Lecture 6: Büchi Automata

B. Srivathsan

Chennai Mathematical Institute

Model Checking and Systems Verification

January - April 2016

Question: How do we model-check LTL and ω-regular properties?

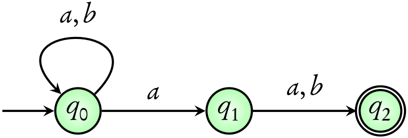
Goal

- Give some kind of an **automaton** for ω-regular expressions and LTL formulas
- Take **synchronous product** with the transition system of the model
- Check **emptiness** of this automaton

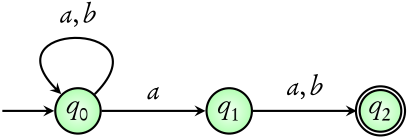
Goal

- Give some kind of an **automaton** for ω-regular expressions and LTL formulas
- Take **synchronous product** with the transition system of the model
- Check **emptiness** of this automaton

Coming next: A short recap of **finite automata**

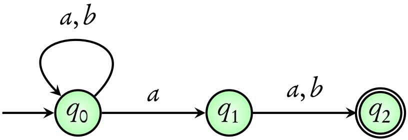

a, b
a
a, b
q_0
q_1
q_2

$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$



$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$

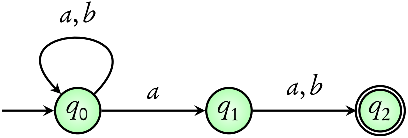
Runs:



$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$

Runs:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0$$

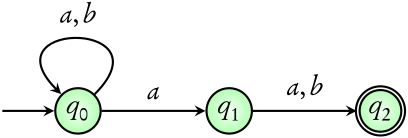


$$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$$

Runs:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$$

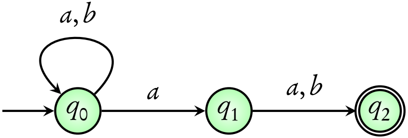


$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$

Runs:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$$ accepting run

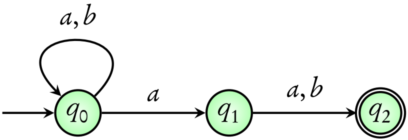


$a \quad b \quad b \quad a \quad a \quad b \quad a \quad b$

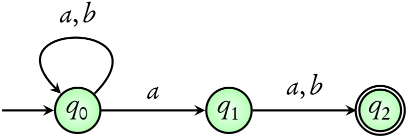
Runs:

$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0$

$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$ accepting run

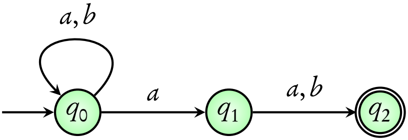


Language: set of words for which **there exists** an accepting run



Language: set of words for which **there exists** an accepting run

$a \quad b \quad b \quad b \quad a$

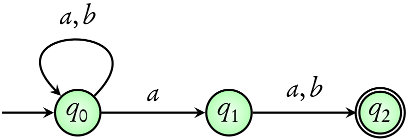


Language: set of words for which **there exists** an accepting run

$$a \quad b \quad b \quad b \quad a$$

Runs:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0$$

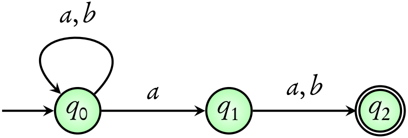


Language: set of words for which **there exists** an accepting run

$a \quad b \quad b \quad b \quad a$

Runs:

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_0$$ Not accepted



Language: set of words for which **there exists** an accepting run

In finite words, there is an **end**

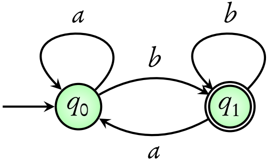
A run is accepting if it **ends in an accepting state**

In finite words, there is an **end**

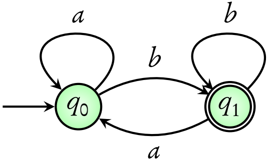
A run is accepting if it **ends in an accepting state**

How do we define **accepting runs** for **infinite words**?

Module 1:
Büchi Automata

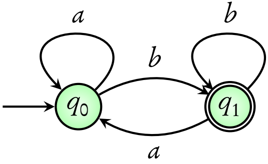

a
b
b
q0
q1
a

$a\ b\ a\ b\ a\ a\ b\ b\ b\ b\ b\ b\ \ldots$



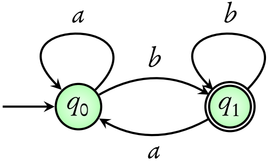
$$a\ b\ a\ b\ a\ a\ b\ b\ b\ b\ b\ b\ \ldots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \cdots$$



$$a\ b\ a\ b\ a\ a\ b\ b\ b\ b\ b\ b\ \ldots$$

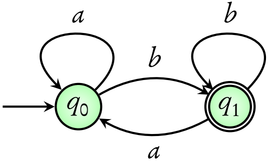
$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \cdots$$



Run is accepting if **some accepting state occurs infinitely often**

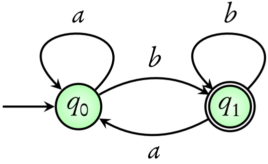
$$a\ b\ a\ b\ a\ a\ b\ b\ b\ b\ b\ b\ \ldots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \cdots$$



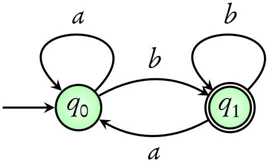
Above word is accepted by this automaton

Run is accepting if **some accepting state occurs infinitely often**



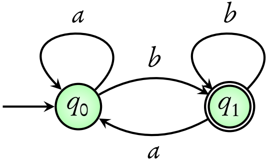
a
b
b
q_0
q_1
a

$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \ldots$



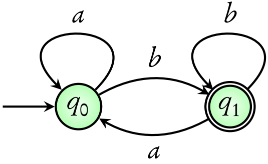
$$a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a\ b\ \ldots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \cdots$$



$a \; b \; a \; b \; a \; b \; a \; b \; a \; b \; a \; b \; \ldots$

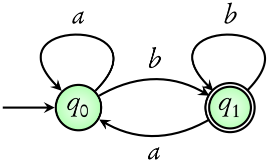
$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \cdots$$



Run is accepting if **some accepting state occurs infinitely often**

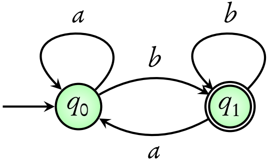
$$a \; b \; a \; b \; a \; b \; a \; b \; a \; b \; a \; b \; \ldots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \cdots$$

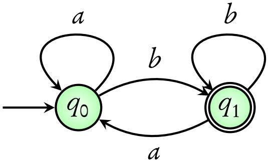


Above word is accepted by this automaton

Run is accepting if **some accepting state occurs infinitely often**

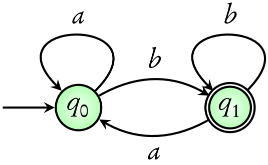

a
b
b
q0
q1
a

$a\ b\ a\ b\ a\ a\ a\ a\ a\ a\ a\ a\ \ldots$



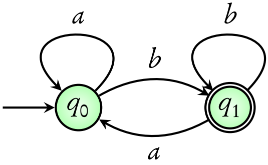
$$a\ b\ a\ b\ a\ a\ a\ a\ a\ a\ a\ a\ \dots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \cdots$$



$a \ b \ a \ b \ a \ a \ a \ a \ a \ a \ a \ a \ \ldots$

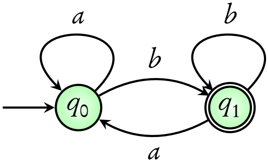
$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \cdots$$



Run is accepting if **some accepting state occurs infinitely often**

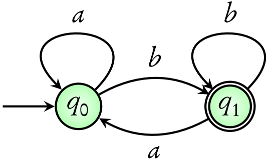
$$a \; b \; a \; b \; a \; a \; a \; a \; a \; a \; a \; a \; \ldots$$

$$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_0 \ldots$$



Above word is **not accepted** by this automaton

Run is accepting if **some accepting state occurs infinitely often**



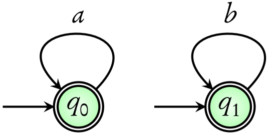
Language: set of infinite words which contain **infinitely many** b-s

Non-deterministic Büchi Automata

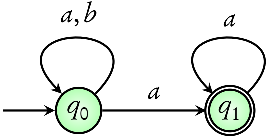
- States, transitions, initial and accepting states like an NFA
- Difference in accepting condition

Word is accepted if it has a run in which **some accepting state occurs infinitely often**

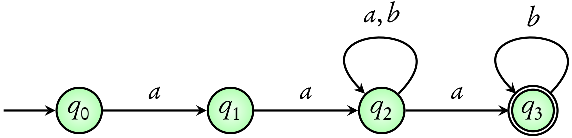
Example: $a^\omega + b^\omega$



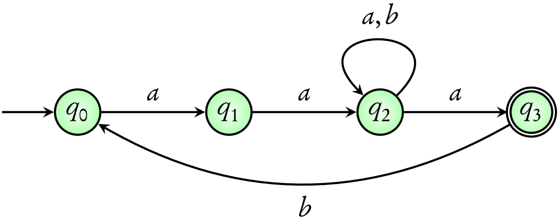
Example: $(a + b)^* a^\omega$



Example: $aa(a+b)^*ab^\omega$



Example: $(aa(a+b)^*ab)^\omega$



Non-deterministic Büchi Automaton

Accepting state occurs infinitely often

Module 2:

Simple properties of NBA

Determinization

Product construction

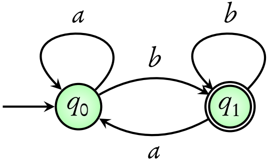
Emptiness

Complementation

Union

Deterministic Büchi Automata

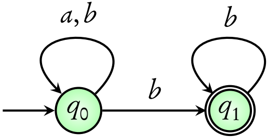
Words where b occurs infinitely often



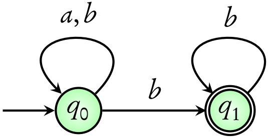
- Single initial state
- From every state - on an alphabet, there is a **unique transition**

Question: Can every NBA be converted to an **equivalent** DBA?

