Languages and Machines:
- sets of sequences/languages (finite/infinite sequences, closure properties, Nerode equivalence, regular languages, automata, transition systems),
- defining sets by expressions and equations (regular expressions, fix-point theory),
- machines, computational models and structures (Turing machines, RAM, PRAM, non-standard automata, etc.).

Rewriting Systems:
- Grammars,
- Post systems,
- Chomsky hierarchy,
- Grammars vs automata
Computability:
- Functions and sets,
- Primitive recursive functions and sets, recursive functions and sets, recursively enumerable sets,
- what does computable and decidable mean (Halting problem),
- limitation of formal systems (Gödel’s theorem),
- Church thesis.

Complexity:
- definition, independence from a computing platform,
- time/space hierarchy,
- \( P \) vs \( NP \), \( NP \)-completeness,
- reduction problem.
Alphabet: an arbitrary (usually finite) set of elements, often denoted by the symbol $\Sigma$.

Sequence:

- an element $x = (a_1, a_2, \ldots, a_k) \in \Sigma^k$, where $\Sigma^k$ is a Cartesian product of $\Sigma$’s.
  For convenience we write $x = a_1 a_2 \ldots a_k$.

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

The two above definitions are in a sense identical since:

$$\underbrace{\Sigma \times \ldots \times \Sigma}_n \equiv \{ f \mid f : \{1, \ldots, k\} \rightarrow \Sigma \}.$$ 

- Frequently a sequence is considered as a primitive undefined concept that is understood and does not need any explanation.
If the elements of $\Sigma$ are symbols, then a finite sequence of symbols is often called a string or a word.

The length of a sequence $x$, denoted $|x|$, is the number of elements composing the sequence.

The empty sequence, $\varepsilon$, is the sequence consisting of zero symbols, i.e. $|\varepsilon| = 0$.

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’).
**Concatenation**

- **Concatenation** (operation)
  Let $x = a_1 \ldots a_k$, $y = b_1 \ldots b_l$. Then

  $$x \circ y = a_1 \ldots a_k b_1 \ldots b_l.$$  

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

- **Properties of concatenation**:
  1. $x(yz) = (xy)z$
  2. $\varepsilon x = x \varepsilon = x$

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

- **Power operator**: $x^0 = \varepsilon$, $x^1 = x$ and $x^k = \underbrace{x \ldots x}_k$.

- **Recursive definition of power**:
  
  - $x^0 = \varepsilon$
  - $x^{k+1} = x^k x$. 

  **Ryszard Janicki**  
  Computability and Complexity  6 / 33
Let $\Sigma$ be a finite alphabet. Then we define $\Sigma^*$ as:

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

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

A (formal) language over $\Sigma$ is any subset of $\Sigma^*$, including the empty set $\emptyset$ and $\Sigma^*$. 
Two Important Models of Sequence Monoid

1. **Algebra of Strings**
   - elements of $\Sigma$ are **symbols**, 
   - elements of $\Sigma^*$ are **words** or **strings**, 
   - concatenation “$\circ$” is just a **juxtaposition**.

2. **Algebra of Relations**
Let $X$ be a set and $\Sigma = \{R_1, \ldots, R_k\}$, where each $R_i \subseteq X \times X$ is a relation.

Now concatenation $\circ$ corresponds to composition of relations, i.e.

$$x(R \circ S)y \iff \exists t \in \Sigma. \; xRt \land tSy,$$

and $\varepsilon$ is the identity relation, i.e. $\varepsilon = \{(x, x) \mid x \in X\}$.

Let $A = \{r_1, \ldots, r_n\}$ be the set of names of relations, i.e. $\text{name}(R_i) = r_i$, $\text{rel}(r_i) = R_i$, where $\text{name} : \Sigma \to A$ and $\text{rel} : A \to \Sigma$.

