#### Automata: Short Course SOFT ENG 3BB4

Ryszard Janicki

Department of Computing and Software McMaster University Hamilton, Ontario, Canada janicki@mcmaster.ca

크

Alphabet: an *arbitrary* (usually finite) set of elements, often denoted by the symbol  $\Sigma$ .

#### Sequence:

• an element  $x = (a_1, a_2, \dots, a_k) \in \Sigma^k$ , where  $\Sigma^k$  is a Cartesian product of  $\Sigma$ 's.

For convenience we write  $x = a_1 a_2 \dots a_k$ .

• a function 
$$\phi: \{1, \ldots, k\} \to \Sigma$$
, such that  $\phi(1) = a_1, \ldots, \phi(k) = a_k$ .

- The two above definitions are in a sense identical since:  $\sum_{n} \sum_{k} \sum_{n} \sum_{k} \sum_{k} \sum_{k} \{f \mid f : \{1, \dots, k\} \rightarrow \Sigma\}.$
- Frequently a *sequence* is considered as a primitive undefined concept that is understood and does not need any explanation.

・ 回 ト ・ ヨ ト ・ ヨ ト

#### Sequences and strings

- If the elements of Σ are symbols, then a finite sequence of symbols is often called a string or a word.
- In concurrency theory sequences are often called traces (for example in the textbook for this course).
  - The *length* of a sequence x, denoted |x|, is the number of elements composing the sequence. For example |aba| = 3, |aabbc| = 5.
  - The *empty sequence*, ε, is the sequence consisting of zero symbols, i.e. |ε| = 0.
  - A prefix of a sequence is any number of leading symbols of that sequence, and a suffix is any number of trailing symbols (any number means 'zero included'). For example a sequence (word, trace) abca has the following prefixes: ε, a, ab, abc, abca, and the following suffixes: abca, bca, ca, a, ε.

・ロト ・回ト ・ヨト ・ヨト

#### Concatenation

• Concatenation (operation) Let  $x = a_1 \dots a_k$ ,  $y = b_1 \dots b_l$ . Then

$$x \circ y = a_1 \dots a_k b_1 \dots b_l$$

#### We usually write xy instead of $x \circ y$ .

• Properties of concatenation:

*Fact.* A triple  $(\Sigma, \circ, \varepsilon)$  is a *monoid* (recall 2LC3).

• Power operator: 
$$x^0 = \varepsilon$$
,  $x^1 = x$  and  $x^k = \underbrace{x \dots x}_k$ .

• Recursive definition of power:

$$x^0 = \varepsilon$$
$$x^{k+1} = x^k x$$

# $\Sigma^*$ and Formal Language

• Let  $\Sigma$  be a finite alphabet. Then we define  $\Sigma^*$  as:

$$\Sigma^* = \{a_1 \dots a_k \mid a_i \in \Sigma \land k \ge 0\},$$

i.e. the set of all sequences, including  $\varepsilon,$  built from the elements of  $\Sigma.$ 

For example

 $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aba, aab, ...\}$ , all sequences built from a and b.

- $\emptyset^* = \{\varepsilon\}$
- If  $\Sigma \neq \emptyset$  then  $|\Sigma^*| = \infty$ .
- A (formal) language over Σ is any subset of Σ\*, including the empty set Ø and Σ\*.
- For example {ab, cba, ba, bbbb} ⊆ {a, b, c}\* is a finite language, while {abc, ba, ab, abb, abbb, ..., ab<sup>k</sup>, ...} ⊆ {a, b}\* in an infinite language.

#### Automata or State Machines

- There is a set of states Q.
   Q may be finite, then we have finite state machines.
- There is a set of **actions/operations** that allow to move from one state to another state.
- There is a **transition function/relation** that allow movement from one state to another state using actions/operations.
- There is an **initial state**.
- There *might be* final states.
- The concept of a current state may easily be introduced.
- The set of actions/operations is finite.
- There is a concept of **nondeterministic choice**.

## (Finite) Automata or (Finite) State Machines: an example



크

## Automata: Non-determinism



æ

<ロ> <同> <同> < 同> < 同>

#### Definition

A **deterministic (finite) automaton (state machine)** is a 5-tuple:

$$M = (\Sigma, Q, \delta, s_0, F),$$

where:  $\Sigma$  is the alphabet (finite) (input alphabet), Q is the set of states (finite),  $\delta: Q \times \Sigma \rightarrow Q$  is the transition function,  $s_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states.

