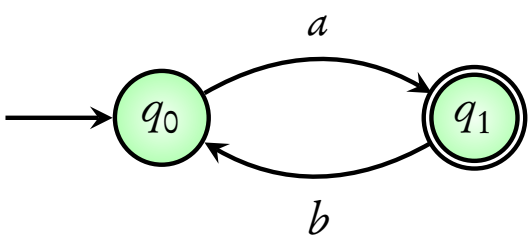


Standardized NFA

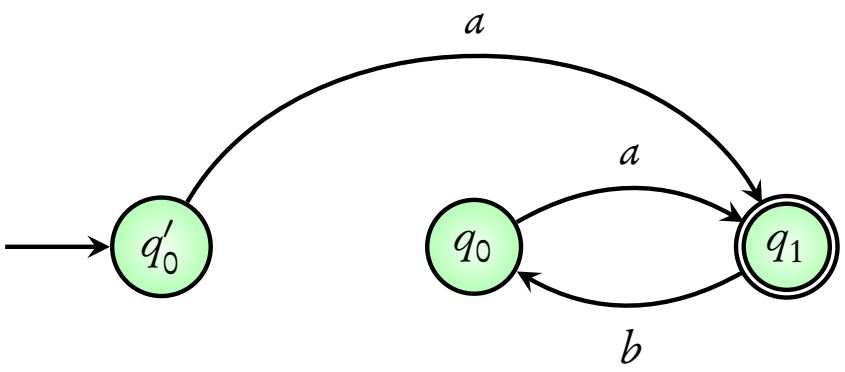




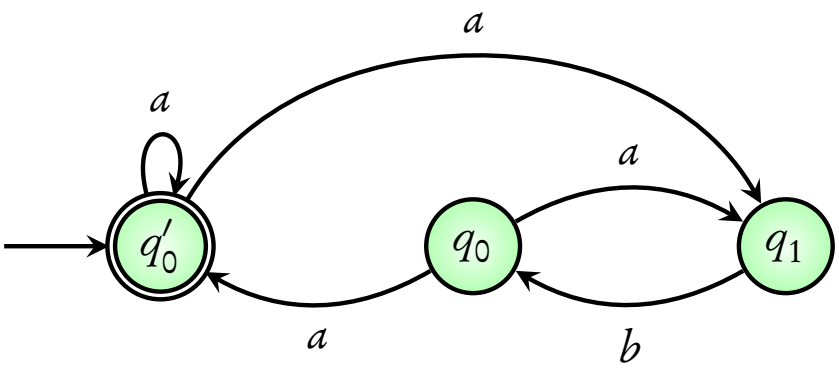
NFA for U



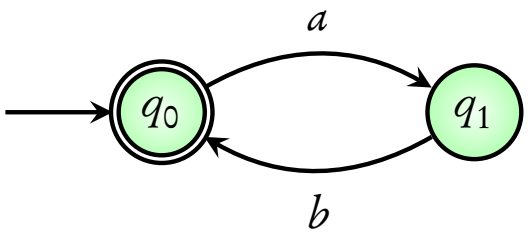
Standardized NFA



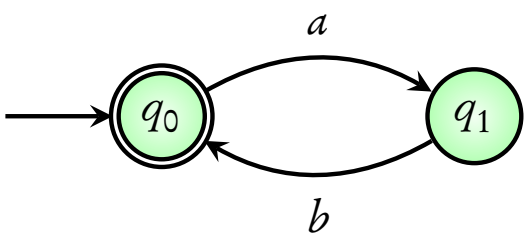
NBA for U^ω

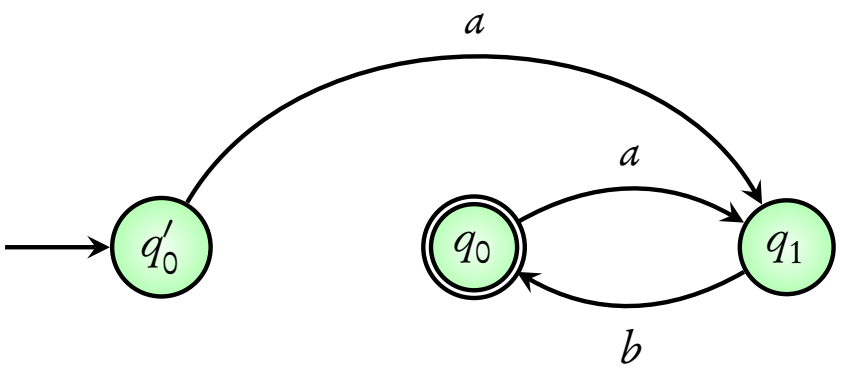


NFA for U

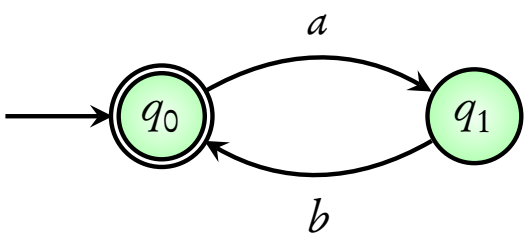


NFA for U

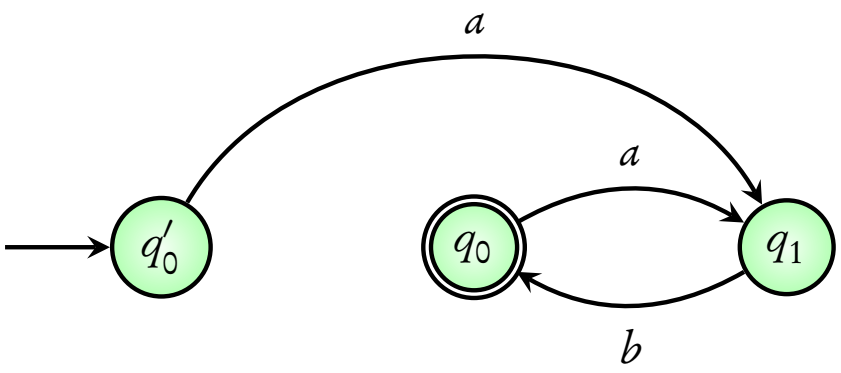




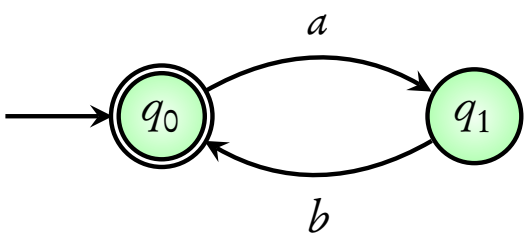
NFA for U



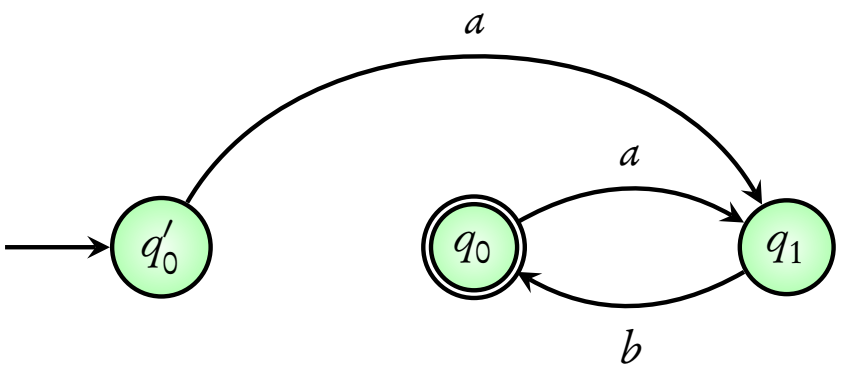
Standardized NFA

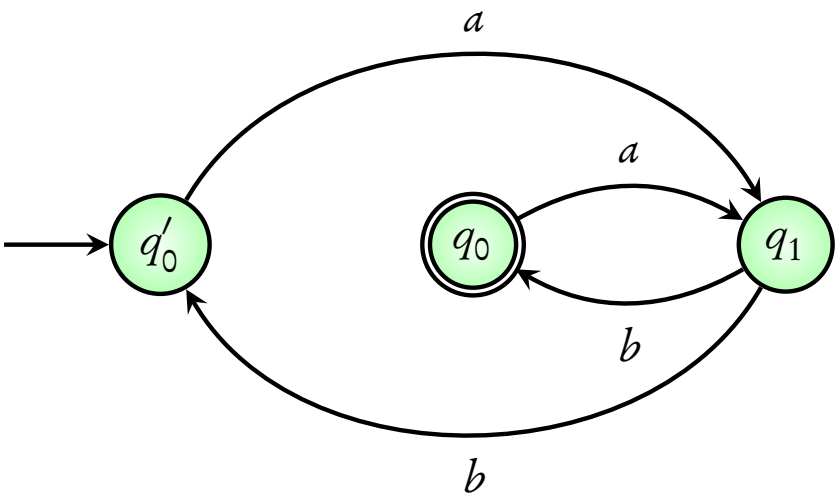


NFA for U

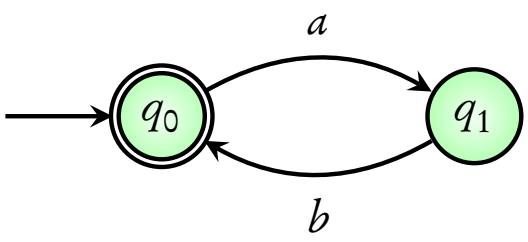


Standardized NFA

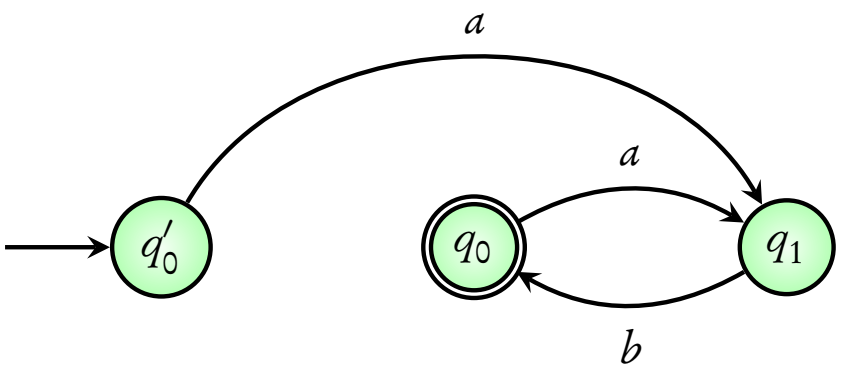


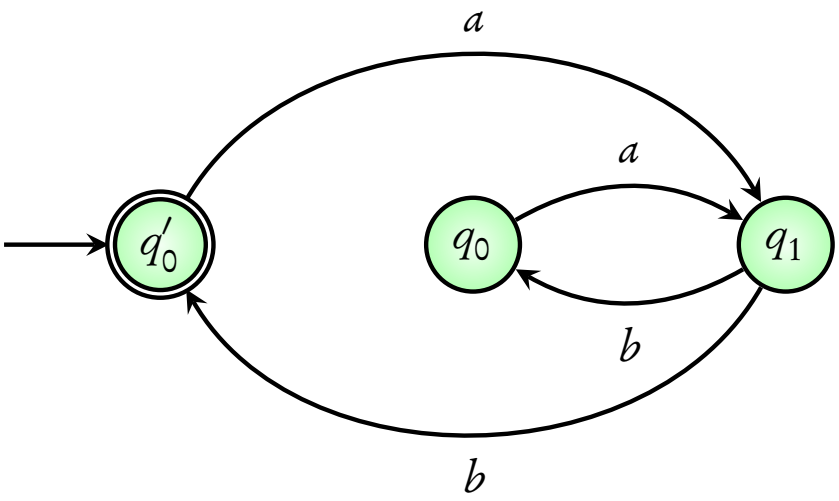


NFA for U

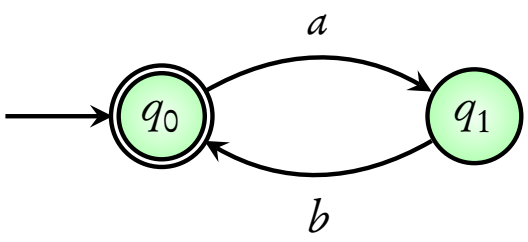


Standardized NFA

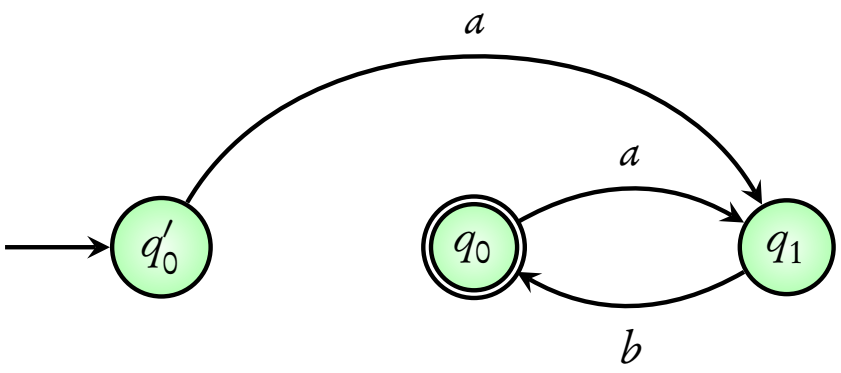




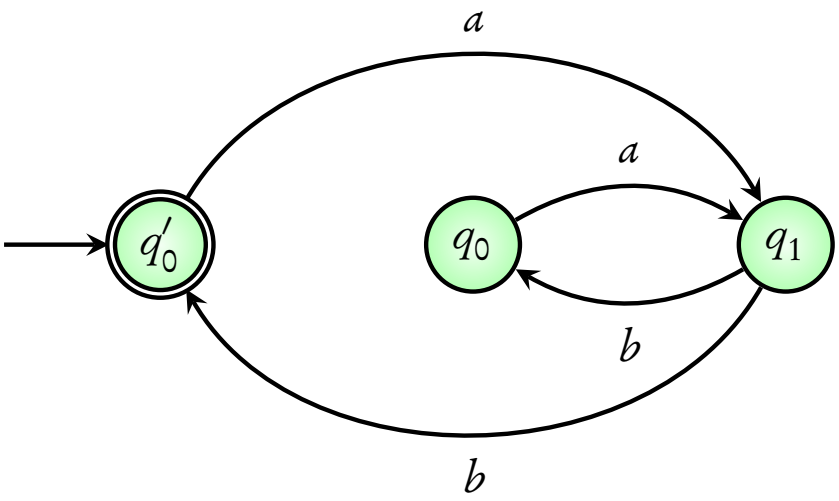
NFA for U



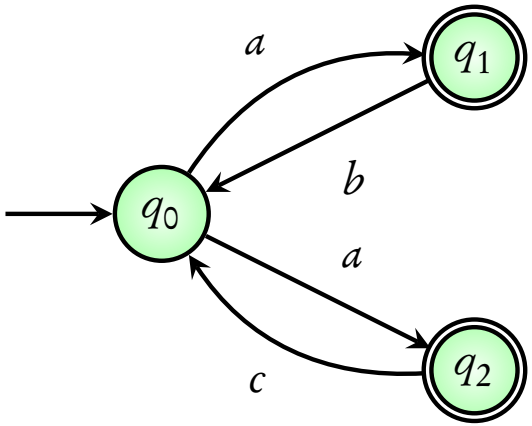
Standardized NFA



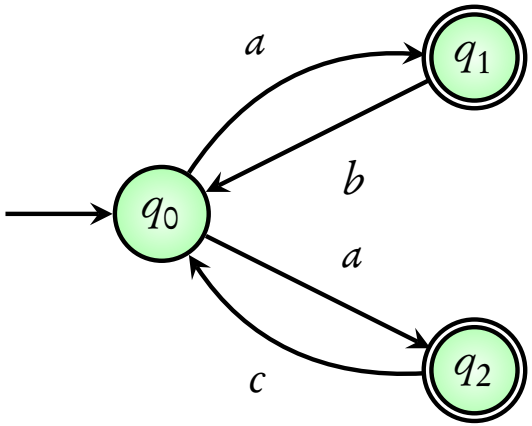
NBA for U^ω

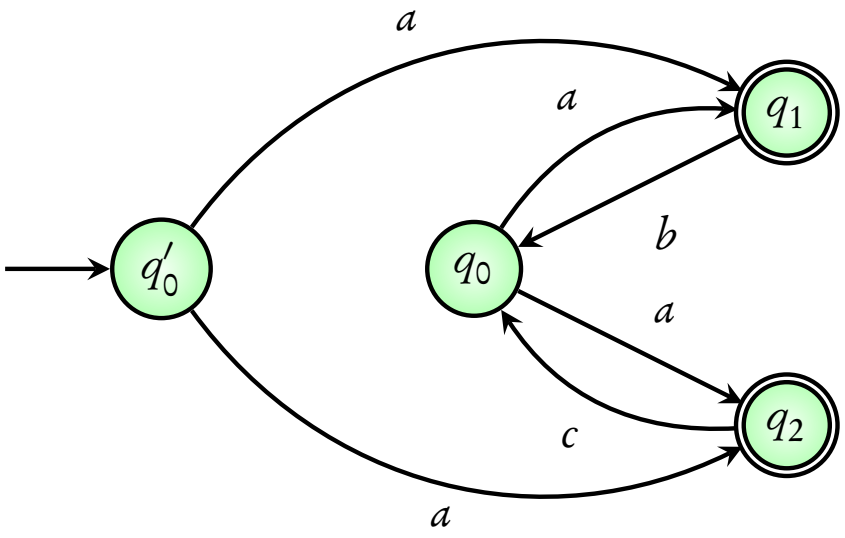


NFA for U

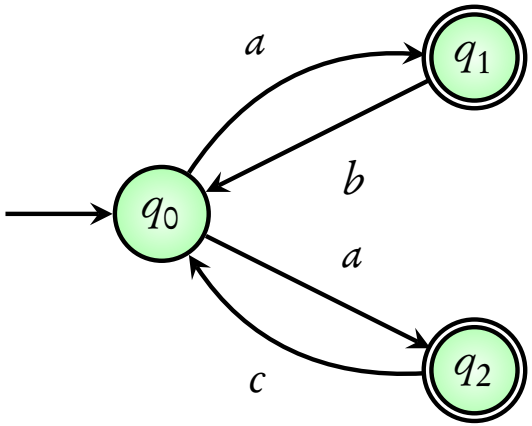


NFA for U

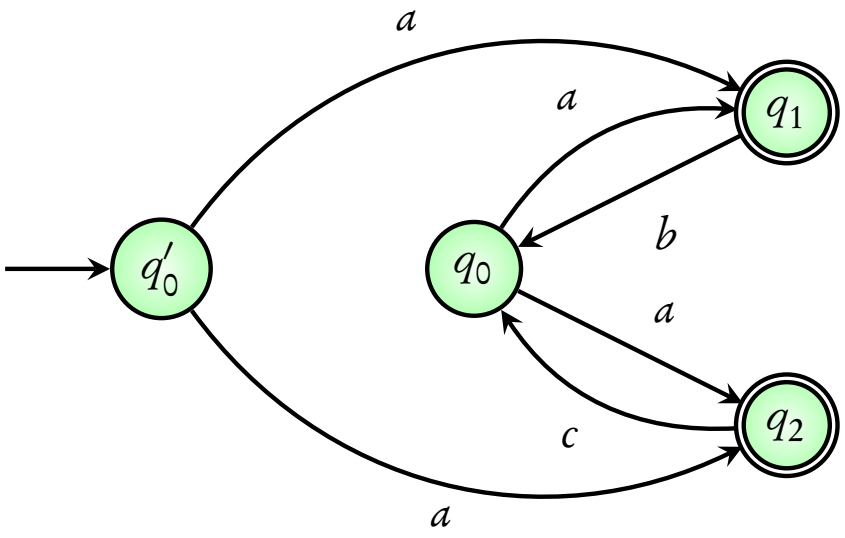




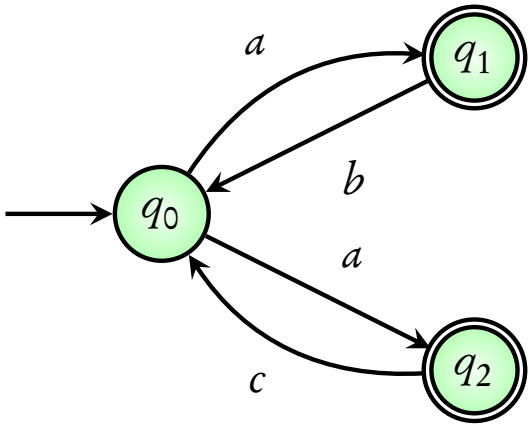
NFA for U



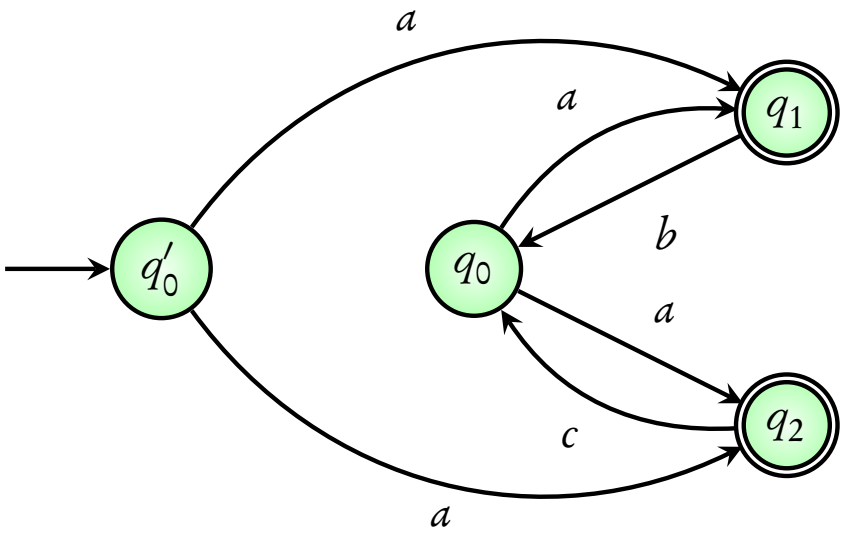
Standardized NFA

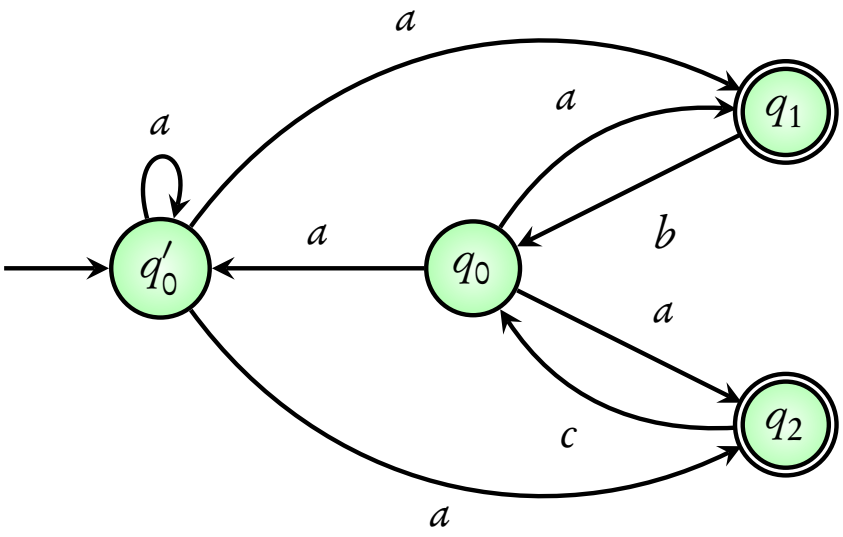


NFA for U

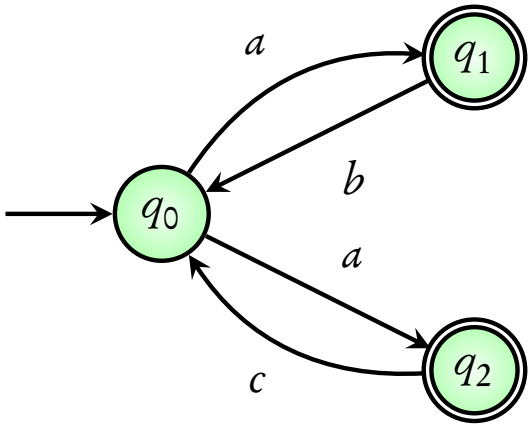


Standardized NFA

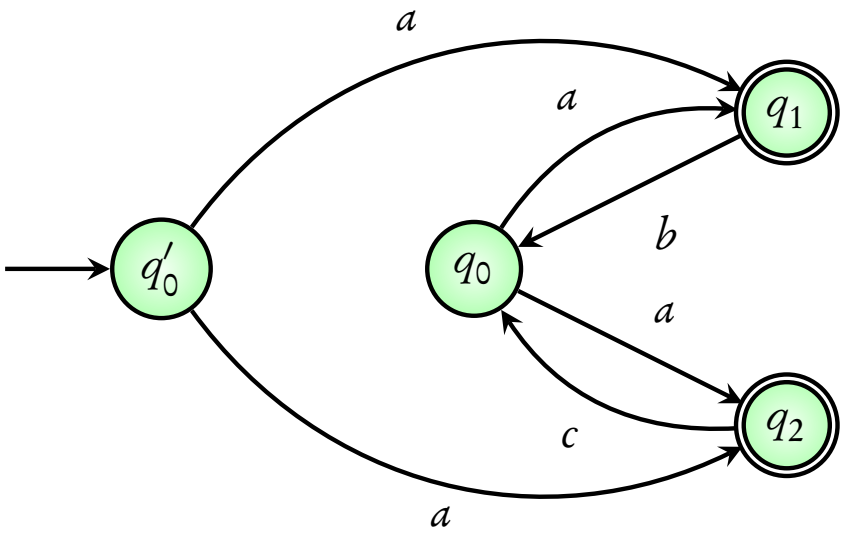


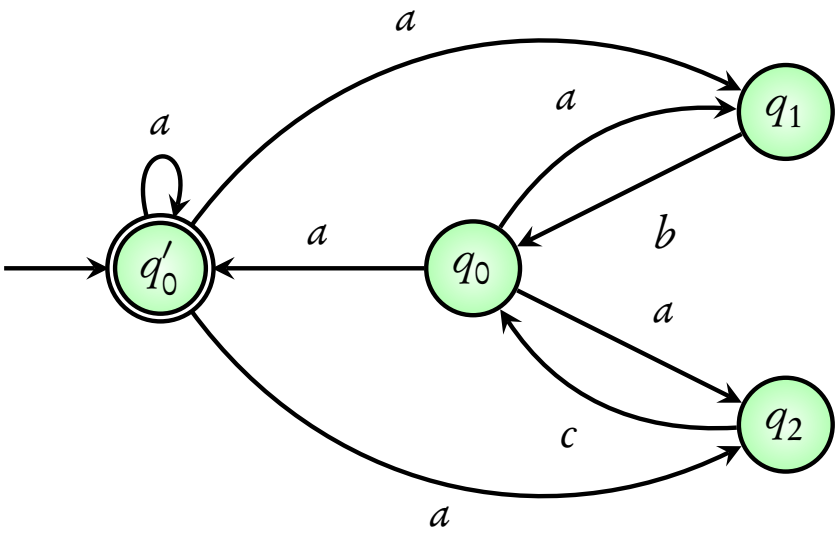


NFA for U

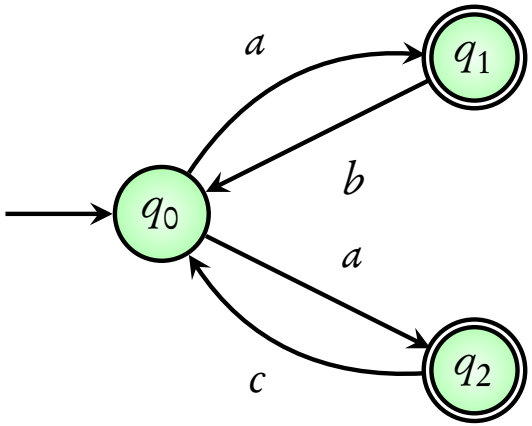


Standardized NFA

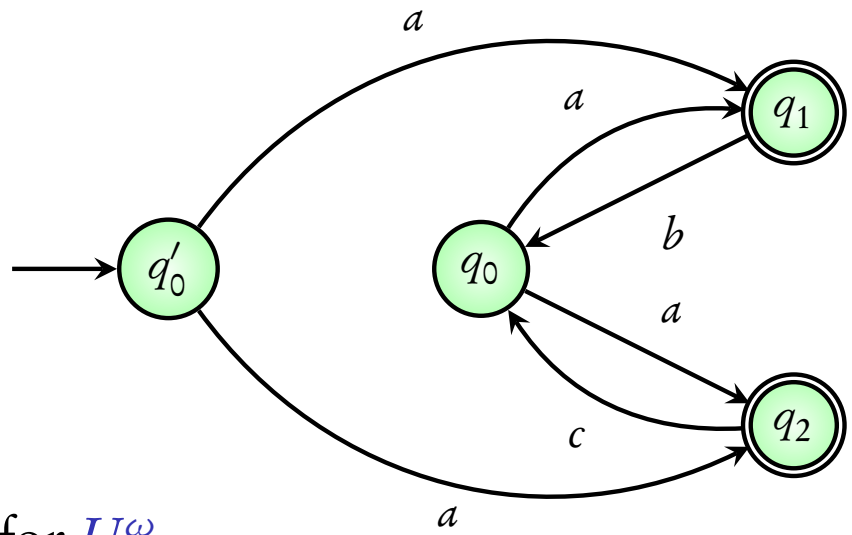