$(a+b)^*b^\omega$: a occurs only finitely often

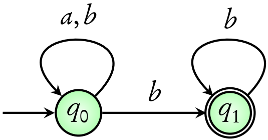


$(a+b)^*b^\omega$: a occurs only finitely often



- Automaton has to **guess** the point from where only b occurs
- A deterministic Büchi automaton cannot make this guess

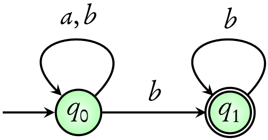
$(a+b)^*b^\omega$: a occurs only finitely often



- Automaton has to **guess** the point from where only b occurs
- A deterministic Büchi automaton cannot make this guess

The above language **cannot be** accepted by a DBA

$(a+b)^*b^\omega$: a occurs only finitely often



- Automaton has to **guess** the point from where only b occurs
- A deterministic Büchi automaton cannot make this guess

The above language **cannot be** accepted by a DBA

Theorem 4.50 (Page 190) of *Principles of Model Checking*, Baier and Katoen. MIT Press (2008)

Determinization

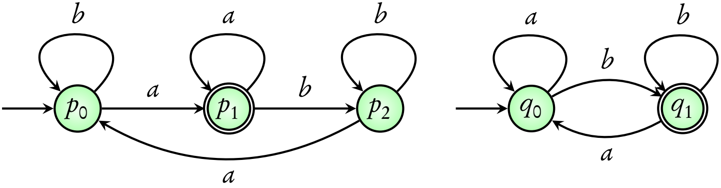
DBA less powerful than NBA

Product construction

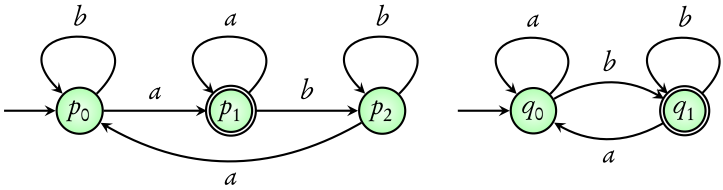
Emptiness

Complementation

Union

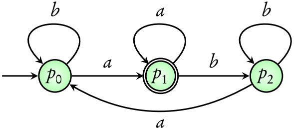


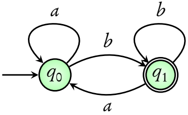
Word $(ab)^\omega$ is **accepted by both automata**

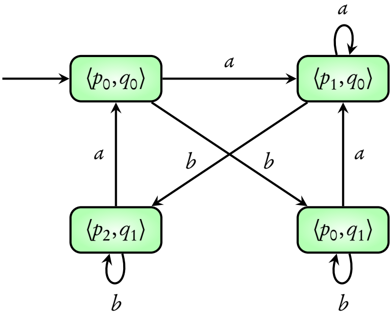


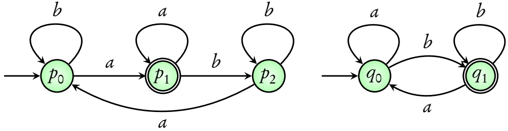
Word $(ab)^\omega$ is **accepted by both automata**

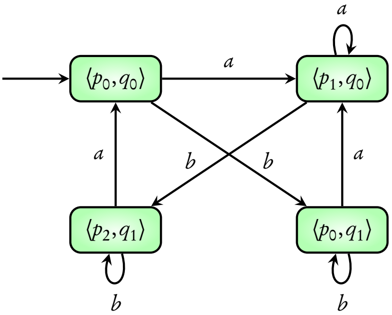
Coming next: The synchronous product construction


b
a
b
a
b
p0
p1
p2
a

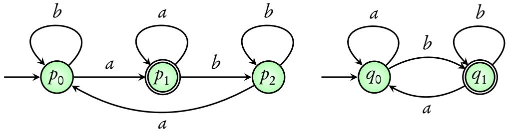

a
b
b
q0
q1
a

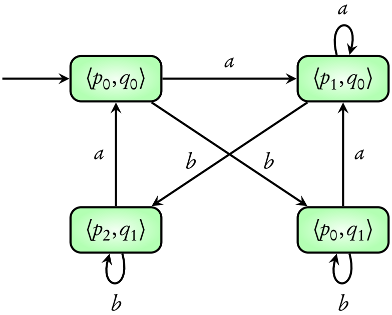

a
a
⟨p0,q0⟩
⟨p1,q0⟩
a
b
b
a
⟨p2,q1⟩
⟨p0,q1⟩
b
b





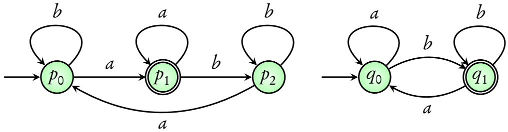
$\langle p_1, q_1 \rangle$ is not present

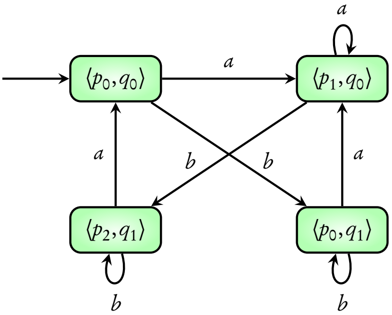




$\langle p_1, q_1 \rangle$ is not present

No accepting state!



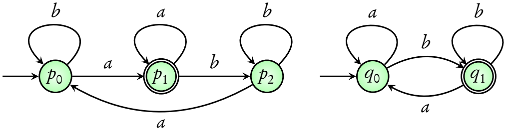


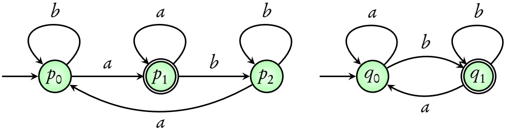
$\langle p_1, q_1 \rangle$ is not present

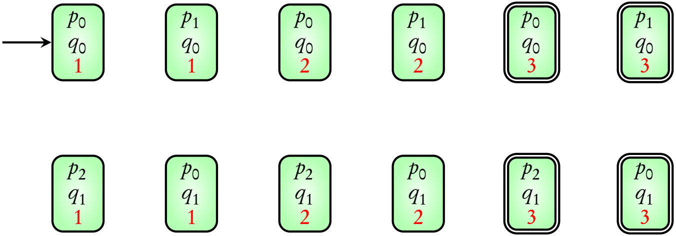
No accepting state!

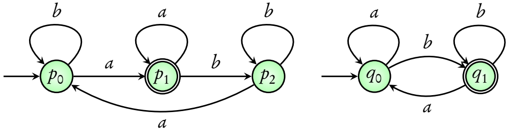
But intersection of the two automata is **not empty**

- Need to **modify** the product construction
- **Track** accepting states of **both automata**
- Ensure that **both** automata visit **accepting states infinitely often**

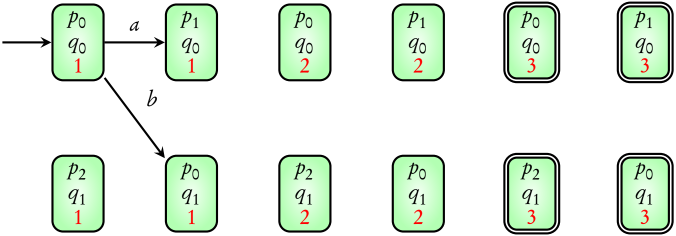

b
a
b
a
b
p_0
p_1
p_2
a
a
b
b
q_0
q_1
a


b
a
b
a
b
b
p_0
p_1
p_2
a
a
q_0
q_1
b
a

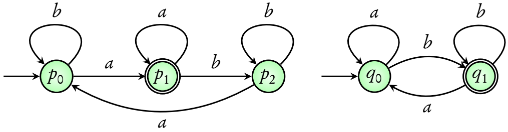

p_0 q_0 1
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3

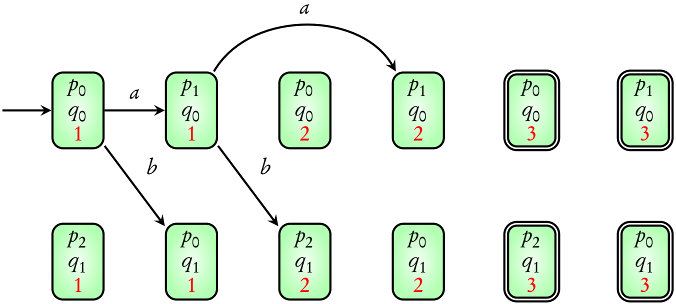


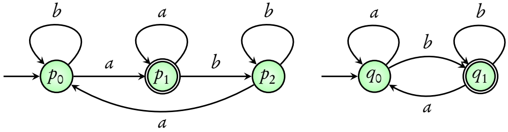
b
a
b
a
b
p0
p1
p2
a
a
b
b
q0
q1
a

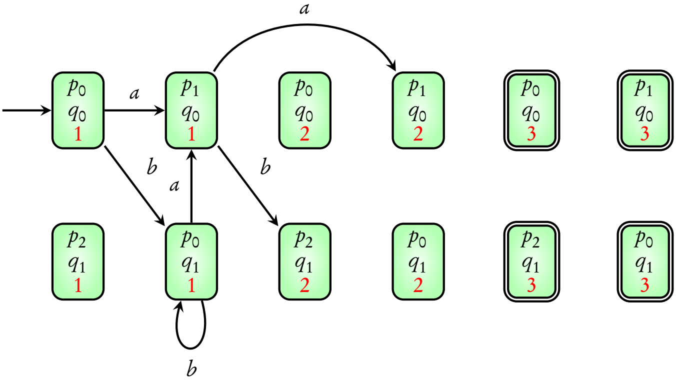


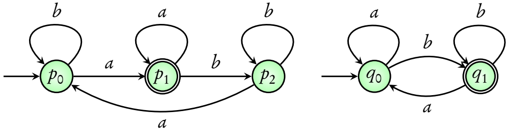
p0 q0 1
a
p1 q0 1
p0 q0 2
p1 q0 2
p0 q0 3
p1 q0 3
b
p2 q1 1
p0 q1 1
p2 q1 2
p0 q1 2
p2 q1 3
p0 q1 3


b
a
b
a
b
p_0
p_1
p_2
a
a
b
b
q_0
q_1
a

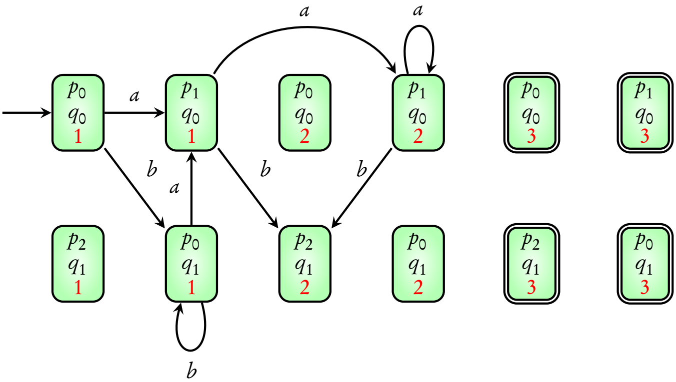

a
a
b
b
p_0 q_0 1
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3


b
a
b
a
b
p_0
p_1
p_2
a
a
b
q_0
q_1
a

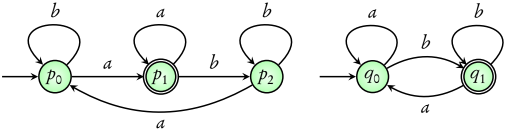

a
p_0 q_0 1
a
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
b
a
b
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3
b