#### Definition ( $\hat{\delta}$ function, also often denoted as $\delta^*$ )

We extend the function  $\delta$  to  $\hat{\delta}: Q imes \Sigma^* o Q$  as follows:

• 
$$\forall q \in Q$$
.  $\hat{\delta}(q, \varepsilon) = q$ 

•  $\forall q \in Q. \forall x \in \Sigma^*. \forall a \in \Sigma. \ \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a).$ 

# $\hat{\delta}$ function, also often denoted as $\delta^*$

#### Definition

We extend the function  $\delta$  to  $\hat{\delta}: \mathcal{Q} imes \Sigma^* o \mathcal{Q}$  as follows:

• 
$$\forall q \in Q$$
.  $\hat{\delta}(q, \varepsilon) = q$ 

• 
$$\forall q \in Q. \forall x \in \Sigma^*. \forall a \in \Sigma. \ \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a).$$

- The above definition of  $\hat{\delta}$  is recursive and the recursion is on the length of x.
- Intuitively δ(q, a) is the state that can be reached from q in one step, while δ(q, x) is the state that can be reached from q in |x| steps by using δ in each step.
- For example  $\hat{\delta}(q, abcd) = \delta(\delta(\delta(\delta(q, a), b), c), d)$ .
- We usually write  $\delta$  instead of  $\hat{\delta}$  when it does not lead to any misunderstanding.
- For example  $\delta(q, abcd) = \delta(\delta(\delta(d(q, a), b), c), d)$ .

#### Definition (Language)

For every automaton M, the set

$$L(M) = \{x \mid \hat{\delta}(s_0, x) \in F\}$$

is called the language accepted/generated by M.

- The language is just a set of all sequences (words, traces) that can be derived by starting from the initial state travelling trough automaton (using  $\delta$  as *next state* function) and ending in some *final state*.
- It is possible to leave a final state!
- In concurrency we often do not have final states! In such a case we assume that each state is a final state, i.e. F = Q!

## Deterministic Automaton: an Example

• Consider the following deterministic automaton M



• We have:  $\Sigma = \{a, b\}$ ,  $Q = \{s_0, s_1, s_2, s_3\}$ ,  $F = \{s_2, s_3\}$  and the below table shows the transition function  $\delta$ .

$\delta$	а	b
<i>s</i> <sub>0</sub>	<i>s</i> <sub>1</sub>	<b>s</b> 2
<i>s</i> <sub>1</sub>	<i>S</i> 3	$s_1$
<i>s</i> <sub>2</sub>	<i>s</i> <sub>1</sub>	<b>S</b> 3
<i>S</i> 3	<b>s</b> <sub>2</sub>	$s_1$

• For example ab,  $bbaa \notin L(M)$ , while aaa,  $abbbaab \in L(M)$ .

#### Problems!

#### (1) We **cannot** specify:

The meaning is pretty simple: "first execute a and next execute any number of b, including none." However, for the first definition we have  $\delta(s_0, a) = s_1$  and  $\delta(s_1, b) = s_1$ , but what about  $\delta(s_0, b) =$ ? and  $\delta(s_1, a) =$ ?. For the second definition we have  $a(s_0) = s_1$  and  $b(s_1) = s_1$  but still  $b(s_0) =$ ? and  $a(s_1) =$ ?.

a

UGLY SOLUTION





- This solution is good for illustration, problematics for real systems as we have to introduce entities that may not exist in the real system!
- The standard solution involves the concept of **non-determinism**.

- Notation for 'power set':  $2^Q = \mathcal{P}(Q) = \{X \mid X \subseteq Q\}$ , and clearly  $\emptyset \in 2^Q$ .
- Problem #1: How to model the below situation?



• The standard solution  $\delta(s, a) = \{s_1, \dots, s_k\}$ , which implies  $\delta : Q \times \Sigma \to 2^Q$ ,

• Consider the following automaton:



• Which is true?  $ab \in L(M)$  or  $ab \notin L(M)$ ?

• Consider the following automaton:



Which is true? ab ∈ L(M) or ab ∉ L(M)?
 Usually it is assumed that ab ∈ L(M). It is called angelic semantics.

16 / 45

• Consider the following automaton:



Which is true? ab ∈ L(M) or ab ∉ L(M)?
 Usually it is assumed that ab ∈ L(M). It is called angelic semantics.