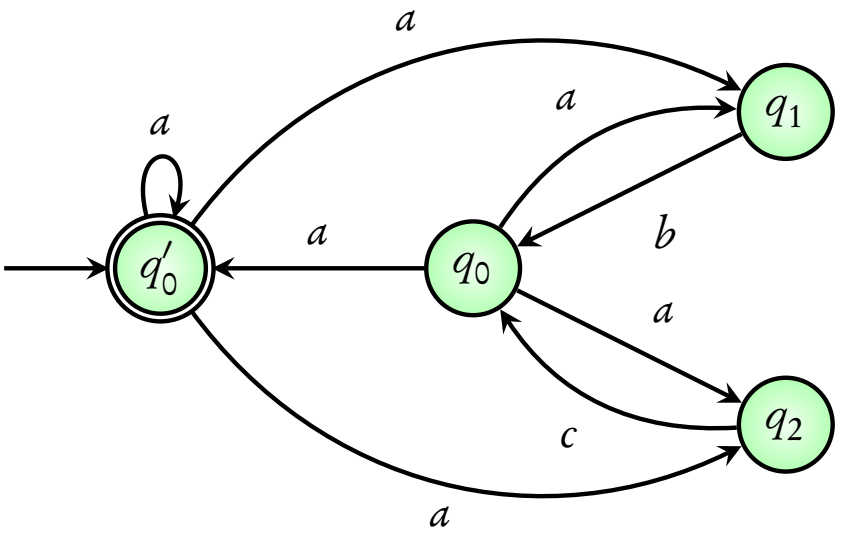
NFA for U



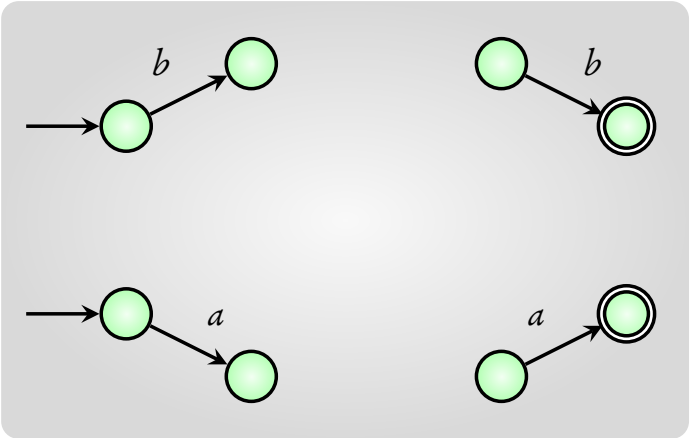
Standardized NFA



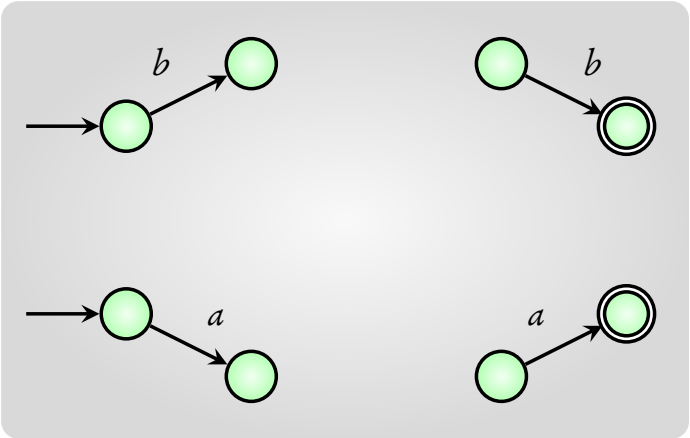
NBA for U^ω



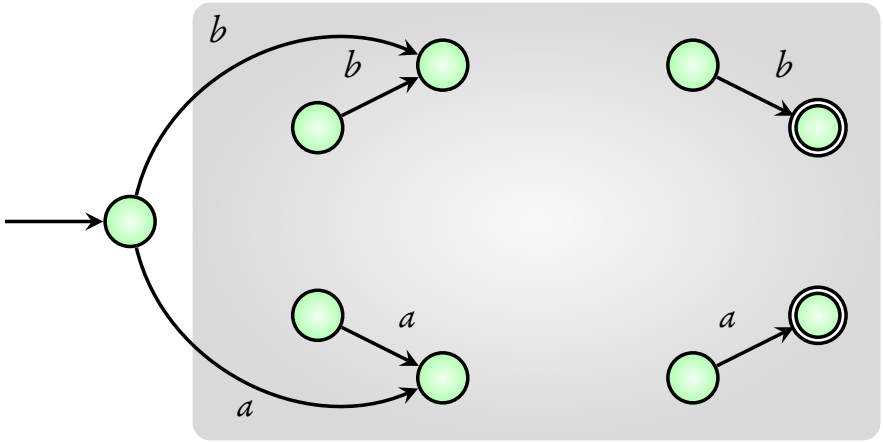
NFA for U



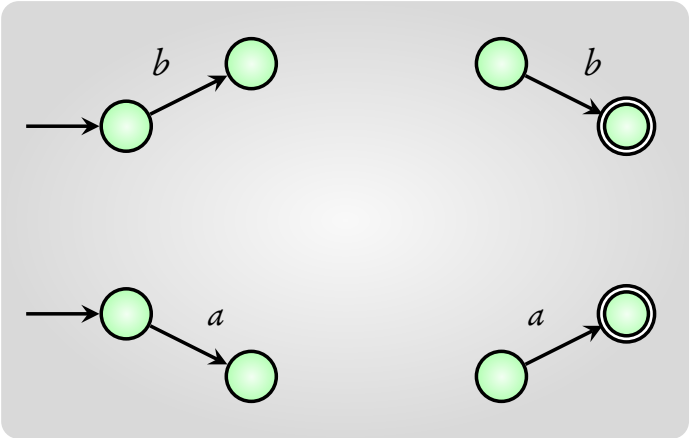
NFA for U



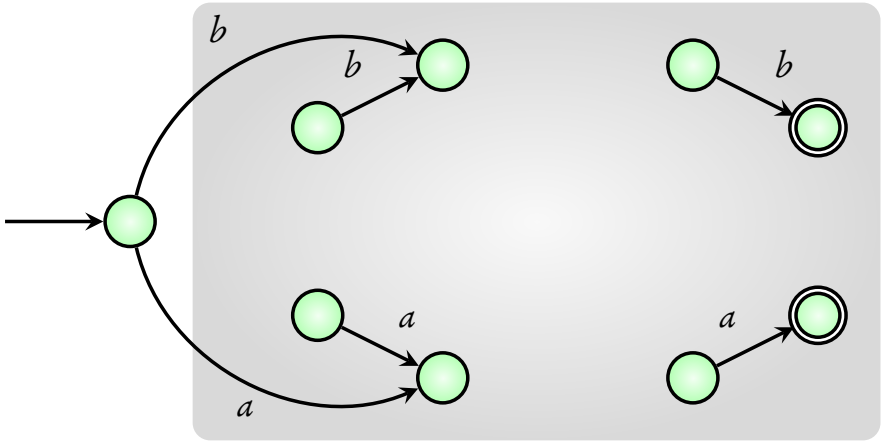
Standardized NFA for U



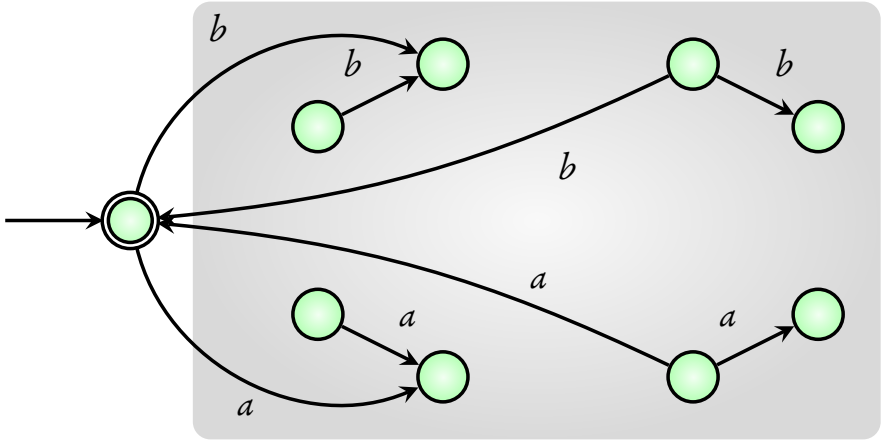
NFA for U



Standardized NFA for U



NBA for U^ω



ω-regular expressions

$$G = E_1 \cdot F_1^\omega + E_2 \cdot F_2^\omega + \cdots + E_n \cdot F_n^\omega$$

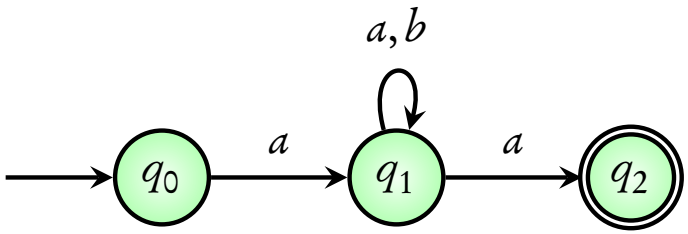
Goal: Convert ω-regular expression to NBA

Part 1: Given regular expression U, find NBA for U^ω

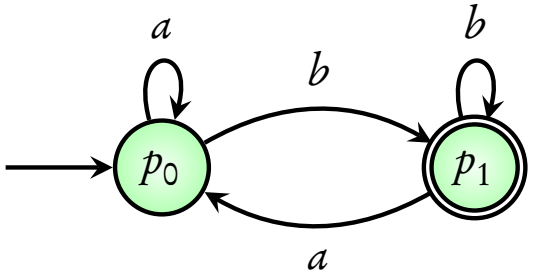
Done!

Part 2: Given regular expression U and NBA for V find NBA for $U \cdot V$

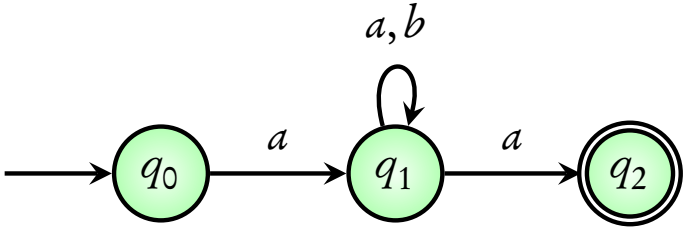
$a\Sigma^* a$



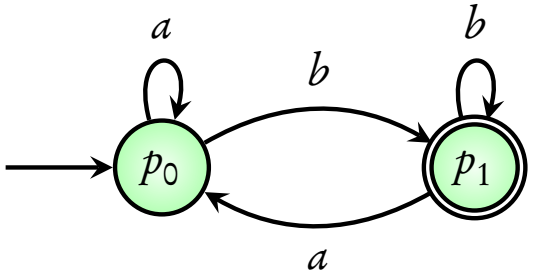
$(a^* b)^\omega$

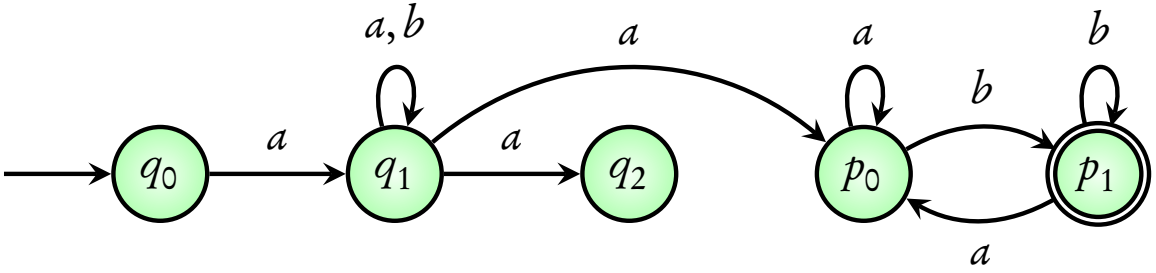


$a\Sigma^* a$

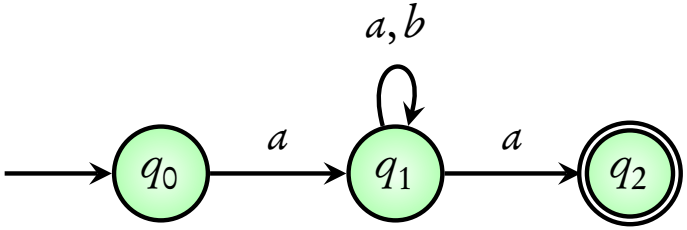


$(a^* b)^\omega$

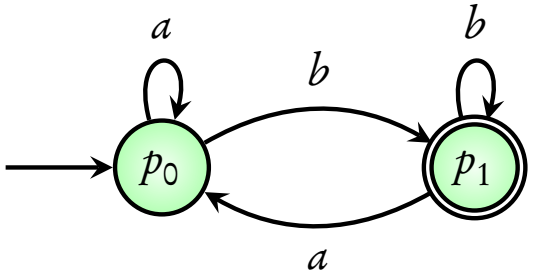




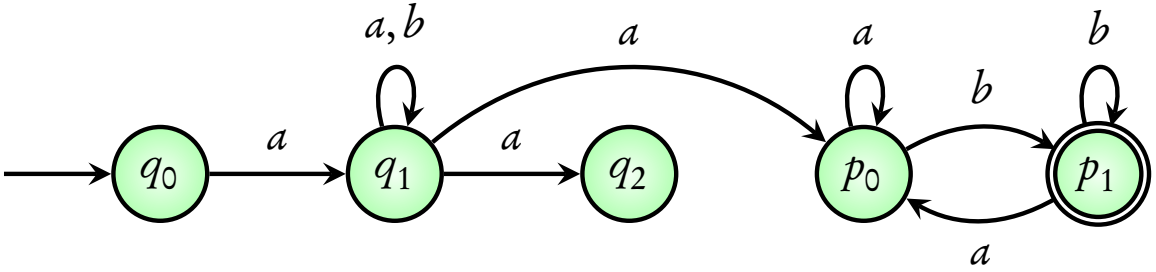
$a\Sigma^* a$



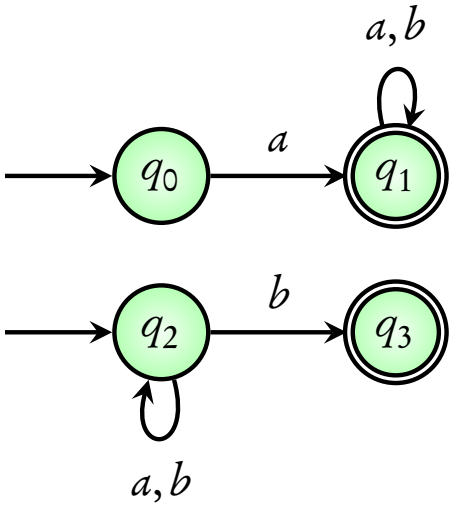
$(a^* b)^\omega$



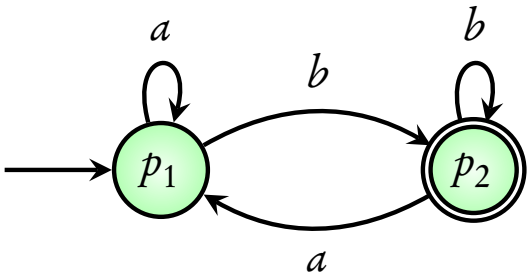
$a\Sigma^* a \cdot (a^* b)^\omega$



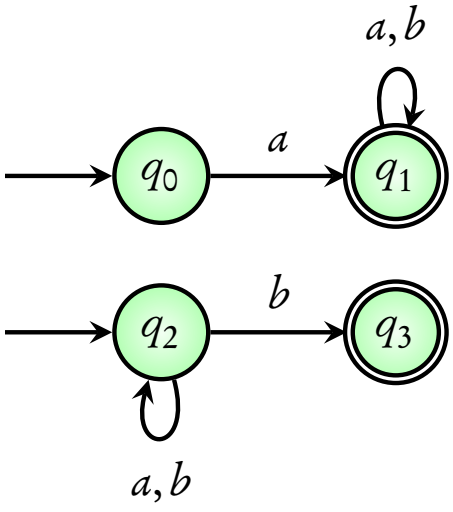
$a\Sigma^* + \Sigma^* b$



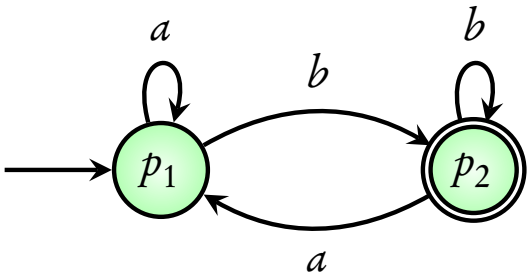
$(a^* b)^\omega$



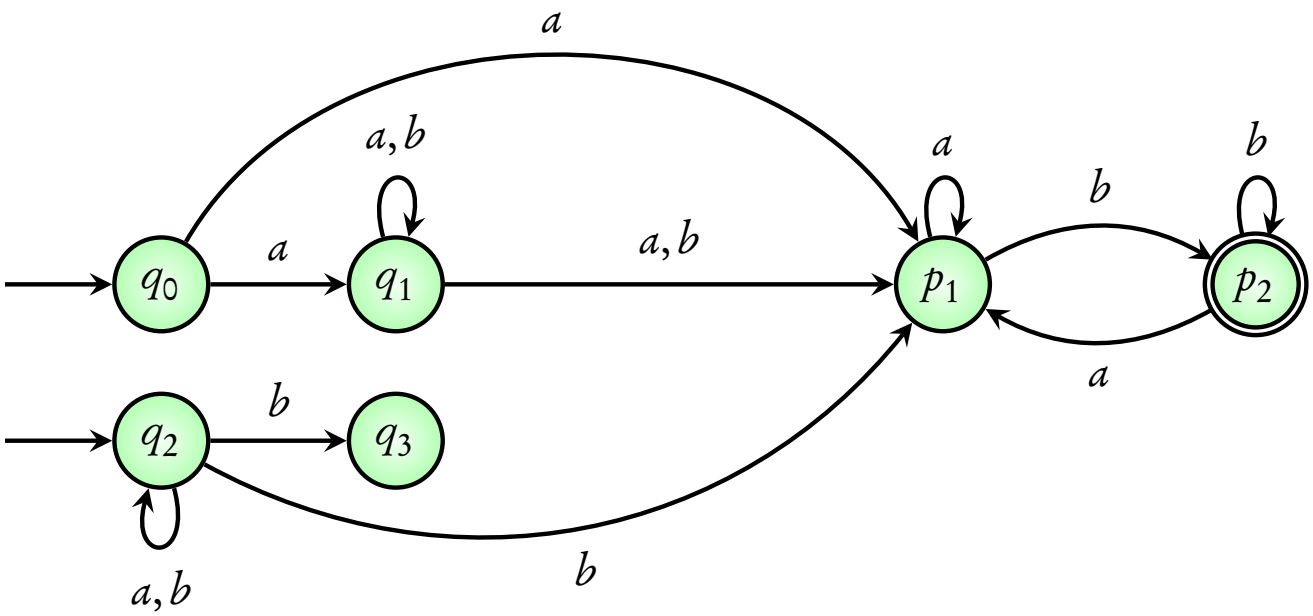
$a\Sigma^* + \Sigma^* b$



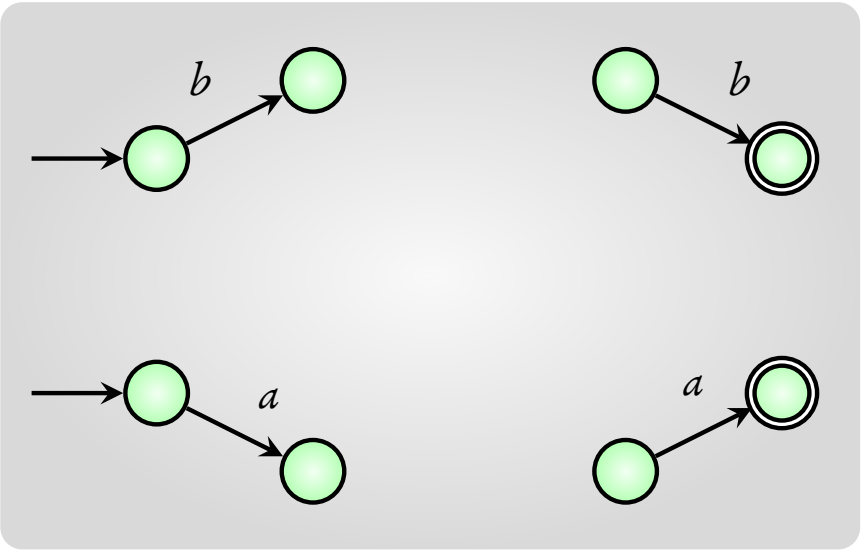
$(a^* b)^\omega$



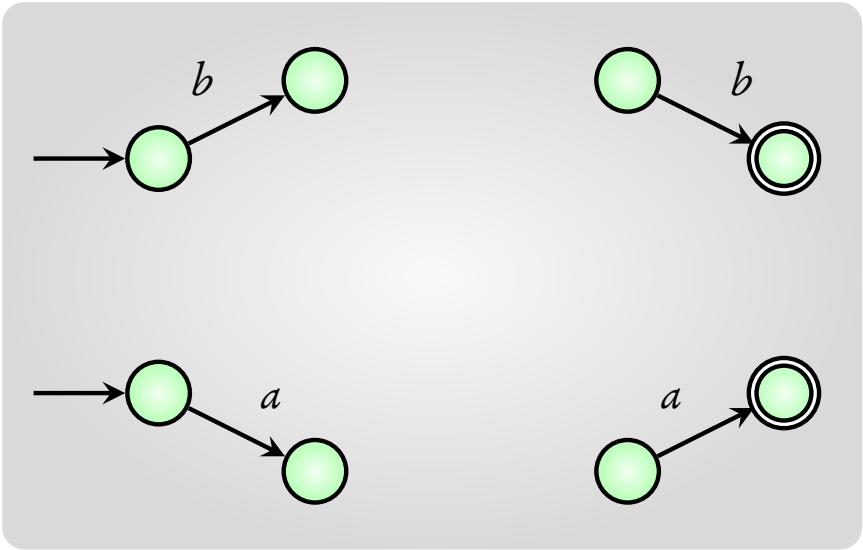
$(a\Sigma^* + \Sigma^* b) \cdot (a^* b)^\omega$