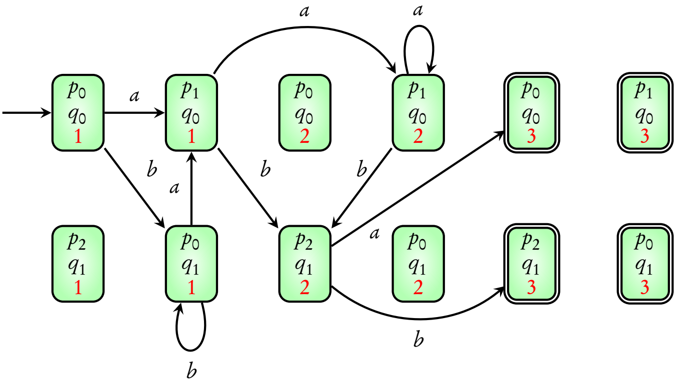


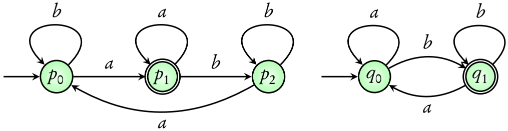
b
a
b
a
b
p_0
p_1
p_2
a
a
b
b
q_0
q_1
a



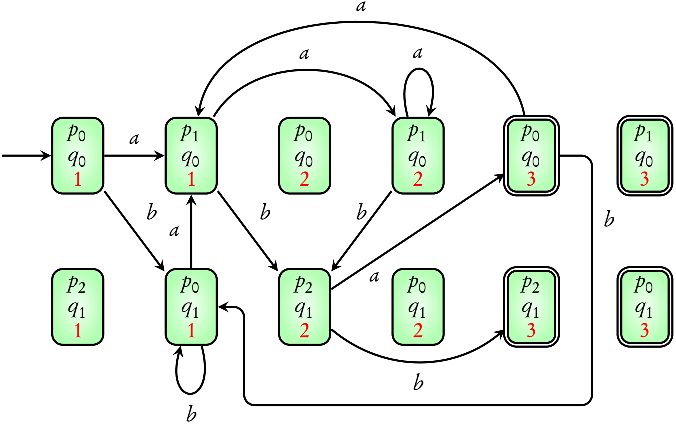
a
a
a
b
a
b
b
b
p_0 q_0 1
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3


p_0
p_1
p_2
q_0
q_1
a
b

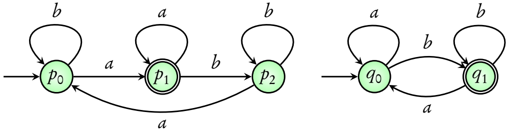

p_0 q_0 1
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3
a
b

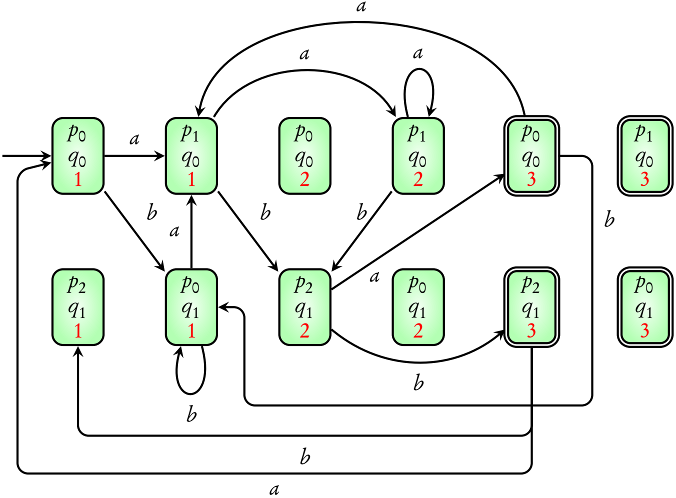


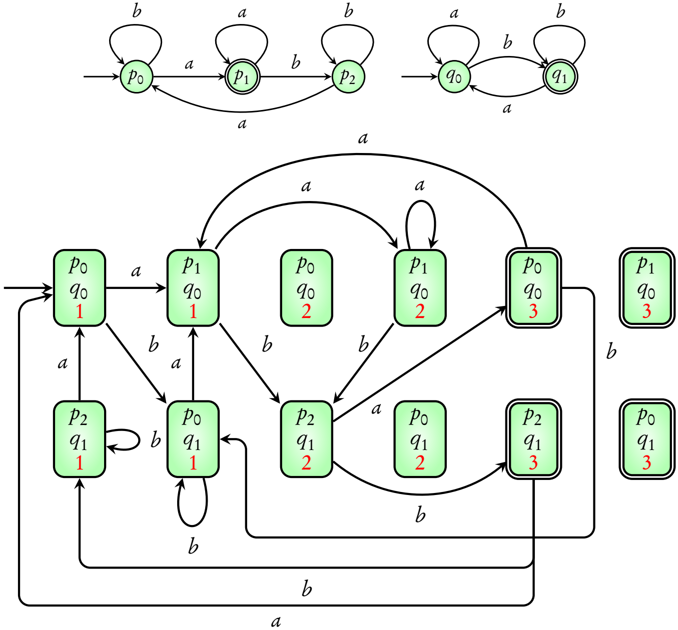
p_0
p_1
p_2
q_0
q_1
a
b

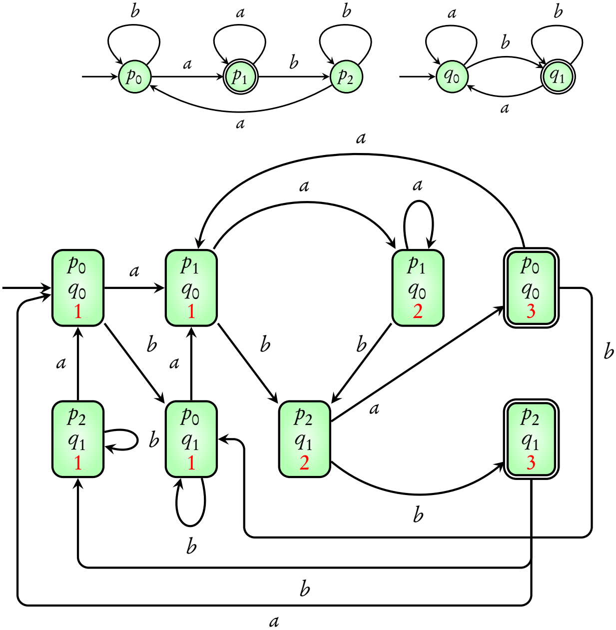


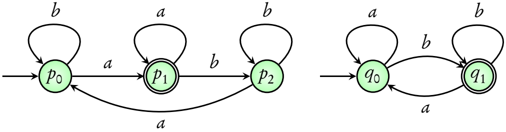
p_0 q_0 1
p_1 q_0 1
p_0 q_0 2
p_1 q_0 2
p_0 q_0 3
p_1 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_0 q_1 2
p_2 q_1 3
p_0 q_1 3
a
b

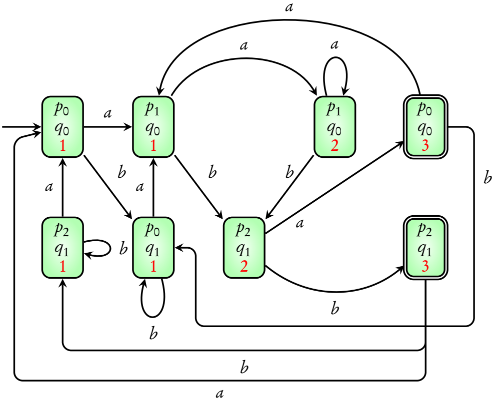

p0
p1
p2
q0
q1
a
b

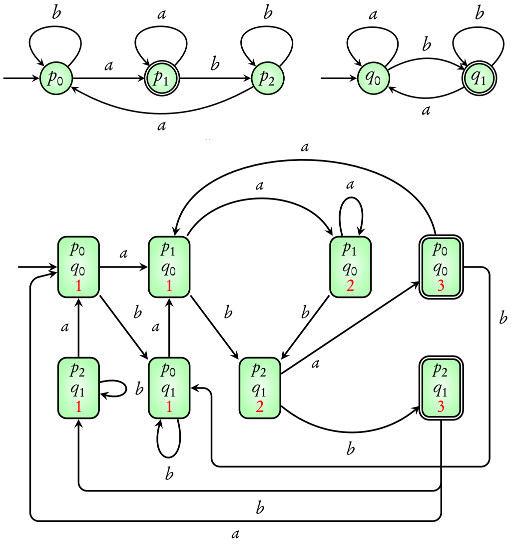

p0 q0 1
p1 q0 1
p0 q0 2
p1 q0 2
p0 q0 3
p1 q0 3
p2 q1 1
p0 q1 1
p2 q1 2
p0 q1 2
p2 q1 3
p0 q1 3
a
b


b
a
b
a
b
p0
p1
p2
a
a
b
b
q0
q1
a
a
a
a
a
p0 q0 1
p1 q0 1
p0 q0 2
p1 q0 2
p0 q0 3
p1 q0 3
b
b
b
b
a
a
a
b
p2 q1 1
p0 q1 1
p2 q1 2
p0 q1 2
p2 q1 3
p0 q1 3
b
b
b
a

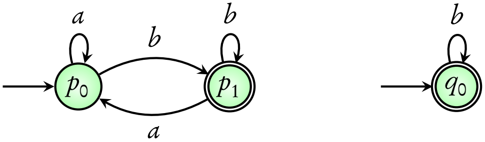

b a b a b b a a
p0 p1 p2
q0 q1
a a a a
p1 q0 1
p1 q0 2
p0 q0 3
p0 q0 1
a b a b b a b
p2 q1 1
p0 q1 1
p2 q1 2
p2 q1 3
b b b a
b b a


p_0
p_1
p_2
q_0
q_1

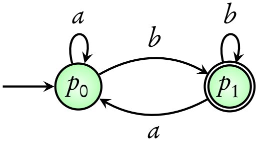

p_0 q_0 1
p_1 q_0 1
p_1 q_0 2
p_0 q_0 3
p_2 q_1 1
p_0 q_1 1
p_2 q_1 2
p_2 q_1 3

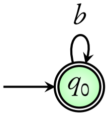


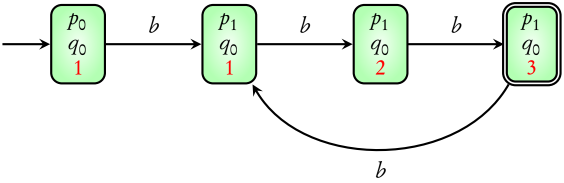
Word is accepted by product $\leftrightarrow$ it is accepted by both component automata



a
b
b
p0
p1
a
b
q0


a
b
b
p_0
p_1
a


b
q_0


p_0 q_0 1
b
p_1 q_0 1
b
p_1 q_0 2
b
p_1 q_0 3
b

Determinization

DBA less powerful than NBA

Product construction

Language intersection

Emptiness

Complementation

Union

Determinization

DBA less powerful than NBA

Product construction

Language intersection

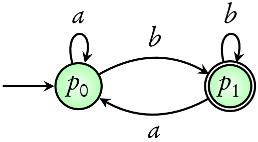
Emptiness

Next unit ...

