MTAT.05.125 Introduction to Theoretical Computer Science

Autumn 2015

Vitaly Skachek
Estonian version by Reimo Palm
English version by Yauhen Yakimenka

Lecture 11. Description of Turing machines. Decidable languages.

Note. Since we use the term "Turing machine" a lot, it will be occasionally shortened as TM.

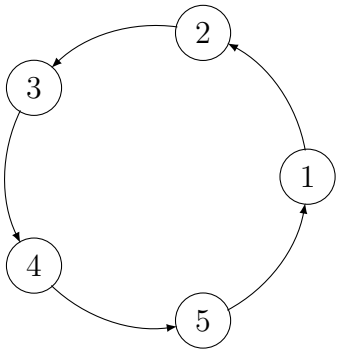
Description of Turing machines

Turing machines do what algorithms do. What is the right level of description of Turing machines?

1. Formal description fully describes all the ingredients.
2. Implementation description is a human language description of what the machine does, how it moves its head, what stores on the tape, etc.
3. High-level description is the algorithm description ignoring the implementation details.

Starting from the next week, the high-level description will be sufficient.

In order to represent an input to TMs, the inputs should be encoded in some agreed format. For example, for the graph G we can first write on the tape number of nodes, then some special separator like #, then number of edges, # again, and finally a list of pairs of connected nodes. For example, the following directed graph



could be encoded as follows:

$$5\#5\#1\#2\#2\#3\#3\#4\#4\#5\#5\#1\text{␣}\ldots$$

Usually we are not interested in the particular way of encoding something. We will use notation $\langle\cdot\rangle$ to stress that we are talking about some encoding[1] in some agreed format. For example, for the graph G, $\langle G\rangle$ is its encoding. For TM M, $\langle M\rangle$ is an encoding of it.

Example 1. Define $L = \{w \mid w \text{ contains equal number of zeros and ones}\}$. Give implementation-level description of TM that decides the language L.

On input w, the machine M:

1. Scans the tape and marks the first 0 that has not been marked. If no unmarked 0s left it goes to 4. Otherwise it moves the head to the beginning of the tape.

2. Scans the tape and marks the first 1 that has not yet been marked. If no unmarked 1 found – rejects.

3. Moves the head to the beginning of the input and goes to 1.

4. Moves the head to the beginning of the input, and scans the input to check if any unmarked 1s remain. If none found – accepts, otherwise – rejects.

We can express different computational problems as languages. For example, testing whether a particular DFA accepts the given string:

$$L_{\text{DFA}} = \{\langle A, w\rangle \mid A \text{ is a DFA,} \\ \text{that accepts the input string } w\}.$$

Here, $\langle A, w\rangle$ represents a pair:

[1] In other words, some string unambiguously describing the object.

- encoding of the DFA A (list of five ingredients: $Q, \Sigma, \delta, q_0, F$);
- input string w.

The task of deciding whether DFA A accepts a string w is equivalent to checking if the pair $\langle A, w \rangle$ is in the language L_{DFA}.

Theorem. *L_{DFA} is a decidable language.*

Proof. We design a TM M that decides the language L_{DFA}.

On the input $\langle A, w \rangle$, the machine M will simulate the automaton A on w, and accept/reject according to the automaton's decision.

First, M scans the input and determines if the input properly represents a DFA (which we denote as A) and a string (which we denote as w). If not, M rejects.

Second, M simulates A. It keeps track of A's current state and A's current position in the input w by writing the information directly on the tape.

In the beginning, the input of M is w, and the head position is the leftmost symbol of w. The states and the positions are updated according to the transition function δ. When M is finishing processing the last symbol of w, it goes to accept/reject state depending on whether A is in the accept/reject state. □

Similarly define

$$L_{\text{NFA}} = \{\langle A, w \rangle \mid A \text{ is an NFA that accepts the input string } w\}.$$

Theorem. *L_{NFA} is a decidable language.*

Proof. We present a TM M' that decides L_{NFA}: on the input $\langle A, w \rangle$, M' does the following:

1. Converts A into equivalent DFA A', by using the procedure that was studied in the course.
2. Run the machine M from the previous theorem on the input $\langle A', w \rangle$.
3. If M accepts – accepts, otherwise – rejects.