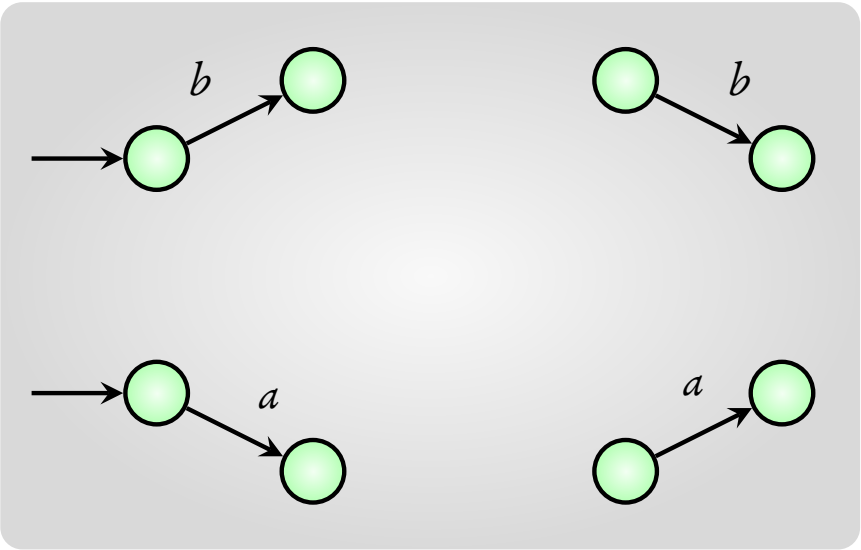
U



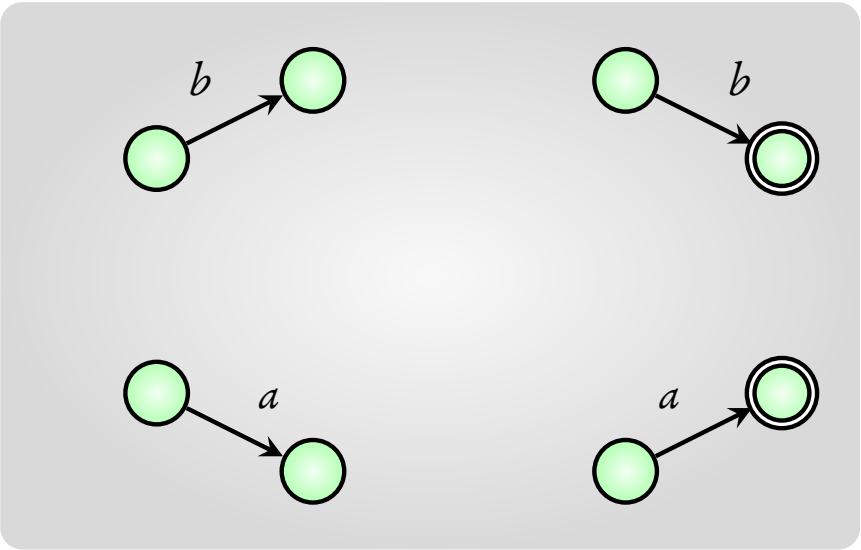
V

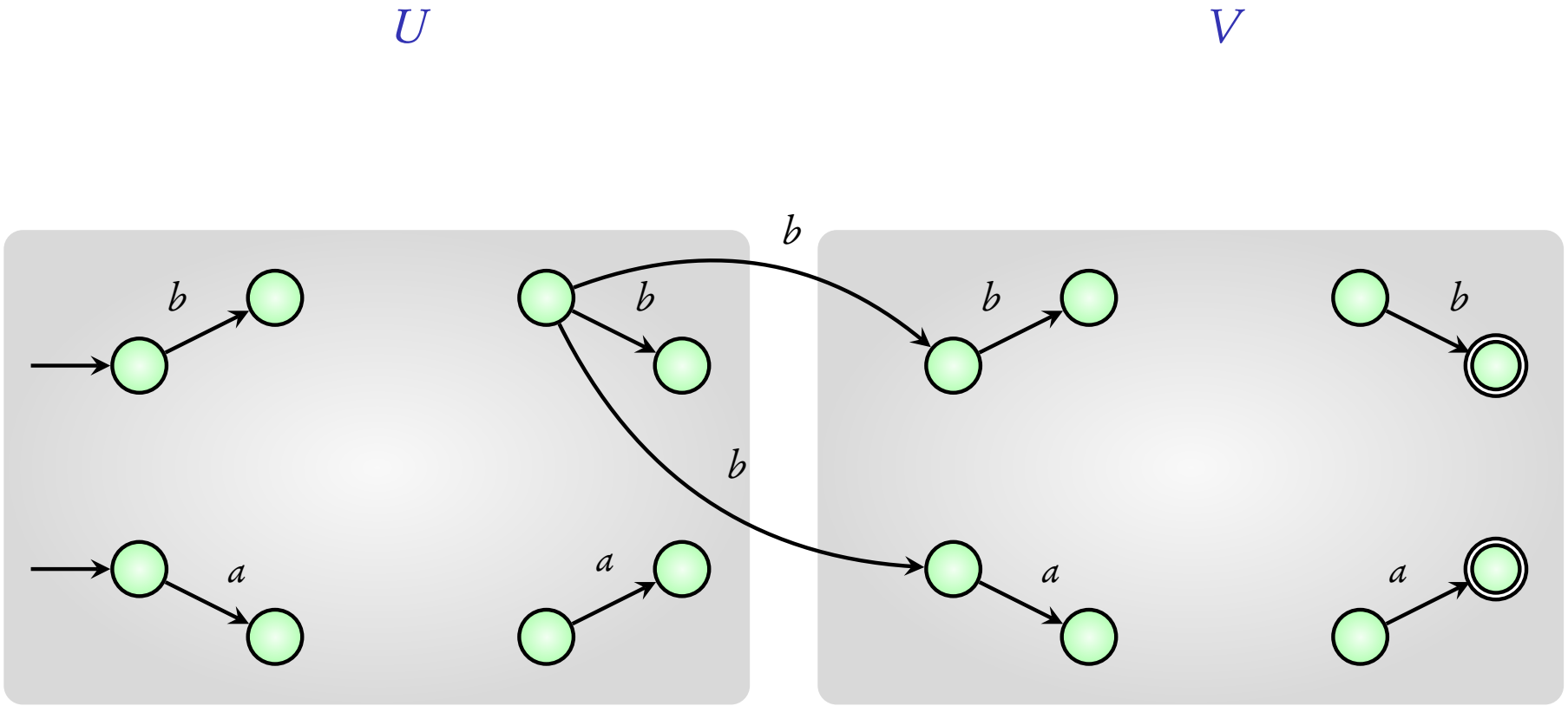


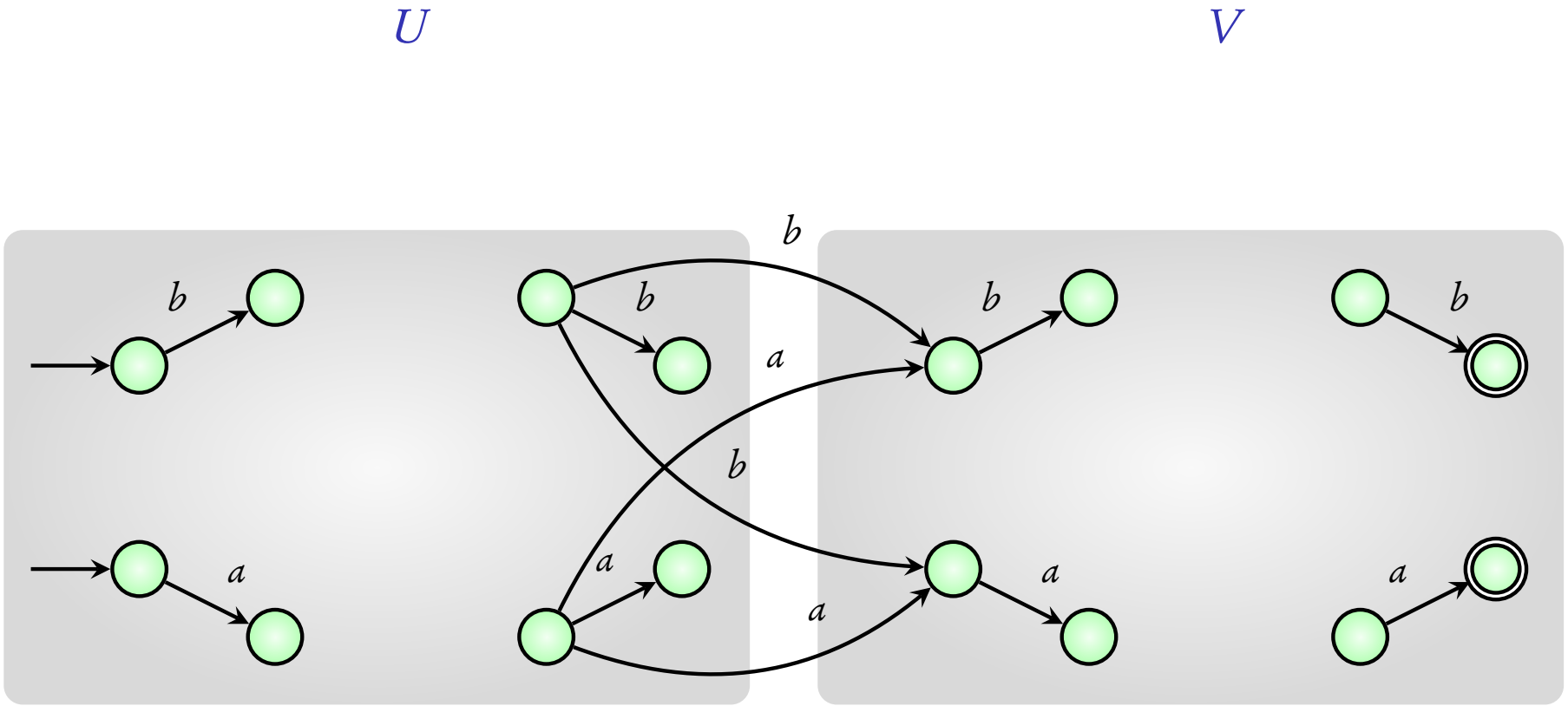
U



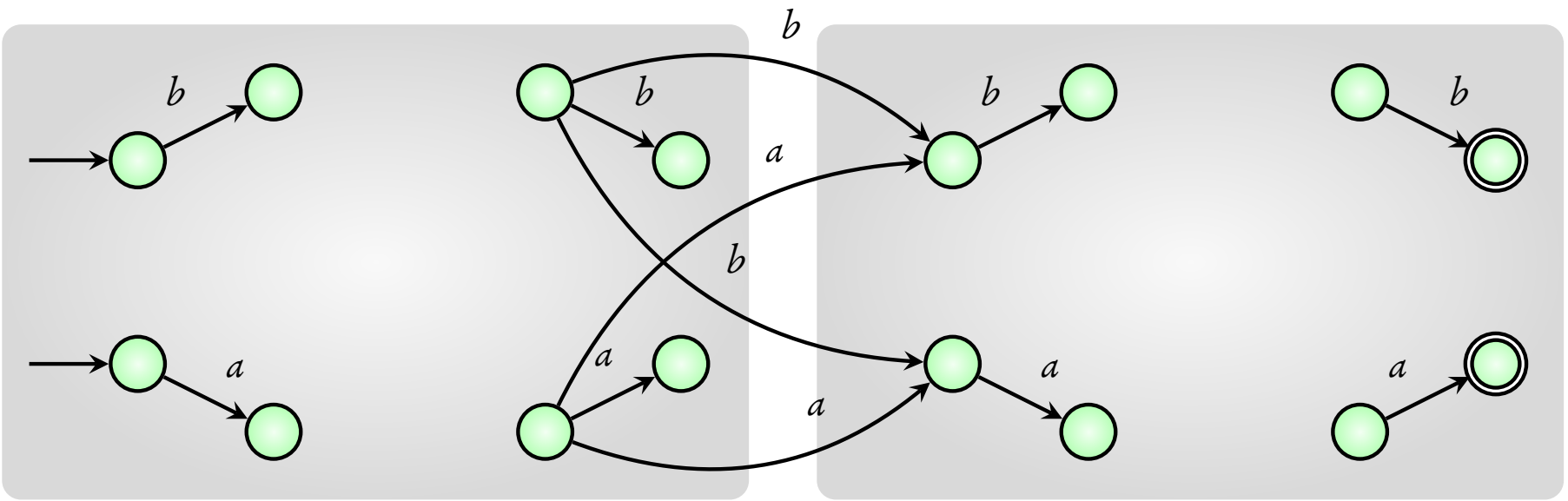
V




U
V
b
b
b
b
b
b
a
a
a
a


U
V
b
b
b
b
a
b
a
a
a
a
a

$U \cdot V$



ω-regular expressions

$$G = E_1 \cdot F_1^\omega + E_2 \cdot F_2^\omega + \cdots + E_n \cdot F_n^\omega$$

Goal: Convert ω-regular expression to NBA

Part 1: Given regular expression U, find NBA for U^ω

Part 2: Given regular expression U and NBA for V find NBA for $U \cdot V$

Done!

Part 3: Given NBA for U and NBA for V find NBA for $U + V$

Part 3: Given NBA for U and NBA for V find NBA for $U + V$

Union of NBA already seen in Unit 5

Part 1: Given regular expression U, find NBA for U^ω

Part 2: Given regular expression U and NBA for V find NBA for $U \cdot V$

Part 3: Given NBA for U and NBA for V find NBA for $U + V$

Theorem

Every ω-regular expression can be **converted** to an NBA

Unit-6: Model-checking ω-regular properties

B. Srivathsan

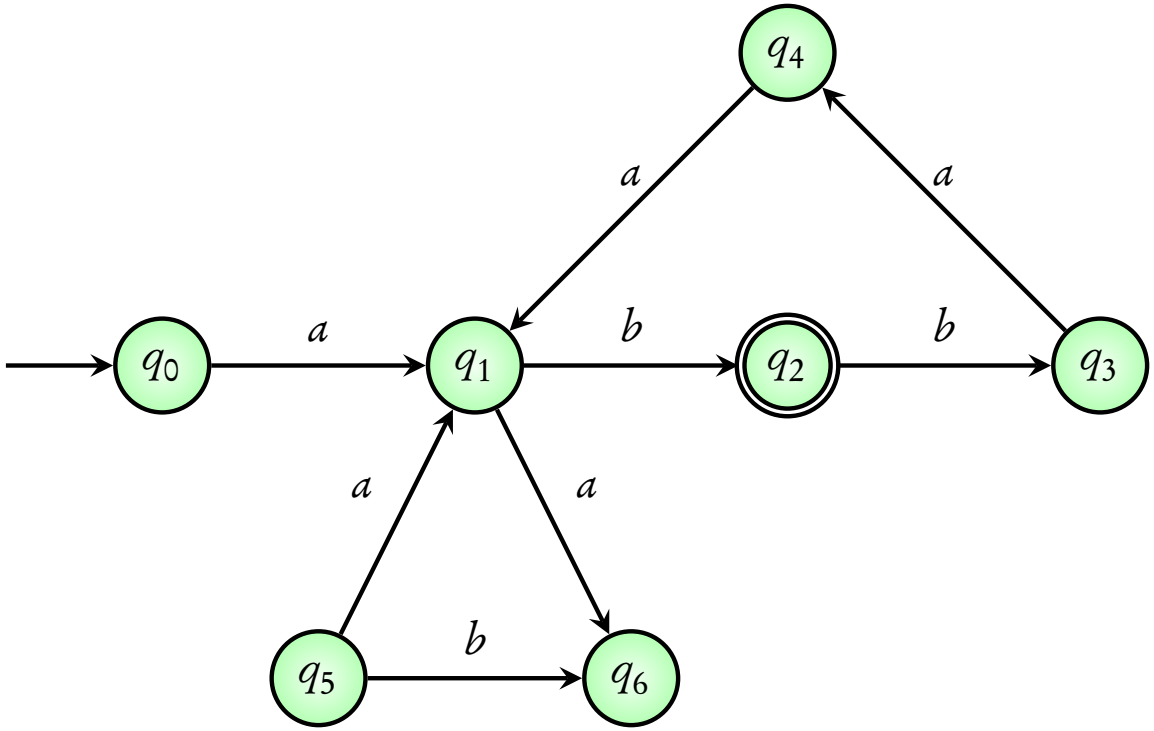
Chennai Mathematical Institute

NPTEL-course

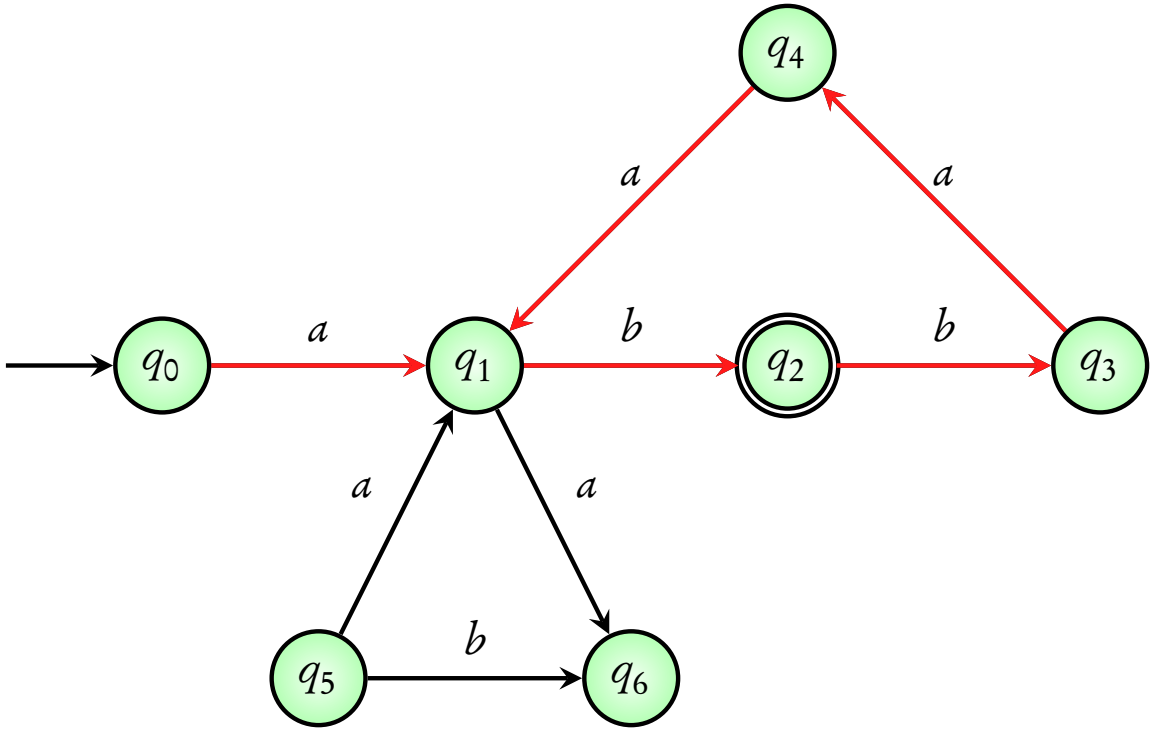
July - November 2015

Module 3:

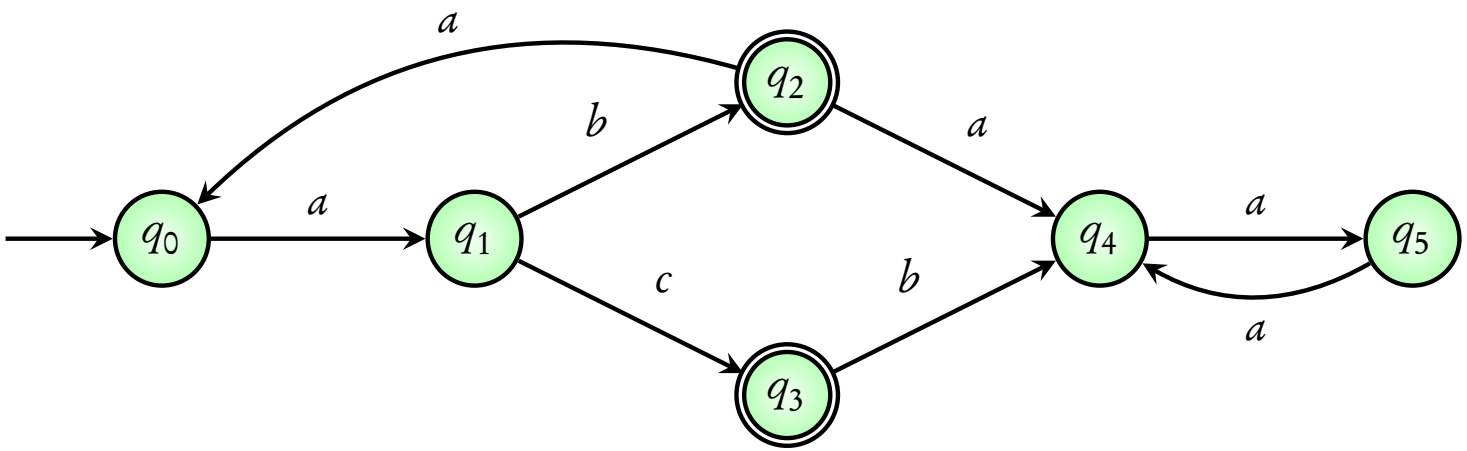
Checking emptiness of Büchi automata



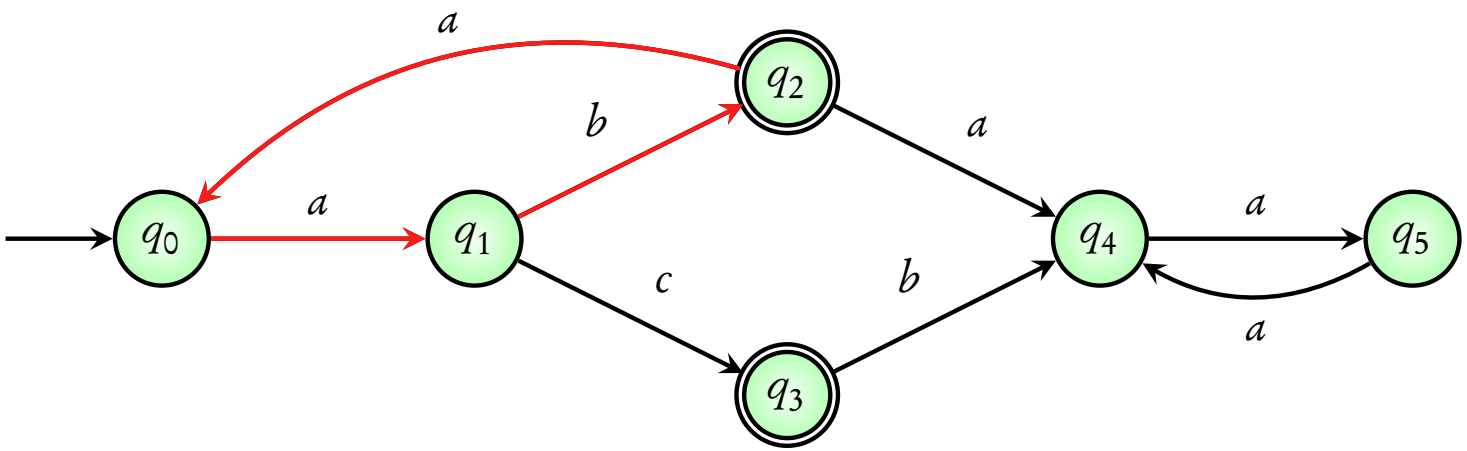
Is the **language** of above NBA **empty**?



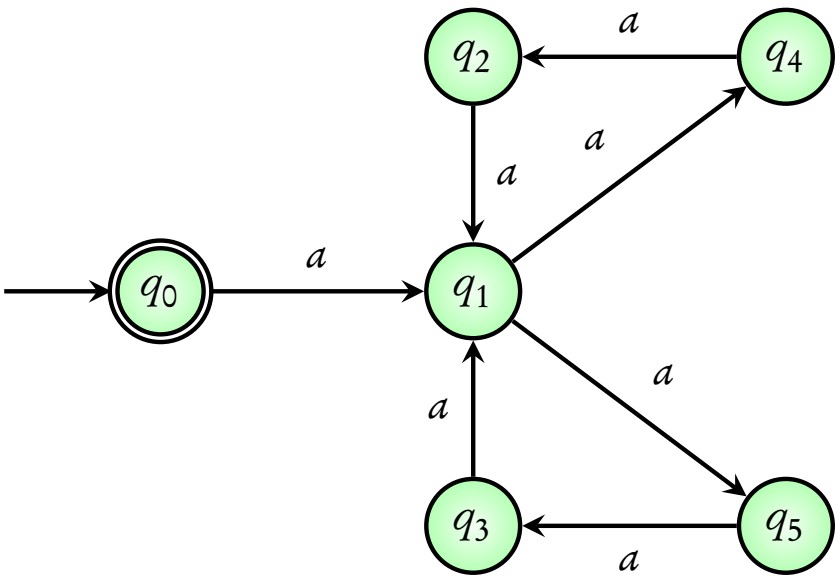
Is the **language** of above NBA **empty**? **No**



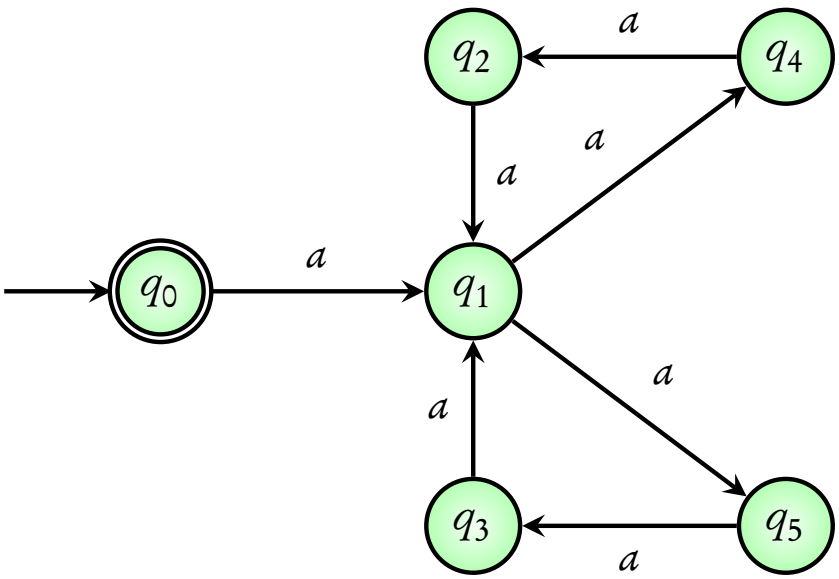
Is the **language** of above NBA **empty**?



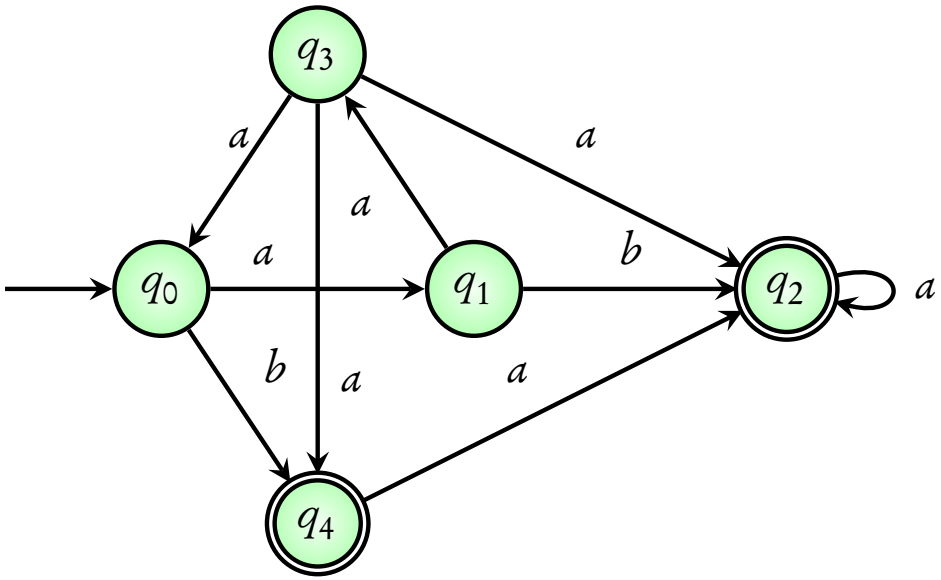
Is the **language** of above NBA **empty**? **No**



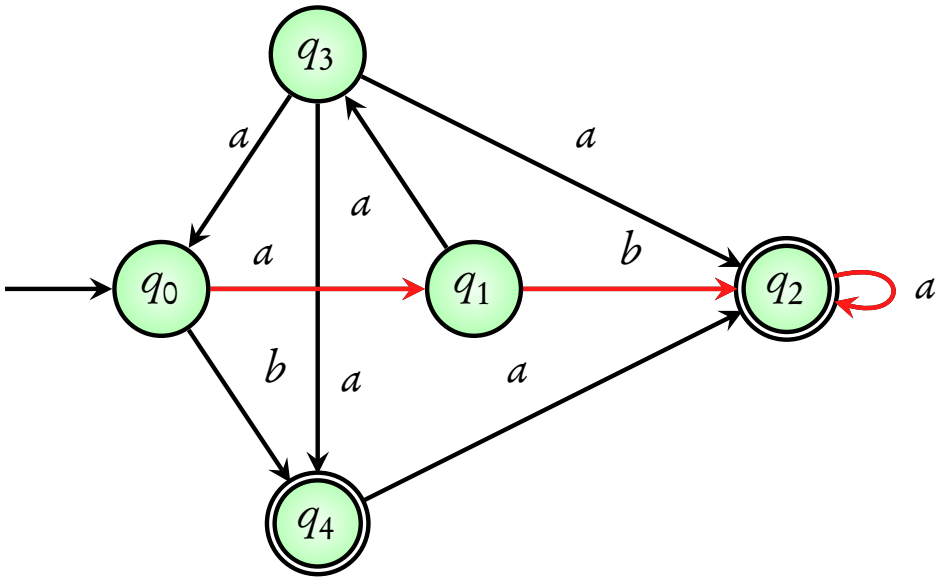
Is the **language** of above NBA **empty**?