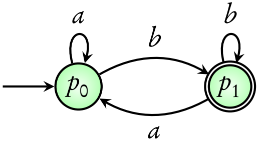
Complementation

Union

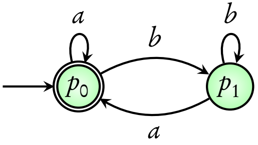
Language: b occurs infinitely often



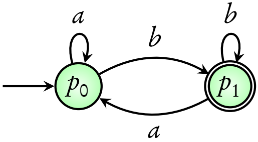
Language: *b* occurs infinitely often



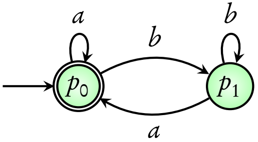
Language: *a* occurs infinitely often



Language: b occurs infinitely often



Language: a occurs infinitely often



Not the complement!

$(ab)^\omega$ present in both

Challenges

- Mere interchange of accepting states does not work

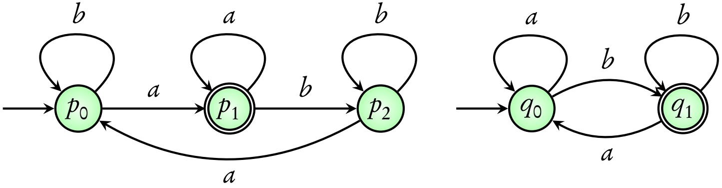
- Moreoever, NBA are more expressive than DBA

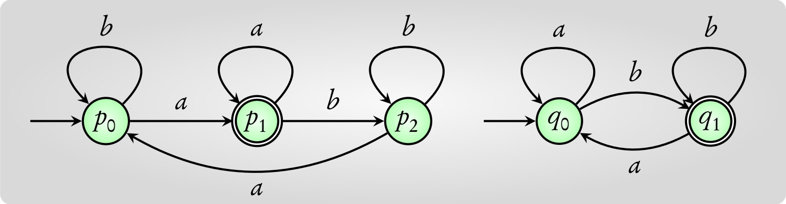
Complementation

Theorem

Given an NBA $\mathscr{A}$, there is an algorithm to compute the NBA accepting the complement language $\mathscr{L}(\mathscr{A})^c$

Proof out of scope of this course


b
a
b
a
b
a
b
p0
p1
p2
a
a
b
q0
q1
a



For **union**, take the disjoint union of the two NBA

Determinization

DBA less powerful than NBA

Product construction

Language intersection

Emptiness

Next unit ...

Complementation

Union

Module 3:
Model-checking schema

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

↓

NBA $\mathscr{A}_\phi$

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

ω-regular expression ϕ

↓ NBA $\mathscr{A}_{T.S}$

↓ NBA $\mathscr{A}_{\phi}$

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

ω-regular expression ϕ

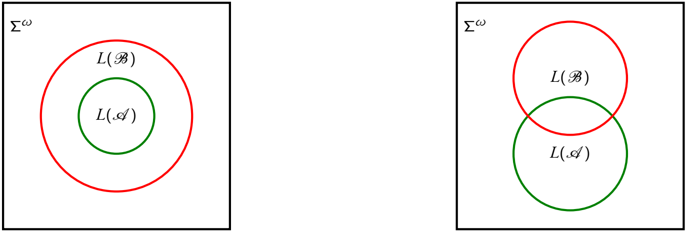
NBA $\mathscr{A}_{T.S}$

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

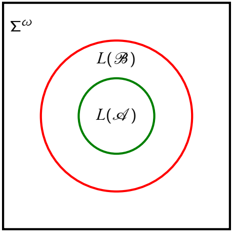
$$L(\mathscr{A}_{T.S.}) \subseteq L(\mathscr{A}_{\phi})\ ?$$

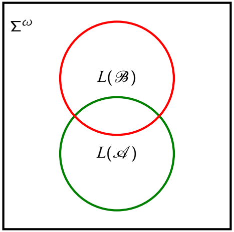
$$L(\mathscr{A}) \subseteq L(\mathscr{B})\ ?$$

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



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





$L(\mathscr{A}) \cap \overline{L(\mathscr{B})}$ 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 $\mathscr{A}_{\phi}$

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

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 4)

- Checking **language emptiness** of NBA (Module 5)

Module 4:

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 $\quad a^\omega + b^\omega$

$$\{ aaaaaaaaaaaaaaa\ldots,\ bbbbbbbbbbb\ldots \}$$

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

$$\{ aaaabbbbbb\ldots,\ aababaabbbbb\ldots,\ aabbbbaabbbbbb\ldots,\ \ldots \}$$

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

$$\{ aaaaaaaaaaaaaaa\ldots,\ baaaaaaaaa\ldots,\ babbaaaaaaaaaaa\ldots,\ \ldots \}$$

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

$$\{ abababababab\ldots,\ bbbabbbabbbabbba\ldots,\ bbbbbbbbbbbb\ldots,\ \ldots \}$$

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**

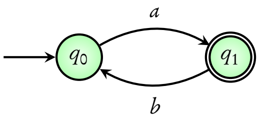
ω-regular expressions

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

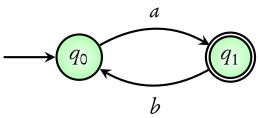
Goal: Convert ω-regular expression to NBA

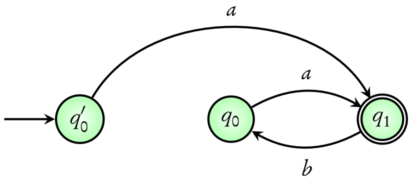
Part 1: Given regular expression U, find NBA for U^ω

