Unit-4: Regular properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

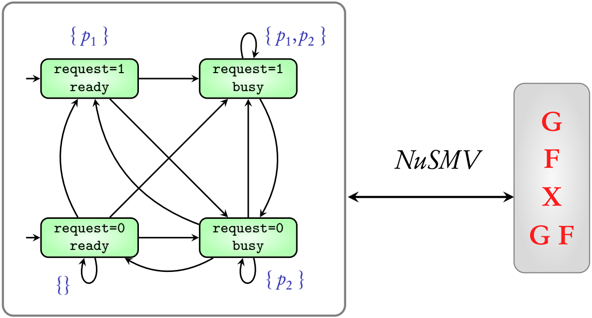
July - November 2015

Module 1:

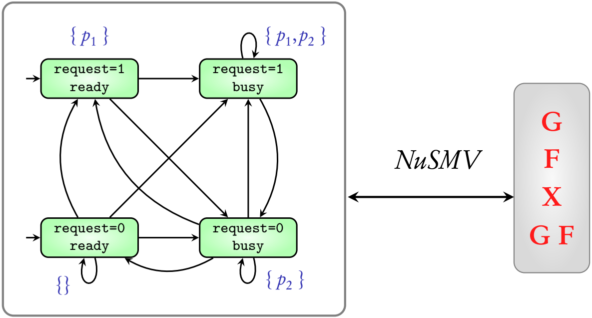
Road Map

Model

Requirements



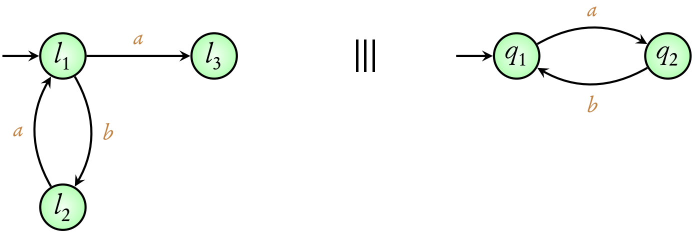
Model   Requirements



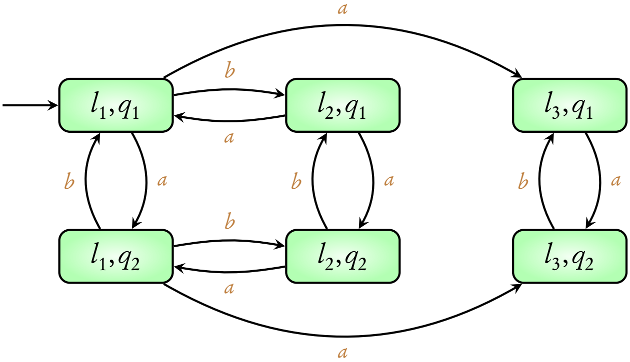
Question 1: What are the **algorithms** used for checking requirements on transition systems?

Coming next: A **major** challenge in designing model-checking algorithms

Recall...



l_1
l_2
l_3
a
a
b
|||
q_1
q_2
a
b



l_1, q_1
l_2, q_1
l_3, q_1
l_1, q_2
l_2, q_2
l_3, q_2
a
b
a
b
a
b
a
b
a
b
a
b
a

|States|: n_1 n_2

$\mathbf{TS_1}$ ||| $\mathbf{TS_2}$

Number of states in the interleaving

$n_1 \cdot n_2$

$|\text{States}|$: n_1 n_2 n_i n_k

$$\mathbf{TS}_1 \;|||\; \mathbf{TS}_2 \;|||\; \cdots \; \mathbf{TS}_i \; \cdots \;|||\; \mathbf{TS}_k$$

Number of states in the interleaving

$$n_1 \cdot n_2 \cdot \ldots \; n_i \cdot \ldots \cdot n_k$$

If there are 10 TS each with 10 states, interleaving would have 10^{10} **states!**

|States|: n_1 n_2 n_i n_k

$\mathbf{TS}_1 \;|||\; \mathbf{TS}_2 \;|||\; \cdots \; \mathbf{TS}_i \; \cdots \;|||\; \mathbf{TS}_k$

Number of states in the interleaving

$$n_1 \cdot n_2 \cdot \ldots \; n_i \cdot \ldots \cdot n_k$$

If there are 10 TS each with 10 states, interleaving would have 10^{10} **states!**

State-space explosion

NuSMV can handle more than 10^{120} states

NuSMV can handle more than 10^{120} states

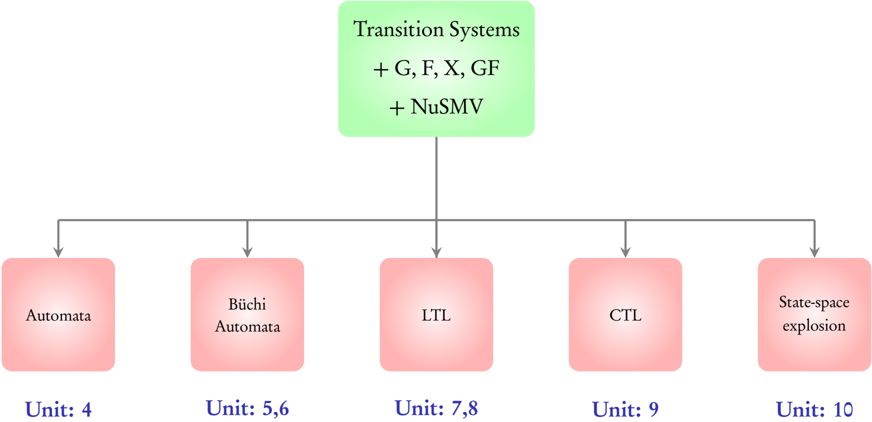
Question 2: How does NuSMV tackle state-space explosion?

Questions

Question 1: What are the **algorithms** used for checking requirements on transition systems?

Question 2: How does NuSMV tackle state-space explosion?

Course plan


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-4: Regular properties

B. Srivathsan

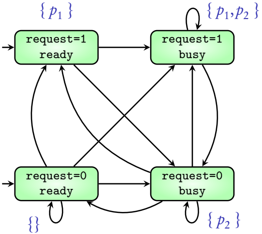
Chennai Mathematical Institute

NPTEL-course

July - November 2015

Module 2:

A gentle introduction to automata



AP = set of **atomic propositions**

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

A property over AP is a **subset** of AP-INF

Goal: Need **finite descriptions** of properties

Goal: Need **finite descriptions** of properties

Here: **Finite state automata** to describe sets of **words**

Goal: Need **finite descriptions** of properties

Here: **Finite state automata** to describe sets of finite **words**

Alphabet: $\{ a, b \}$

Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \ldots \}$$

Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \ldots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

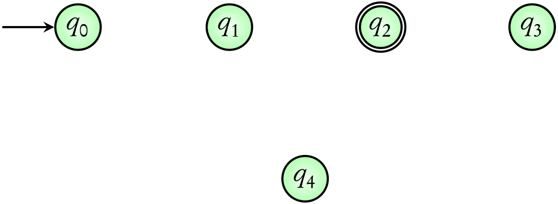
the set of **all paths** from an initial state to an accepting state equals L_1

Alphabet: $\{ a, b \}$

$$L_1 = \{ ab,\ abab,\ ababab,\ \dots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1

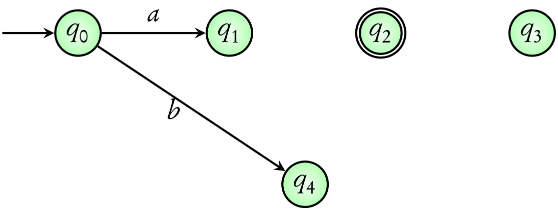


Alphabet: $\{ a, b \}$

$$L_1 = \{ ab,\ abab,\ ababab,\ \ldots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1

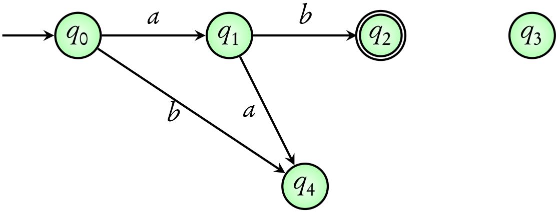


Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \dots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1

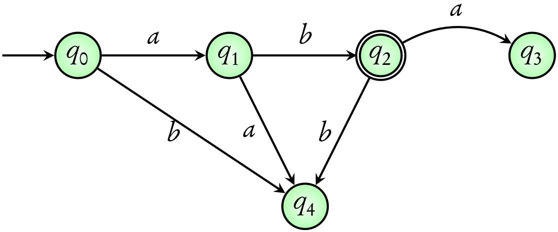


Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \ldots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1

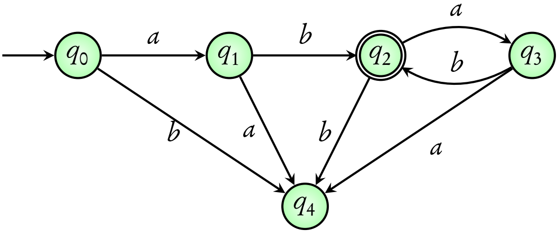


Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \dots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1

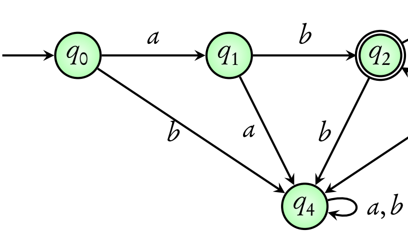


Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \ldots \}$$

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that

the set of **all paths** from an initial state to an accepting state equals L_1



Alphabet: $\{\, a, b \,\}$

$$L_2 = \{\, a,\ aa,\ ab,\ aaa,\ aab,\ aba,\ abb\ ,\ \ldots \}$$

L_2 is the set of all words starting with a

Alphabet: $\{ a, b \}$

$$L_2 = \{ a,\ aa,\ ab,\ aaa,\ aab,\ aba,\ abb\ ,\ \ldots \}$$

L_2 is the set of all words starting with a

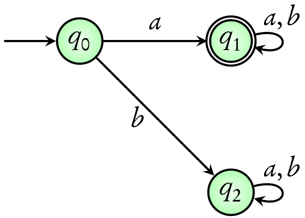
Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that the set of **all paths** from an initial state to an accepting state equals L_2

Alphabet: $\{ a, b \}$

$$L_2 = \{ a,\ aa,\ ab,\ aaa,\ aab,\ aba,\ abb\ ,\ \ldots \}$$

L_2 is the set of all words starting with a

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that the set of **all paths** from an initial state to an accepting state equals L_2

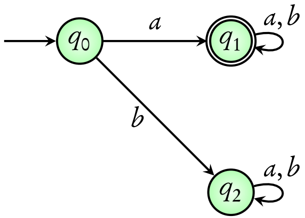


Alphabet: $\{ a, b \}$

$$L_2 = \{ a, aa, ab, aaa, aab, aba, abb, \ldots \}$$

L_2 is the set of all words starting with a

Design a TS with actions $\{ a, b \}$ and mark some states as **accepting** so that the set of **all paths** from an initial state to an accepting state equals L_2



Finite Automaton

Coming next: Some terminology

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

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

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

$$
\begin{aligned}
\textbf{Alphabet} \quad \Sigma &= \{ a, b \} \\[1em]
\Sigma \cdot \Sigma &= \{ a, b \} \cdot \{ a, b \} \\
&= \{ aa, ab, ba, bb \}
\end{aligned}
$$

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

$$\Sigma \cdot \Sigma = \{ a, b \} \cdot \{ a, b \} = \{ aa, ab, ba, bb \}$$

Σ^1 = words of length 1

Σ^2 = words of length 2

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

$$\begin{aligned} \Sigma \cdot \Sigma &= \{ a, b \} \cdot \{ a, b \} \\ &= \{ aa, ab, ba, bb \} \end{aligned}$$

Σ^1 = words of length 1

Σ^2 = words of length 2

Σ^3 = words of length 3

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

$$\Sigma \cdot \Sigma = \{ a, b \} \cdot \{ a, b \} = \{ aa, ab, ba, bb \}$$

Σ^1 = words of length 1

Σ^2 = words of length 2

Σ^3 = words of length 3

$\vdots$

Σ^k = words of length k

$\vdots$

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

$$\begin{aligned} \Sigma \cdot \Sigma &= \{ a, b \} \cdot \{ a, b \} \\ &= \{ aa, ab, ba, bb \} \end{aligned}$$

$$\begin{aligned} aba \cdot \epsilon &= aba \\ \epsilon \cdot bbb &= bbb \\ w \cdot \epsilon &= w \\ \epsilon \cdot w &= w \end{aligned}$$