16 / 45

- **Angelic**: At each state an *angel* will tell you where to go, so if there is a good choice you will make it. The only bad case is when all choices are bad.
- **Demonic**: At each state a *demon* will tell you where to go, so if there is a bad choice you will make it. The only good case is when all choices are good.
- Demonic semantics is much less popular. It is relatively new and was motivated by *fault tolerant systems*. In this class we will use only *angelic* semantics. I have mentioned demonic, to show that non-determinism is more complex than the one presented in most textbooks.

## Angelic vs Demonic Semantics - An Example

• Consider the three automata below:



• Let  $L_A(M_1)$ , i = 1, 2, 3 denote a language defined by  $M_i$ under angelic semantics, and let  $L_D(M_1)$ , i = 1, 2, 3 denote a language defined by  $M_i$  under demonic semantics. Note that  $L_A(M_1) = L_D(M_1) = \emptyset$ ,  $L_A(M_2) = \{ab\}$ ,  $L_D(M_2) = \emptyset$  and  $L_A(M_3) = L_D(M_3) = \{ab\}$ .

#### Definition (Non-deterministic Automaton)

A **non-deterministic (finite) automaton (state machine**) is a 5-tuple:

$$M = (\Sigma, Q, \delta, s_0, F),$$

where:  $\Sigma$  is the **alphabet** (finite) (**input alphabet**), Q is the **set of states** (finite),  $\delta: Q \times \Sigma \rightarrow 2^{Q}$  is the **transition function**,  $s_{0} \in Q$  is the **initial state**,  $F \subseteq Q$  is the set of **final states**.

#### Definition (non-deterministic $\hat{\delta}$ function)

We extend the function  $\delta$  to  $\hat{\delta}: Q \times \Sigma^* \to 2^Q$  as follows:

• 
$$\forall q \in Q$$
.  $\hat{\delta}(q, \varepsilon) = \{q\}$ 

• 
$$\forall q \in Q. \forall x \in \Sigma^*. \forall a \in \Sigma. \ \hat{\delta}(q, xa) = \bigcup_{s \in \hat{\delta}(q, x)} \delta(s, a).$$

Sometimes, by a small abuse of notation, we write  $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a).$ 

## Language Defined by a Non-deterministic Automaton

#### Definition (Angelic semantics)

For every automaton M, the set

$$L(M) = \{x \mid \hat{\delta}(s_0, x) \cap F \neq \emptyset\}$$

is called the language accepted/generated by M.

#### Definition (Demonic semantics)

For every automaton M, the set

$$L(M) = \{x \mid \hat{\delta}(s_0, x) \subseteq F\}$$

is called the language accepted/generated by M.

We will not consider demonic semantics in this course.

## Non-determinism: Example 1

• Consider the following example:



• Standard model: 
$$\Sigma = \{a, b\}, Q = \{s_0, s\}, F = \{s_1\}.$$
  
 $\delta(s_0, a) = \{s_1\}, \delta(s_0, b) = \emptyset,$   
 $\delta(s_1, a) = \emptyset, \delta(s_1, b) = \{s_1\}.$ 

- In this case we do not have 'splits' like the one from page 15, all outcomes of the function  $\delta$  are either singletons or empty set, so intuitively this is rather a deterministic system.
- However formally the automaton is non-deterministic.

## Non-determinism: Example 2

• Consider the following example:



- Classical model:  $\Sigma = \{a, b, c\}, Q = \{q_0, q_1, q_2, q_3\}, F = \{q_3\}.$  $\delta(q_0, a) = \{q_1, q_2\}, \delta(q_0, b) = \delta(q_0, c) = \emptyset, \delta(q_1, a) = \{q_3\}, \delta(q_1, b) = \emptyset, \delta(q_1, c) = \{q_1\}, \delta(q_2, a) = \{q_3\}, \delta(q_2, b) = \{q_2\}, \delta(q_2, c) = \emptyset, \delta(q_3, a) = \delta(q_3, b) = \delta(q_3, c) = \emptyset$
- 'Local' model:  $\Sigma = \{a, b, c\}, Q = \{q_0, q_1, q_2, q_3\}, F = \{q_3\}.$  $a = \{(q_0, q_1), (q_0, q_2), (q_1, q_3), (q_2, q_3)\}, b = \{(q_2, q_2)\}, c = \{(q_1, q_1)\}.$

## Another Approach to Non-determinism

#### Definition

An **automaton** (state machine) is a 5-tuple:

$$M = (\Sigma, Q, \delta, s_0, F),$$

where:  $\Sigma$  is the **alphabet** (finite), Q is the **set of states** (finite),  $\delta: Q \times \Sigma \rightarrow 2^{Q}$  is the **transition function**,  $s_{0} \in Q$  is the **initial state**,  $F \subseteq Q$  is the set of **final states**.