Is the **language** of above NBA **empty**? **Yes**



Is the **language** of above NBA **empty**?



Is the **language** of above NBA **empty**? **No**

Main idea of algorithm

Find a **reachable cycle** in the automaton that contains an **accepting state**

Main idea of algorithm

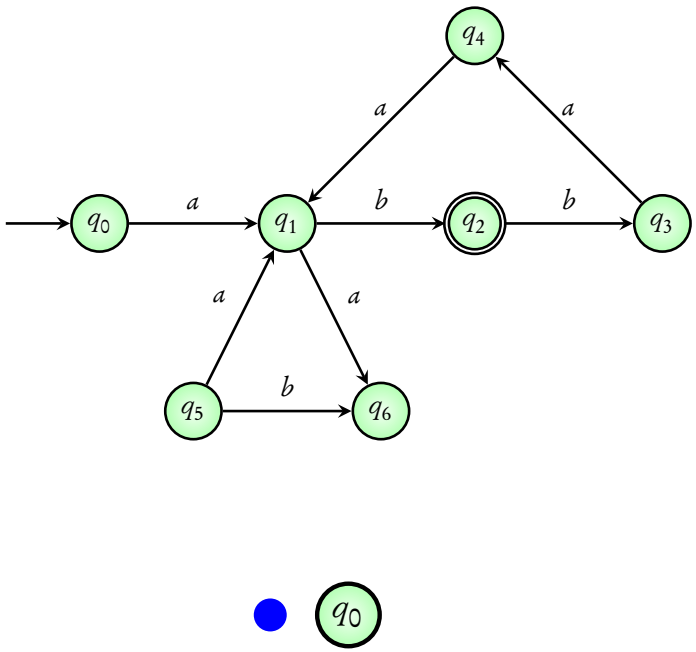
Find a **reachable cycle** in the automaton that contains an **accepting state**

- Do a preliminary DFS to get all **reachable states**
- From every **accepting state**, do a secondary DFS to see if it can **come back to itself**

Coming next: A **nested-DFS** algorithm

Courcoubetis, Vardi, Wolper, Yannakakis. Memory-efficient algorithms for the verification of temporal properties

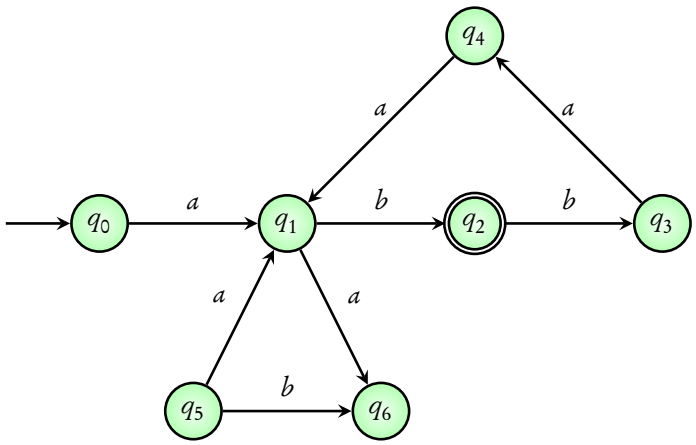
Formal Methods in System Design, 1992

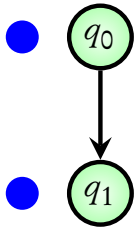


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

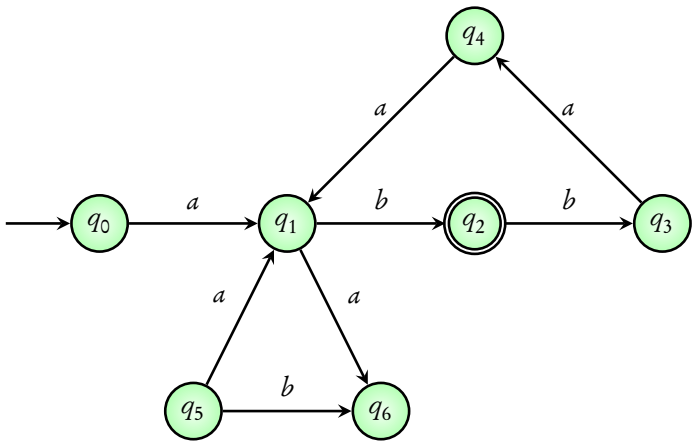


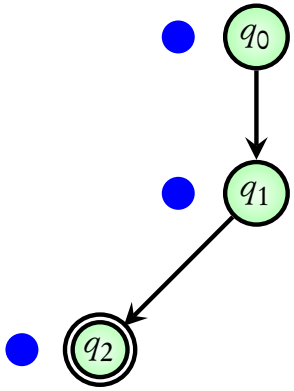


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

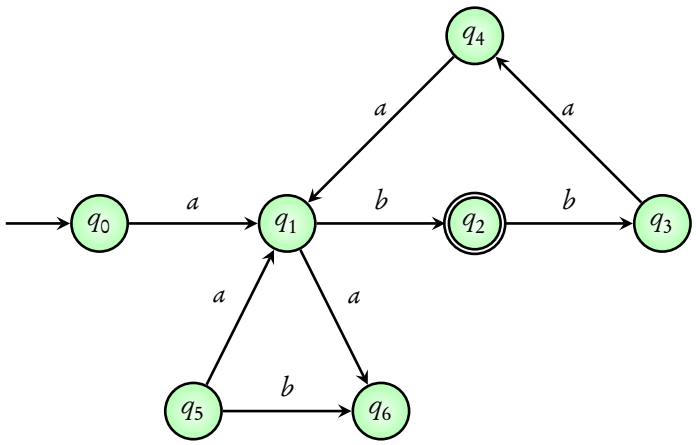


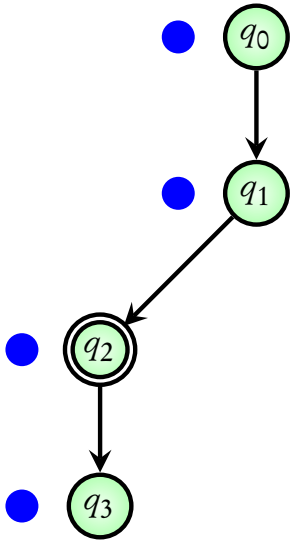


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

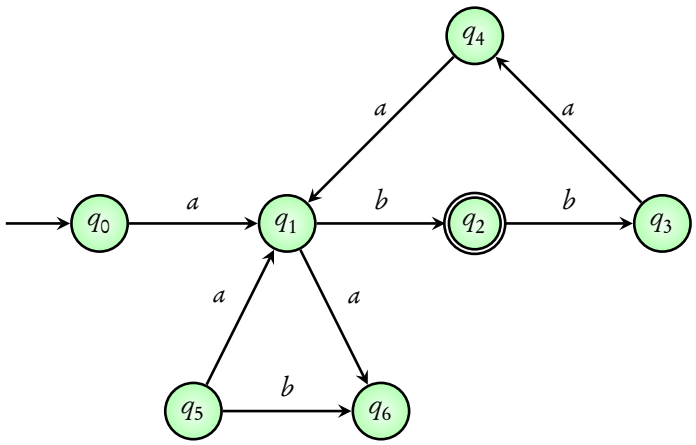


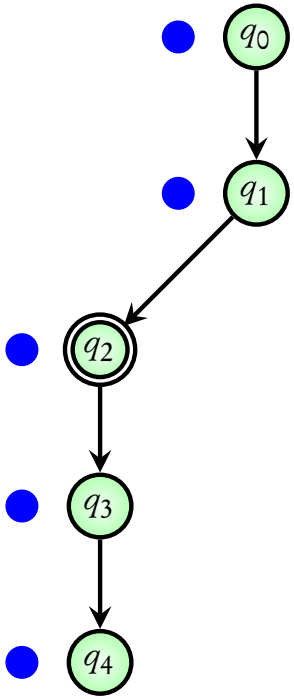


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

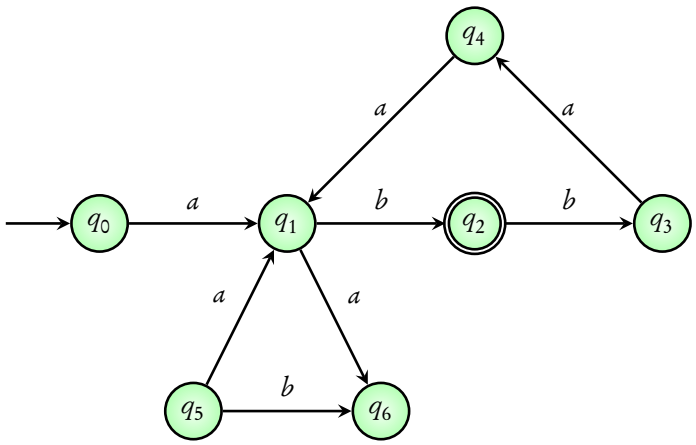


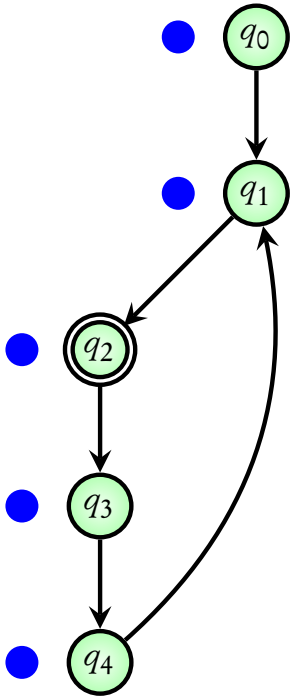


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

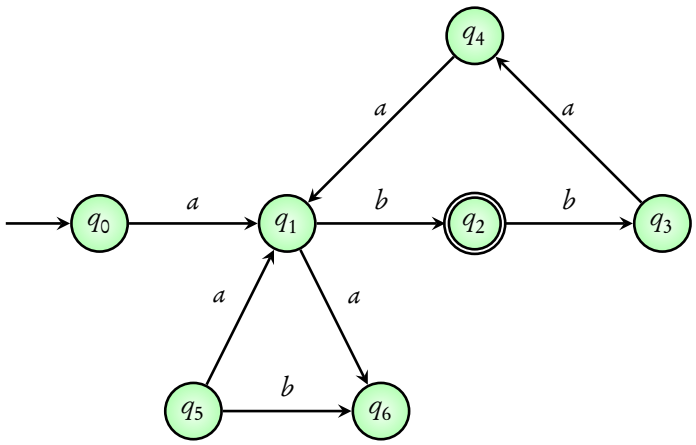


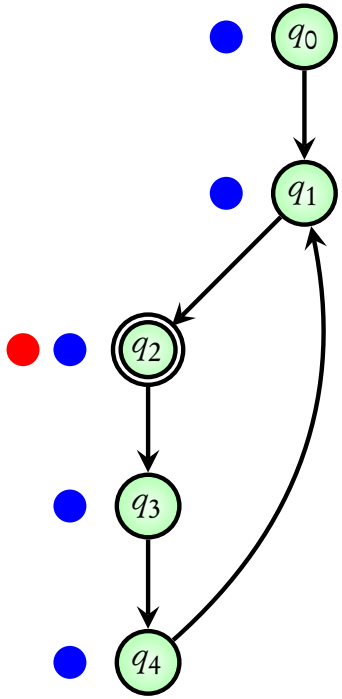


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

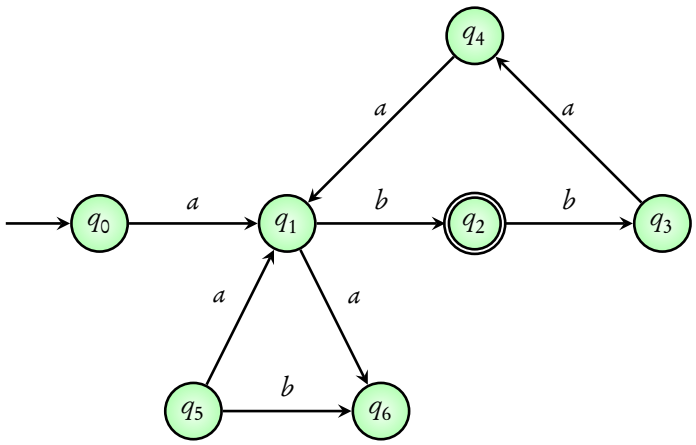


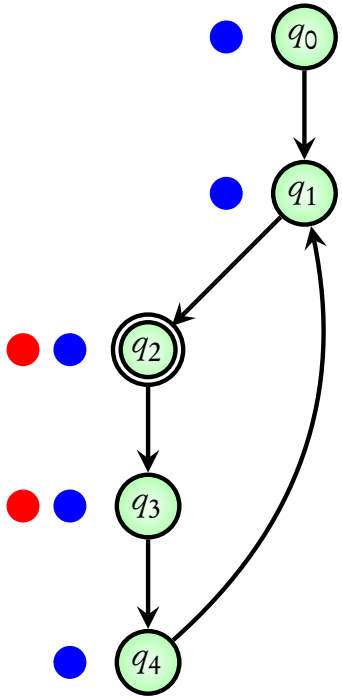


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

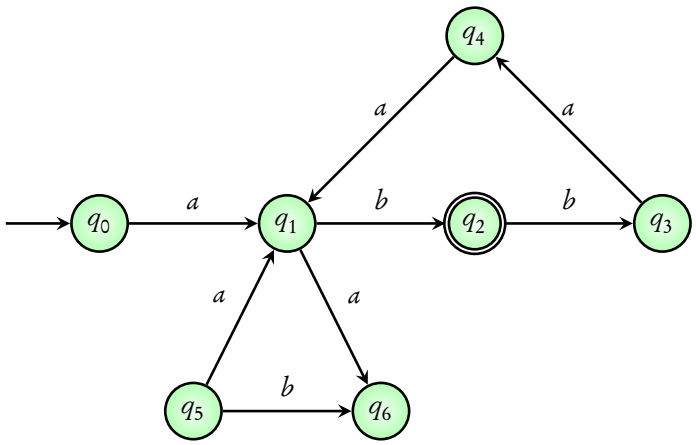


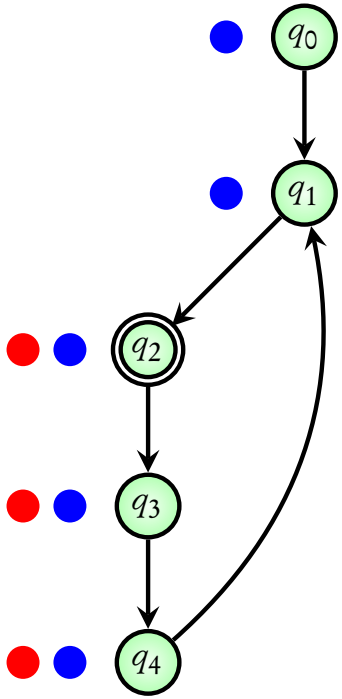


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

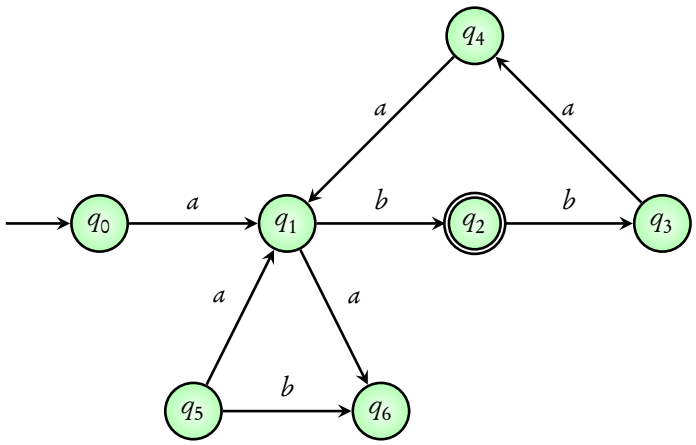


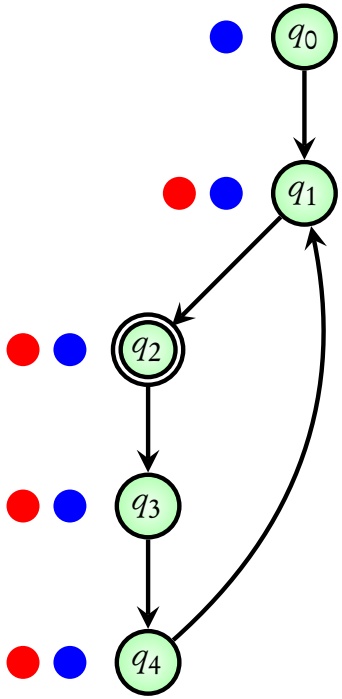


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

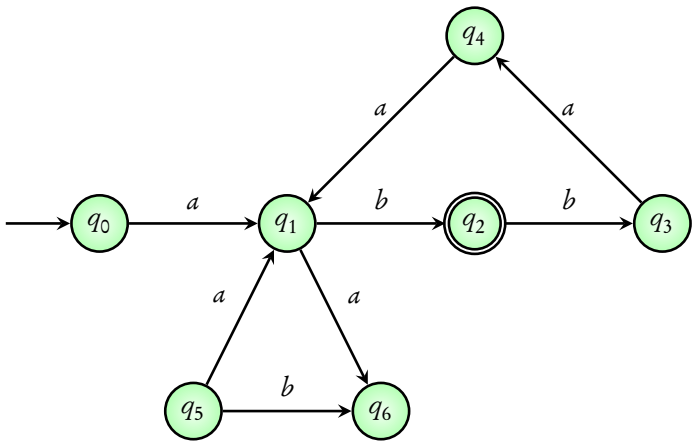


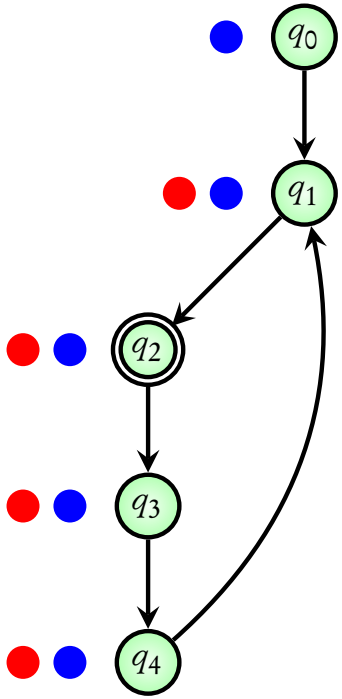


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle



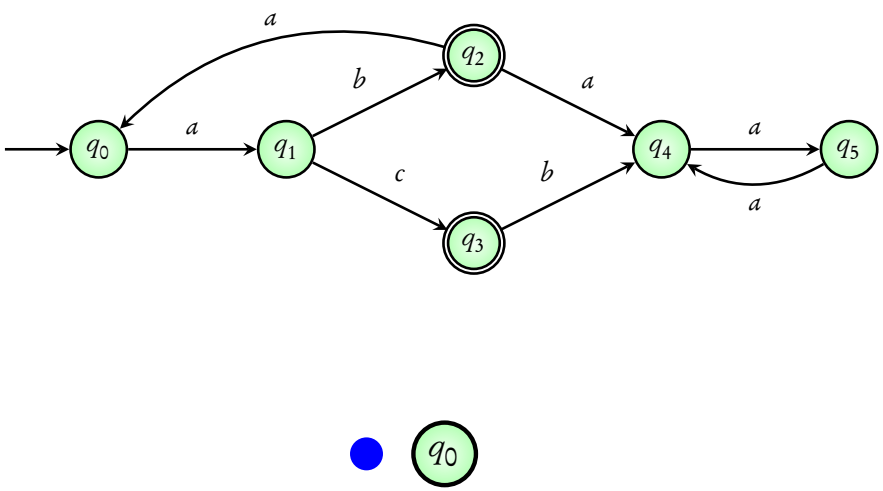


report cycle!

procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

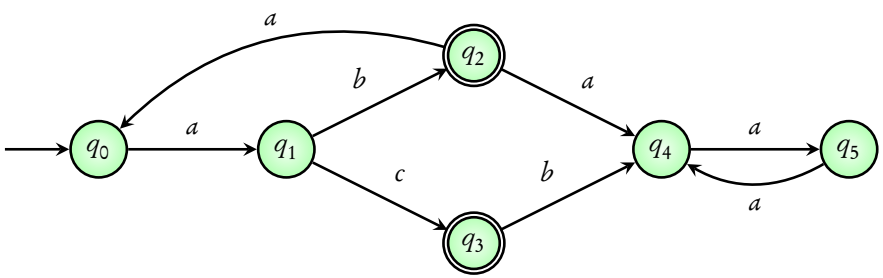
procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

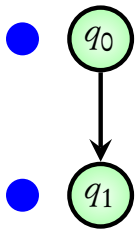


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

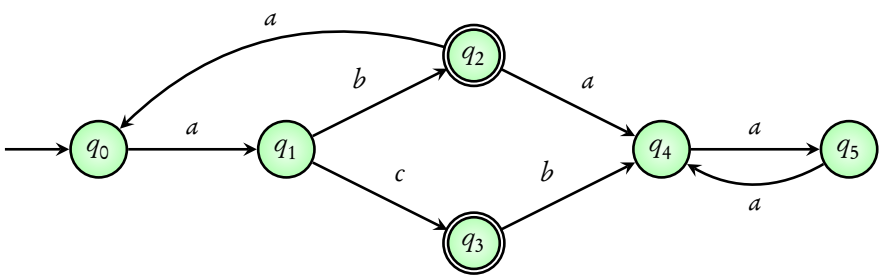


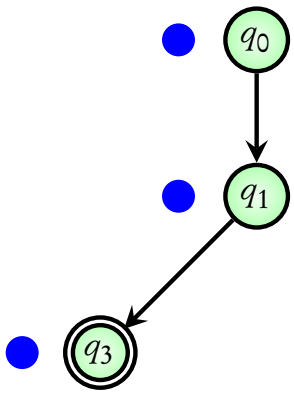


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

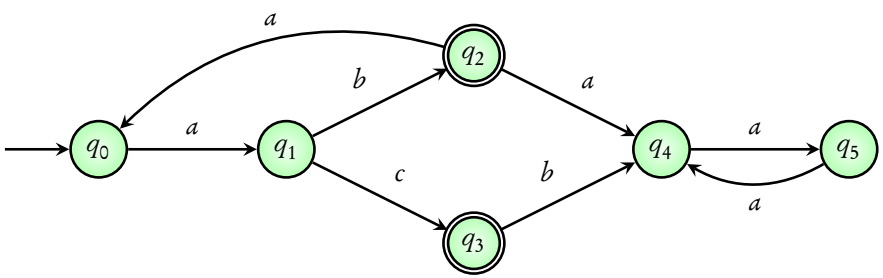


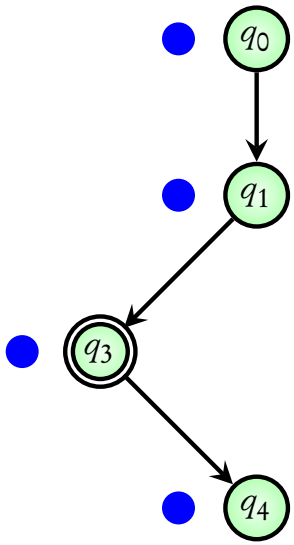


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

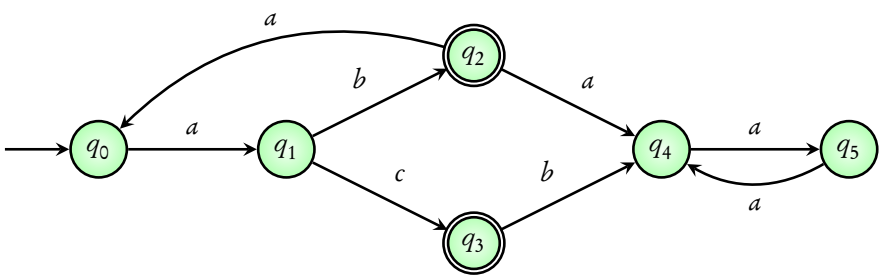


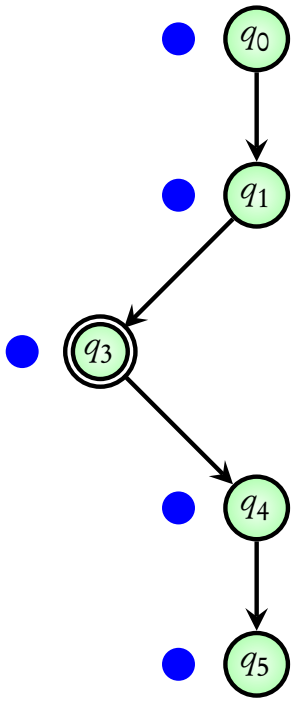


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

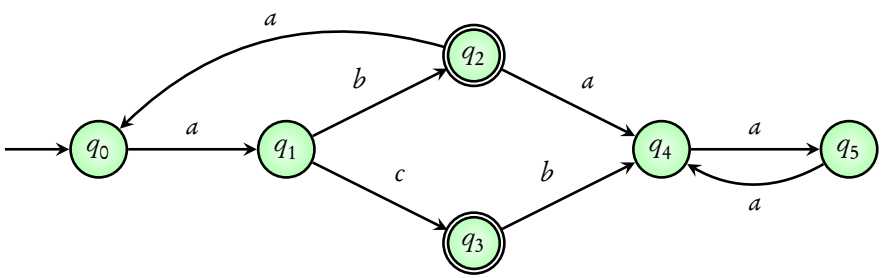


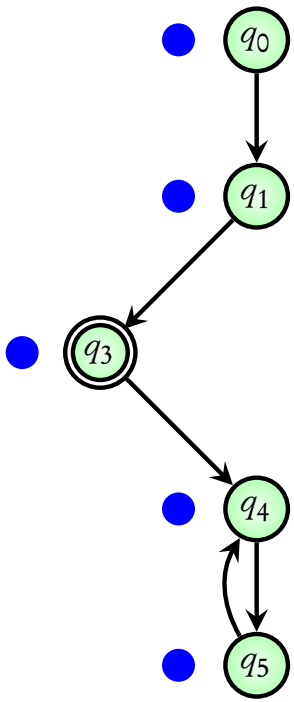


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed := s$
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