$\Sigma^0 = \{ \epsilon \}$ (empty word, with length 0)

$\Sigma^1 =$ words of length 1

$\Sigma^2 =$ words of length 2

$\Sigma^3 =$ words of length 3

$\vdots$

$\Sigma^k =$ words of length k

$\vdots$

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

$$\Sigma \cdot \Sigma = \{ a, b \} \cdot \{ a, b \} = \{ aa, ab, ba, bb \}$$

$$aba \cdot \epsilon = aba$$

$$\epsilon \cdot bbb = bbb$$

$$w \cdot \epsilon = w$$

$$\epsilon \cdot w = w$$

$\Sigma^0 = \{ \epsilon \}$ (empty word, with length 0)

Σ^1 = words of length 1

Σ^2 = words of length 2

Σ^3 = words of length 3

$\vdots$

Σ^k = words of length k

$\vdots$

$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$

= **set of all finite length words**

$\Sigma^* \ = \$ set of **all words** over Σ

$$\Sigma^* \;=\; \text{set of } \textbf{all words} \text{ over } \Sigma$$

Any set of words is called a **language**

$\Sigma^* =$ set of **all words** over Σ

Any set of words is called a **language**

$\{ ab, abab, ababab, \ldots \}$

words starting with an a

words starting with a b

$\{ \epsilon, b, bb, bbb, \ldots \}$

$\{ \epsilon, ab, abab, ababab, \ldots \}$

$\{ \epsilon, bbb, bbbbbb, (bbb)^3, \ldots \}$

words starting and ending with an a

$\{ \epsilon, ab, aabb, aaabbb, a^4b^4 \ldots \}$

$\Sigma^* =$ set of **all words** over Σ

Any set of words is called a **language**

$\{ab, abab, ababab, \ldots\}$

$a\Sigma^*$ words starting with an a

words starting with a b

$\{\epsilon, b, bb, bbb, \ldots\}$

$\{\epsilon, ab, abab, ababab, \ldots\}$

$\{\epsilon, bbb, bbbbbb, (bbb)^3, \ldots\}$

words starting and ending with an a

$\{\epsilon, ab, aabb, aaabbb, a^4b^4 \ldots\}$

$\Sigma^* \;=\;$ set of **all words** over Σ

Any set of words is called a **language**

$\{ ab, abab, ababab, \ldots \}$

$a\Sigma^*$ words starting with an a

$b\Sigma^*$ words starting with a b

$\{ \epsilon, b, bb, bbb, \ldots \}$

$\{ \epsilon, ab, abab, ababab, \ldots \}$

$\{ \epsilon, bbb, bbbbbb, (bbb)^3, \ldots \}$

words starting and ending with an a

$\{ \epsilon, ab, aabb, aaabbb, a^4b^4 \ldots \}$

$\Sigma^* \; = \;$ set of **all words** over Σ

Any set of words is called a **language**

$\{\, ab,\; abab,\; ababab,\; \ldots \}$

$a\Sigma^*$ words starting with an a

$b\Sigma^*$ words starting with a b

b^* $\{\, \epsilon,\; b,\; bb,\; bbb,\; \ldots \}$

$\{\, \epsilon,\; ab,\; abab,\; ababab,\; \ldots \}$

$\{\, \epsilon,\; bbb,\; bbbbbb,\; (bbb)^3,\; \ldots \}$

words starting and ending with an a

$\{\, \epsilon,\; ab,\; aabb,\; aaabbb, a^4b^4\; \ldots \}$

$\Sigma^* \; = \;$ set of **all words** over Σ

Any set of words is called a **language**

$\{\, ab,\ abab,\ ababab,\ \ldots \}$

$a\Sigma^*$ words starting with an a

$b\Sigma^*$ words starting with a b

b^* $\{\, \epsilon,\ b,\ bb,\ bbb,\ \ldots \}$

$(ab)^*$ $\{\, \epsilon,\ ab,\ abab,\ ababab,\ \ldots \}$

$\{\, \epsilon,\ bbb,\ bbbbbb,\ (bbb)^3,\ \ldots \}$

words starting and ending with an a

$\{\, \epsilon,\ ab,\ aabb,\ aaabbb, a^4 b^4\ \ldots \}$

$\Sigma^* \;=\;$ set of **all words** over Σ

Any set of words is called a **language**

$\{\, ab,\ abab,\ ababab,\ \ldots \}$

$a\Sigma^*$ words starting with an a

$b\Sigma^*$ words starting with a b

b^* $\{\, \epsilon,\ b,\ bb,\ bbb,\ \ldots \}$

$(ab)^*$ $\{\, \epsilon,\ ab,\ abab,\ ababab,\ \ldots \}$

$(bbb)^*$ $\{\, \epsilon,\ bbb,\ bbbbbb,\ (bbb)^3,\ \ldots \}$

words starting and ending with an a

$\{\, \epsilon,\ ab,\ aabb,\ aaabbb, a^4b^4\ \ldots \}$

Σ^* = set of **all words** over Σ

Any set of words is called a **language**

$\{ab, abab, ababab, \ldots\}$

$a\Sigma^*$ words starting with an a

$b\Sigma^*$ words starting with a b

b^* $\{\epsilon, b, bb, bbb, \ldots\}$

$(ab)^*$ $\{\epsilon, ab, abab, ababab, \ldots\}$

$(bbb)^*$ $\{\epsilon, bbb, bbbbbb, (bbb)^3, \ldots\}$

$a\Sigma^*a$ words starting and ending with an a

$\{\epsilon, ab, aabb, aaabbb, a^4b^4 \ldots\}$

In this module...

Task: Design **Finite Automata** for some languages

Words

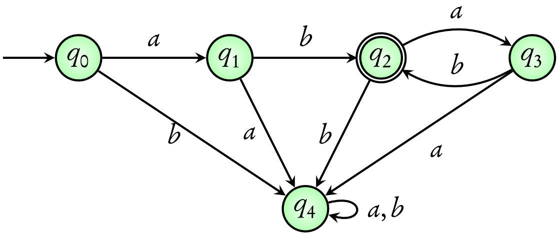
Languages

Finite Automata

Alphabet: $\{ a, b \}$

$$L_1 = \{ ab, abab, ababab, \ldots \}$$

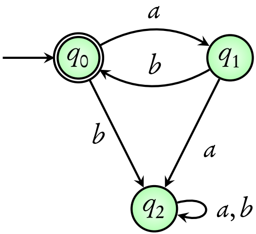
Design a Finite automaton for L_1



Alphabet: $\{a, b\}$

$$L_3 = \{\, \epsilon,\ ab,\ abab,\ ababab,\ \ldots\}$$

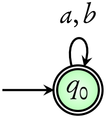
Design a Finite automaton for L_3



Alphabet: $\{ a, b \}$

$$\Sigma^* = \{ \epsilon, a, b, aa, ab, ba, bb \ldots \}$$

Design a Finite automaton for Σ^*



Alphabet: $\{ a, b \}$

$$a^* = \{ \epsilon, a, aa, aaa, aaaa, a^5, \ldots \}$$

a^* is the set of all words having only a

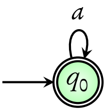
Design a Finite automaton for a^*

Alphabet: $\{ a, b \}$

$$a^* = \{ \epsilon,\ a,\ aa,\ aaa,\ aaaa,\ a^5,\ \ldots \}$$

a^* is the set of all words having only a

Design a Finite automaton for a^*

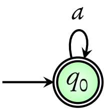


Alphabet: $\{ a, b \}$

$$a^* = \{ \epsilon,\ a,\ aa,\ aaa,\ aaaa,\ a^5,\ \ldots \}$$

a^* is the set of all words having only a

Design a Finite automaton for a^*



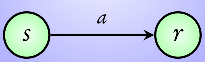
Non-deterministic automaton

Transition Systems

Deterministic

Single initial state

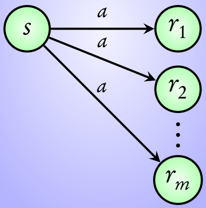
and



Non-deterministic

Multiple initial states

or

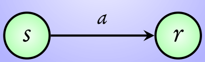


Transition Systems

Deterministic

Single initial state

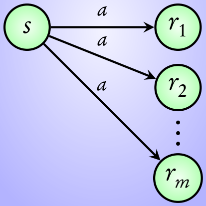
and



Non-deterministic

Multiple initial states

or



Same applies in the case of Finite Automata

Alphabet: $\{ a, b \}$

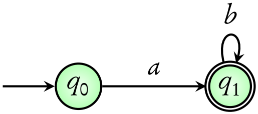
$$ab^* = \{ a, ab, ab^2, ab^3, ab^4, \ldots \}$$

Design a Finite automaton for ab^*

Alphabet: $\{ a, b \}$

$$ab^* = \{ a, ab, ab^2, ab^3, ab^4, \ldots \}$$

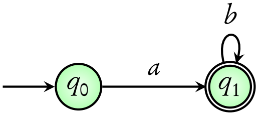
Design a Finite automaton for ab^*



Alphabet: $\{ a, b \}$

$$ab^* = \{ a,\ ab,\ ab^2,\ ab^3,\ ab^4,\ \ldots \}$$

Design a Finite automaton for ab^*



Non-deterministic automaton

Alphabet: $\{a, b\}$

$$ab^* = \{a, ab, ab^2, ab^3, ab^4, \dots\}$$

$$ba^* = \{b, ba, ba^2, ba^3, ba^4, \dots\}$$

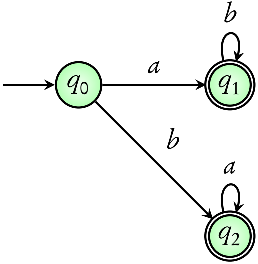
Design a Finite automaton for $ab^* \cup ba^*$

Alphabet: $\{a, b\}$

$$ab^* = \{a, ab, ab^2, ab^3, ab^4, \ldots\}$$

$$ba^* = \{b, ba, ba^2, ba^3, ba^4, \ldots\}$$

Design a Finite automaton for $ab^* \cup ba^*$

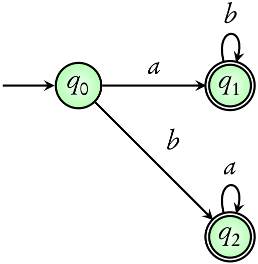


Alphabet: $\{a, b\}$

$$ab^* = \{a, ab, ab^2, ab^3, ab^4, \ldots\}$$

$$ba^* = \{b, ba, ba^2, ba^3, ba^4, \ldots\}$$

Design a Finite automaton for $ab^* \cup ba^*$



Non-deterministic automaton

Alphabet: $\{ a, b \}$

$$ab^* = \{ a,\ ab,\ ab^2,\ ab^3,\ ab^4,\ \ldots \}$$

$$ba^* = \{ b,\ ba,\ ba^2,\ ba^3,\ ba^4,\ \ldots \}$$

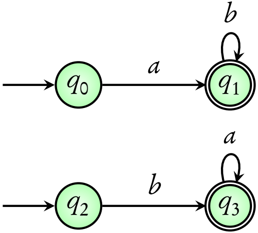
Design a Finite automaton for $ab^* \cup ba^*$

Alphabet: $\{ a, b \}$

$$ab^* = \{ a,\ ab,\ ab^2,\ ab^3,\ ab^4,\ \ldots \}$$

$$ba^* = \{ b,\ ba,\ ba^2,\ ba^3,\ ba^4,\ \ldots \}$$

Design a Finite automaton for $ab^* \cup ba^*$

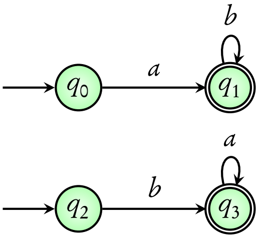


Alphabet: $\{ a, b \}$

$$ab^* = \{ a,\ ab,\ ab^2,\ ab^3,\ ab^4,\ \ldots \}$$

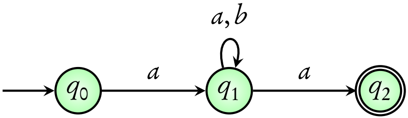
$$ba^* = \{ b,\ ba,\ ba^2,\ ba^3,\ ba^4,\ \ldots \}$$

Design a Finite automaton for $ab^* \cup ba^*$

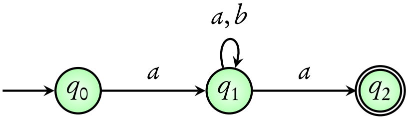


Multiple initial states: **non-deterministic** automaton

What is the language of the following automaton?