#### Definition

- *M* is deterministic iff  $\forall q \in Q. \forall a \in \Sigma$ .  $|\delta(q, a)| \leq 1$
- *M* is strictly deterministic iff  $\forall q \in Q. \forall a \in \Sigma$ .  $|\delta(q, a)| = 1$

## Labelled Transition Systems

- **Transition** (from Collins Dictionary):
  - "a passing or change from one place, state, condition, etc., to another."
- Consider the case: (5) <sup>a</sup>→(5) <sup>a</sup>→(5)
   Can "a" be called a transition?
- Transitions, state and labels:



• Transitions are **unique**:  $t_1 \iff \mathfrak{S}$ 



• Transitions (labelled) are usually needed and used for modelling concurrency.

## Non-determinism vs Determinism: Example

• Does non-determinism increases the descriptive power?

## Non-determinism vs Determinism: Example

• Does non-determinism increases the descriptive power? NO. See an example below and Theorem on next page.



#### Theorem

Let L be a language accepted by a non-deterministic finite automaton M, i.e. L = L(M). There exists a deterministic finite automaton M' such that L(M') = L.

26 / 45

## Invisible (silent) Actions or $\varepsilon$ -moves



• In many cases we need to model invisible actions!

#### Definition

A non-deterministic automaton with  $\varepsilon$ -moves is:  $M = (Q, \Sigma, \delta, q_0, F),$ where:  $Q, \Sigma, q_0, F$  are as usual, and:

 $\delta: \boldsymbol{Q} \times (\boldsymbol{\Sigma} \cup \{\varepsilon\}) \to 2^{\boldsymbol{Q}}$ 

- **Problem**:  $\varepsilon$  has **two** interpretations:
  - Do nothing,
  - Go to another state by executing invisible (silent) action

#### Theorem

If L is accepted by a nondeterministic automaton with  $\varepsilon$ -moves, then L is also accepted by a nondeterministic automaton without  $\varepsilon$ -moves.

28 / 45

#### Removal of $\varepsilon$ -moves: An example



 We may then transform the non-deterministic automaton from the right hand side into appropriate deterministic one.

# Equivalence of Finite Automata

#### Definition

Two automata  $M_1$  and  $M_2$  are **equivalent** if and only if  $L(M_1) = L(M_2)$ .

#### Conclusion

Non-deterministic automata, nondeterministic automata with  $\varepsilon$ -moves, and deterministic automata are all equivalent.

- *Equivalence* means only the same language, other properties may differ.
- For example the concept of 'demonic semantics' does not make much sense for deterministic automata as we do not have non-deterministic choices.
- For any deterministic automaton M, if all states are finite then then  $L(M) = \Sigma^*$ , so this concept also has very little sense.

#### No Final States

- No final states is equivalent to all states are final, i.e.  $F = \emptyset \equiv F = Q$ .
- But this makes sense only for nondeterministic automata.



# Modelling Dynamic Systems With Automata: Hotel Reservation



• No arrow to initial state and no arrow from final state.

32 / 45

æ

▶ < E ▶ < E ▶</p>



• Neither initial nor final state specified.

E ► < E ►</p>

Image: A matrix

## Simplified Home Security



• Neither initial nor final state specified.

Ryszard Janicki

34 / 45

æ

イロト イヨト イヨト イヨト

#### Modelling Dynamic Systems With Automata

• FOR THIS KIND OF APPLICATIONS AUTOMATA ARE USUALLY NONDETERMINISTIC, or deterministic in the sense of the definition from page 23.