Let extend $\text{rel} : A \to \Sigma$ to $\text{rel} : A^* \to \Sigma^*$, where $\Sigma^*$ is the set of all relations on $X$ that could be composed from the elements of $\Sigma$, plus identity relation, by

$$\text{rel}(r_{i_1} \ldots r_{i_k}) = \text{rel}(r_{i_1}) \circ \ldots \circ \text{rel}(r_{i_k}) = R_{i_1} \circ \ldots \circ R_{i_k}.$$

All properties of the names algebra hold for the relation algebra.

**Algebra of Relations**
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.

The last two statements are equivalent.

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 maybe infinite, but usually is finite.

There is a concept of nondeterministic choice.
Automata or State Machines: an example

- **Initial State**: $S_1$
- **States**: $S_1, S_2, S_3, S_4$
- **Actions**: $a, b, c$
- **Final States**: $S_3, S_4$
Automata: Non-determinism

Diagram:

```
   a
  / \  \\
 a   a \\
   \   \\
    \  \\
    a
```
Definition (Automaton - 1)

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)

We extend the function \( \delta \) to \( \hat{\delta} : Q \times \Sigma^* \rightarrow Q \) as follows:

- \( \forall q \in Q. \ \hat{\delta}(q, \varepsilon) = q \)
- \( \forall q \in Q. \forall x \in \Sigma^*. \forall a \in \Sigma. \ \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) \).
Definition (Language - 1)

For every automaton $M$, the set

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

is called the language accepted/generated by $M$.

We will write $\delta$ instead of $\hat{\delta}$ if it will not lead to any misunderstanding.
A deterministic (finite) automaton (state machine) is a 4-tuple:

\[ M = (\Sigma, Q, s_0, F), \]

where: \( \Sigma \) is the set of actions, i.e. each \( a \in \Sigma \) is a function \( a : Q \to Q, \)

\( Q \) is the set of states (finite),
\( s_0 \in Q \) is the initial state,
\( F \subseteq Q \) is the set of final states.

The relationship between Definition 1 and Definition 2:

\[ \delta(s, a) = q \iff a(s) = q. \]
We define $\Sigma^*$ as the set of all functions that can be constructed from the elements of $\Sigma$ and the function composition. For example the sequence $x = a_1 \ldots a_k \in \Sigma^*$ defines the function $x : Q \rightarrow Q$,

$$x(s) = a_1 \ldots a_k(s) = a_k(\ldots a_2(a_1(s))\ldots).$$

**Definition (Language - 2)**

For every automaton $M$, the set

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

is called the language accepted/generated by $M$.

**RULE:**

$$s \xrightarrow{a} p \iff \delta(s, a) = p \iff a(s) = p.$$
Consider the following deterministic automaton:

In both models: \( \Sigma = \{a, b\} \), \( Q = \{s_0, s_1, s_2, s_3\} \), \( F = \{s_2, s_3\} \) and the below table shows both transition function \( \delta \) (classical model) and actions \( a(\ldots) \), \( b(\ldots) \) (‘local’ model).

<table>
<thead>
<tr>
<th></th>
<th>( a(s) )</th>
<th>( b(s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \delta )</td>
<td>( a )</td>
<td>( b )</td>
</tr>
<tr>
<td>( s_0 )</td>
<td>( s_1 )</td>
<td>( s_2 )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( s_3 )</td>
<td>( s_1 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_1 )</td>
<td>( s_3 )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( s_2 )</td>
<td>( s_1 )</td>
</tr>
</tbody>
</table>
(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 $a(s_1) = ?$.

**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!
A solution with Definition 2: for each $a \in \Sigma$, assume that $a : Q \mapsto Q$ is a partial function.

The solution is elegant only on the surface as dealing with partial functions is tricky and often problematic.