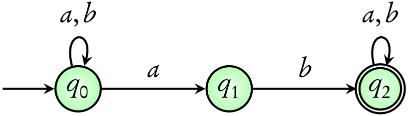
What is the language of the following automaton?



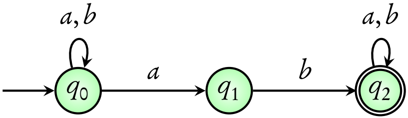
Answer: $a\,\Sigma^* a$

words starting and ending with *a*

What is the language of the following automaton?



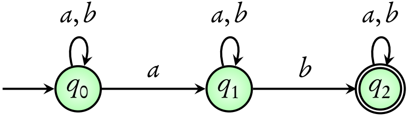
What is the language of the following automaton?



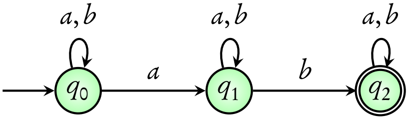
Answer: $\Sigma^* ab\, \Sigma^*$

words **containing** *ab*

What is the language of the following automaton?



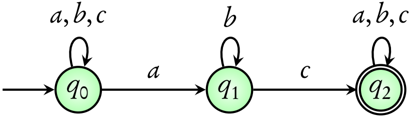
What is the language of the following automaton?



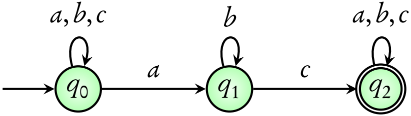
Answer: $\Sigma^* a \, \Sigma^* \, b \, \Sigma^*$

words where there exists an **a** followed by a **b** after sometime

What is the language of the following automaton?



What is the language of the following automaton?



Answer: $\Sigma^* a\, b^*\, c\, \Sigma^* \quad (\Sigma = \{ a, b, c \})$

words where there exists an **a** followed by only **b**'s and after sometime a **c** occurs

Alphabet: $\{ a, b \}$

$$L = \{ \epsilon, ab, aabb, aaabbb, \ldots, a^i b^i, \ldots \}$$

Can we design a Finite automaton for L?

Alphabet: $\{ a, b \}$

$$L = \{ \epsilon, ab, aabb, aaabbb, \ldots, a^i b^i, \ldots \}$$

Can we design a Finite automaton for L?

Need **infinitely many states** to remember the number of a's

Alphabet: $\{ a, b \}$

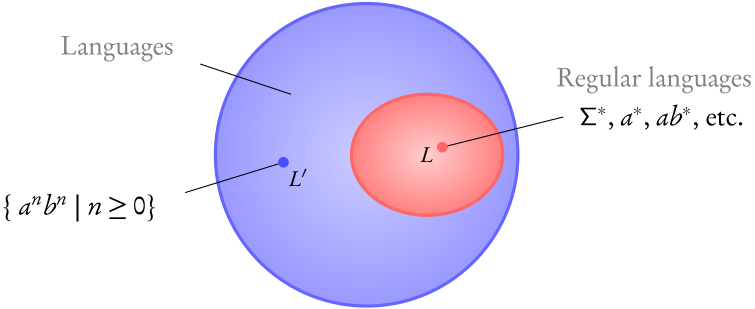
$$L = \{ \epsilon, ab, aabb, aaabbb, \ldots, a^i b^i, \ldots \}$$

Can we design a Finite automaton for L?

Need **infinitely many states** to remember the number of a's

Cannot construct finite automaton for this language

Regular languages



Definition

A language is called **regular** if it can be **accepted** by a finite automaton

Words

Languages

Finite Automata

Deterministic (DFA)

Non-deterministic (NFA)

Regular languages

Words

Languages

Finite Automata

Deterministic (DFA)

Non-deterministic (NFA)

Regular languages

Next module: Are DFA and NFA **equivalent**?

Unit-4: Regular properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Module 3:

Simple properties of finite automata

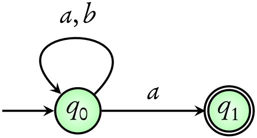
Determinization

Product construction

Emptiness

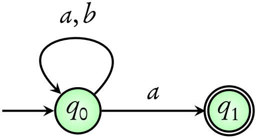
Complementation

Union



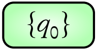
Non-deterministic automaton

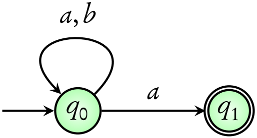
$\Sigma^* a$: words ending with an a



Non-deterministic automaton

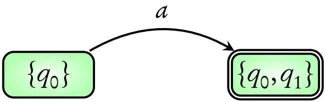
$\Sigma^* a$: words ending with an a

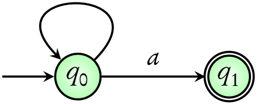




Non-deterministic automaton

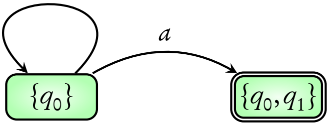
$\Sigma^* a$: words ending with an a

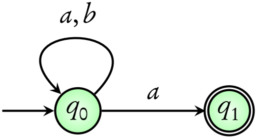




Non-deterministic automaton

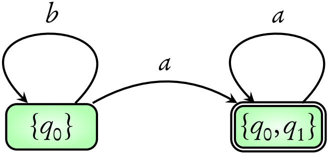
$\Sigma^* a$: words ending with an a

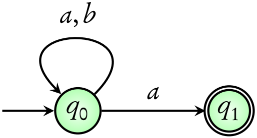




Non-deterministic automaton

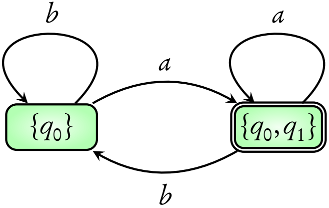
$\Sigma^* a$: words ending with an a

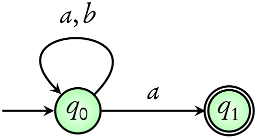




Non-deterministic automaton

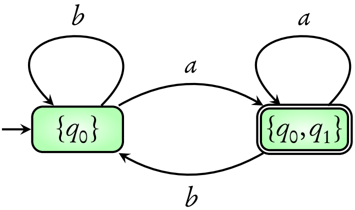
$\Sigma^* a$: words ending with an a

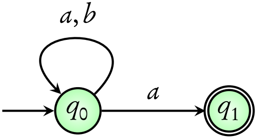




Non-deterministic automaton

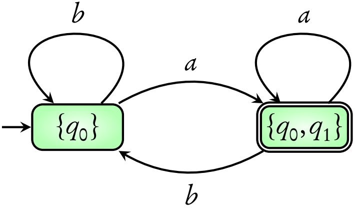
$\Sigma^* a$: words ending with an a



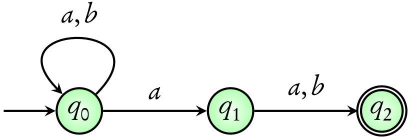


Non-deterministic automaton

$\Sigma^* a$: words ending with an a

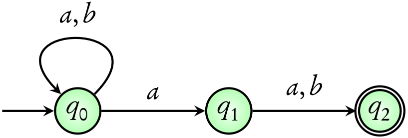


Deterministic automaton



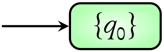
NFA

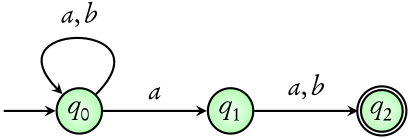
$\Sigma^* a \, \Sigma$: words where the second last letter is a



NFA

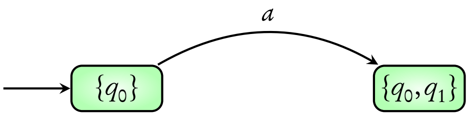
$\Sigma^* a\, \Sigma$: words where the second last letter is a

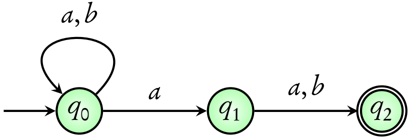




NFA

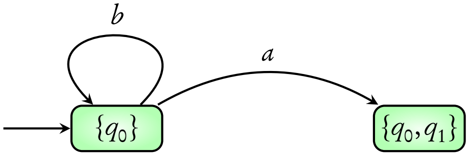
$\Sigma^* a\, \Sigma$: words where the second last letter is a

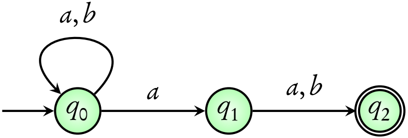




NFA

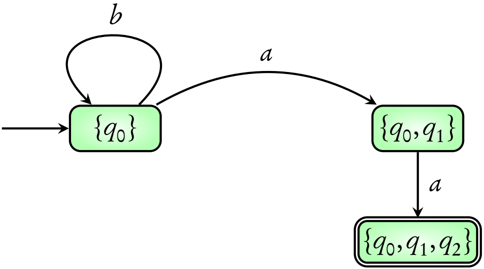
$\Sigma^* a\, \Sigma$: words where the second last letter is a

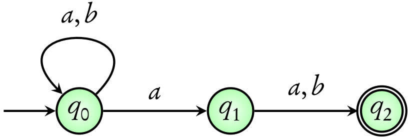




NFA

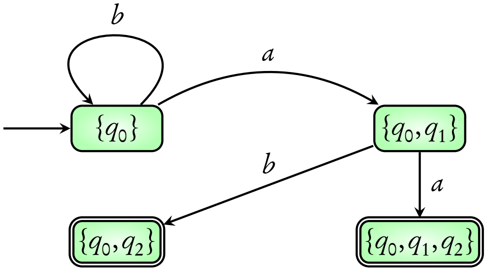
$\Sigma^* a\, \Sigma$: words where the second last letter is a

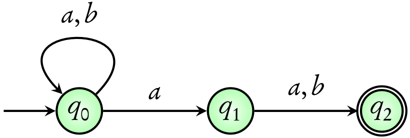




NFA

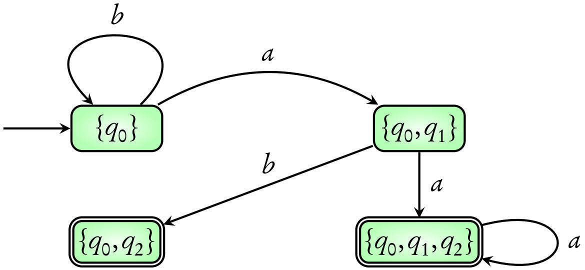
$\Sigma^* a\ \Sigma$: words where the second last letter is a

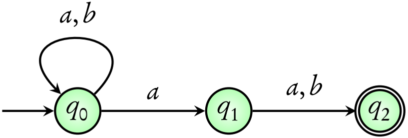




NFA

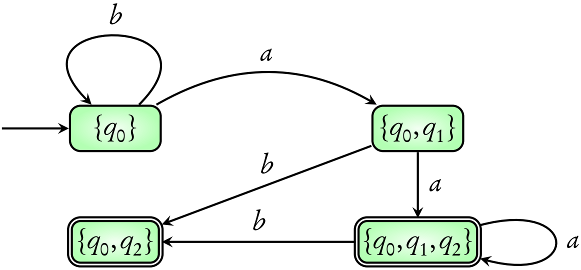
$\Sigma^* a \, \Sigma$: words where the second last letter is a

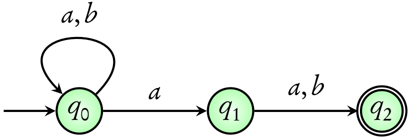




NFA

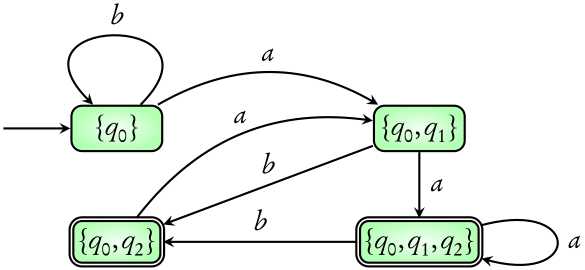
$\Sigma^* a\, \Sigma$: words where the second last letter is a