□

Practise session

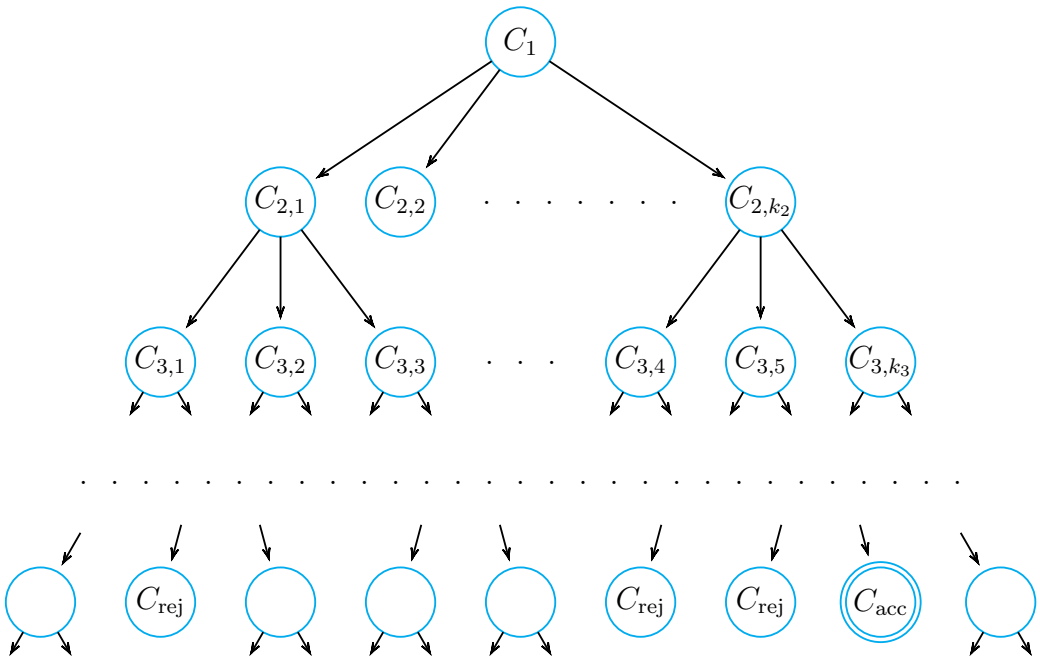
Non-deterministic TM (NTM)

This generalisation allows TM at any point in a computation to proceed to *several* configurations. The transition function:

$$\delta\colon Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\}).$$

Here $\mathcal{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\})$ is a set of all subsets of $Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\}$.

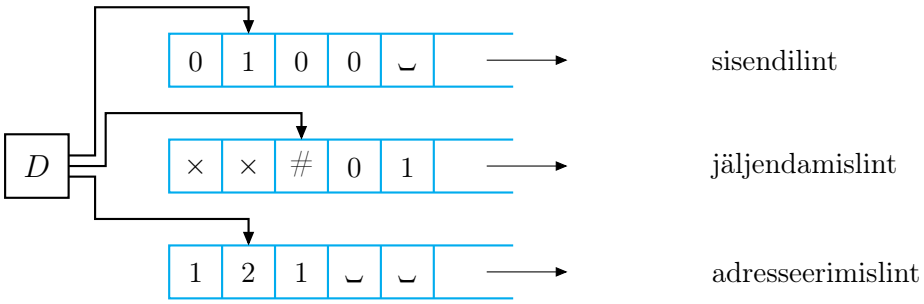
For NTM the computation is a tree, whose branches represent different computation paths for the machine. If some branch of the computation leads to an accept state, the machine accepts.



Theorem. *Every NTM has an equivalent deterministic Turing machine (DTM).*

Proof idea. We simulate NTM N with DTM D, which has 3 tapes. D tries all possible computational paths of N and if some of them leads to accept state, D accepts.

Proof. More formally, the simulating machine D has three tapes:



Every node in the tree can have at most b children, where b is the upper bound on the size of possible choices given by the transition function of N. We assign to every node in the tree a string over $\Gamma_b = \{1, 2, \ldots, b\}$. For example, 315 means "from the root take the third child, then the first child, and then the fifth child". It is possible, that a string does not represent any node in the tree.

Description of D:

1. Tape 1 contains the input, tapes 2 and 3 are empty in the beginning.
2. Copy tape 1 to tape 2.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each choice of N check the next symbol of tape 3 to decide what choice to make.
   - If accepting configuration occurred – accept.
   - If rejecting configuration occurred – goto 4.
   - If no more symbols appear on tape 3 or the configuration is invalid – goto 4.
4. Replace string on tape 3 with the next string in that ordering Goto 2. □

Let

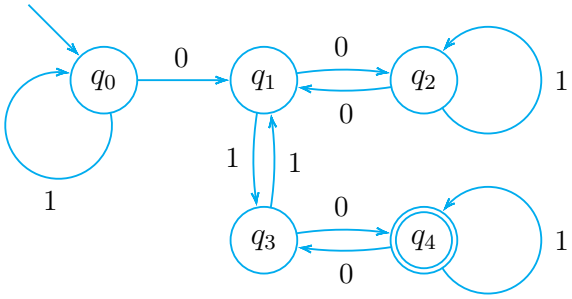
$$L_\varnothing = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \varnothing\}.$$

I.e. all DFAs that do not accept anything.

Theorem. *$L_\varnothing$ is a decidable language.*

Proof. A DFA A accepts some string if and only if reaching one of the accept states by travelling along the arrows of the DFA is possible. Therefore, a TM $\widehat{M}$ will test if there exists such a path.