When Definition 1 is used we have to introduce the concept of Non-determinism. This concept can also be used for Definition 2.
Non-determinism: Problem #1

- 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?

\[\begin{array}{c}
S_1 \\
\vdots \\
S_k
\end{array}\]

- Functional solutions:
  1. \(\delta(s, a) = \{s_1, \ldots, s_k\}\), which implies \(\delta: Q \times \Sigma \rightarrow 2^Q\),
  2. \(a(s) = \{s_1, \ldots, s_k\}\), which implies \(\forall a \in \Sigma. \ a: Q \rightarrow 2^Q\).

- Relational solutions:
  3. \((s, a, s_i) \in \delta \text{ for } i = 1, \ldots, k, \implies \delta \subseteq Q \times \Sigma \times Q\),
  4. \((s, s_i) \in a \text{ for } i = 1, \ldots, k, \implies \forall a \in \Sigma. \ a \subseteq Q \times Q\).

- Solutions (1) and (4) are the most popular.
Consider the following automaton:

Which is true? \( ab \in L(M) \) or \( ab \notin L(M) \)?
Non-determinism: Problem #2

Consider the following automaton:

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

Usually it is assumed that \( ab \in L(M) \). It is called angelic semantics.
Consider the following automaton:

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

Usually it is assumed that \( ab \in L(M) \). It is called **angelic semantics**.
**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.

How does this distinction affects computability and complexity? How does it affect theory of Turing Machines? The ‘angelic’ approach is the classical one, the ‘demonic’ is about 25 years old.
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 - 1)

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 \( \hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q \) 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

\[ \hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a). \]
**Definition (Angellic 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 again write $\delta$ instead of $\hat{\delta}$ if it will not lead to any misunderstanding.
A non-deterministic (finite) automaton (state machine) is a 4-tuple:

\[ M = (\Sigma, Q, s_0, F), \]

where: \( \Sigma \) is the set of actions, i.e. each \( a \in \Sigma \) is a relation

\[ a \subseteq Q \times Q, \]

\( Q \) is the set of states (finite),
\( s_0 \in Q \) is the initial state,
\( F \subseteq Q \) is the set of final states.

The relationship is the following:

\[ q \in \delta(s, a) \iff (s, q) \in a. \]
Here $\Sigma^*$ is the set of all relations that can be built form the elements of $\Sigma$, i.e. $a_1 \ldots a_k = a_1 \circ a_2 \circ \ldots \circ a_k$, there “$\circ$” is the composition of relations given on page 9 of this lecture notes.

**Definition (Angelic Semantics)**
For every automaton $M$, the set

$$L(M) = \{x \in \Sigma^* \mid \{s \mid (s_0, s) \in x\} \cap F \neq \emptyset\}$$

is called the language accepted/generated by $M$.

**Definition (Demonic Semantics)**
For every automaton $M$, the set

$$L(M) = \{x \in \Sigma^* \mid \{s \mid (s_0, s) \in x\} \subseteq F\}$$

is called the language accepted/generated by $M$.

**RULE:**

$\delta(s, a) \iff p \in \delta(s, a) \iff (s, p) \in a$. 

Ryszard Janicki
Computability and Complexity
Consider the following example:

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

- ‘Local’ model: $\Sigma = \{a, b\}$, $Q = \{s_0, s\}$, $F = \{s_1\}$.
  $a = \{(s_0, s_1)\}$, $b = \{(s_1, s_1)\}$.
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 \( \forall q \in Q. \forall a \in \Sigma. |\delta(q, a)| = 1 \)
Transition Systems

- **Transition** (from Collins Dictionary): “a passing or change from one place, state, condition, etc., to another.”

- Consider the case: $s_1 \xrightarrow{a} s_2 \xrightarrow{a} s_3$
  Can “a” be called a transition?

- Transitions, state and labels:

  ![Diagram showing transitions between states](image)

- Transitions are **unique**: $t_1 \iff s_1 \xrightarrow{a} s_2$
Transitions are usually needed for modelling concurrency.

Consider the following very simple Petri Net:

There is a structural difference between the label $a$ attached to the transition $t_1$ and the same label $a$ attached to the transition $t_3$. 