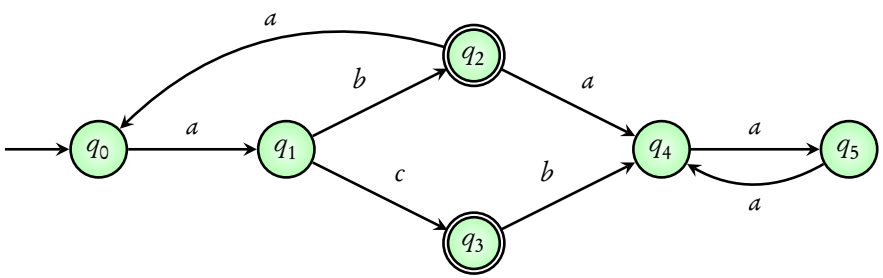


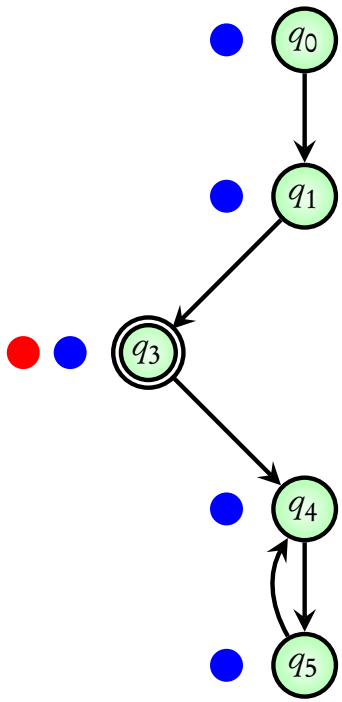


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

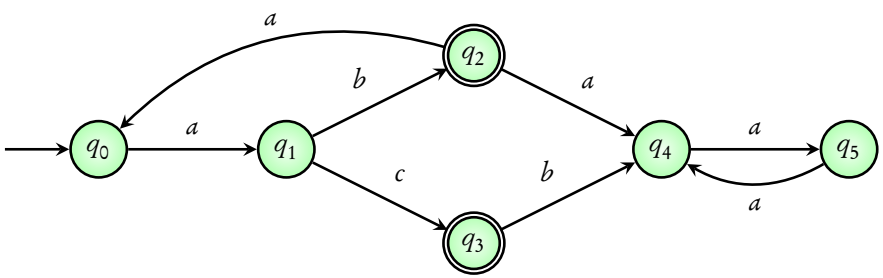


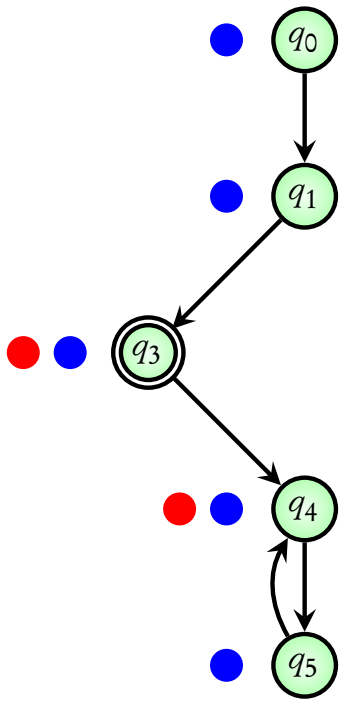


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

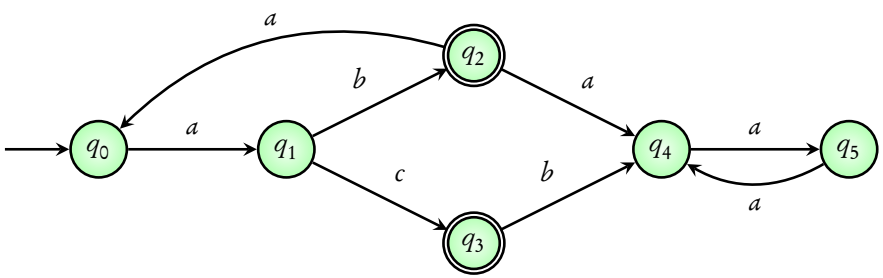


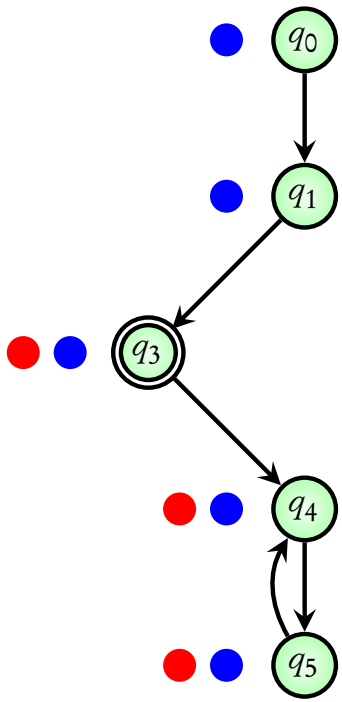


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

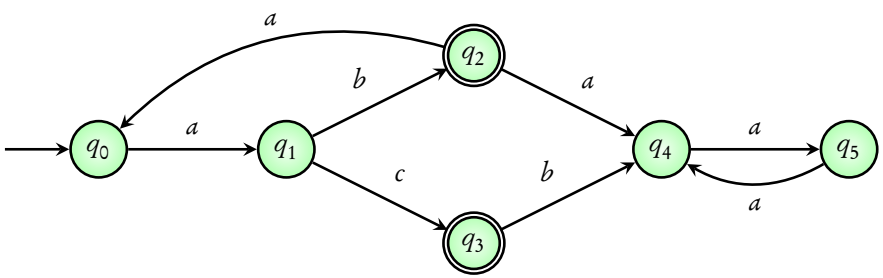


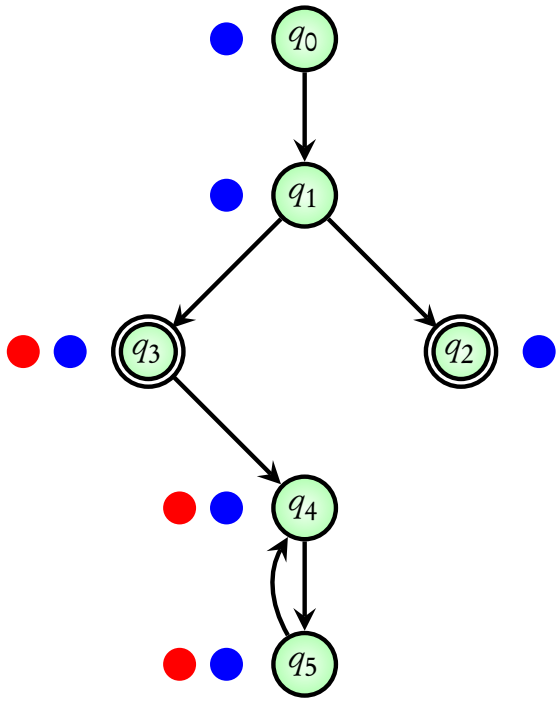


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

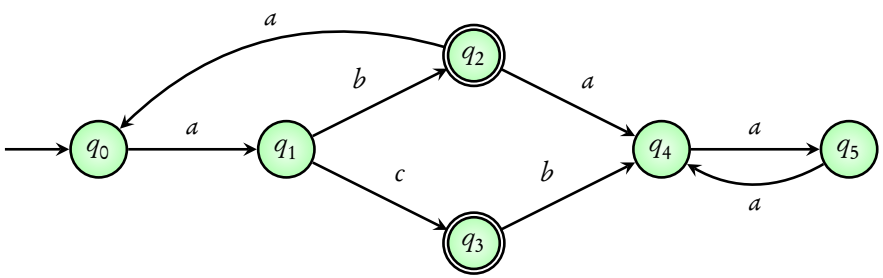


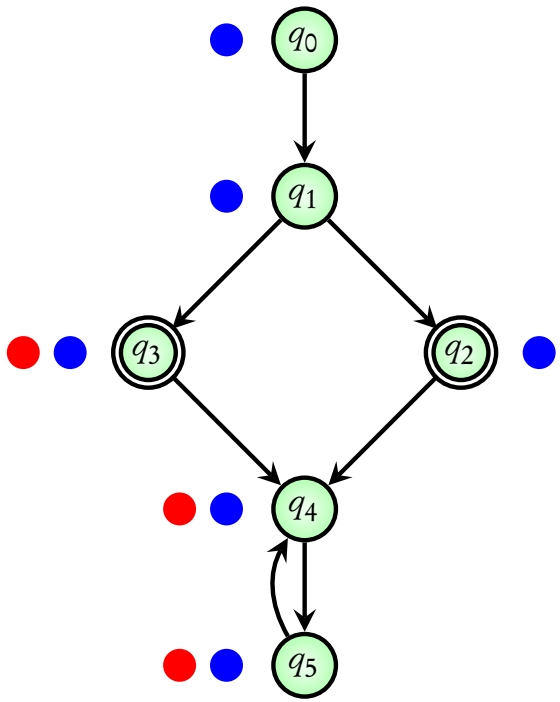


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

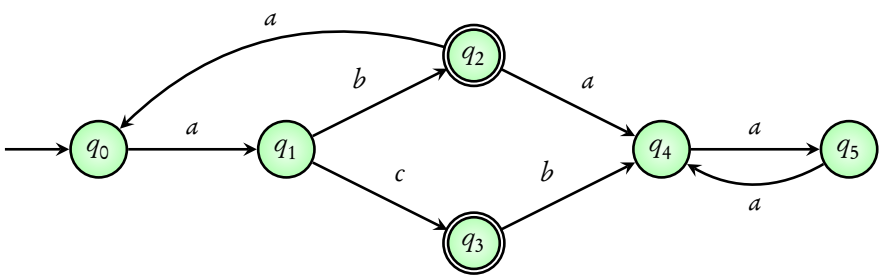


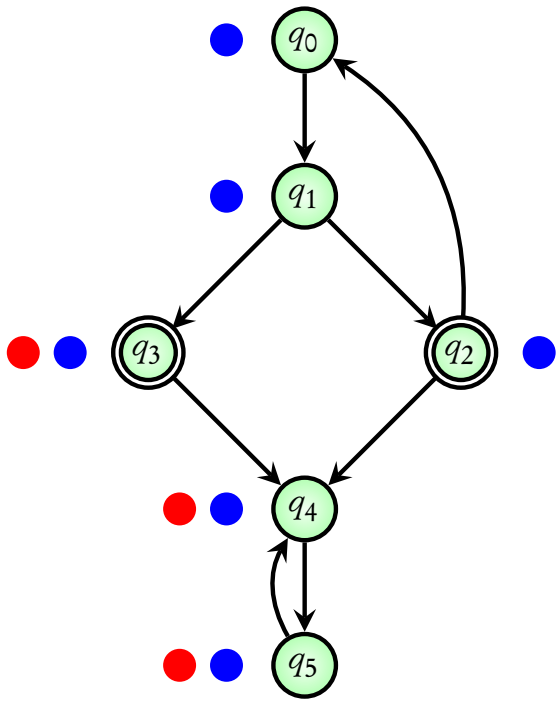


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

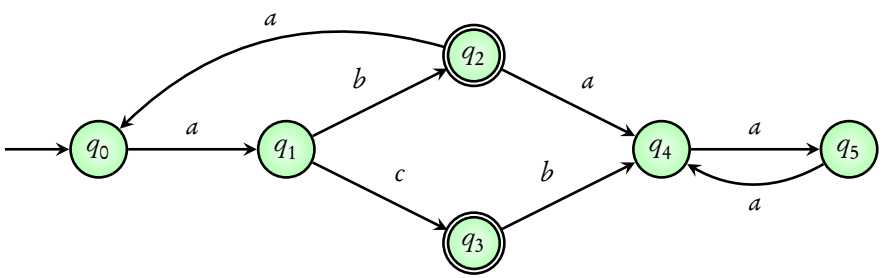


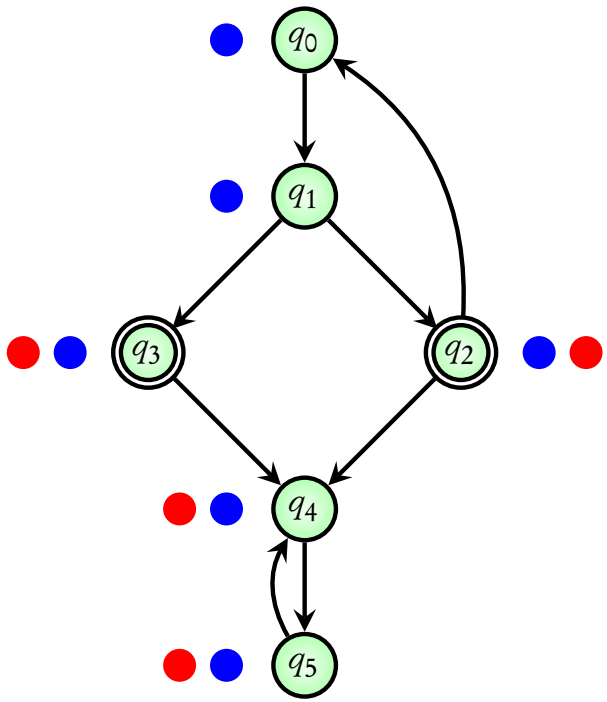


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

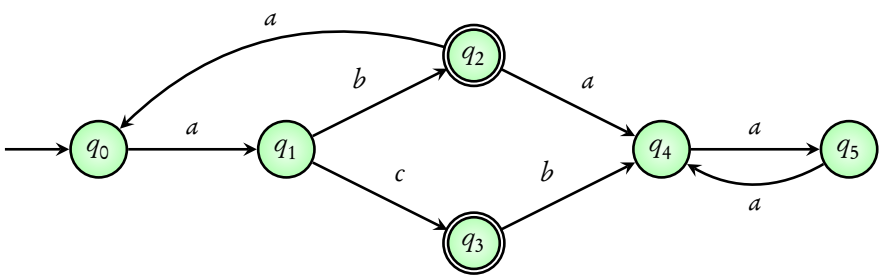


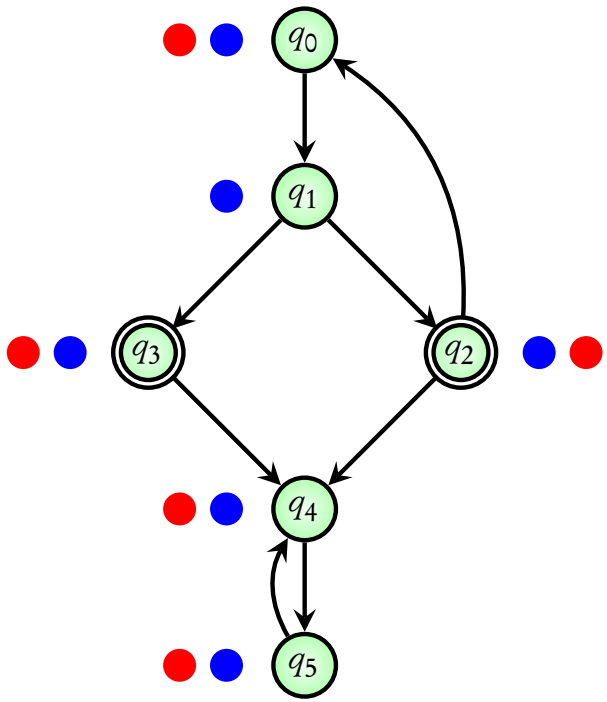


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

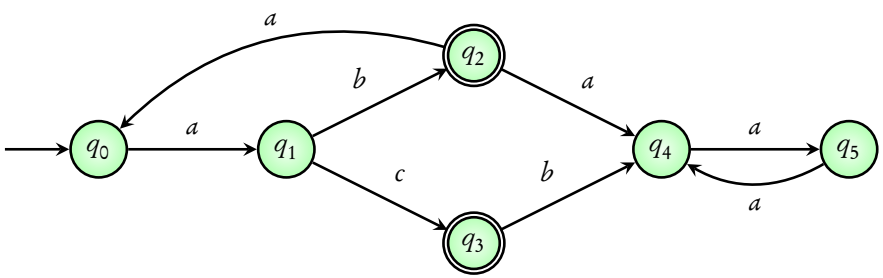


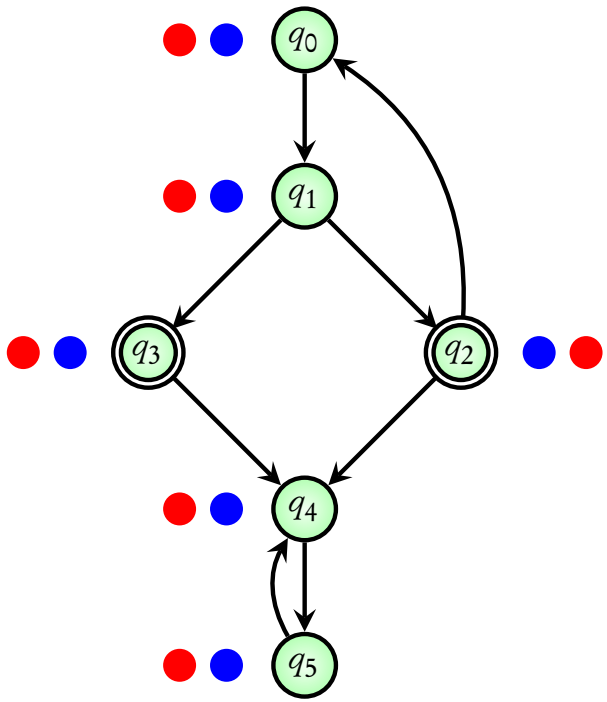


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

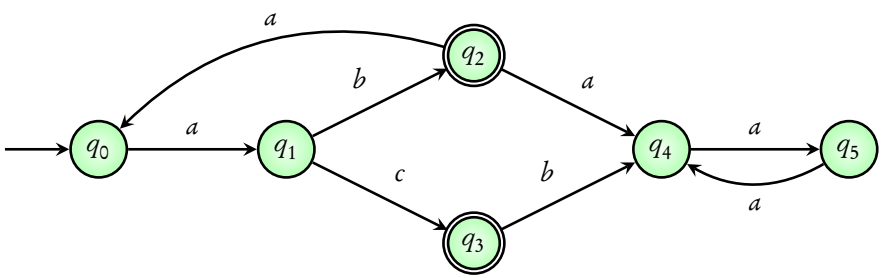


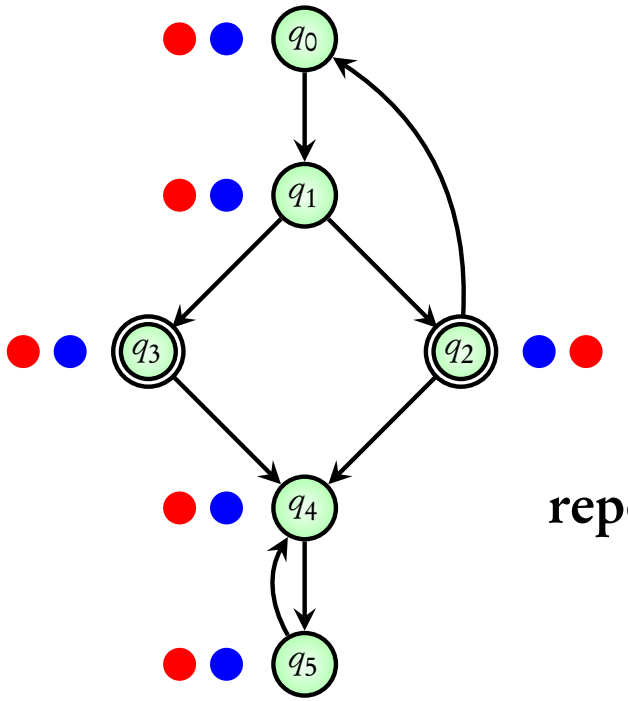


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle



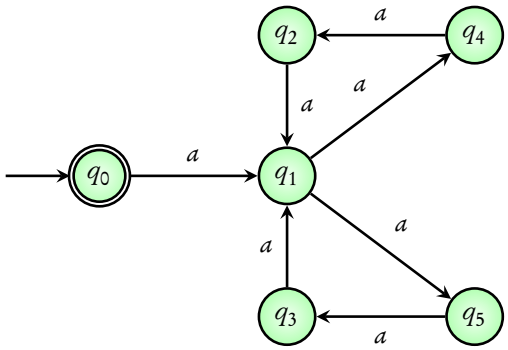


report cycle!

procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

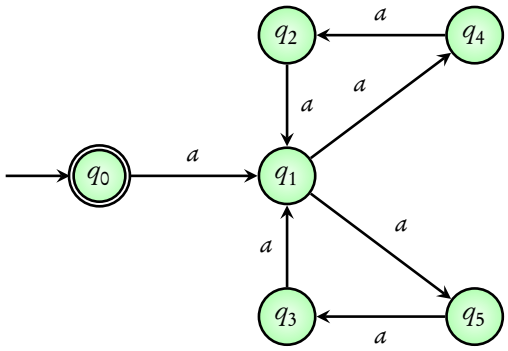
procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

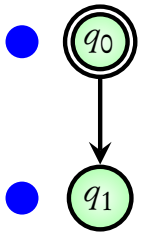



```
procedure nested_dfs( )
    call dfs_blue ( s0 )

procedure dfs_blue( s )
    s.blue := true
    for all t ∈ post( s ) do
        if ¬t.blue then
            call dfs_blue ( t )
    if s ∈ Accept then
        seed := s
        call dfs_red ( s )

procedure dfs_red ( s )
    s.red := true
    for all t ∈ post(s) do
        if ¬t.red then
            call dfs_red ( t )
        else if t = seed
            report cycle
```

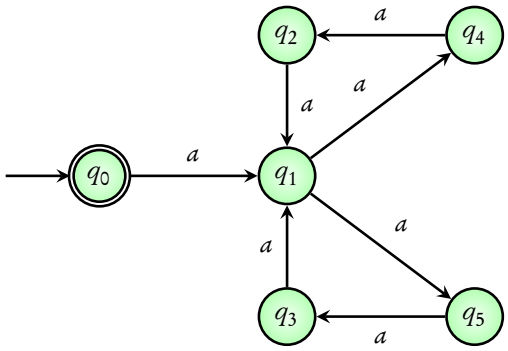


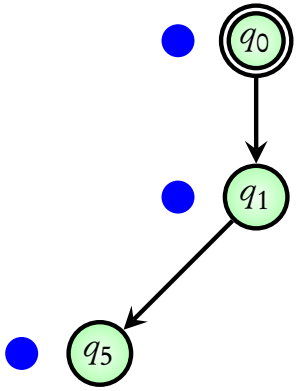


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

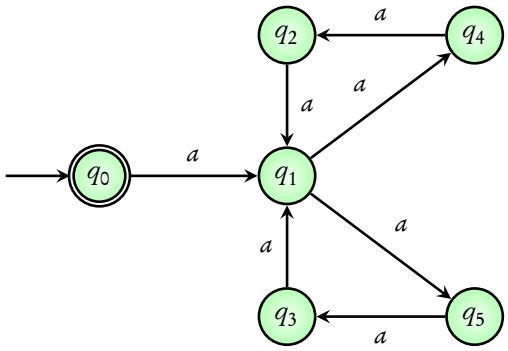


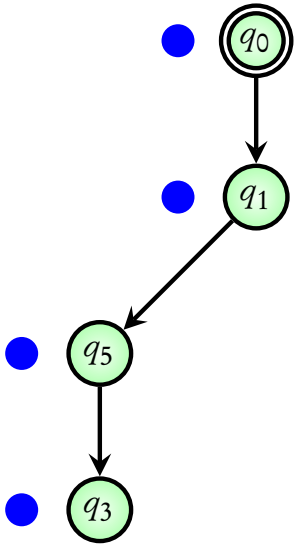


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

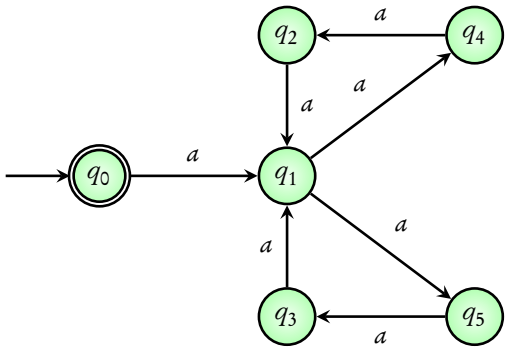


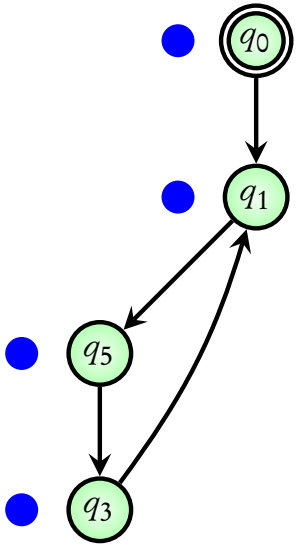


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

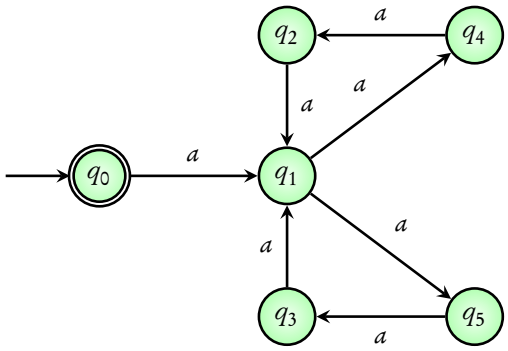


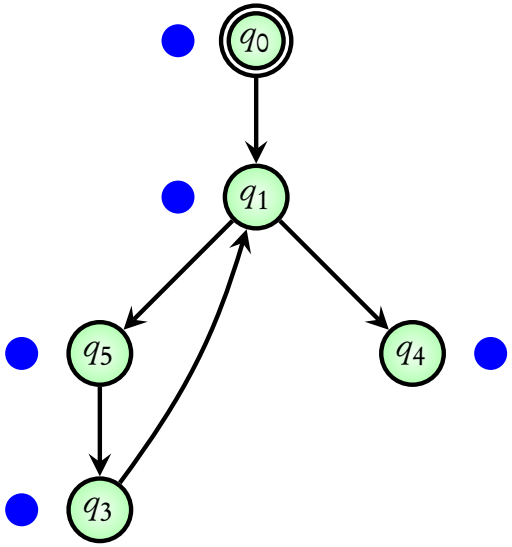


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.blue* **then**
      call *dfs_blue* (*t*)
  if $s \in$ *Accept* **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.red* **then**
      call *dfs_red* (*t*)
    else if $t =$ *seed*
      report cycle