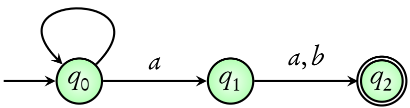




NFA

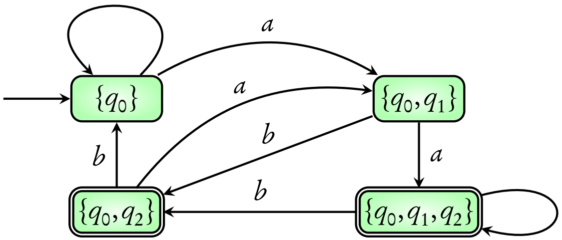
$\Sigma^* a\, \Sigma$: words where the second last letter is a

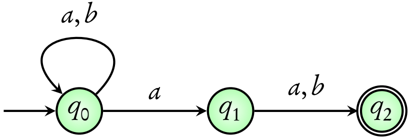




NFA

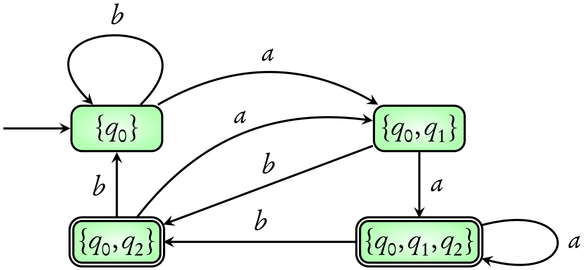
$\Sigma^* a \, \Sigma$: words where the second last letter is a



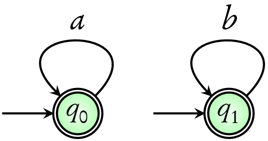


NFA

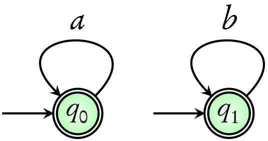
$\Sigma^* a\ \Sigma$: words where the second last letter is a



DFA

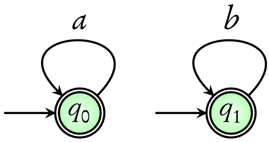


NFA



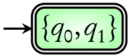
NFA

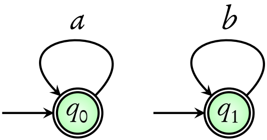
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ



NFA

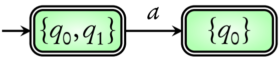
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

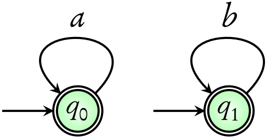




NFA

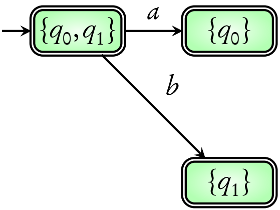
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

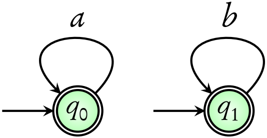




NFA

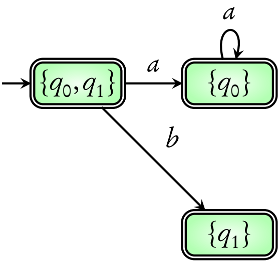
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

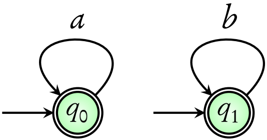




NFA

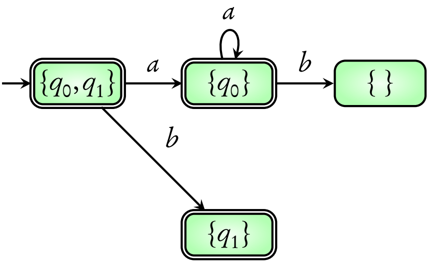
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

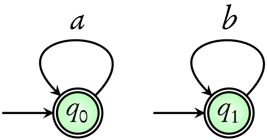




NFA

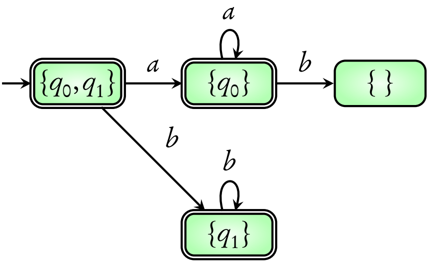
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

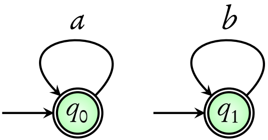




NFA

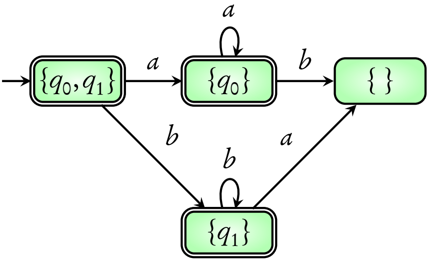
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

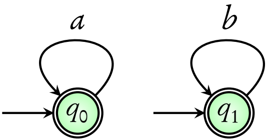




NFA

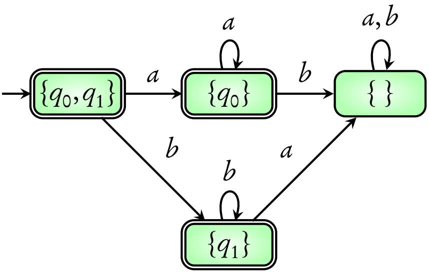
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ

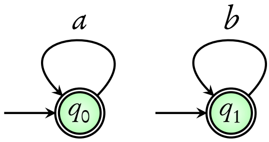




NFA

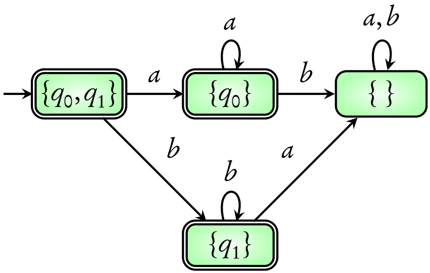
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ





NFA

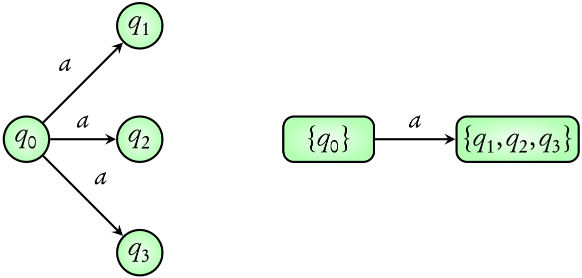
$a^* \cup b^*$: words of the form a^i, b^i, or ϵ