NFA for U

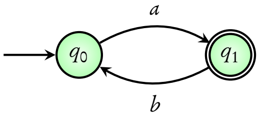


NFA for U

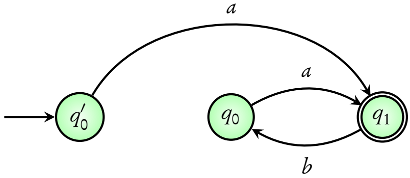




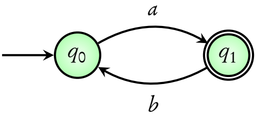
NFA for U



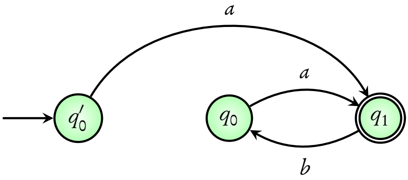
Standardized NFA

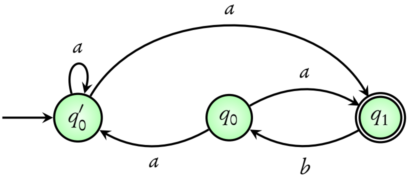


NFA for U

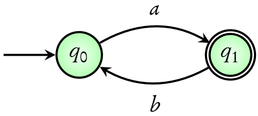


Standardized NFA

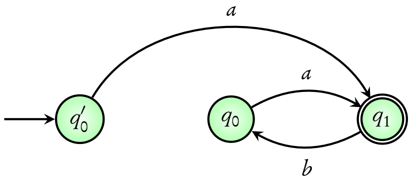


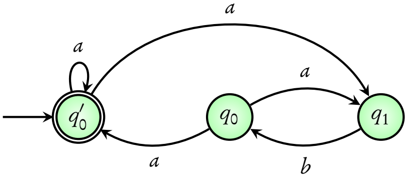


NFA for U

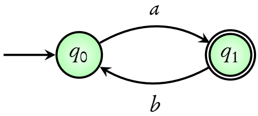


Standardized NFA

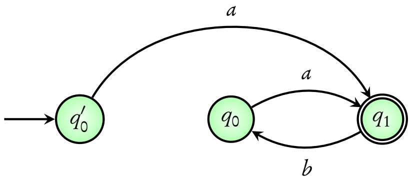




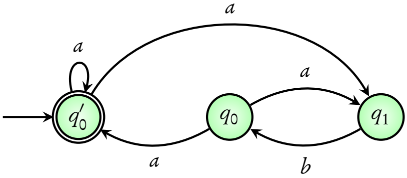
NFA for U



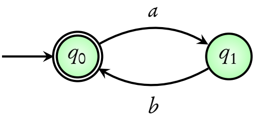
Standardized NFA



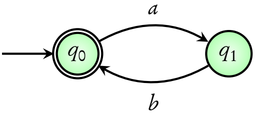
NBA for U^ω

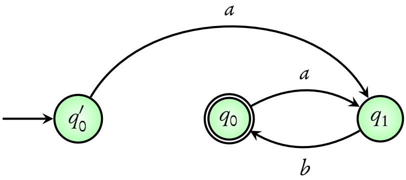


NFA for U

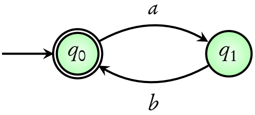


NFA for U

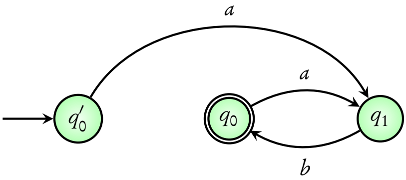




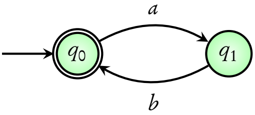
NFA for U



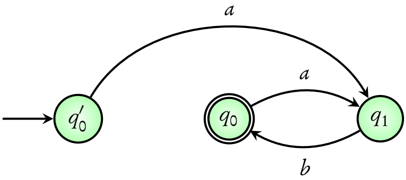
Standardized NFA

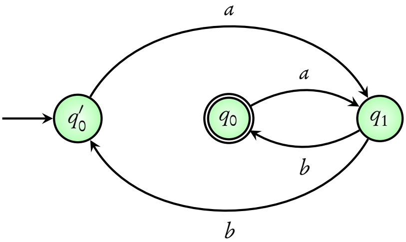


NFA for *U*

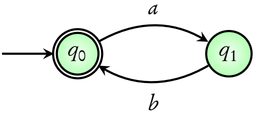


Standardized NFA

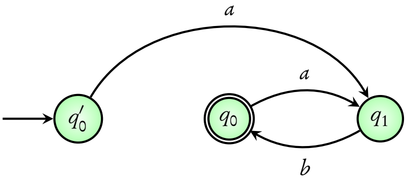


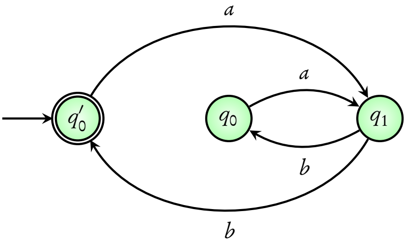


NFA for U

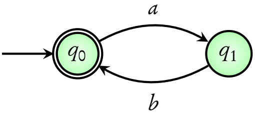


Standardized NFA

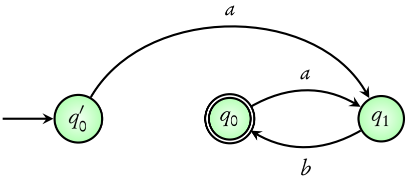




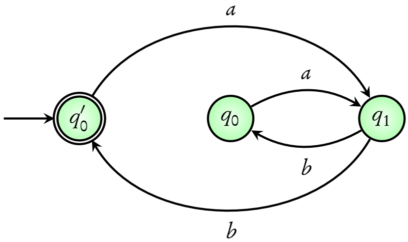
NFA for U



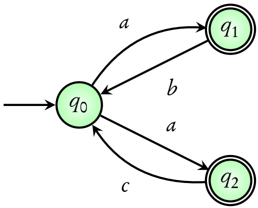
Standardized NFA



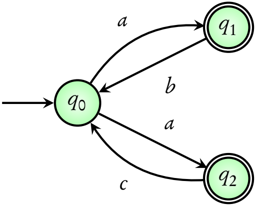
NBA for U^ω

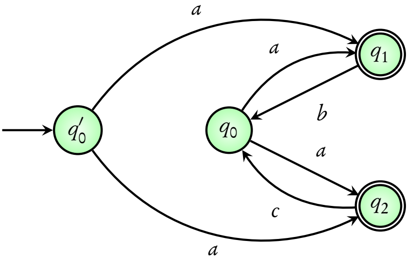


NFA for U

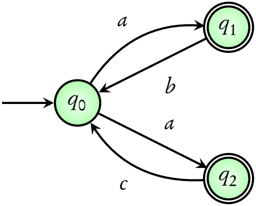


NFA for U

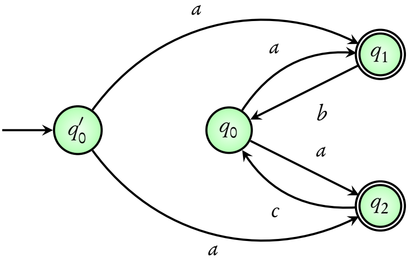




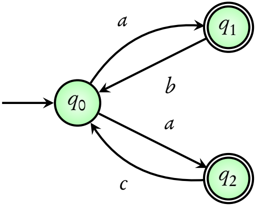
NFA for U



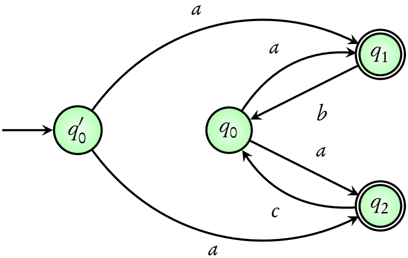
Standardized NFA

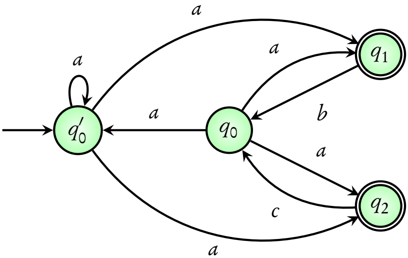


NFA for U

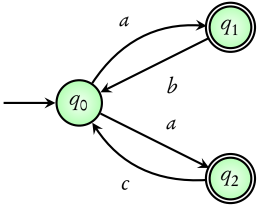


Standardized NFA

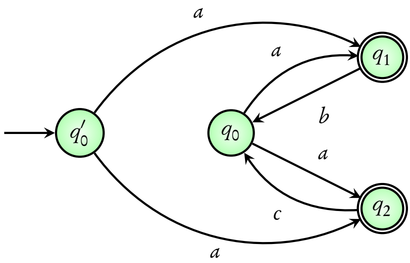


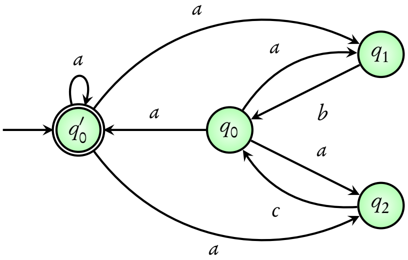


NFA for U



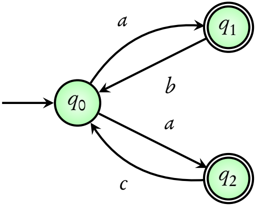
Standardized NFA

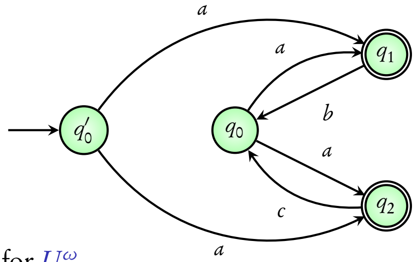




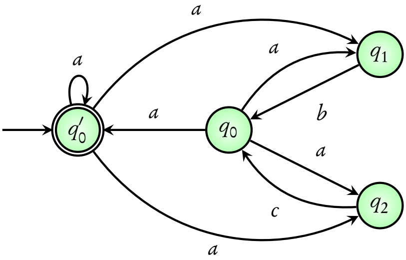
Standardized NFA

NFA for U

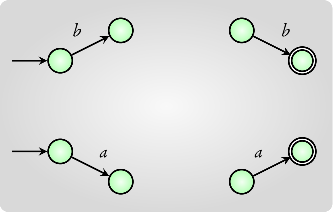




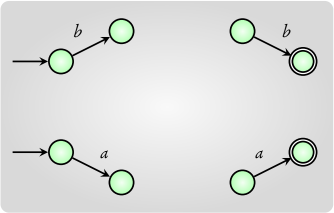
NBA for U^ω



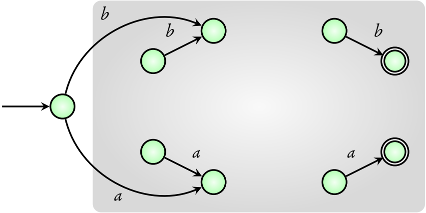
NFA for U



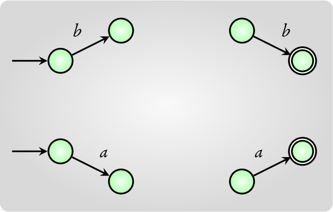
NFA for U



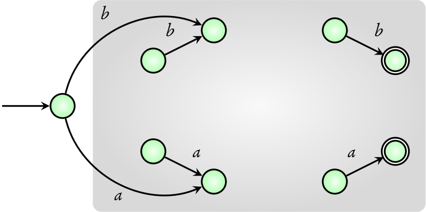
Standardized NFA for U



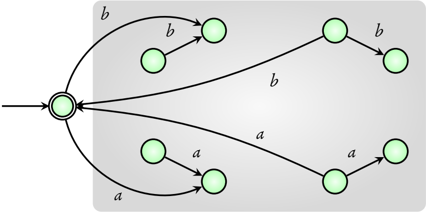
NFA for U



Standardized NFA for U



NBA for U^ω



ω-regular expressions

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

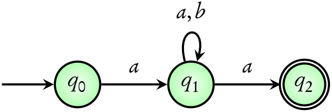
Goal: Convert ω-regular expression to NBA

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

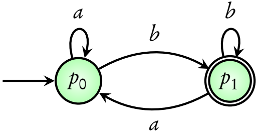
Done!

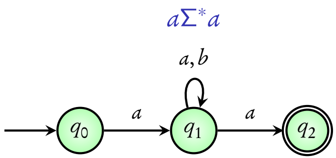
Part 2: Given regular expression U and NBA for V find NBA for $U \cdot V$