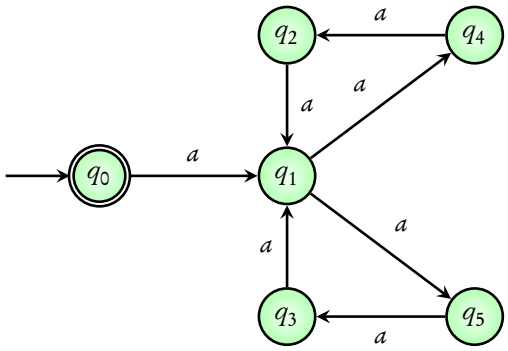


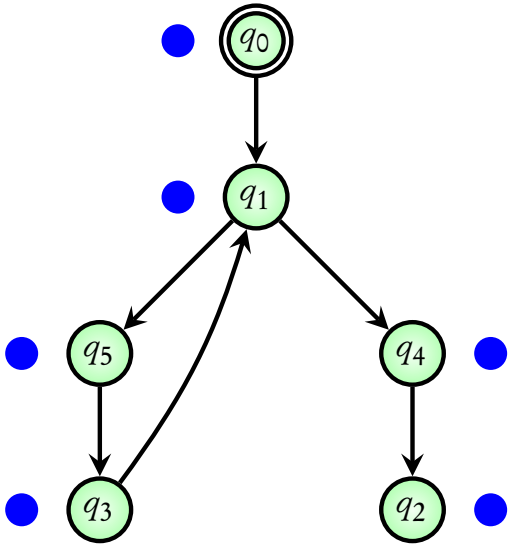


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

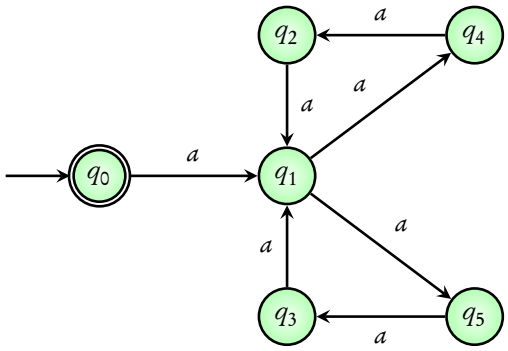


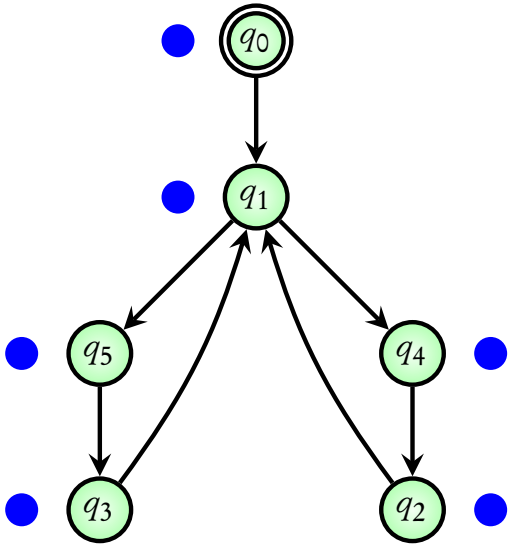


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

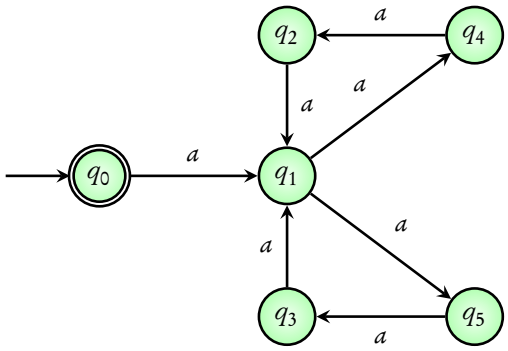


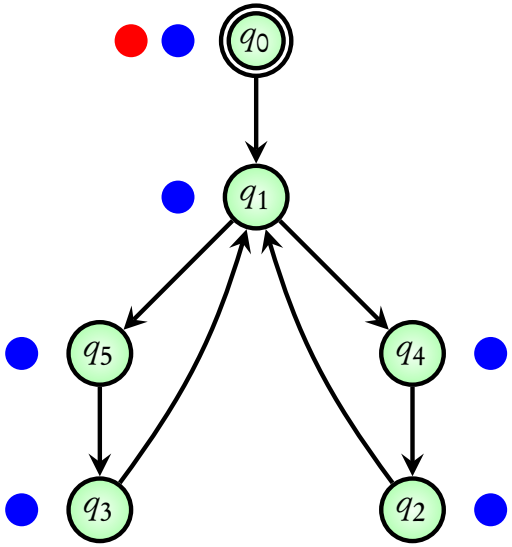


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

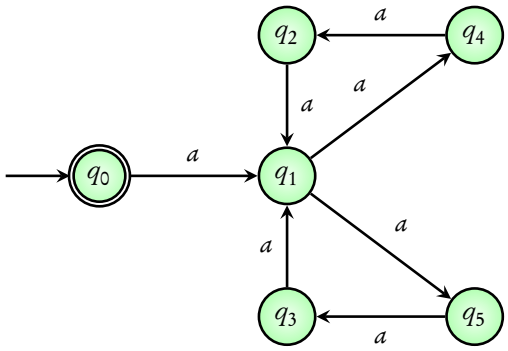


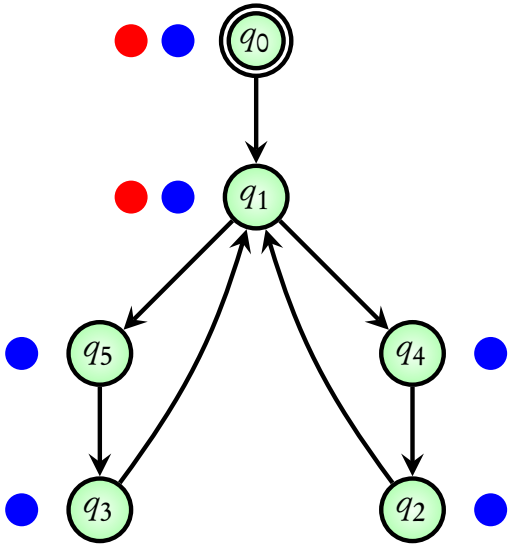


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

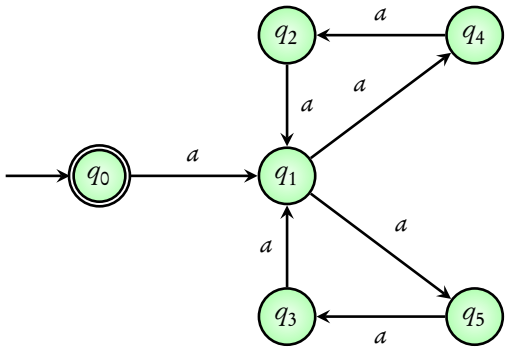


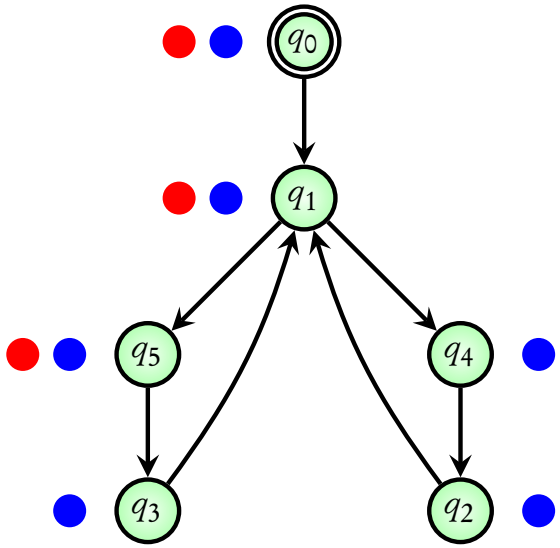


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

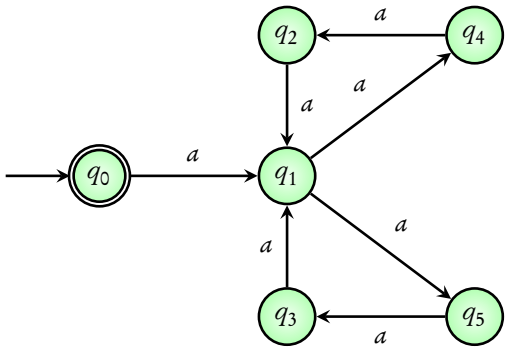


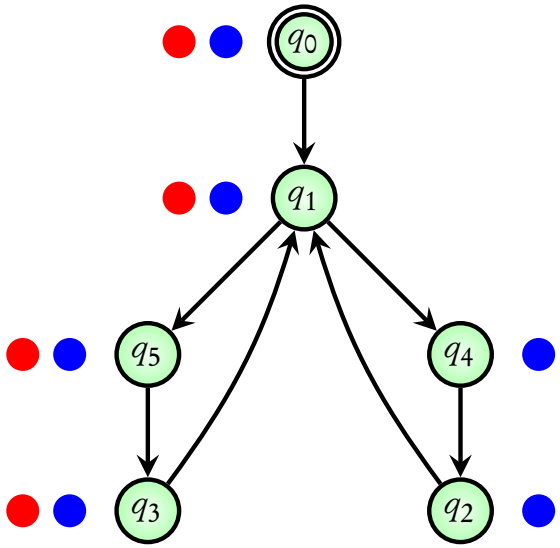


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.blue* **then**
      call *dfs_blue* (*t*)
  if $s \in$ *Accept* **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.red* **then**
      call *dfs_red* (*t*)
    else if $t =$ *seed*
      report cycle

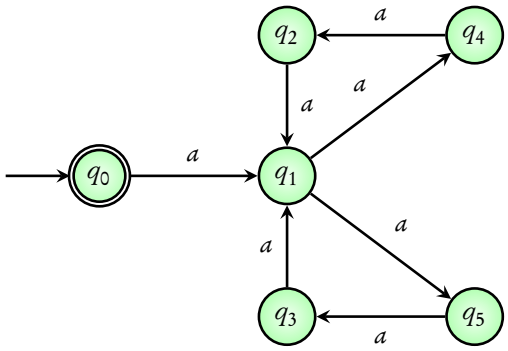


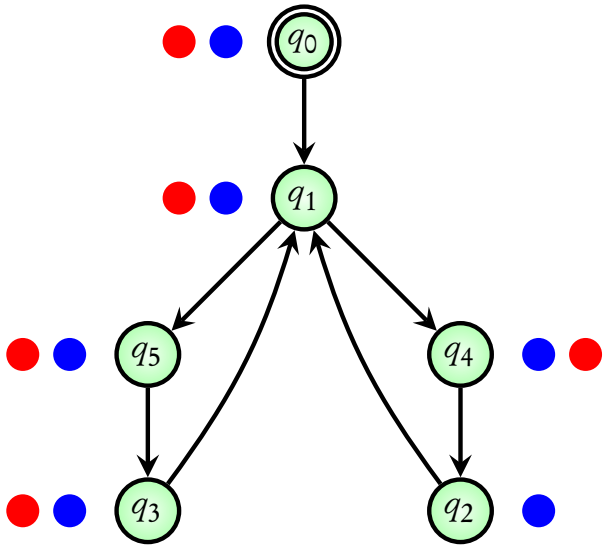


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.blue* **then**
      call *dfs_blue* (*t*)
  if $s \in$ *Accept* **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg$*t.red* **then**
      call *dfs_red* (*t*)
    else if $t =$ *seed*
      report cycle

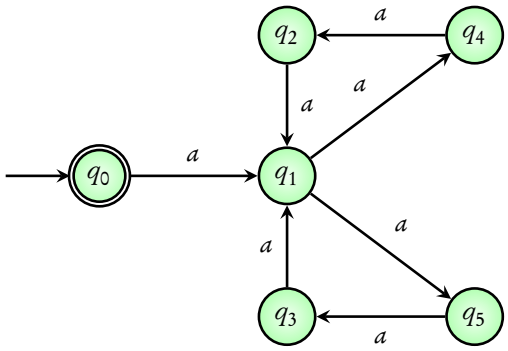


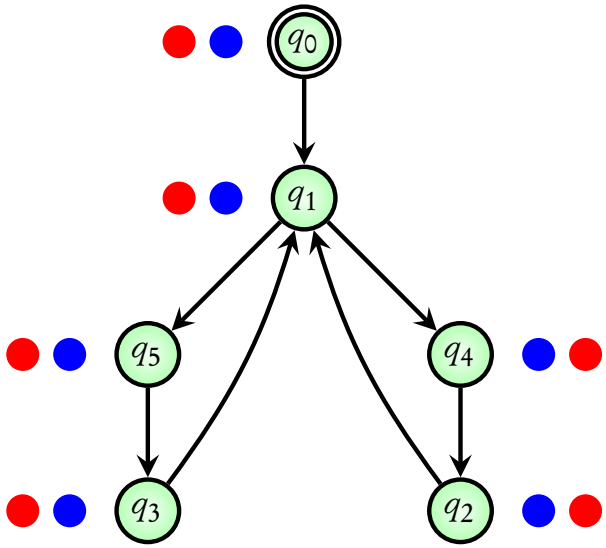


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

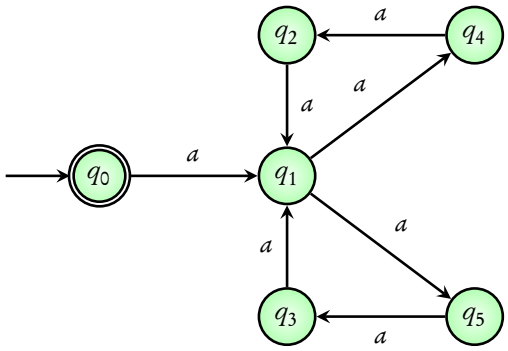


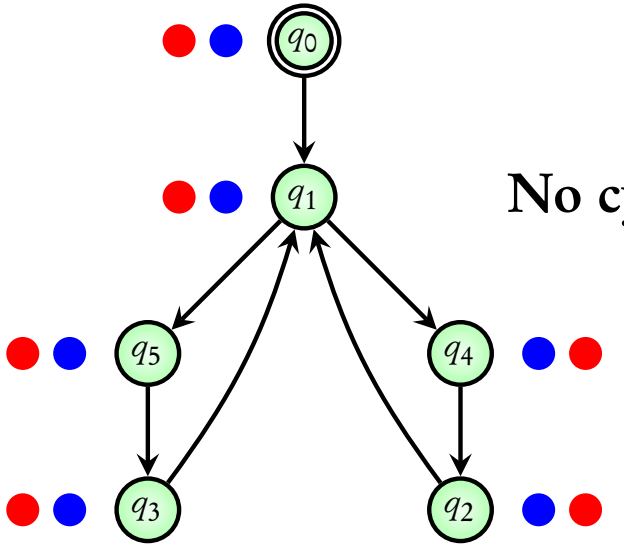


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle



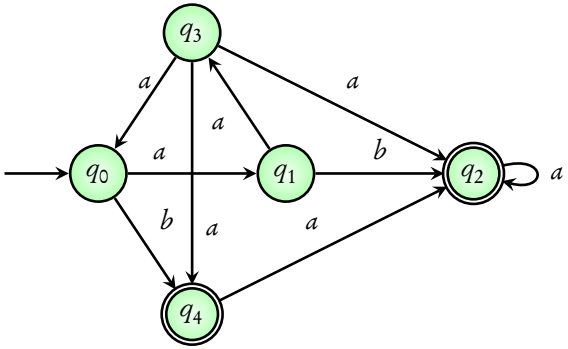


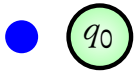
No cycle!

procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
  s.blue := **true**
  for all $t \in$ *post*(*s*) **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (*t*)
  if $s \in Accept$ **then**
    seed := *s*
    call *dfs_red* (*s*)

procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

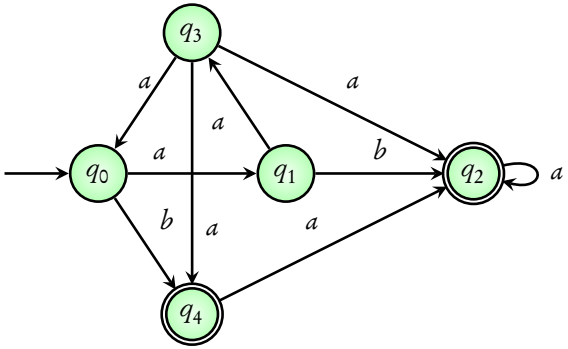


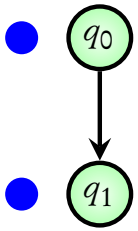


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

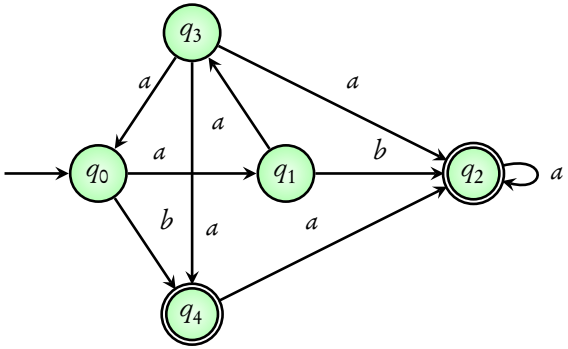


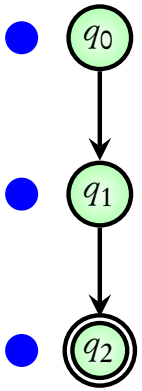


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

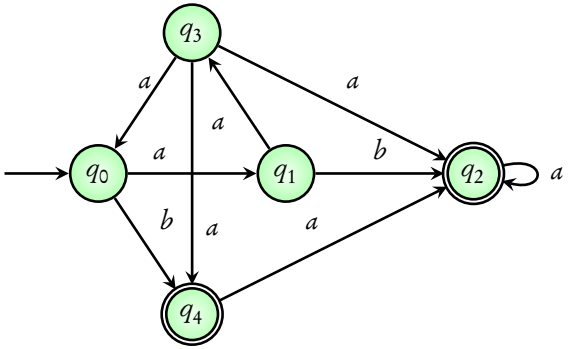


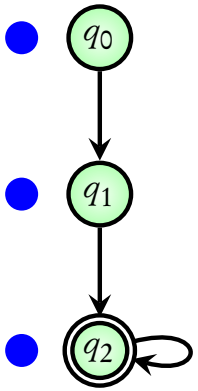


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

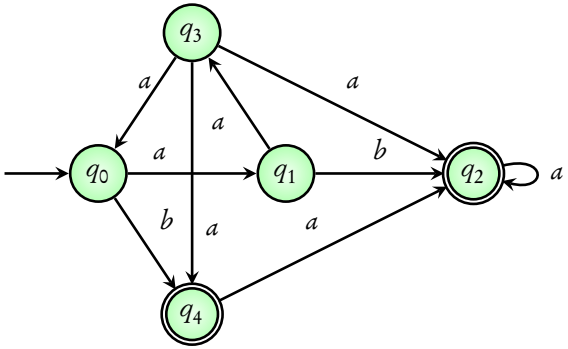


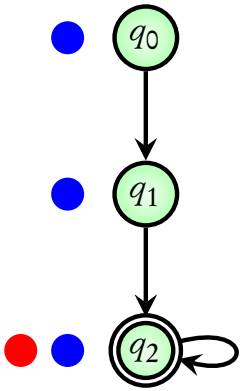


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

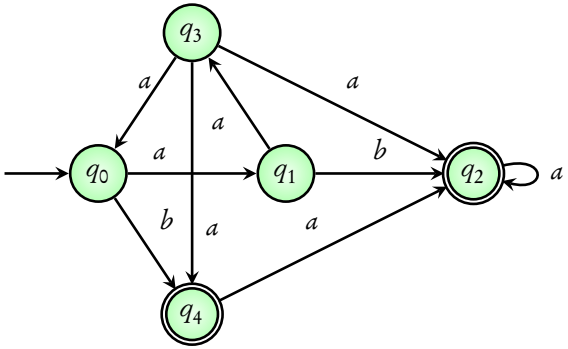


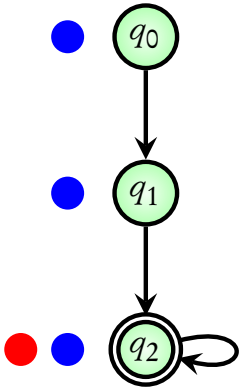


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle





report cycle!

procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

procedure *dfs_blue*(s)
  $s.blue$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.blue$ **then**
      call *dfs_blue* (t)
  if $s \in Accept$ **then**
    $seed$:= s
    call *dfs_red* (s)

procedure *dfs_red* (s)
  $s.red$:= **true**
  for all $t \in post(s)$ **do**
    if $\neg t.red$ **then**
      call *dfs_red* (t)
    else if $t = seed$
      report cycle

Does **Transition system** satisfy ω-regular property ?

ω-regular expression ϕ

NBA $\mathscr{A}_{T.S}$

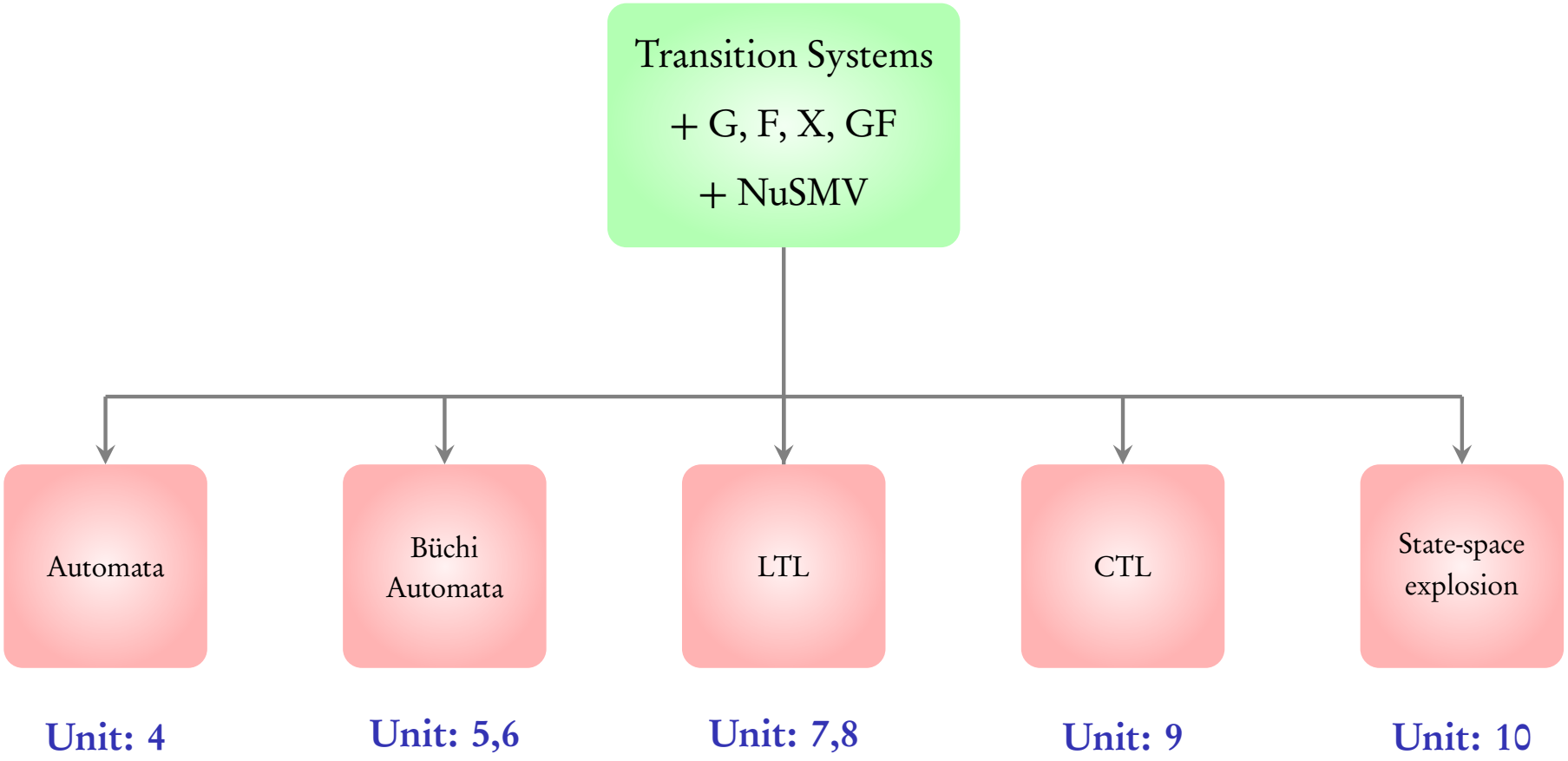
NBA $\mathscr{A}_{\phi}$

$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\,?$$

Is $L(\mathscr{A}_{T.S.}) \cap \overline{L(\mathscr{A}_{\phi})}$ empty ?

Is $L(\mathscr{A}_{T.S.}) \cap L(\overline{\mathscr{A}_{\phi}})$ empty ?

Is $L(\mathscr{A}_{T.S.} \times \overline{\mathscr{A}_{\phi}})$ empty ?