DFA

Subset construction

Every NFA can be converted to an **equivalent** DFA



Determinization

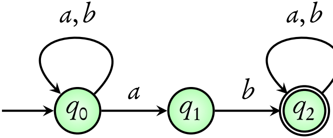
Subset construction

Product construction

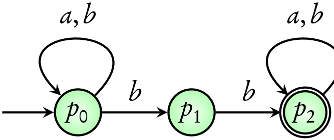
Emptiness

Complementation

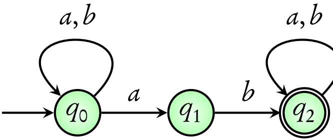
Union



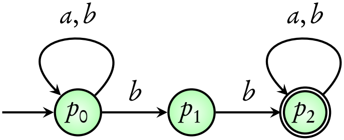
$\Sigma^* ab \Sigma^*$



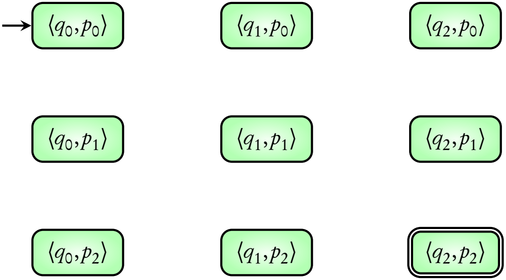
$\Sigma^* bb \Sigma^*$

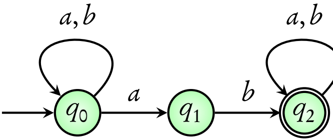


$\Sigma^* ab \Sigma^*$

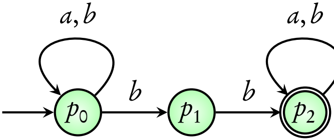


$\Sigma^* bb \Sigma^*$

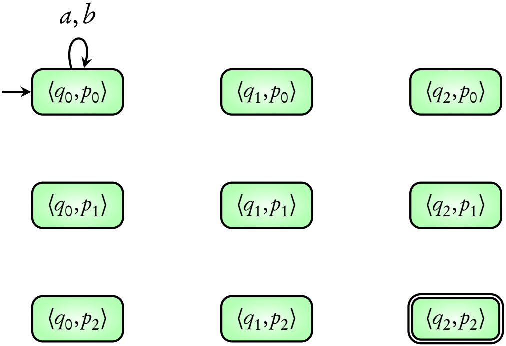


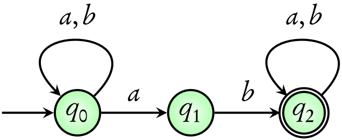


$\Sigma^* ab \Sigma^*$

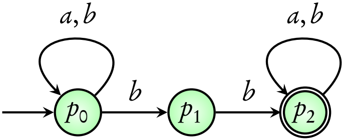


$\Sigma^* bb \Sigma^*$

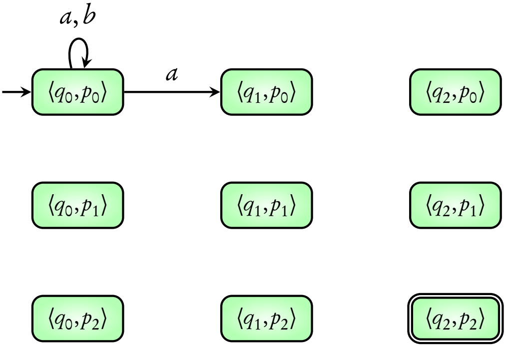


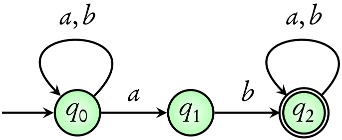


$\Sigma^* ab \Sigma^*$

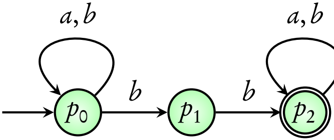


$\Sigma^* bb \Sigma^*$

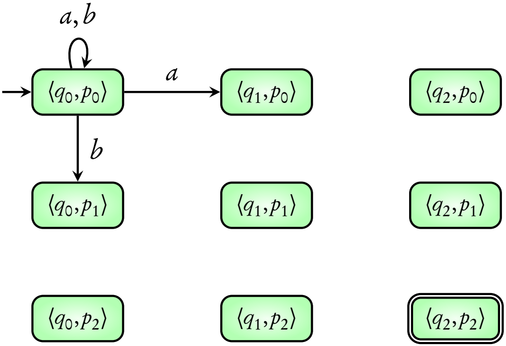


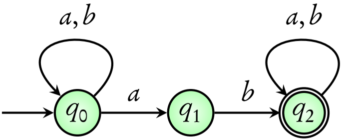


$\Sigma^* ab \Sigma^*$

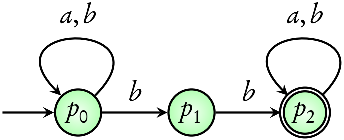


$\Sigma^* bb \Sigma^*$

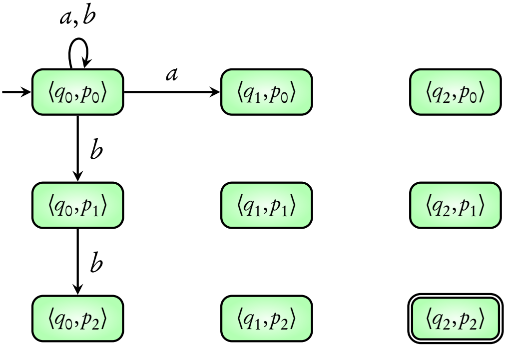


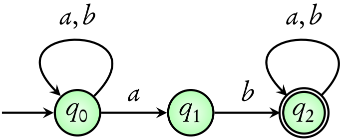


$\Sigma^* ab \Sigma^*$

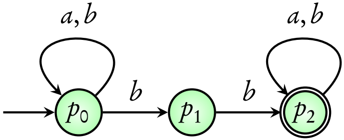


$\Sigma^* bb \Sigma^*$

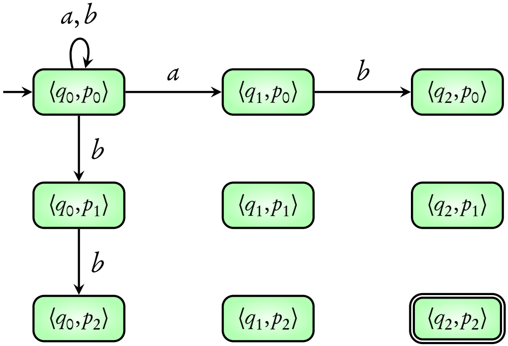


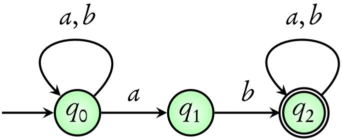


$\Sigma^* ab \Sigma^*$

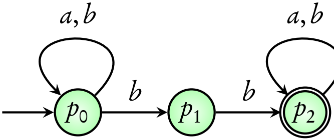


$\Sigma^* bb \Sigma^*$

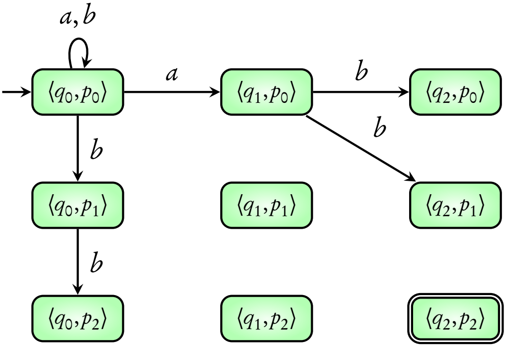


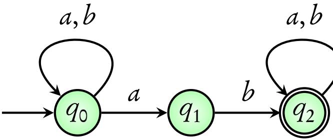


$\Sigma^* ab \Sigma^*$

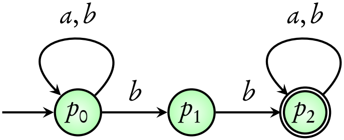


$\Sigma^* bb \Sigma^*$

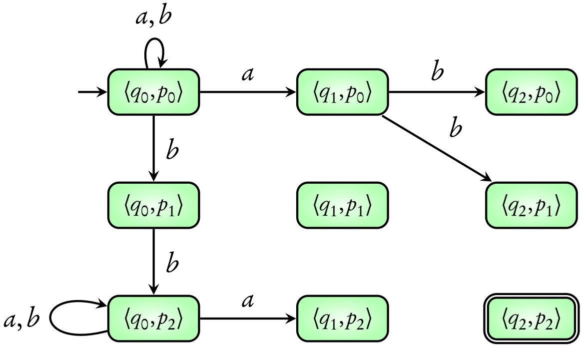


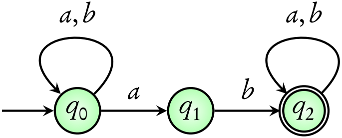


$\Sigma^* ab \Sigma^*$

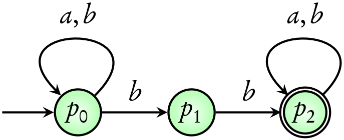


$\Sigma^* bb \Sigma^*$

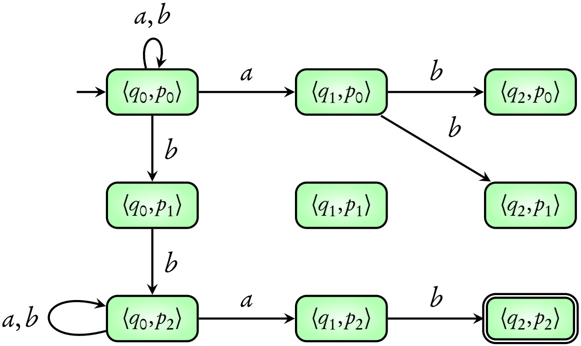


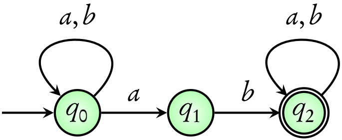


$\Sigma^* ab \Sigma^*$

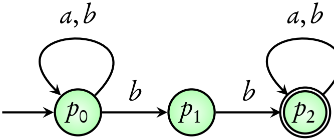


$\Sigma^* bb \Sigma^*$

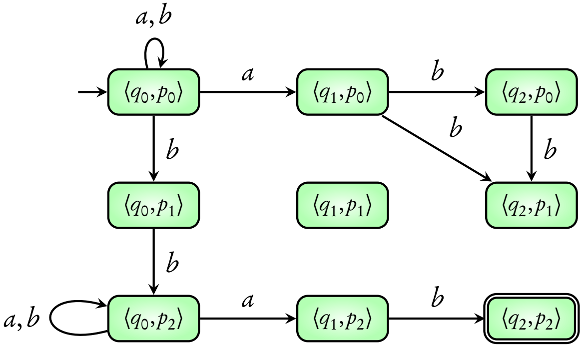


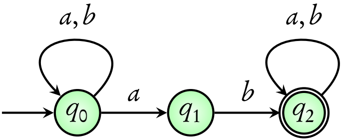


$\Sigma^* ab \Sigma^*$

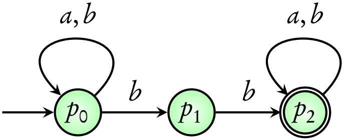


$\Sigma^* bb \Sigma^*$

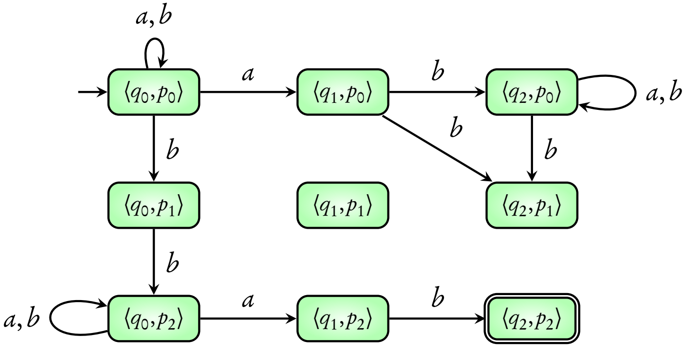


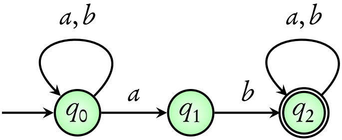


$\Sigma^* ab \Sigma^*$

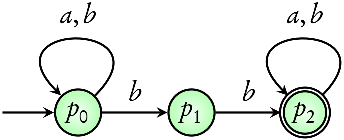


$\Sigma^* bb \Sigma^*$

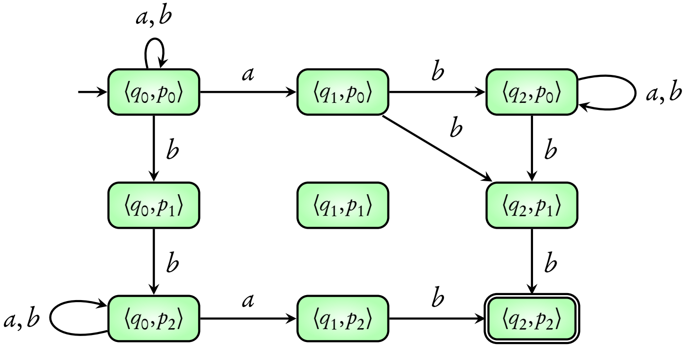


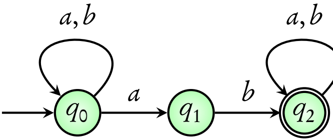


$\Sigma^* ab \Sigma^*$

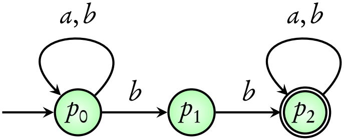


$\Sigma^* bb \Sigma^*$

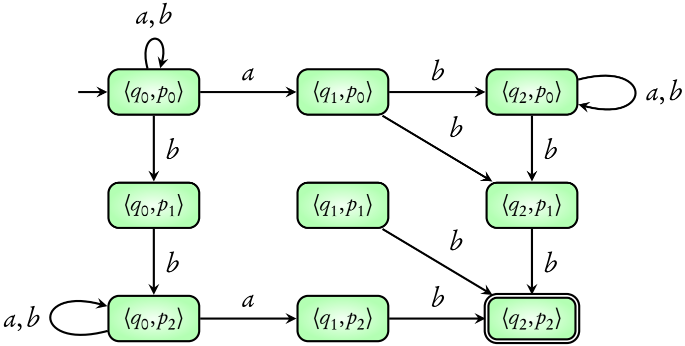


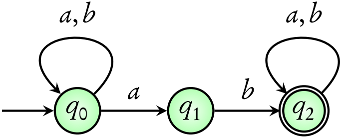


$\Sigma^* ab \Sigma^*$

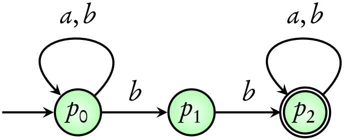


$\Sigma^* bb \Sigma^*$

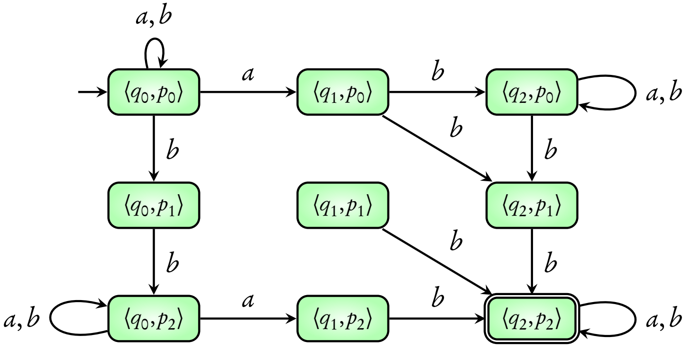


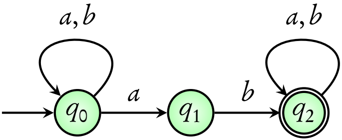


$\Sigma^* ab \Sigma^*$

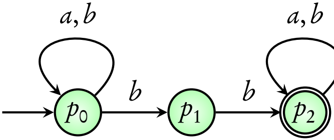


$\Sigma^* bb \Sigma^*$

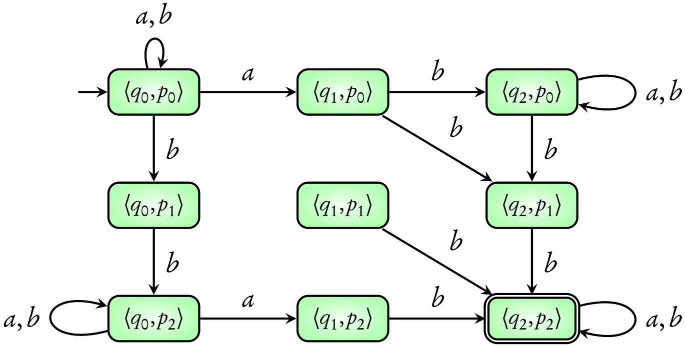




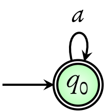
$\Sigma^* ab \Sigma^*$



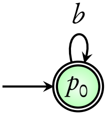
$\Sigma^* bb \Sigma^*$



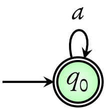
$\Sigma^* ab \Sigma^* \cap \Sigma^* bb \Sigma^*$: words containing both *ab* and *bb*