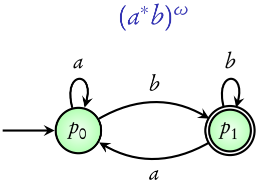
$a\Sigma^*a$

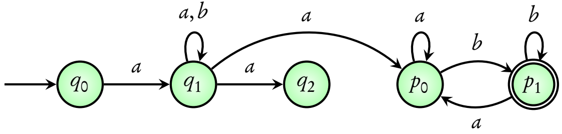


$(a^*b)^\omega$

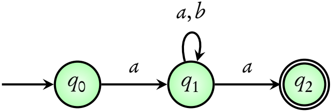



$a\Sigma^*a$
a,b
a
a
q_0
q_1
q_2

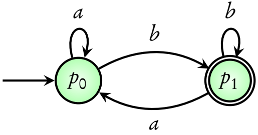

$(a^*b)^\omega$
a
b
b
a
p_0
p_1


a,b
a
a
a
a
b
b
a
q_0
q_1
q_2
p_0
p_1

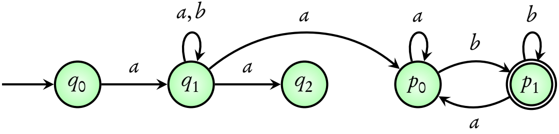
$a\Sigma^* a$



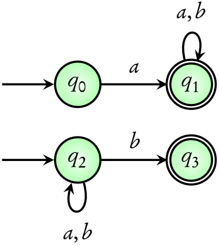
$(a^* b)^\omega$



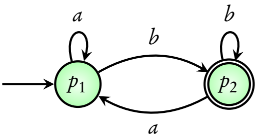
$a\Sigma^* a \cdot (a^* b)^\omega$



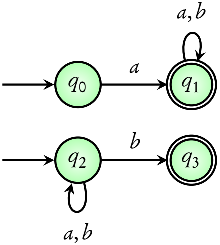
$a\Sigma^* + \Sigma^* b$



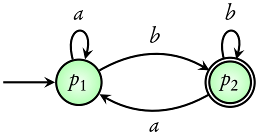
$(a^* b)^\omega$



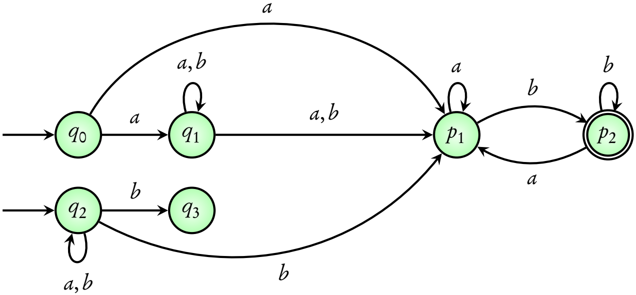
$a\Sigma^* + \Sigma^* b$



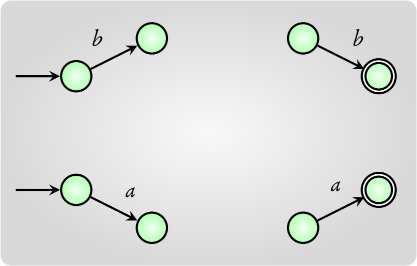
$(a^* b)^\omega$



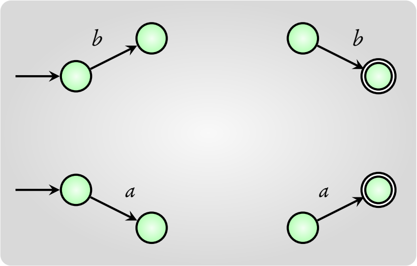
$(a\Sigma^* + \Sigma^* b) \cdot (a^* b)^\omega$



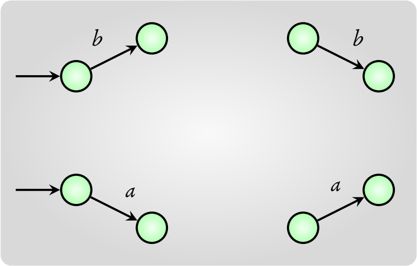
U



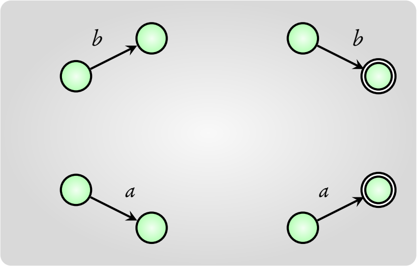
V



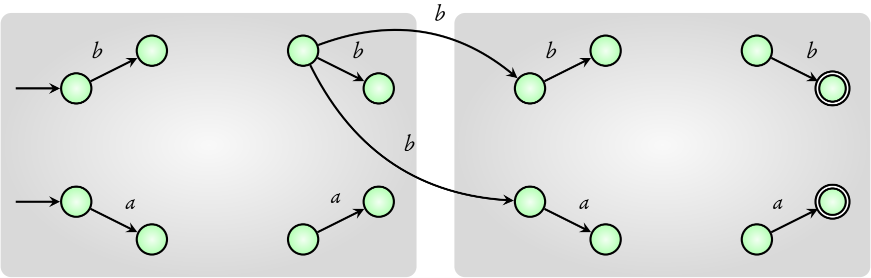
U

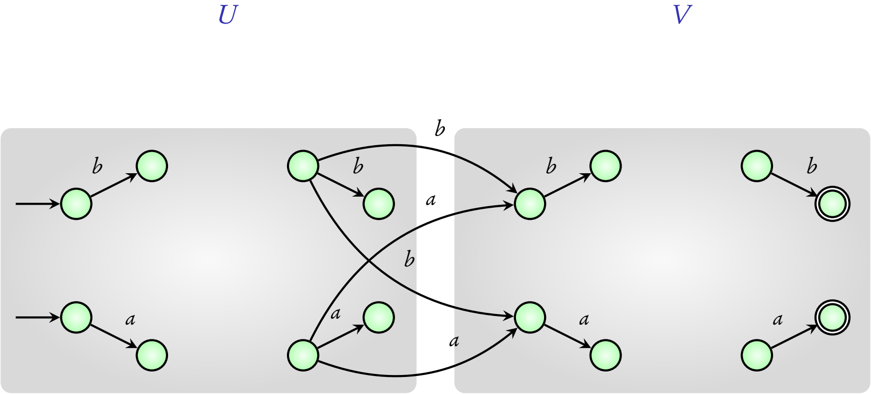


V

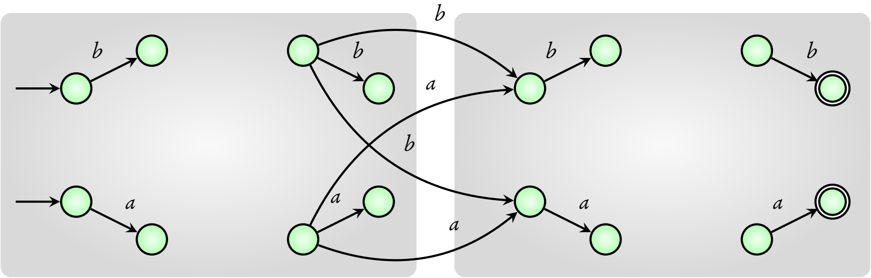


U V




U
V
b
b
b
a
b
b
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 $U \cdot V$

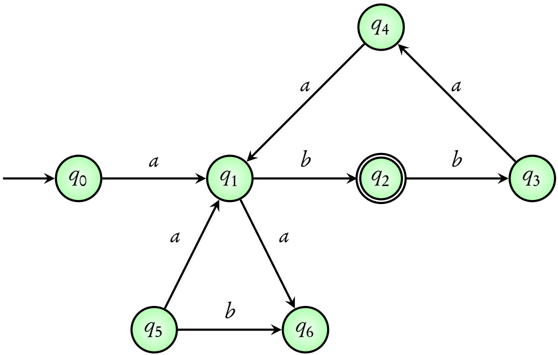
Part 3: Given NBA for U and NBA for V find NBA for $U + V$

Theorem

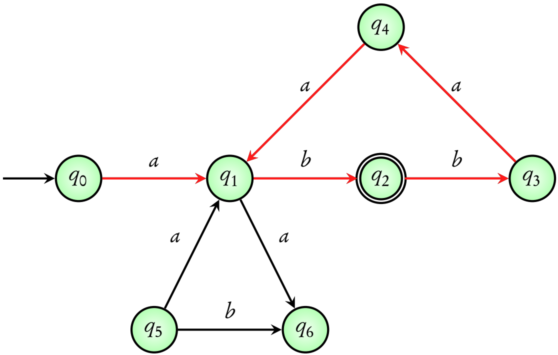
Every ω-regular expression can be **converted** to an NBA

Module 5:

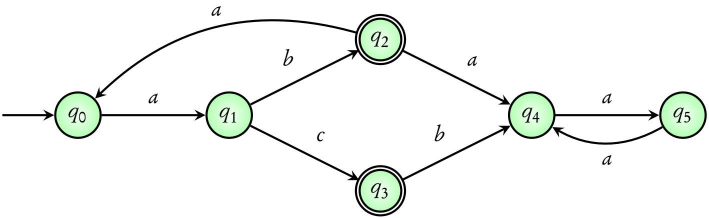
Checking emptiness of Büchi automata



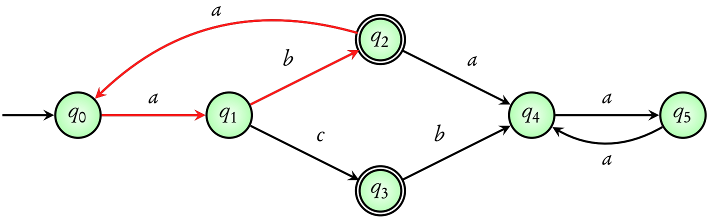
Is the **language** of above NBA **empty**?



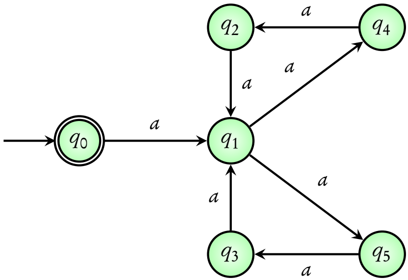
Is the **language** of above NBA **empty**? **No**



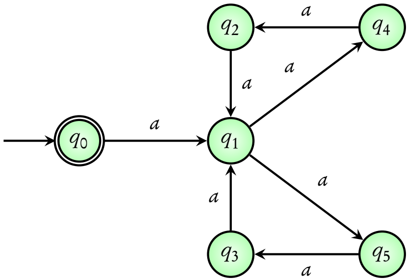
Is the **language** of above NBA **empty**?



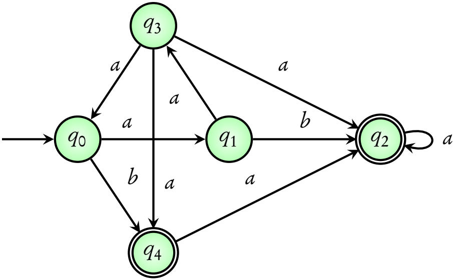
Is the **language** of above NBA **empty**? **No**



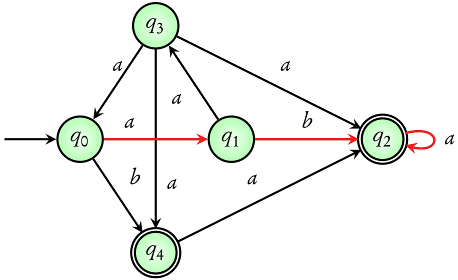
Is the **language** of above NBA **empty**?