Transition Systems
+ G, F, X, GF
+ NuSMV
Automata
Büchi Automata
LTL
CTL
State-space explosion
Unit: 4
Unit: 5,6
Unit: 7,8
Unit: 9
Unit: 10

Unit-6: Model-checking ω-regular properties

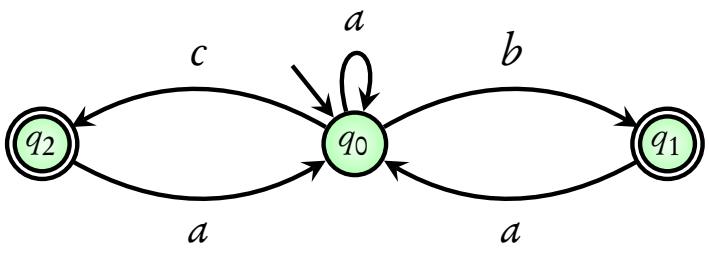
B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

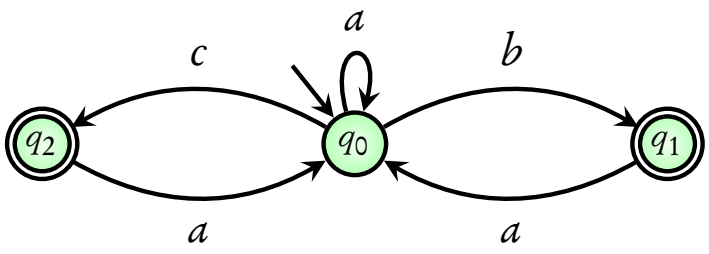
July - November 2015

Module 4:
Generalized Büchi Automata



$$(a^*(b+c)a)^\omega$$

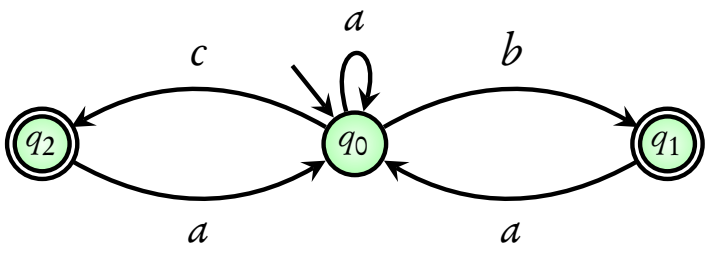
Accept states: $\{q_1, q_2\}$



$(a^*(b+c)a)^\omega$

Accept states: $\{q_1, q_2\}$

Above NBA also accepts *abababababab.....*



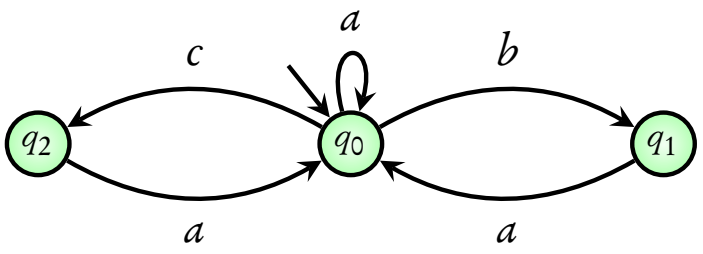
$(a^*(b+c)a)^\omega$

Accept states: $\{q_1, q_2\}$

Above NBA also accepts *abababababab.....*

Suppose we want NBA for **subset** of $(a^*(b+c)a)^\omega$ where

both b and c occur infinitely often



$(a^*(b+c)a)^\omega$

~~Accept states: $\{q_1, q_2\}$~~

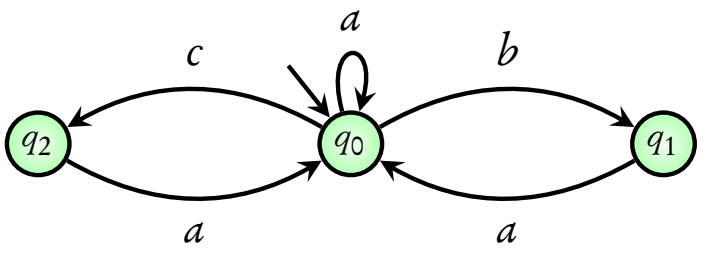
Above NBA also accepts *abababababab.....*

Suppose we want NBA for **subset** of $(a^*(b+c)a)^\omega$ where

both b and c occur infinitely often

Modified accepting condition: $\{\ \{q_1\},\ \{q_2\}\ \}$

Generalized NBA



$(a^*(b+c)a)^\omega$

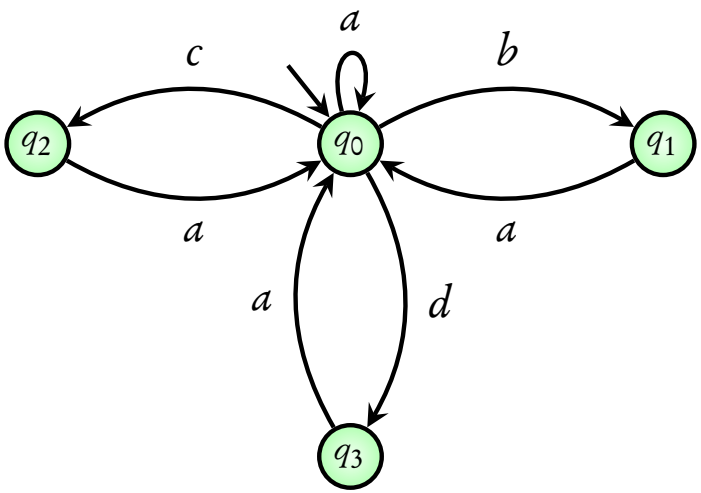
~~Accept states: $\{q_1, q_2\}$~~

Above NBA also accepts *abababababab.....*

Suppose we want NBA for **subset** of $(a^*(b+c)a)^\omega$ where

both b and c occur infinitely often

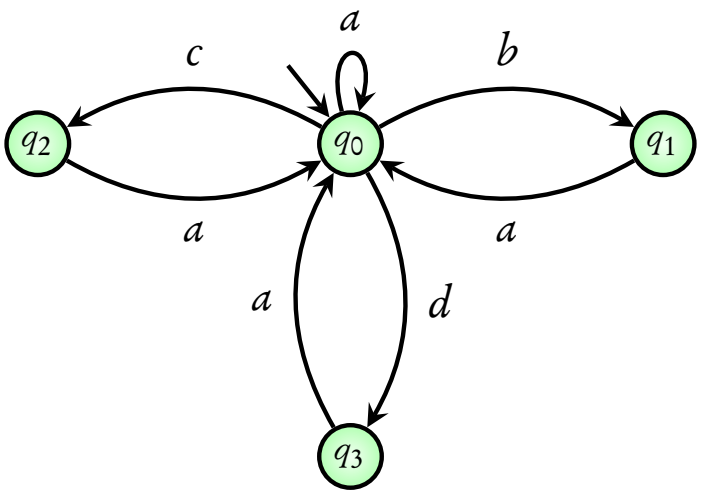
Modified accepting condition: $\{\ \{q_1\},\ \{q_2\}\ \}$



Get GNBA for subset of $(a^*(b+c+d)a)^\omega$ where:

d occurs infinitely often **and**

either b **or** c occur infinitely often



Get GNBA for subset of $(a^*(b+c+d)a)^\omega$ where:

d occurs infinitely often **and**

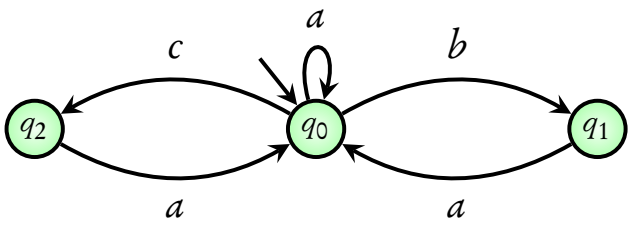
either b **or** c occur infinitely often

Accepting condition: $\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

Generalized Büchi Automata

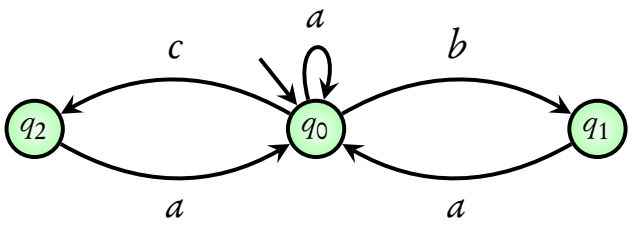
- States, transitions, initial states as in an NBA
- Accepting condition: $\{ F_1, F_2, \ldots, F_k \}$
- Run is accepting if **some state from each of the** F_i occurs infinitely often