For example, in the automaton

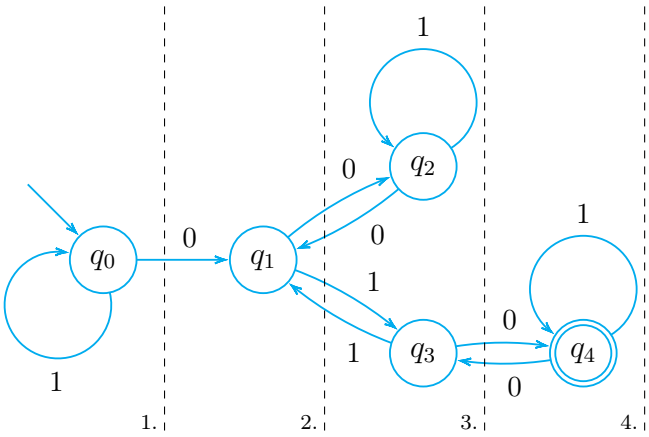


there is a path $q_0 \to q_1 \to q_3 \to q_4$. This correspond to the input 010. Therefore $L(A) \neq \varnothing$ as $010 \in L(A)$.

TM $\widehat{M}$ works as follows.

1. Mark the start state of A.
2. Repeat until no new states are marked:
   - Mark any unmarked state that has an incoming arrow from any state that was marked already.
3. If no accept state is marked – accept, otherwise – reject.

For the example above, $\widehat{M}$ will mark the states in the following order:



q_4 is marked, so $\widehat{M}$ rejects (A accepts at least one string). □

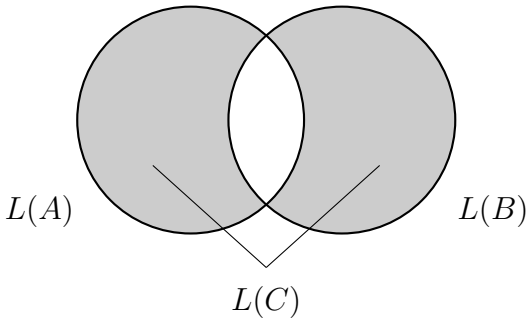
1. Define the language:

$$L_{\text{DFAEQ}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFA and } L(A) = L(B)\}.$$

Prove that L_{DFAEQ} is a decidable language.

Solution. We construct a new DFA C, which accepts strings that are accepted by either A or B, but not by both[2]. Then

$$L(C) = \big(L(A) \cap \overline{L(B)}\,\big) \cup \big(\,\overline{L(A)} \cap L(B)\big).$$



- If $L(A) = L(B)$, then $L(A) \cap \overline{L(B)} = \varnothing$ and $\overline{L(A)} \cap L(B) = \varnothing$, hence $L(C) = \varnothing$.
- If $L(A) \neq L(B)$, then there exists $w \in L(A)$, $w \notin L(B)$ (or vice versa). Then $w \in L(A) \cap \overline{L(B)}$ (or, respectively, $w \in \overline{L(A)} \cap L(B)$) and therefore $w \in L(C)$ and $L(C) \neq \varnothing$.

So $L(A) = L(B)$ if and only if $L(C) = \varnothing$.

We construct a TM M as follows. On the input $\langle A, B\rangle$ it does the following:

1. Constructs C as described.
2. Runs TM that decides the language $L_\varnothing$ on $\langle C\rangle$.
3. If $\langle C\rangle \in L_\varnothing$ – accepts. If $\langle C\rangle \notin L_\varnothing$ – rejects.

2. Define the language

$$L_1 = \{\langle A\rangle \mid A \text{ is a DFA that accepts at least one string of the form } 1^*\}.$$

Prove that L_1 is decidable.

Solution. We construct TM M that decides L_1. On the input $\langle A\rangle$, M does the following:

1. Constructs a DFA B that accepts exactly language described by 1^*.

[2]Such an automaton is easy to build: it runs A and B in parallel and accepts if and only if exactly one of A and B accepts

2. Constructs a DFA C, such that

$$L(C) = L(A) \cap L(B).$$

3. Checks if $\langle C \rangle \in L_{\varnothing}$. If no – accepts, if yes – rejects.

Let us justify the construction.

- If $\langle C \rangle \in L_{\varnothing}$ then $L(C) = \varnothing$ and so $L(A) \cap L(B) = \varnothing$. This means that for each $w \in L(A)$, it holds that $w \notin L(B)$ and therefore w does not have the form 1^*.
- If $\langle C \rangle \notin L_{\varnothing}$, then $L(C) \neq \varnothing$ and $L(A) \cap L(B) \neq \varnothing$. Thus there exists w, such that $w \in L(A)$ and $w \in L(B)$. This means that w has the form 1^* and $w \in L(A)$. Correct.