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

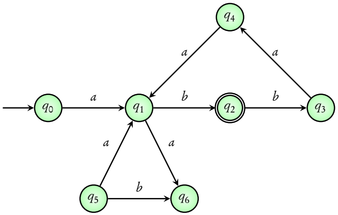
Find a **reachable cycle** in the automaton that contains an **accepting state**

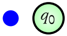
- Do a preliminary DFS to get all **reachable states**
- From every **accepting state**, do a secondary DFS to see if it can **come back to itself**

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

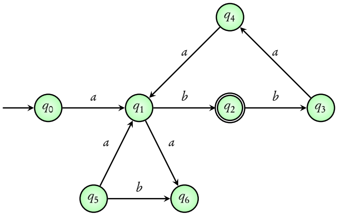


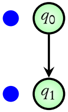


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

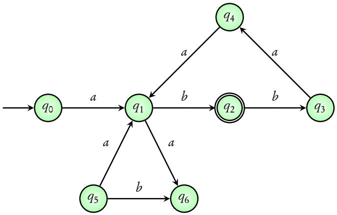


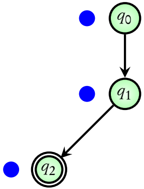



```
procedure nested_dfs( )
    call dfs_blue ( s0 )

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
```

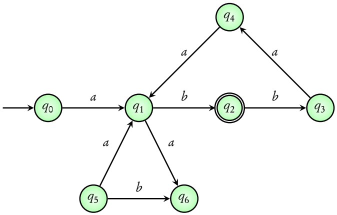


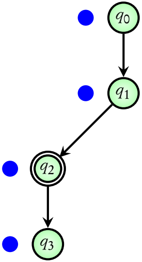


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

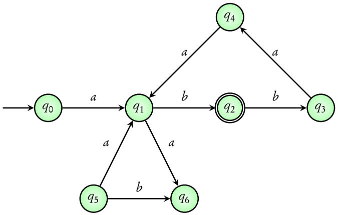


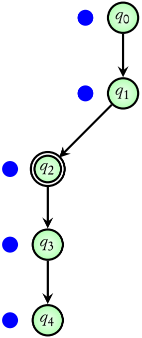


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

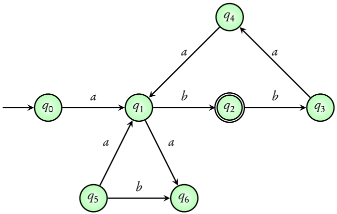




procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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*)

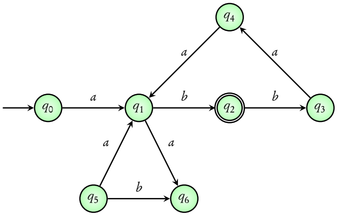
procedure *dfs_red* (*s*)
    s.red := **true**
    for all $t \in$ *post*(*s*) **do**
        if $\neg$*t.red* **then**
            call *dfs_red* (*t*)
        else if $t =$ *seed*
            report cycle

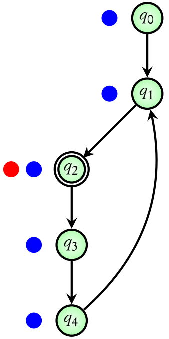


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

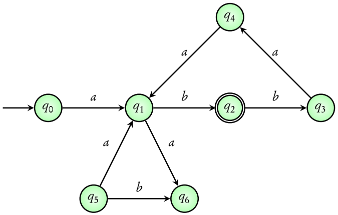


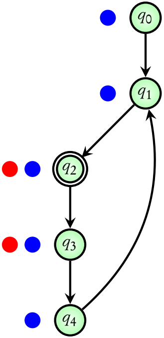



```
procedure nested_dfs()
    call dfs_blue(s0)

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
```

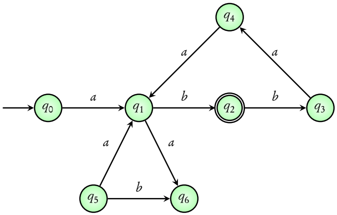


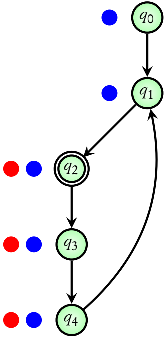


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

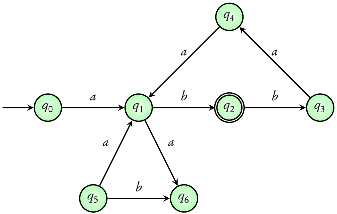


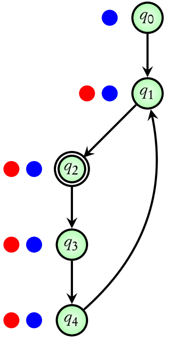



```
procedure nested_dfs()
    call dfs_blue(s0)

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
```

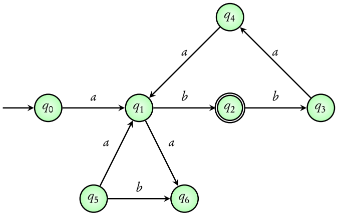


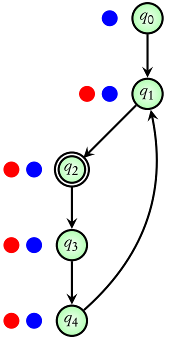


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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



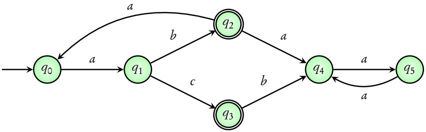


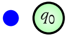
report cycle!

```
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
```

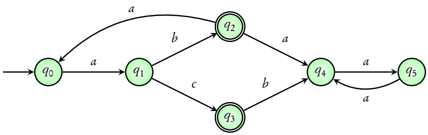


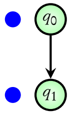


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

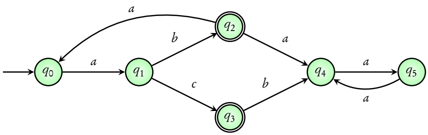


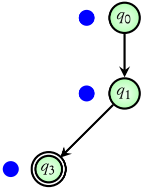


procedure *nested_dfs*()
    call *dfs_blue* (s_0)

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

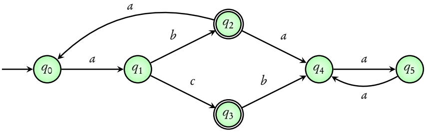


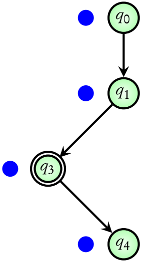


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

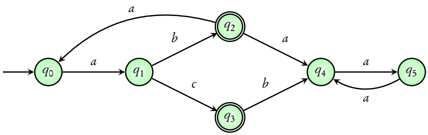


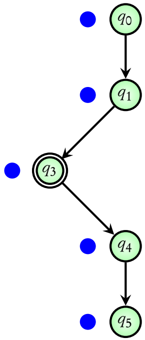



```
procedure nested_dfs()
    call dfs_blue ( s0 )

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
```

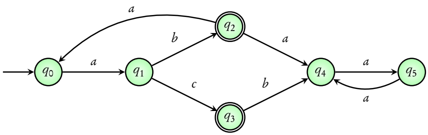


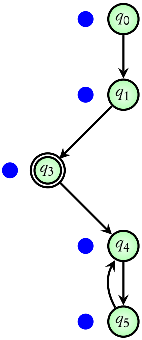



```
procedure nested_dfs()
    call dfs_blue ( s0 )

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**

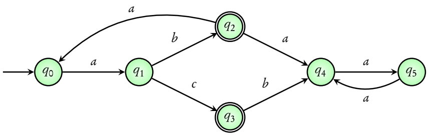
  for all $t \in post(s)$ **do**

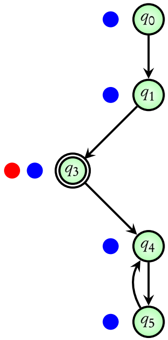
    if $\neg t.red$ **then**

      call *dfs_red* (t)

    else if $t = seed$

      report cycle

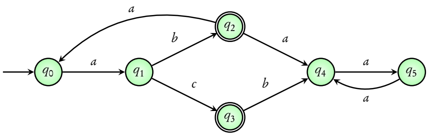


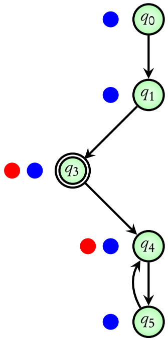



```
procedure nested_dfs( )
    call dfs_blue ( s0 )

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
```

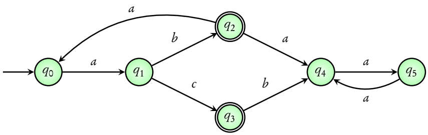


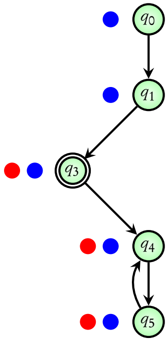


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

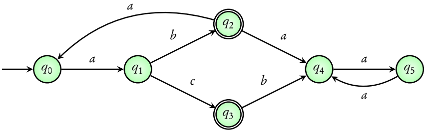


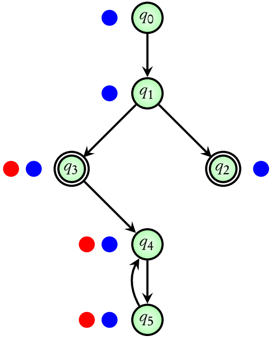



```
procedure nested_dfs()
    call dfs_blue ( s0 )

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
```

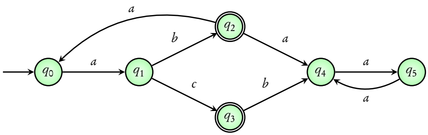


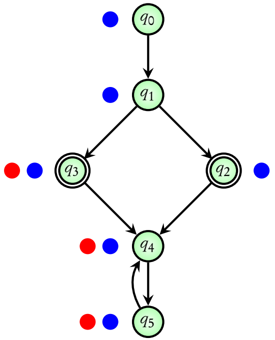


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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**

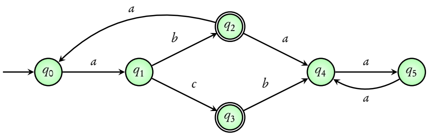
  for all $t \in post(s)$ **do**

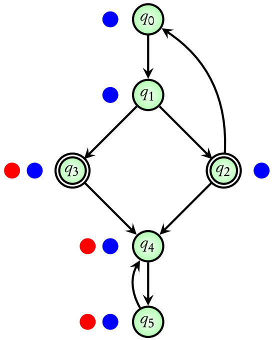
    if $\neg t.red$ **then**

      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**

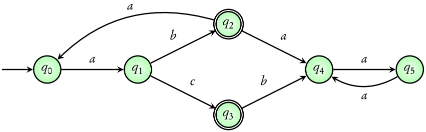
  for all $t \in post(s)$ **do**

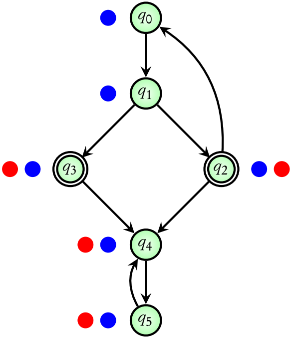
    if $\neg t.red$ **then**

      call *dfs_red* (t)

    else if $t = seed$

      report cycle

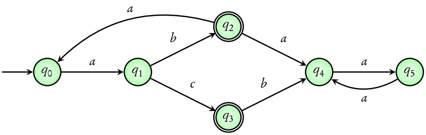


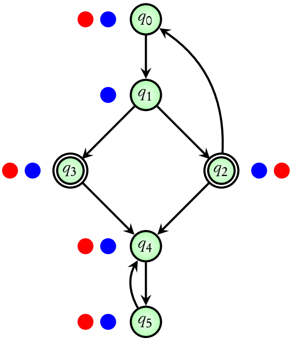


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

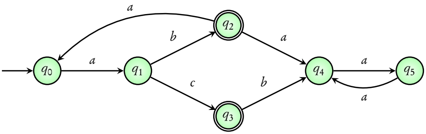


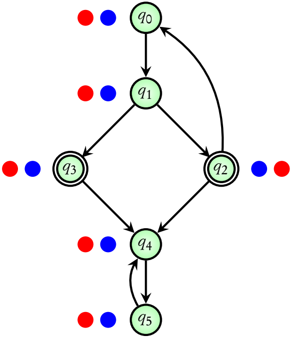