## Regular Expressions: Intuition

$(0\cup 1)0^*$	$\rightarrow$	{0,00,000,,1,10,100,1000,} 'zero or one followed by any number of zeros including none'
ab*	$\rightarrow$	{ <i>a</i> , <i>ab</i> , <i>abb</i> , <i>abbb</i> , }
$(a \cup b)^*$	$\rightarrow$	$\{a, b\}^*$
		'all strings (including $arepsilon$ ) that can be
		built from <i>a</i> and <i>b</i> '
$(a \cup \varepsilon)(b \cup \varepsilon) =$		
$ab \cup \varepsilon b \cup a\varepsilon \cup \varepsilon\varepsilon =$		
$ab \cup b \cup a \cup \varepsilon$	$\rightarrow$	$\{\varepsilon, a, b, ab\}$

< A >

æ

#### Definition (Formal Definition of Regular Expressions)

Let  $\Sigma$  be an alphabet. A string R built from the elements of  $\Sigma \cup \{\varepsilon, \emptyset, (, ), \cup, *\}$  is a **regular expression**, if it is defined by the following rules:

- **(**)  $\emptyset, \varepsilon$  and each  $a \in \Sigma$  are *regular expressions*.
- **2**  $(R_1 \cup R_2)$  is a *regular expression* if  $R_1$  and  $R_2$  are regular expressions.
- (R<sub>1</sub>R<sub>2</sub>) is a regular expression if R<sub>1</sub> and R<sub>2</sub> are regular expressions.
- $(R)^*$  is a regular expression if R is a regular expression.
- There are no other regular expressions.

The set of all regular expressions over the alphabet  $\Sigma$  will be denoted by  $Rex(\Sigma)$ .

- We usually skip some parenthesis.
- Rules: \* first, followed by concatenation, and finally ∪, unless parentheses say differently.

37 / 45

## Languages Defined by Regular Expressions

#### Definition (Interpretation)

Let  $L : \frac{Rex}{\Sigma} \to 2^{\Sigma^*}$  be the following function called interpretation:

• 
$$L(\emptyset) = \emptyset, \ L(\varepsilon) = \varepsilon, \ L(a) = \{a\}$$

$$L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$$

• 
$$L((R)^*) = L(R)^*$$

#### Definition (Language)

For every regular expression R, L(R) is a **language defined by** R.

A class of all languages defined by regular expressions will be denoted by  $\mathcal{L}_{\textit{REX}}.$ 

#### Languages Defined by Regular Expressions: Examples

- $L((0 \cup 1)0^*) = \{0, 00, 000, \dots, 1, 10, 100, 1000, \dots\}$
- *L*(*ab*\*) = {*a*, *ab*, *abb*, *abbb*, *abbbb*, ...}
- $L((a \cup \varepsilon)(b \cup \varepsilon)) = \{\varepsilon, a, b, ab\}$
- We customarily often identify a regular expression *R* with *L*(*R*) but technically *R* is not *L*(*R*).
- **A question:** What is the relationship between  $\mathcal{L}_R$  and  $\mathcal{L}_{REX}$ ?

#### Theorem

$$\mathcal{L}_{R} = \mathcal{L}_{REX}$$

39 / 45

・ 同 ト ・ ヨ ト ・ ヨ ト

## Minimization of Deterministic Finite Automata

• Consider the following two deterministic automata:



- It can be proved that  $L(M_1) = L(M_2)$ , and clearly  $M_2$  has less states.
- It can be proved that  $M_2$  is the minimum state automaton that is equivalent to  $M_1$ , i.e.  $L(M_1) = L(M_2)$ .

## Intuitions for Minimization

Consider the following two automata, both generating the language  $ab^*$ :



The states  $q_1$  and  $q_2$  of the left automaton and *equivalent*, so they, and appropriate arrows, can be glued together.

#### Minimum State Deterministic Finite Automata

#### Theorem

For every deterministic automaton  $M_1$  there is the minimum state deterministic automaton  $M_2$  such that  $L(M_1) = L(M_2)$ . The automaton  $M_2$  is unique up to the names of states.



# Non-deterministic Automata and Minimization Problem

- The word 'deterministic' is important and cannot be omitted!
- Consider the following two non-deterministic automata, both generating the language

$$L = \{ab, ac, bc, ba, ca, cb\}$$





• Both automata above are minimum state and there is no way to make them identical!

43 / 45

# Non-deterministic Automata and Minimization Problem

 When talking about Minimum State Non-deterministic automaton (usually when we discuss some application), we usually mean the case as below (no non-deterministic splits):



- If we forgot about red part, both automata are non-deterministic and the black automaton on the right can be interpreted as the minimum state automaton equivalent to the blue automaton on the left.
- However, to derive formally the back automaton on the right from the blue on left we need to add the red parts.

## Non-deterministic Automata and Minimization Problem



 Labelled Transition Systems are almost always non-deterministic and the statements 'minimal', 'minimization', etc., in the textbook, refer to the meaning from the previous slide.