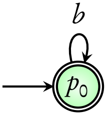
a^*



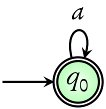
b^*

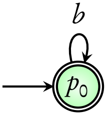


a^*



b^*





a^* b^*

$a^* \cap b^* = \{ \epsilon \}$

Synchronous product

Gives the **intersection** of the two languages

Determinization

Subset construction

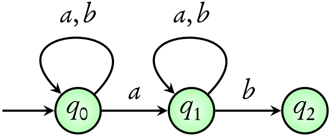
Product construction

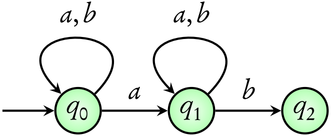
Intersection of languages

Emptiness

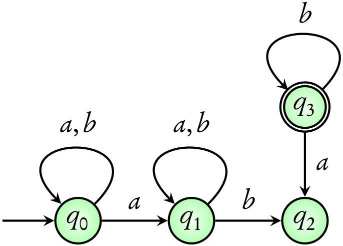
Complementation

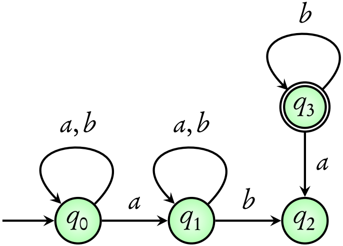
Union


a, b
a, b
a
b
q0
q1
q2

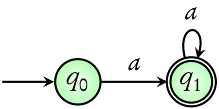


Language is empty as there is no accepting state

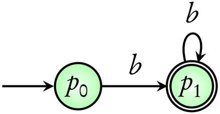

b
a, b
a, b
q_3
a
a
b
q_0
q_1
q_2



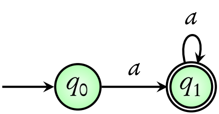
Language is empty as accepting state is **not reachable**

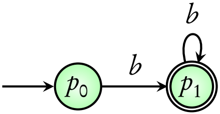


aa^*



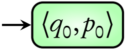
bb^*

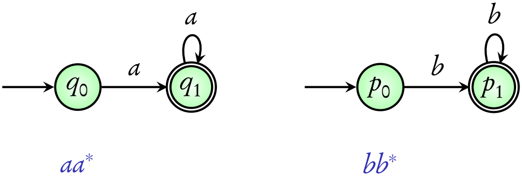


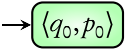


aa^*

bb^*







Language is empty as there is **no accepting state**

Question: Given NFA $\mathscr{A}$, is language accepted by $\mathscr{A}$ empty?

Question: Given NFA $\mathscr{A}$, is language accepted by $\mathscr{A}$ empty?

Emptiness of NFA

Language of an NFA is empty if and only if it has **no reachable accepting states**

Question: Given NFA $\mathscr{A}$, is language accepted by $\mathscr{A}$ empty?

Emptiness of NFA

Language of an NFA is empty if and only if it has **no reachable accepting states**

Algorithm

Run a **depth-first or breadth-first search** to find if there is a path to an accepting state

Determinization

Subset construction

Product construction

Intersection of languages

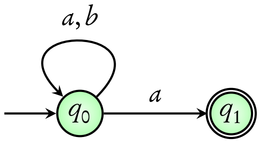
Emptiness

Algorithm for emptiness

Complementation

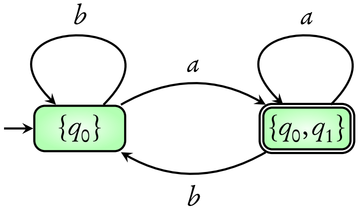
Union

NFA

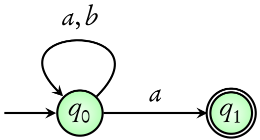


$\Sigma^* a$: words ending with an a

DFA

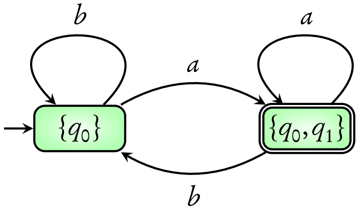


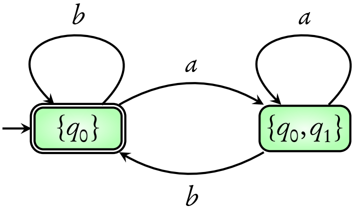
NFA



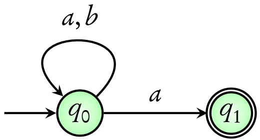
$\Sigma^* a$: words ending with an a

DFA



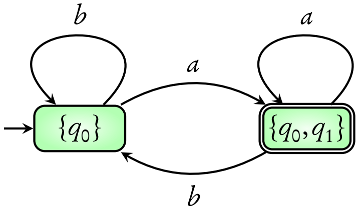


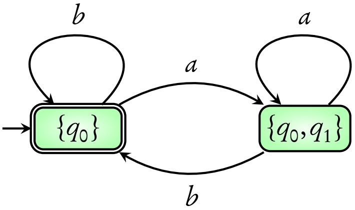
NFA



$\Sigma^* a$: words ending with an a

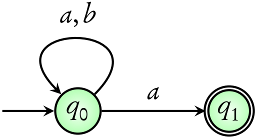
DFA

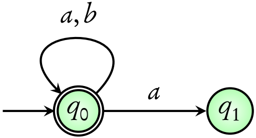




complement of $\Sigma^* a$

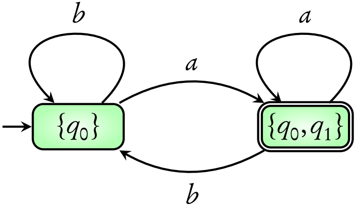
NFA

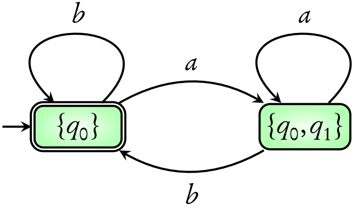




$\Sigma^* a$: words ending with an a

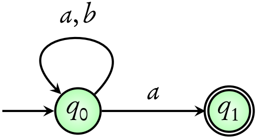
DFA



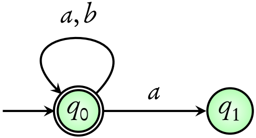


complement of $\Sigma^* a$

NFA

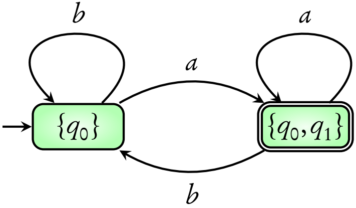


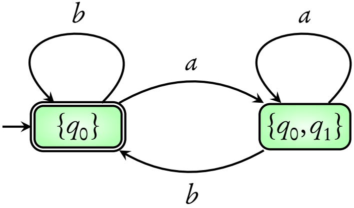
$\Sigma^* a$: words ending with an a



not the complement!

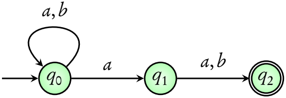
DFA





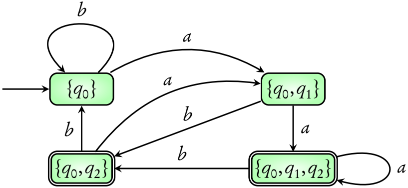
complement of $\Sigma^* a$

NFA

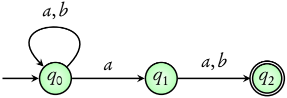


$\Sigma^* a\, \Sigma$: words where the second last letter is a

DFA

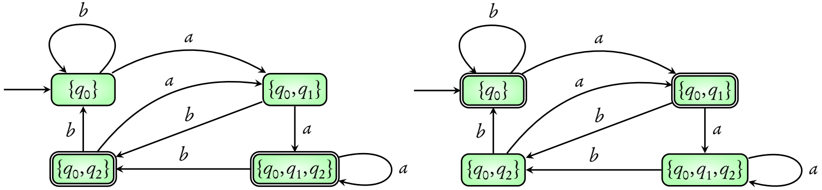


NFA

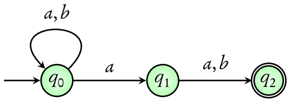


$\Sigma^* a\ \Sigma$: words where the second last letter is a

DFA

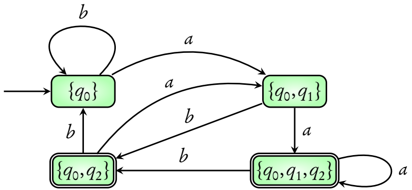


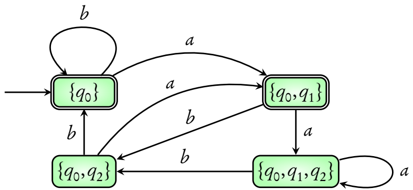
NFA



$\Sigma^* a\, \Sigma$: words where the second last letter is a

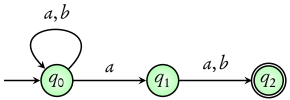
DFA

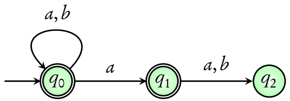




complement of $\Sigma^* a \Sigma$

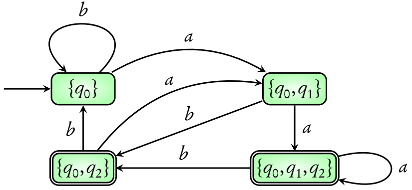
NFA

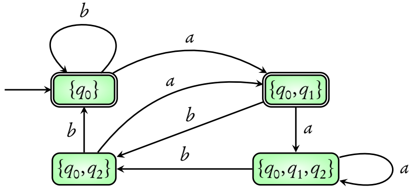




$\Sigma^* a\ \Sigma$: words where the second last letter is a

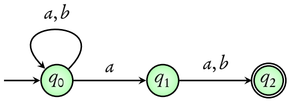
DFA

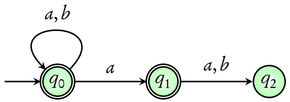




complement of $\Sigma^* a \Sigma$

NFA

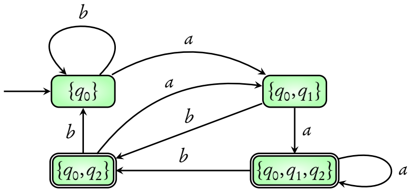


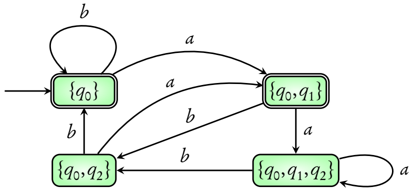


$\Sigma^* a \, \Sigma$: words where the second last letter is a

not the complement!

DFA





complement of $\Sigma^* a \Sigma$

Complementation

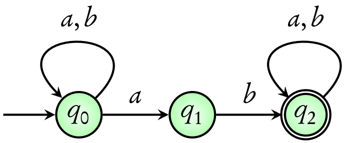
Interchange accepting and non-accepting states in a DFA

Complementation

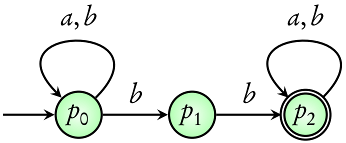
Interchange accepting and non-accepting states in a DFA

Does not work in the case of NFA

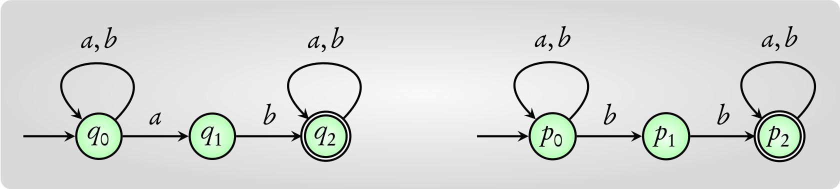
Coming next: Union of two regular languages



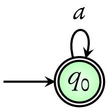
$\Sigma^* ab \Sigma^*$

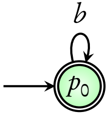


$\Sigma^* bb \Sigma^*$



$\Sigma^* ab \Sigma^*$ $\quad\cup\quad$ $\Sigma^* bb \Sigma^*$





a^* b^*

a^* $\cup$ b^*

Union

Consider the two automata as a **single automaton**

Determinization

Subset construction

Product construction

Intersection of languages

Emptiness

Algorithm for emptiness

Complementation

Union

Unit-4: Regular properties

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

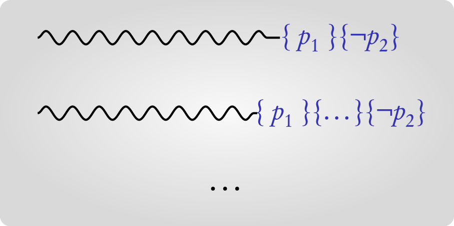
July - November 2015

Module 4:

Safety properties described by automata

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

P: a property over AP



Bad-Prefixes

P is a safety property if there **exists** a set Bad-Prefixes such that

P is the set of **all words** that **do not start** with a Bad-Prefix

Property 1: if p_1 is true, then p_2 should be true in the next step

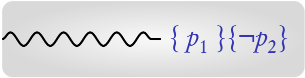
$\sim\sim\sim\sim\sim\ \{p_1\}\{\neg p_2\}$ BadPrefixes

Property 1: if p_1 is true, then p_2 should be true in the next step

$\rightsquigarrow \{ p_1 \}\{\neg p_2\}$ BadPrefixes

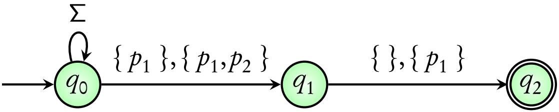
$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$

Property 1: if p_1 is true, then p_2 should be true in the next step



BadPrefixes

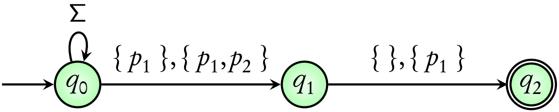
$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$



Property 1: if p_1 is true, then p_2 should be true in the next step

$\sim\sim\sim\sim\sim \{ p_1 \}\{\neg p_2\}$ BadPrefixes

$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$



This **BadPrefixes** set is a **regular language**

Property 2: if p_1 is true, then p_2 should be true in the next to next step

$$\leadsto \{ p_1 \}\{\ldots\}\{\neg p_2\}$$

"BadPrefixes"

Property 2: if p_1 is true, then p_2 should be true in the next to next step

$$\rightsquigarrow \{\, p_1 \,\}\{\ldots\}\{\neg p_2\}$$

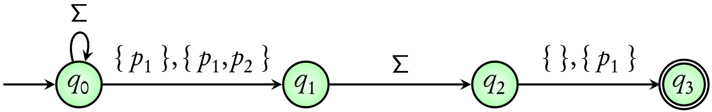
"BadPrefixes"

$$\Sigma \;=\; \{\, \{\,\},\; \{\, p_1 \,\},\; \{\, p_2 \,\},\; \{\, p_1, p_2 \,\} \,\}$$

Property 2: if p_1 is true, then p_2 should be true in the next to next step

$\rightsquigarrow \{ p_1 \}\{ \dots \}\{ \neg p_2 \}$ “BadPrefixes”

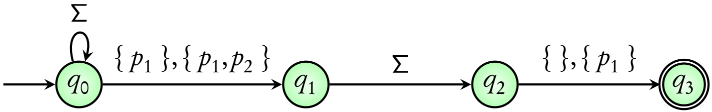
$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$



Property 2: if p_1 is true, then p_2 should be true in the next to next step

$\sim\sim\sim\{p_1\}\{\ldots\}\{\neg p_2\}$ “BadPrefixes”

$$\Sigma = \{\,\{\,\},\ \{p_1\},\ \{p_2\},\ \{p_1,p_2\}\,\}$$



This **BadPrefixes** set is a **regular language**

Property 3: at any point, the number of times p_1 has occured should be less than the number of times p_2 has occured

Property 3: at any point, the number of times p_1 has occured should be less than the number of times p_2 has occured

$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$

Property 3: at any point, the number of times p_1 has occured should be less than the number of times p_2 has occured

$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$

BadPrefixes = words where number of times p_1 occurs is more than that of p_2

Property 3: at any point, the number of times p_1 has occured should be less than the number of times p_2 has occured

$$\Sigma = \{ \{ \}, \{ p_1 \}, \{ p_2 \}, \{ p_1, p_2 \} \}$$

BadPrefixes = words where number of times p_1 occurs is more than that of p_2

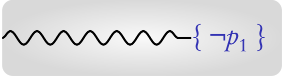
This **BadPrefixes** set is **not a regular language**

Regular safety properties

A safety property is **regular** if the associated **BadPrefixes** set is a **regular language**

Invariants are **regular safety properties**

Property: Always p_1 is true



"Bad-Prefixes"

$\Sigma^* \{\neg p_1\}$

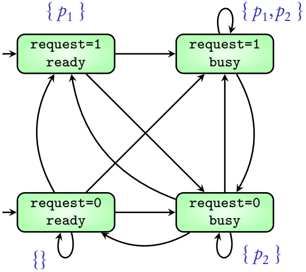
BadPrefixes set for invariant properties is a **regular language**

Coming next: An algorithm to model-check safety properties

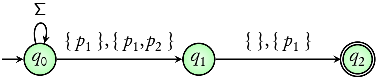
Model

Atomic propositions $\mathrm{AP} = \{\, p_1, p_2 \,\}$

p_1: `request=1` p_2: `status=busy`



Safety property

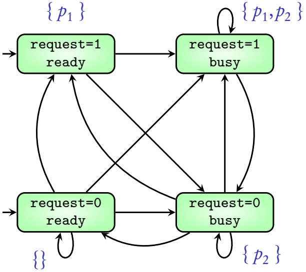


BadPrefixes

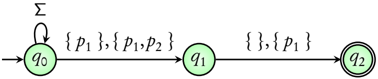
Model

Atomic propositions AP = $\{ p_1, p_2 \}$

p_1: `request=1` p_2: `status=busy`



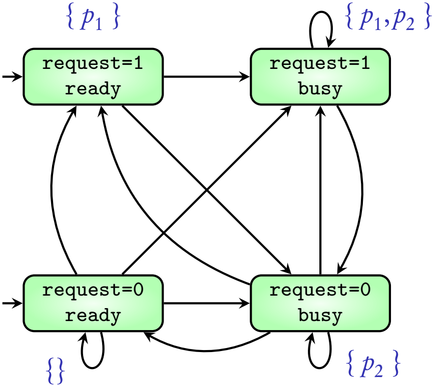
Safety property

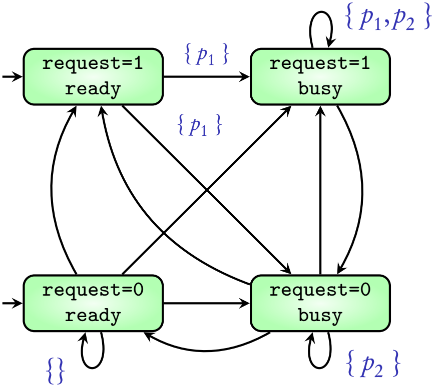


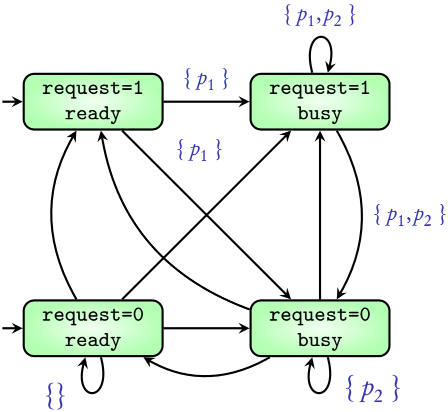
BadPrefixes

Does the model satisfy the safety property?

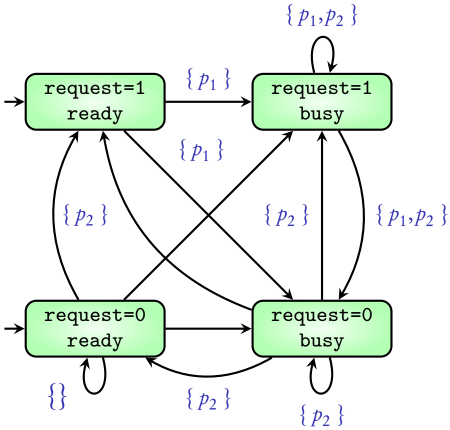
Step 1: Transition system ⟶ automaton

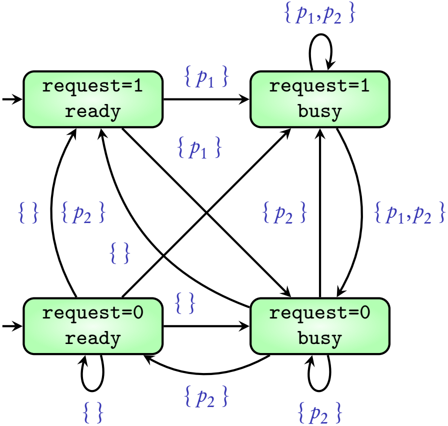

{ p1 }
{ p1,p2 }
request=1
ready
request=1
busy
request=0
ready
request=0
busy
{}
{ p2 }


request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p_1 }
{ p_1 }
{ p_1,p_2 }
{ p_2 }
{}

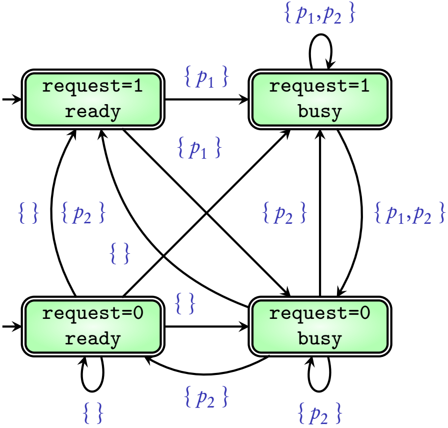


{ p_1,p_2 }
request=1
ready
{ p_1 }
request=1
busy
{ p_1 }
{ p_1,p_2 }
request=0
ready
request=0
busy
{}
{ p_2 }

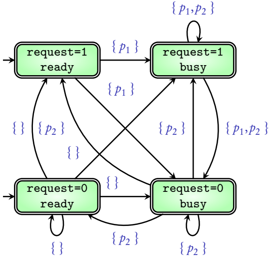

{ p_1,p_2 }
request=1
ready
{ p_1 }
request=1
busy
{ p_1 }
{ p_2 }
{ p_2 }
{ p_1,p_2 }
request=0
ready
request=0
busy
{}
{ p_2 }
{ p_2 }



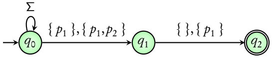
{ p1,p2 }
{ p1 }
request=1
ready
request=1
busy
{ p1 }
{ } { p2 }
{ p2 }
{ p1,p2 }
{ }
{ }
request=0
ready
request=0
busy
{ p2 }
{ }
{ p2 }


{ p1,p2 }
{ p1 }
request=1
ready
request=1
busy
{ p1 }
{ } { p2 }
{ p2 }
{ p1,p2 }
{ }
{ }
request=0
ready
request=0
busy
{ p2 }
{ }
{ p2 }

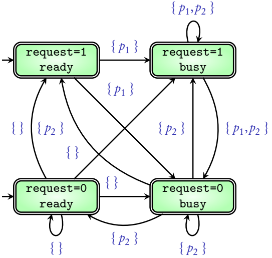
Step 2: Take a synchronous product with property automaton

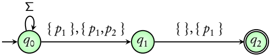


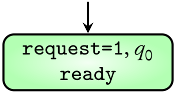
request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p1 }
{ p1,p2 }
{ p1 }
{ } { p2 }
{ }
{ p2 }
{ p1,p2 }
{ }
{ }
{ p2 }
{ p2 }

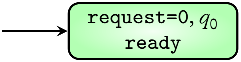


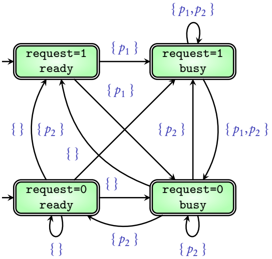
Σ
q0
{ p1 },{ p1,p2 }
q1
{ },{ p1 }
q2

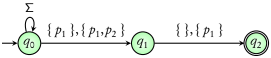

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p1 }
{ p1,p2 }
{ p1 }
{ } { p2 }
{ }
{ p2 }
{ p1,p2 }
{ }
{ }
{ p2 }
{ p2 }