```
procedure nested_dfs()
    call dfs_blue ( s_0 )

procedure dfs_blue( s )
    s.blue := true
    for all t ∈ 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
```

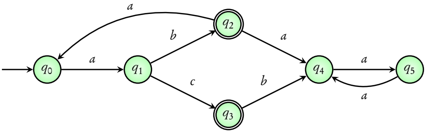


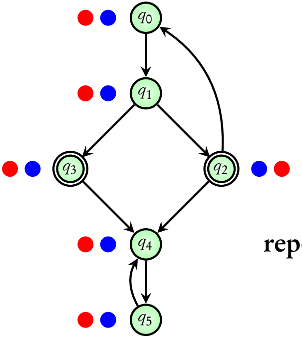


```
procedure nested_dfs()
    call dfs_blue ( s0 )

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
```



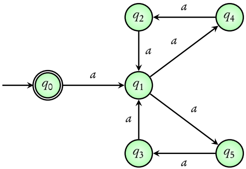


report cycle!

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

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

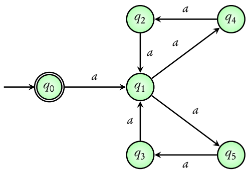
procedure *dfs_red* (*s*)
  s.red := **true**
  for all $t \in post(s)$ **do**
    if ¬*t.red* **then**
      call *dfs_red* (*t*)
    else if $t = seed$
      report cycle

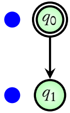


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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**

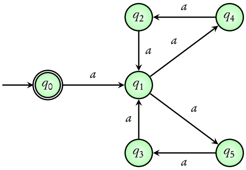
  for all $t \in post(s)$ **do**

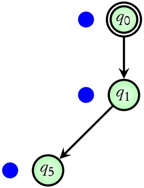
    if $\neg t.red$ **then**

      call *dfs_red* (t)

    else if $t = seed$

      report cycle

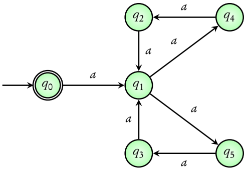


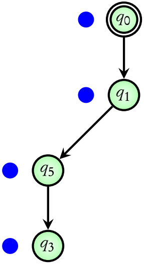



```
procedure nested_dfs()
    call dfs_blue(s0)

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
```

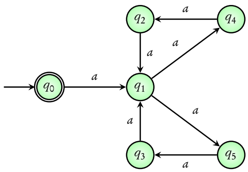


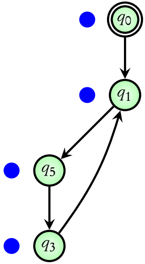


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

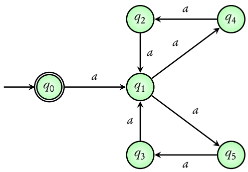


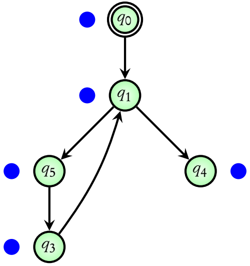


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

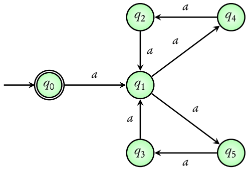


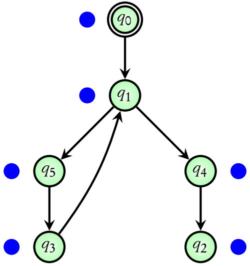


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

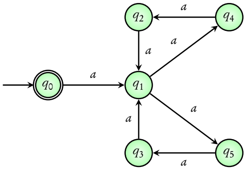


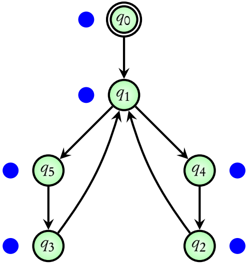



```
procedure nested_dfs()
    call dfs_blue ( s0 )

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
```

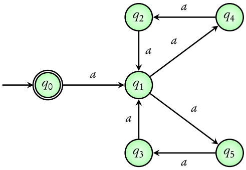


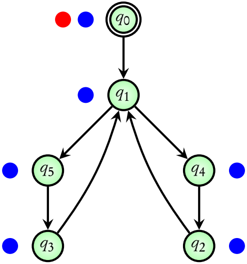



```
procedure nested_dfs( )
    call dfs_blue ( s0 )

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
```

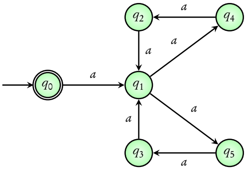


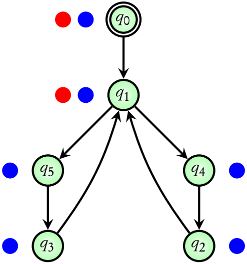


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

procedure *dfs_blue*(*s*)
    s.blue := **true**
    for all $t \in$ *post*(*s*) **do**
        if ¬*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 ¬*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**

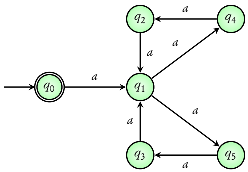
  for all $t \in post(s)$ **do**

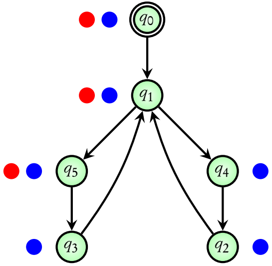
    if $\neg t.red$ **then**

      call *dfs_red* (t)

    else if $t = seed$

      report cycle

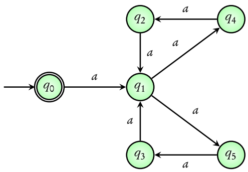


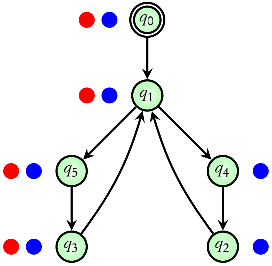


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

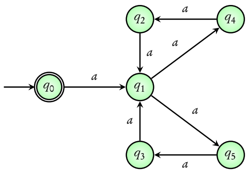


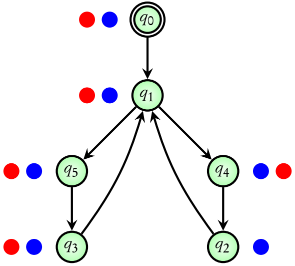


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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

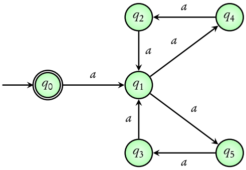


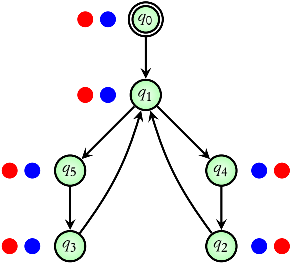


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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

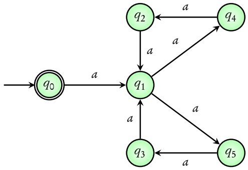


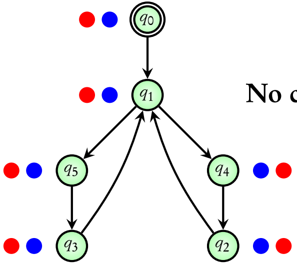


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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



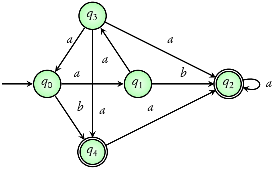


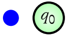
No cycle!

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

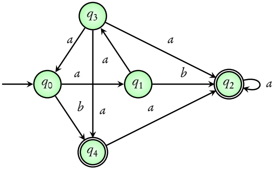


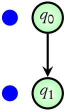


procedure *nested_dfs*( )
    call *dfs_blue* (s_0)

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 ¬*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**

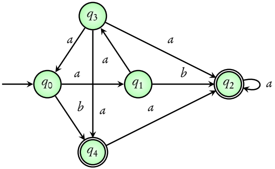
  for all $t \in$ *post*(*s*) **do**

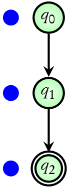
    if ¬*t.red* **then**

      call *dfs_red* (*t*)

    else if $t = seed$

      report cycle

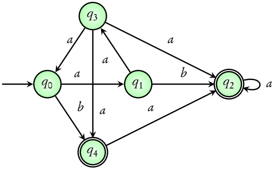


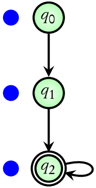



```
procedure nested_dfs()
    call dfs_blue (s0)

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
```

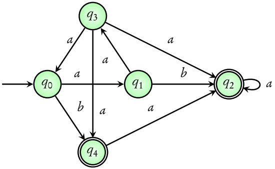


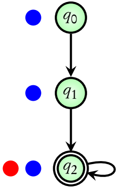



```
procedure nested_dfs()
    call dfs_blue (s0)

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
```

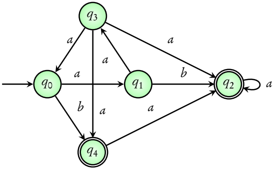


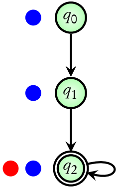


procedure *nested_dfs*( )
  call *dfs_blue* (s_0)

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





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

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

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 ?

Take-away

- **Büchi automata:** an automaton model for languages over infinite words

- **Closure properties** of Büchi Automata

- **Converting** ω-regular expressions to NBA

- **Nested DFS algorithm** for emptiness of NBA