GNBA

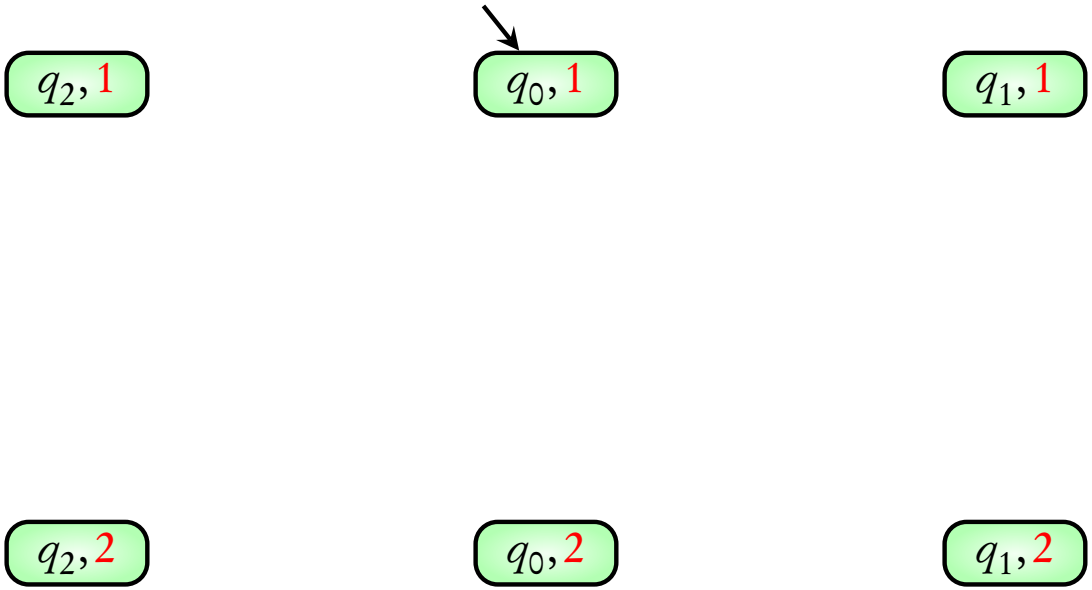


$\{\ \{q_1\},\ \{q_2\}\ \}$

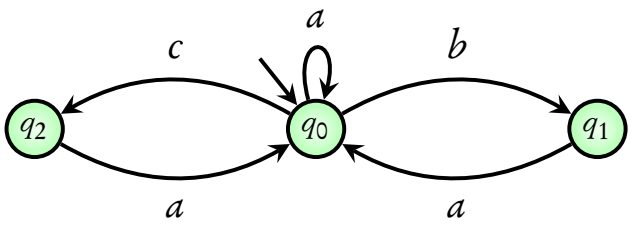
GNBA



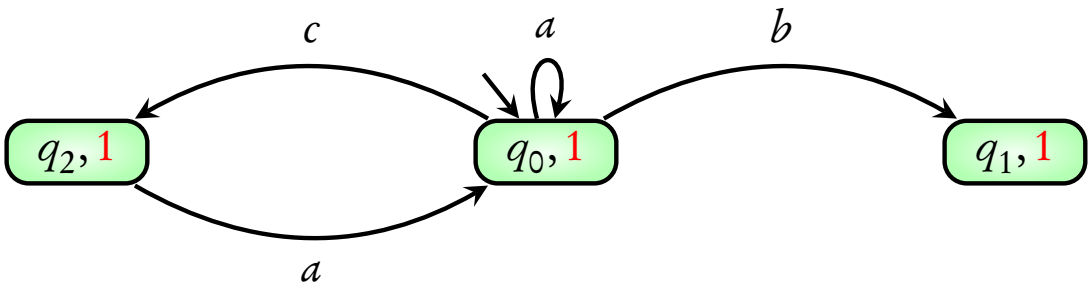
$\{\ \{q_1\},\ \{q_2\}\ \}$

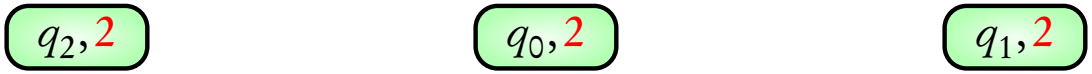


GNBA

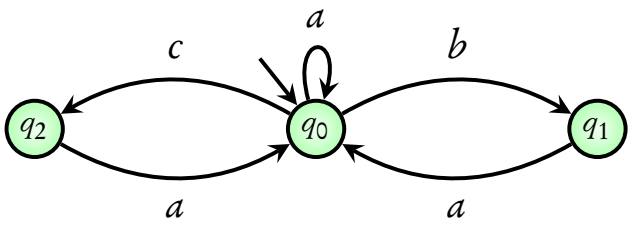


$\{\ \{q_1\},\ \{q_2\}\ \}$

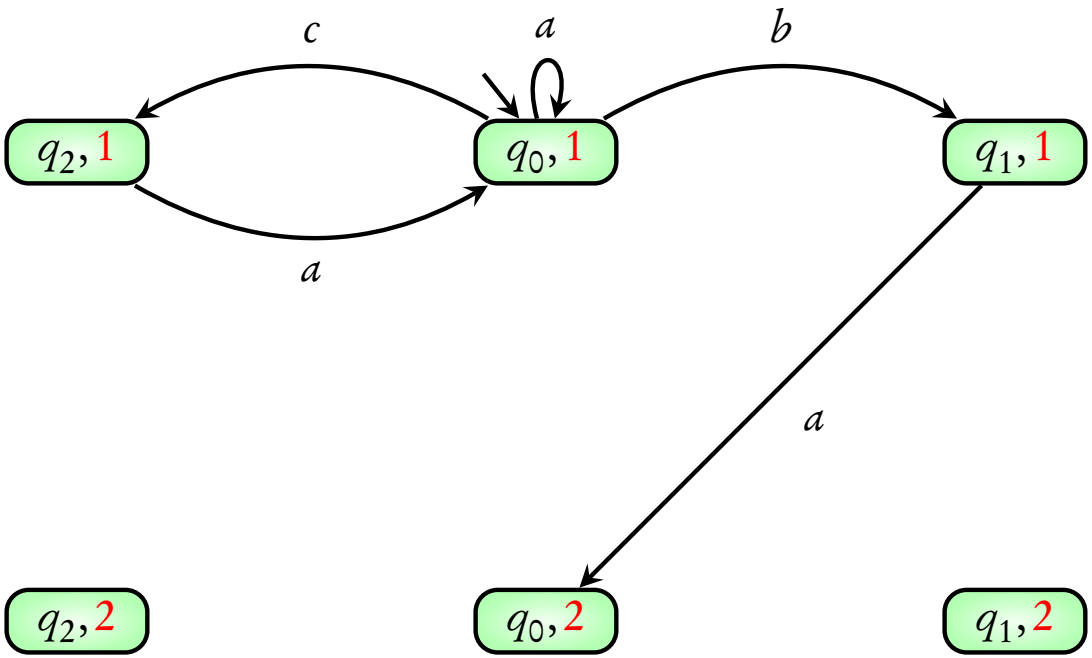




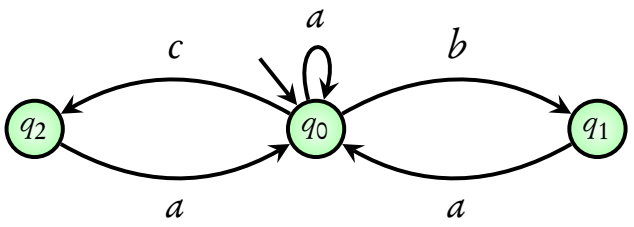
GNBA



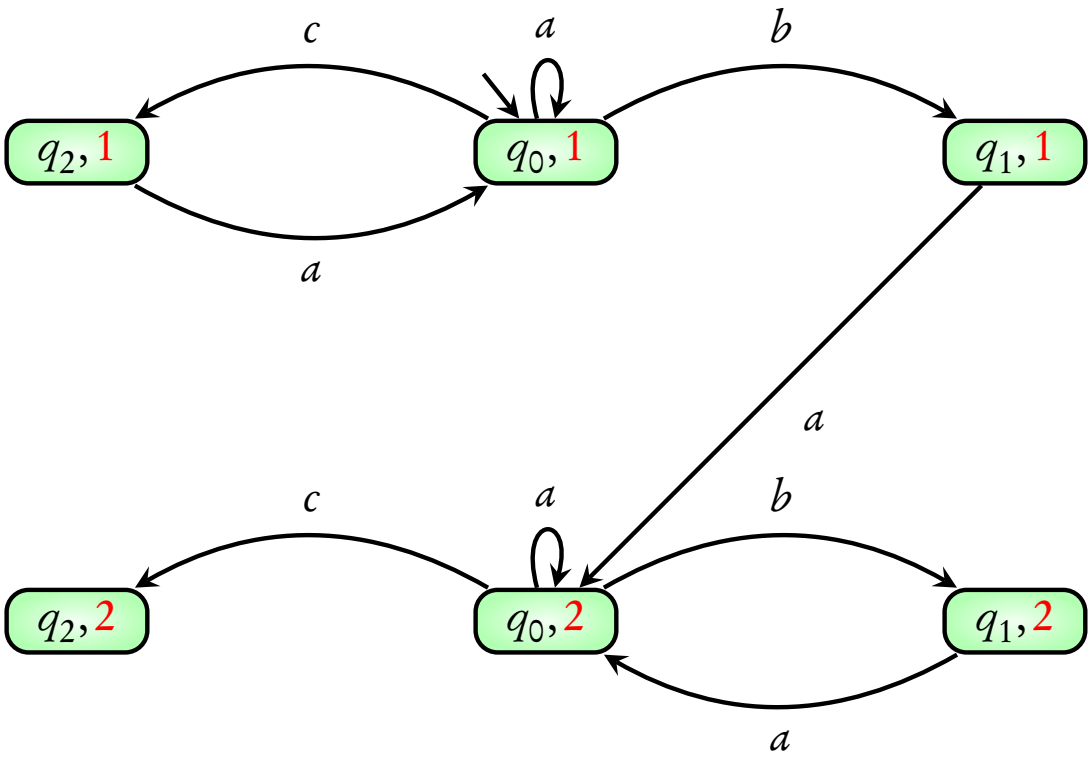
$\{\ \{q_1\},\ \{q_2\}\ \}$



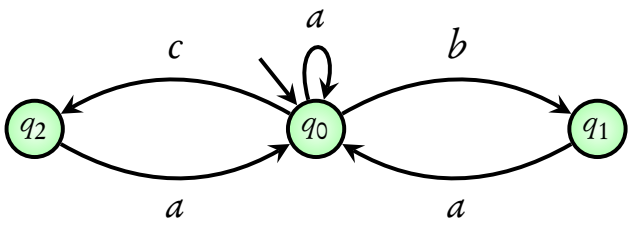
GNBA



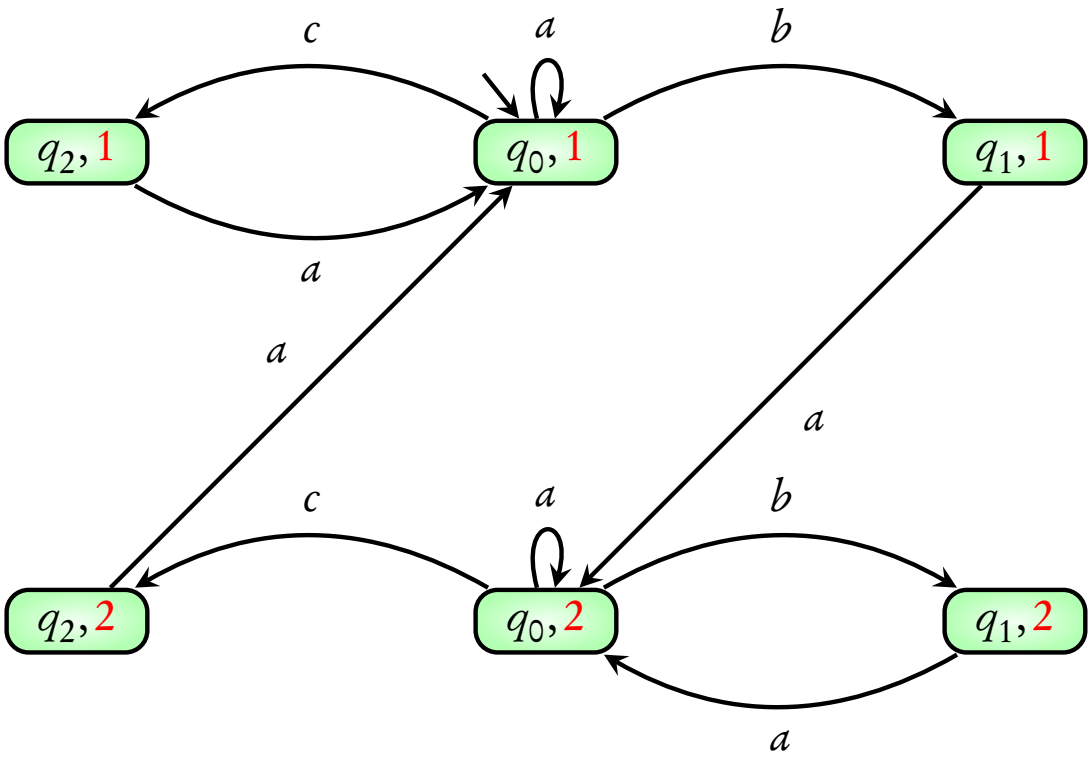
$\{\ \{q_1\},\ \{q_2\}\ \}$



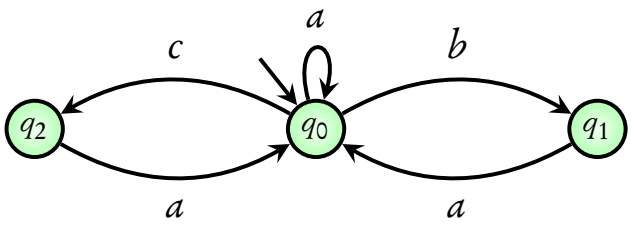
GNBA



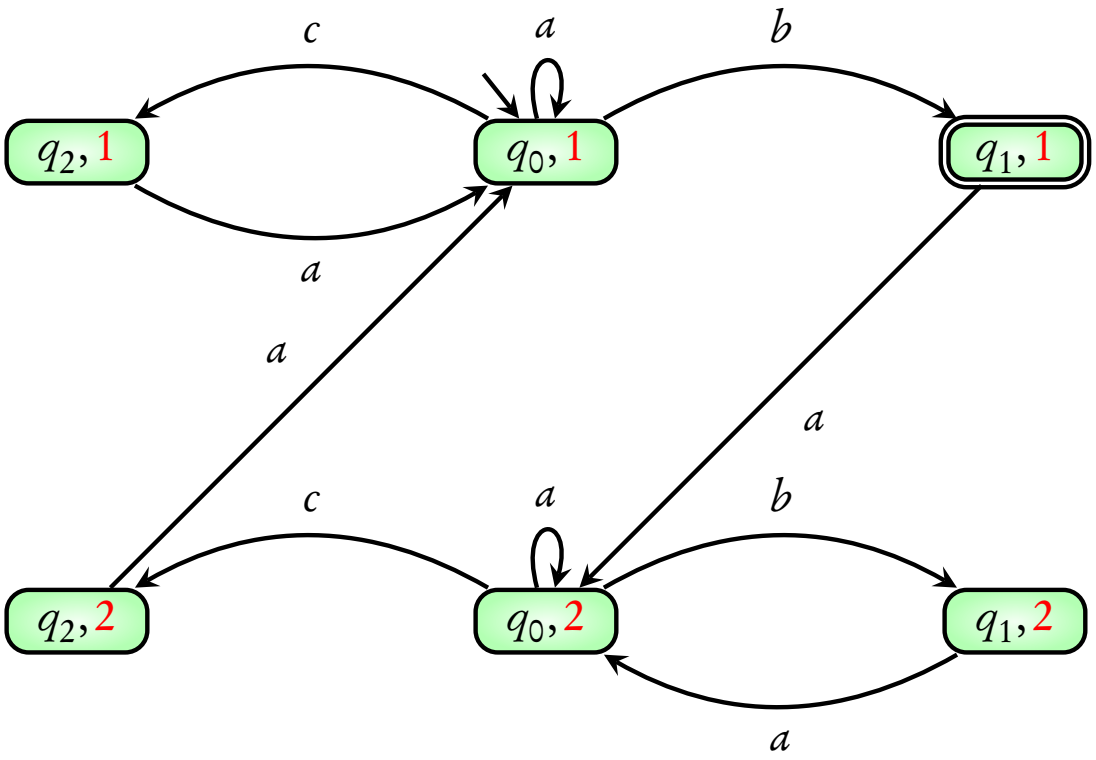
$\{\ \{q_1\},\ \{q_2\}\ \}$



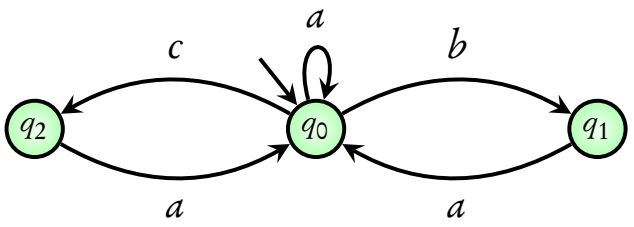
GNBA



$\{\ \{q_1\},\ \{q_2\}\ \}$

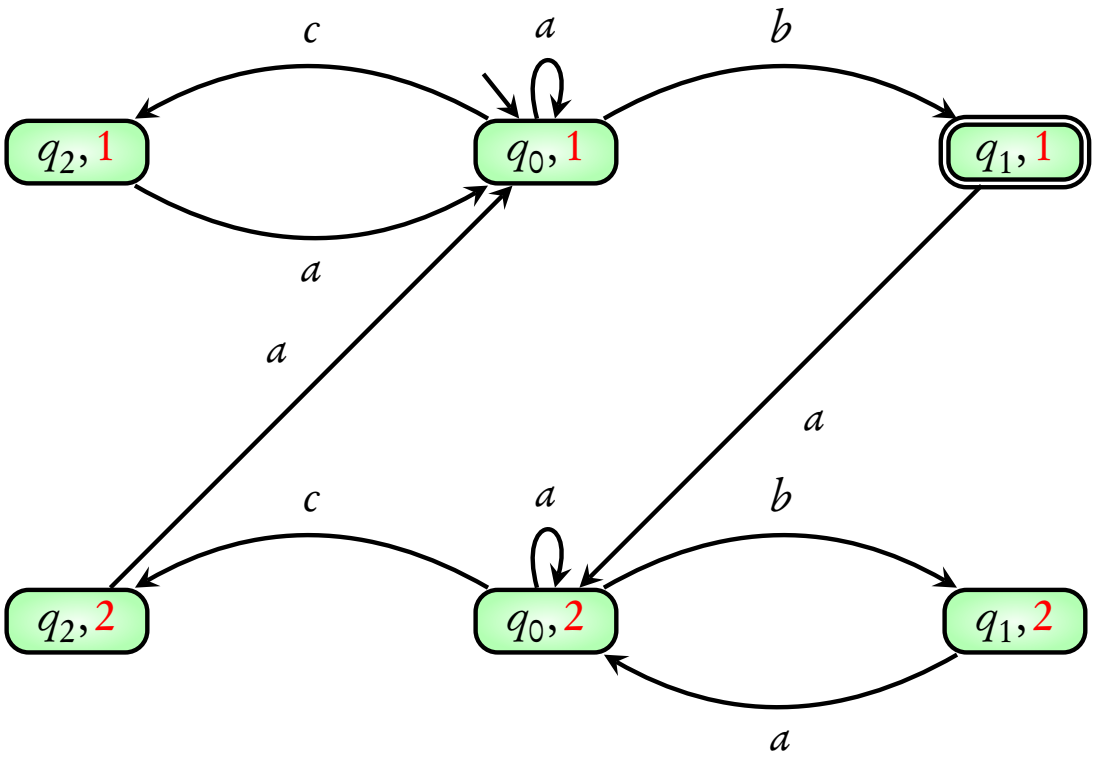


GNBA

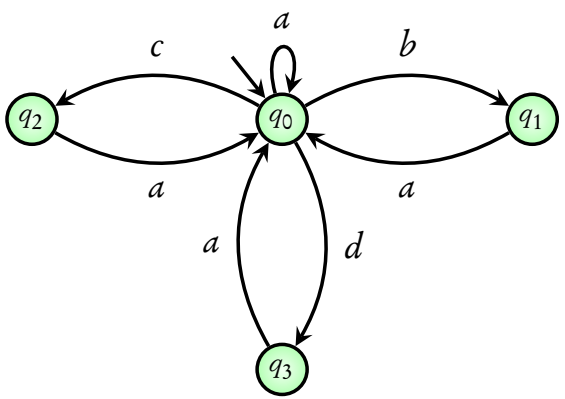


$\{\ \{q_1\},\ \{q_2\}\ \}$

NBA

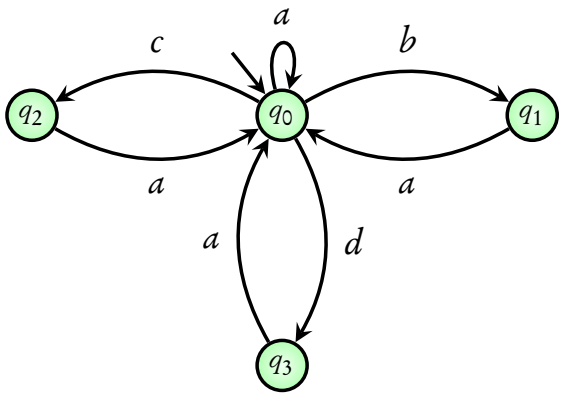


GNBA

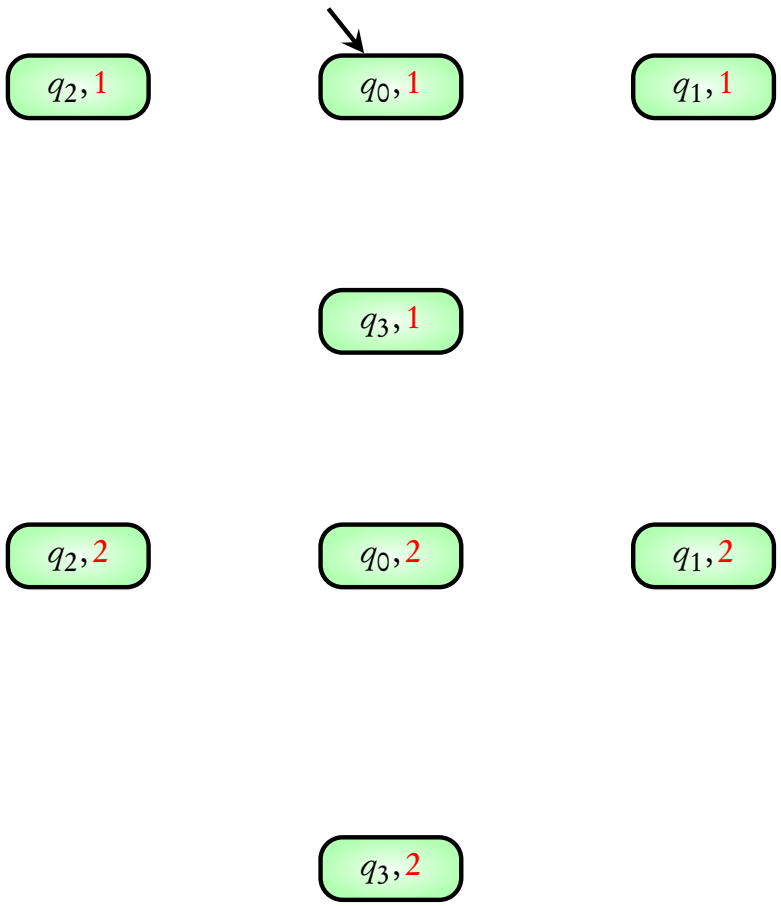


$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

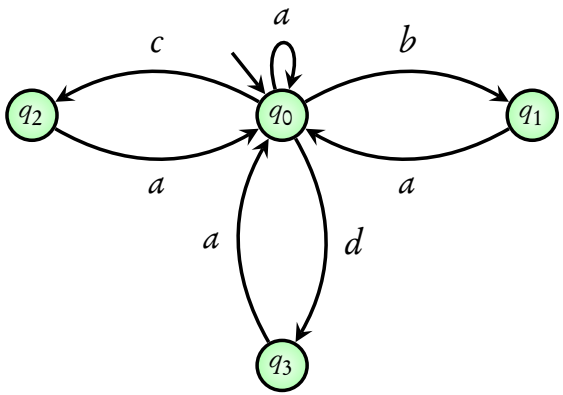
GNBA



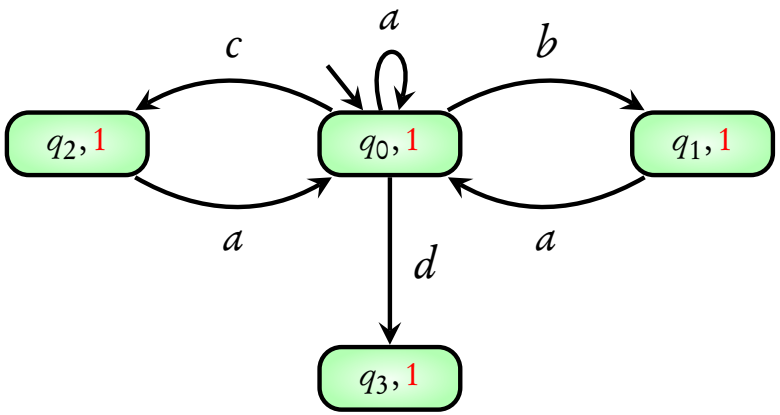
$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

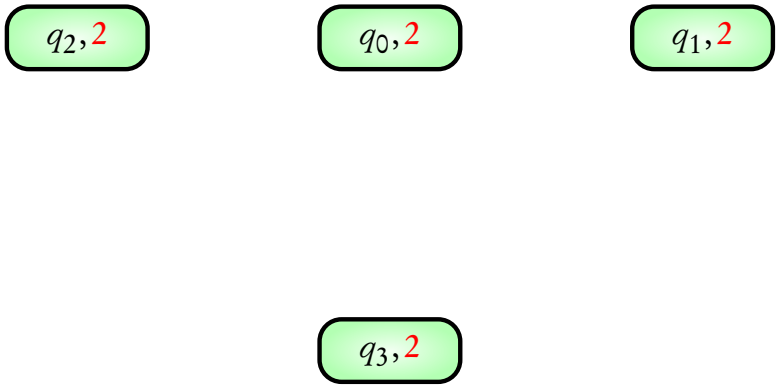


GNBA

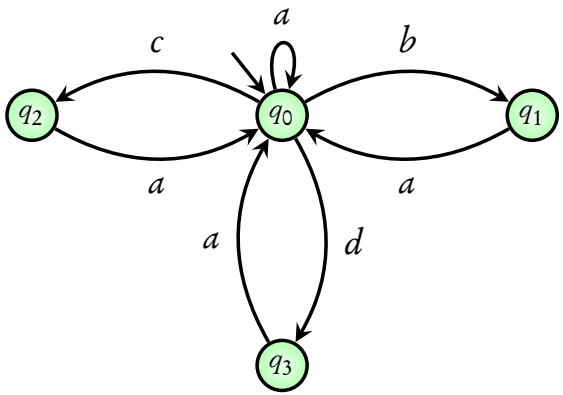


$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

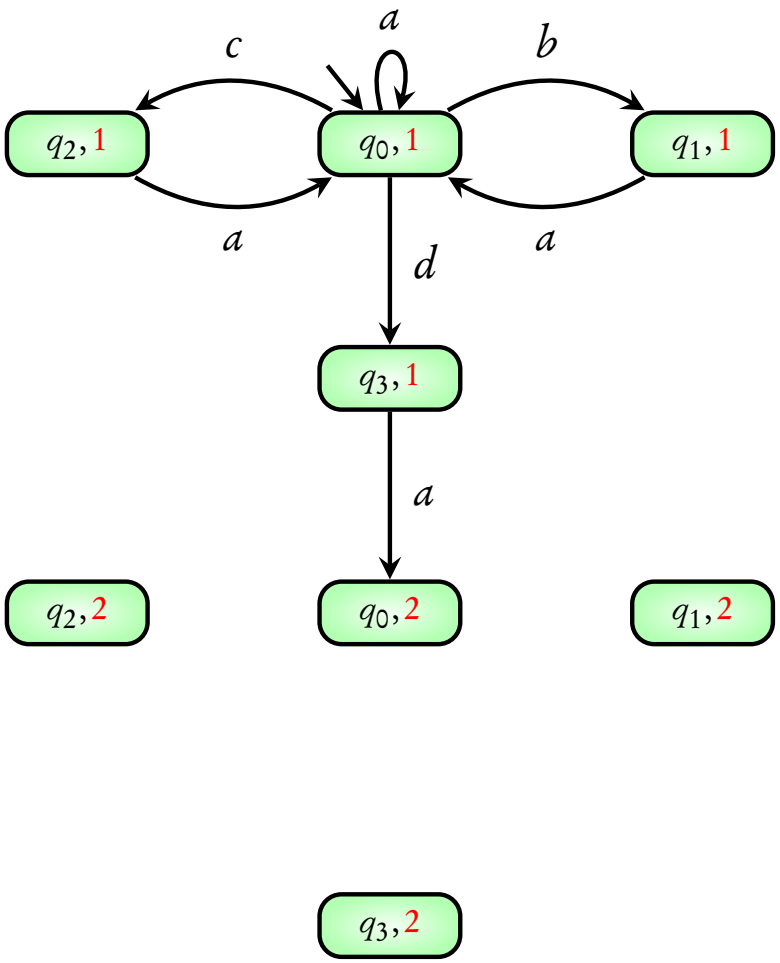




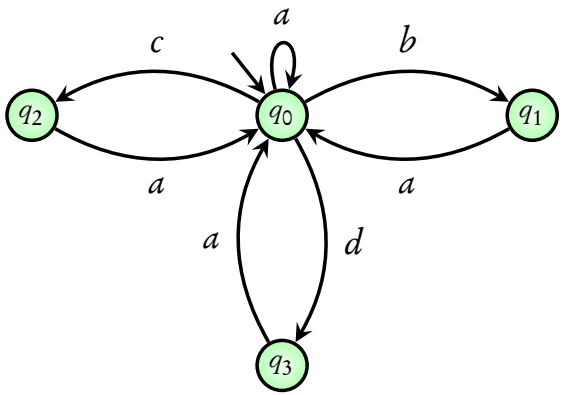
GNBA



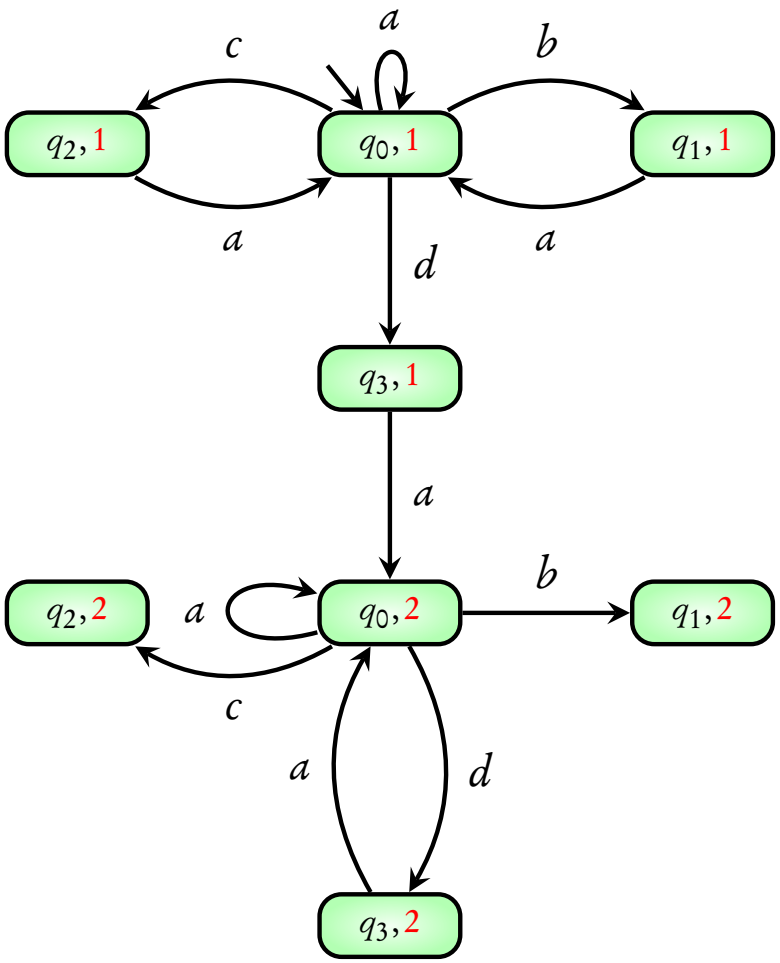
$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$



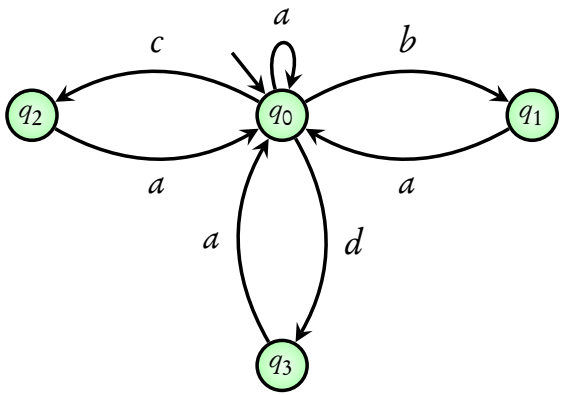
GNBA


a
c
b
q_2
q_0
q_1
a
a
a
d
q_3

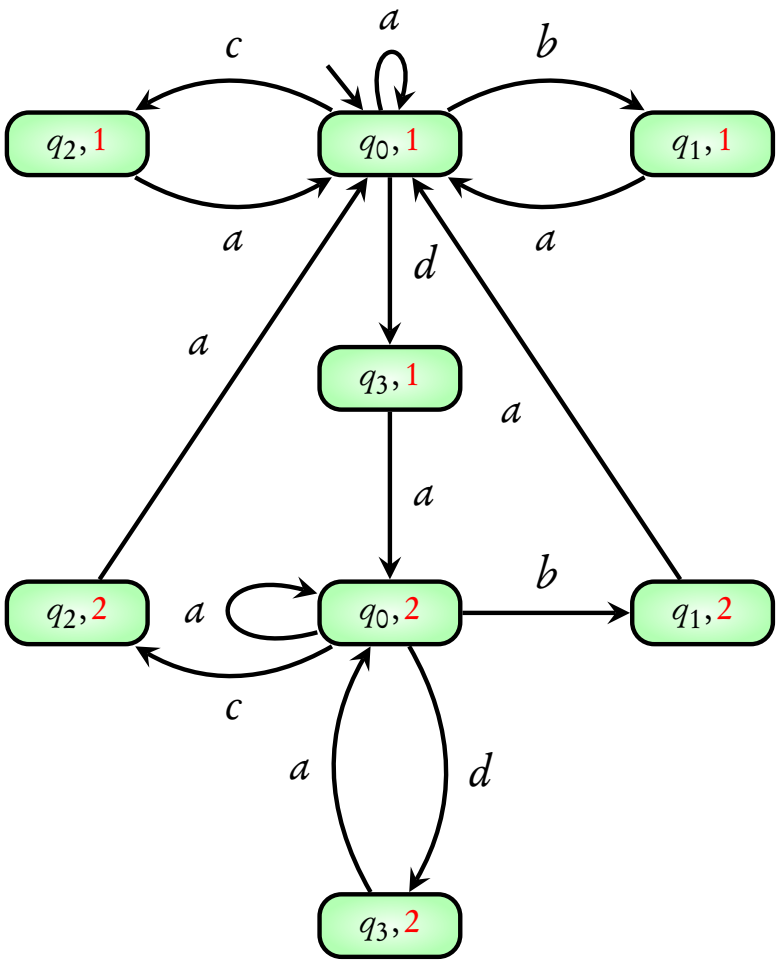
$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$


a
c
b
$q_2, 1$
$q_0, 1$
$q_1, 1$
a
a
d
$q_3, 1$
a
b
$q_2, 2$
a
$q_0, 2$
$q_1, 2$
c
a
d
$q_3, 2$

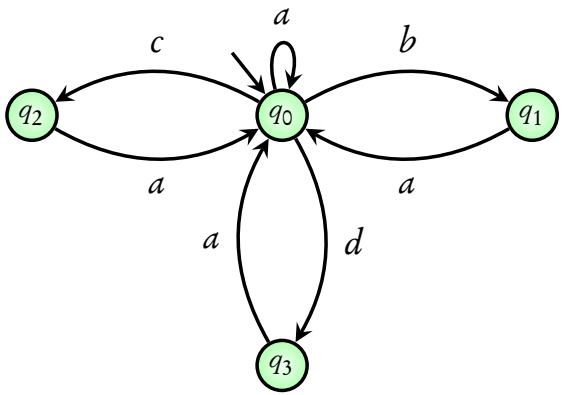
GNBA


a
c
b
q_2
q_0
q_1
a
a
a
d
q_3

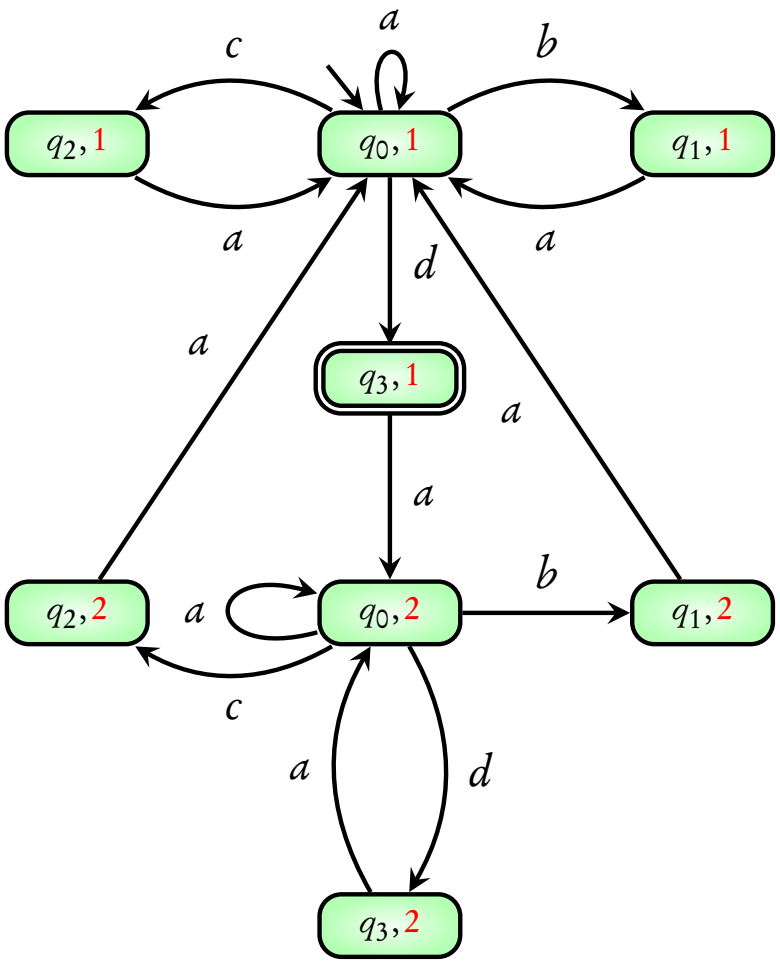
$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$


a
c
b
$q_2, 1$
$q_0, 1$
$q_1, 1$
a
d
a
a
$q_3, 1$
a
a
$q_2, 2$
a
$q_0, 2$
b
$q_1, 2$
c
a
d
$q_3, 2$

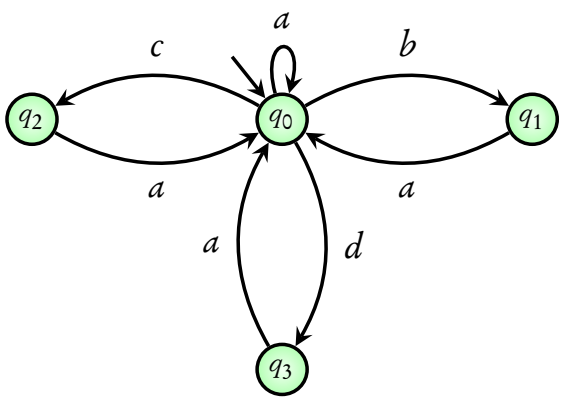
GNBA


a
c
b
q_2
q_0
q_1
a
a
a
d
q_3

$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

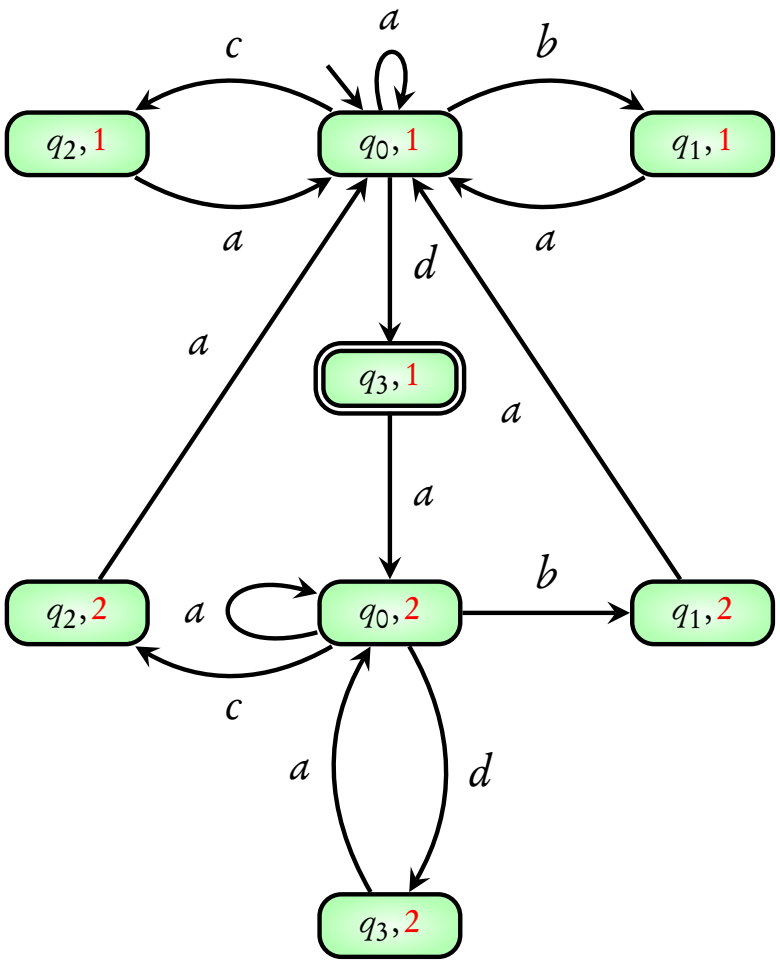

a
c
b
$q_2, 1$
$q_0, 1$
$q_1, 1$
a
d
a
a
$q_3, 1$
a
a
$q_2, 2$
a
$q_0, 2$
b
$q_1, 2$
c
a
d
$q_3, 2$

GNBA



$\{\ \{q_3\},\ \{q_1, q_2\}\ \}$

NBA



Generalized Büchi Automata

- States, transitions, initial states as in an NBA
- Accepting condition: $\{ F_1, F_2, \ldots, F_k \}$
- Run is accepting if **some state from each of the** F_i occurs infinitely often

Every GNBA can be **converted** to an **equivalent NBA**

Unit-6: Model-checking ω-regular properties

B. Srivathsan

Chennai Mathematical Institute

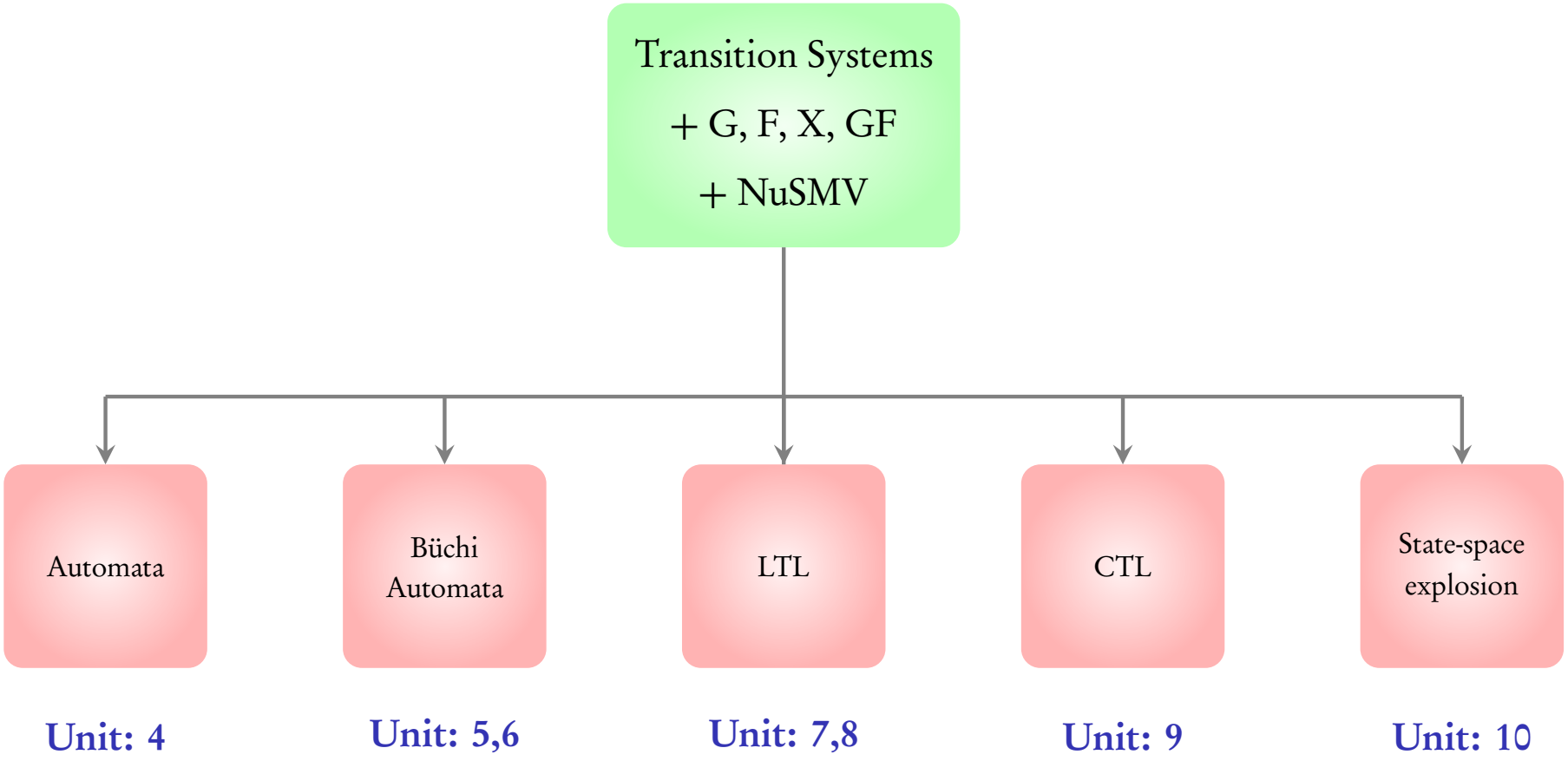
NPTEL-course

July - November 2015

Summary

- Model-checking problem reduced to emptiness of NBA
- ω-regular expressions can be converted to equivalent NBA
- Algorithm for emptiness check of NBA
- GNBA: every GNBA can be converted to equivalent NBA

Important concepts: Nested-dfs algorithm


Transition Systems
+ G, F, X, GF
+ NuSMV
Automata
Büchi Automata
LTL
CTL
State-space explosion
Unit: 4
Unit: 5,6
Unit: 7,8
Unit: 9
Unit: 10