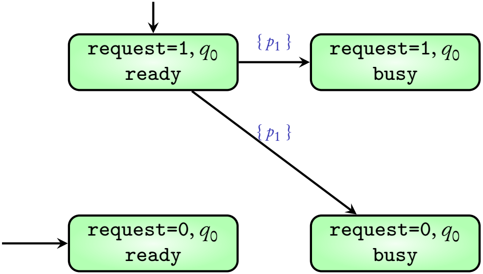

Σ
q0
{ p1 },{ p1,p2 }
q1
{ },{ p1 }
q2

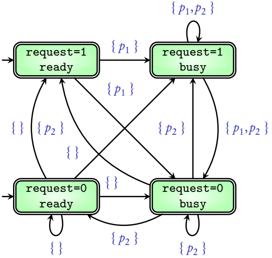

request=1, q0
ready

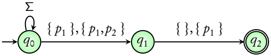

request=0, q0
ready

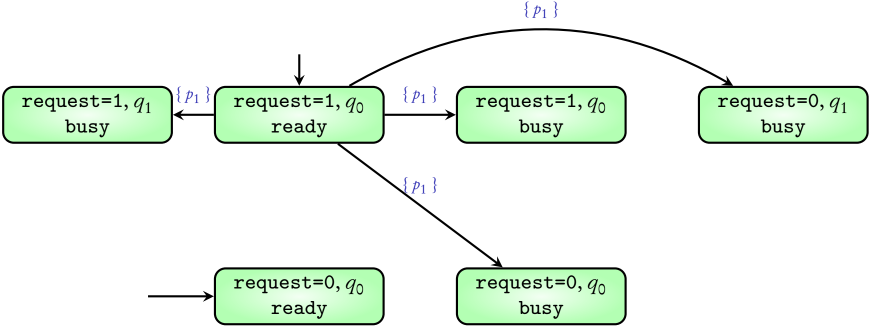

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p_1 }
{ p_1,p_2 }
{ p_1 }
{ } { p_2 }
{ p_2 }
{ p_1,p_2 }
{ }
{ }
{ p_2 }
{ }
{ p_2 }

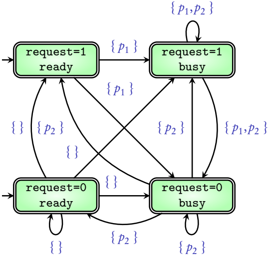

Σ
q_0
{ p_1 },{ p_1,p_2 }
q_1
{ },{ p_1 }
q_2

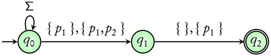

request=1, q_0
ready
{ p_1 }
request=1, q_0
busy
{ p_1 }
request=0, q_0
ready
request=0, q_0
busy

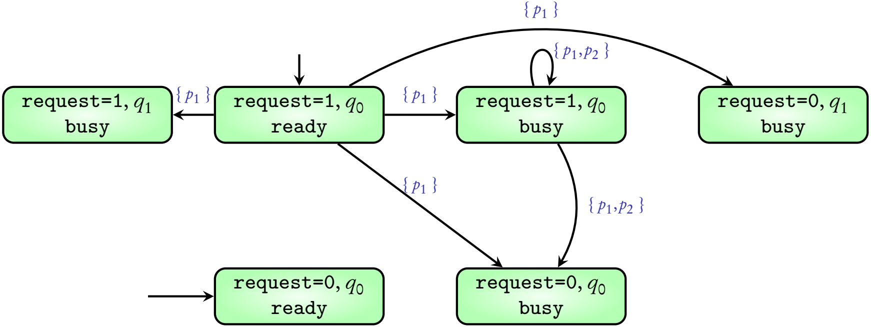

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p1 }
{ p1,p2 }
{ p1 }
{ }
{ p2 }
{ p2 }
{ p1,p2 }
{ }
{ }
{ }
{ p2 }
{ p2 }

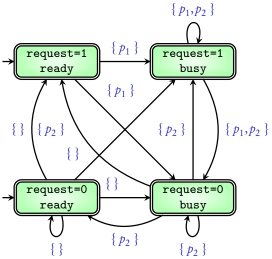

Σ
{ p1 },{ p1,p2 }
{ },{ p1 }
q0
q1
q2

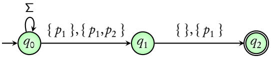

{ p1 }
request=1, q1
busy
{ p1 }
request=1, q0
ready
{ p1 }
request=1, q0
busy
request=0, q1
busy
{ p1 }
request=0, q0
ready
request=0, q0
busy

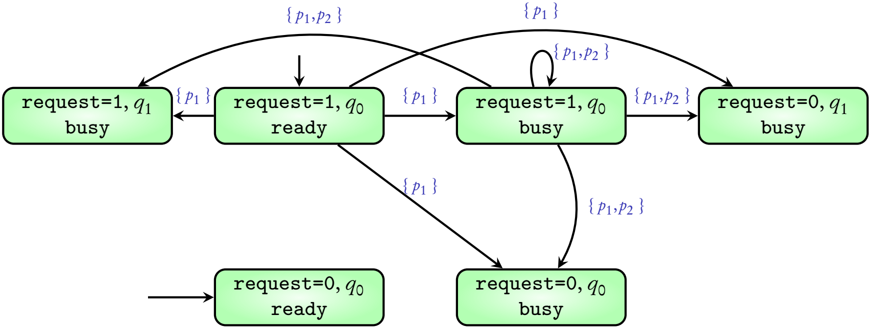

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p_1 }
{ p_1,p_2 }
{ } { p_2 }
{ p_2 }
{ }
{ }
{ p_2 }

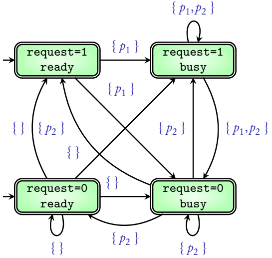

Σ
q_0
{ p_1 },{ p_1,p_2 }
q_1
{ },{ p_1 }
q_2

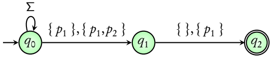

request=1, q_1
busy
request=1, q_0
ready
request=1, q_0
busy
request=0, q_1
busy
request=0, q_0
ready
request=0, q_0
busy
{ p_1 }
{ p_1,p_2 }

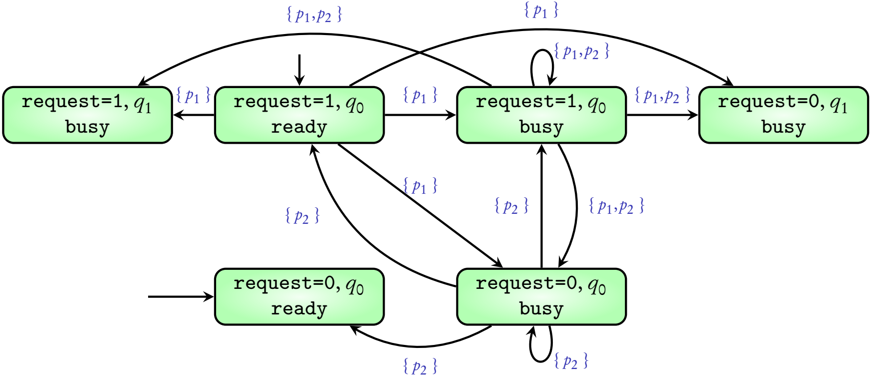

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p_1 }
{ p_1,p_2 }
{ p_1 }
{ } { p_2 }
{ }
{ p_2 }
{ p_1,p_2 }
{ }
{ p_2 }
{ }
{ p_2 }

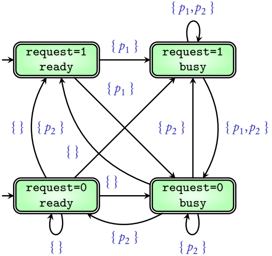

Σ
q_0
{ p_1 },{ p_1,p_2 }
q_1
{ },{ p_1 }
q_2

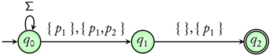

{ p_1,p_2 }
{ p_1 }
{ p_1,p_2 }
request=1, q_1
busy
{ p_1 }
request=1, q_0
ready
{ p_1 }
request=1, q_0
busy
{ p_1,p_2 }
request=0, q_1
busy
{ p_1 }
{ p_1,p_2 }
request=0, q_0
ready
request=0, q_0
busy

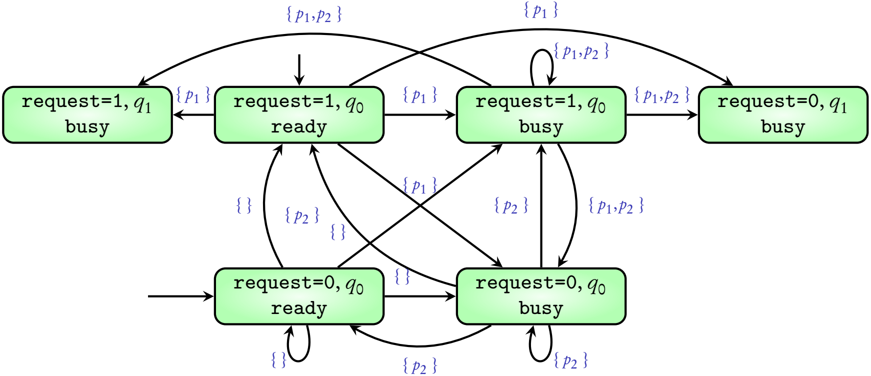

request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p1 }
{ p1,p2 }
{ p2 }
{ }


Σ
q0
q1
q2
{ p1 },{ p1,p2 }
{ },{ p1 }


request=1, q1
busy
request=1, q0
ready
request=1, q0
busy
request=0, q1
busy
request=0, q0
ready
request=0, q0
busy
{ p1,p2 }
{ p1 }
{ p2 }


request=1
ready
request=1
busy
request=0
ready
request=0
busy
{ p_1 }
{ p_1,p_2 }
{ p_1 }
{ }
{ p_2 }
{ }
{ p_2 }
{ p_1,p_2 }
{ }
{ }
{ p_2 }
{ p_2 }


Σ
q_0
{ p_1 },{ p_1,p_2 }
q_1
{ },{ p_1 }
q_2


request=1, q_1
busy
request=1, q_0
ready
request=1, q_0
busy
request=0, q_1
busy
request=0, q_0
ready
request=0, q_0
busy
{ p_1,p_2 }
{ p_1 }
{ p_1,p_2 }
{ p_1 }
{ p_1 }
{ p_1,p_2 }
{ p_1 }
{ }
{ p_2 }
{ }
{ p_2 }
{ p_1,p_2 }
{ }
{ }
{ p_2 }
{ p_2 }

Step 3: Check if the language of the product automaton is empty

Step 3: Check if the language of the product automaton is empty

If language is empty, there are **no bad prefixes**

Step 3: Check if the language of the product automaton is empty

If language is empty, there are **no bad prefixes**

- Language empty → model satisfies safety property
- Language non-empty → model does not satisfy safety property

- Step 1: Convert model to automaton
- Step 2: Take synchronous product with **BadPrefixes** automaton
- Step 3: Check if language of product is empty

- Language empty ⟶ model satisfies safety property
- Language non-empty ⟶ model does not satisfy safety property

Regular safety properties

BadPrefixes is regular

Algorithm

Unit-4: Regular properties

B. Srivathsan

Chennai Mathematical Institute

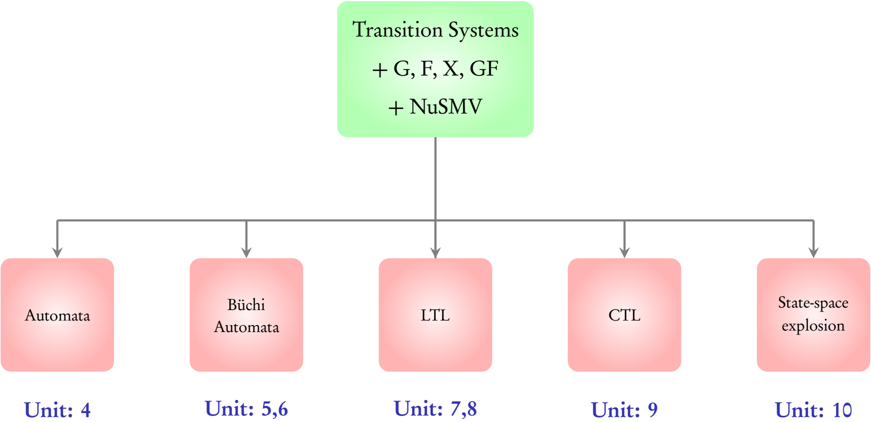
NPTEL-course

July - November 2015

Summary

- Introduction to automata
- Simple properties of automata
- Regular safety properties
- Algorithm for regular safety properties

Important concepts: NFA, DFA, subset construction, synchronous product, complementation


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