Variants of Turing machines

What happens if we modify the transition function \( \delta : Q \times \Gamma \to Q \times \Gamma \times \{ \text{L, R} \} \) to be \( \delta : Q \times \Gamma \to Q \times \Gamma \times \{ \text{L, R, S} \} \), where \( \text{S} \) means “stay on the same place”. Does this generalisation allow Turing machines to recognise more languages?

The answer is no. We can convert any Turing machine with “stay on the same place” option into the original model with left and right movements only. We achieve this by replacing “stay on the same place” command by moving to the right (and into a special state), and then moving back to the left.

Turing machines with multiple tapes

It is Turing machine with several read-write tapes (and heads). More formally:

\[
\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{ \text{L, R, S} \}^k,
\]

where \( k \) is the number of tapes. Now \( \delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \text{L, R, \ldots, L}) \) means that if heads 1, 2, \ldots, \( k \) read symbols \( a_1, a_2, \ldots, a_k \) respectively, the the machine moves to the state \( q_j \) and writes \( b_1, b_2, \ldots, b_k \) to the respective tapes.
Theorem. Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Proof. We convert a multi-tape TM $M$ into an equivalent single-tape machine $S$.

If $M$ has $k$ tapes, then $S$ simulates these $k$ tapes by storing their content on a single tape. It uses the new special symbol $\#$ as a delimiter to separate the content of different tapes. Additionally, it should keep track of the positions of the heads. It is achieved by writing special “dotted” symbols, which will be added to the alphabet.

On input $w = w_1\ldots w_n$, $S$ does the following:

1. First, $S$ puts its tape into the format that represents all $k$ tapes on $M$.
   The formatted tape contains\(^1\)
   
   $\# w_1 w_2 \ldots w_n \# \# \# \# \ldots \#$

\(^1\)We assume tapes 2, 3, \ldots, $k$ are empty in the beginning.
2. To simulate a single move, $S$ scans its tape from the first # to the $(k+1)$-st #, in order to determine the symbols pointed by the “virtual heads”. $S$ makes the second pass to update the new tapes according to the way that $M$ would do.

3. If at some point $S$ moves one of the “virtual heads” to the right onto #, this means that $M$ has moved the corresponding head onto the previously unread blank position of the tape. Then, $S$ writes a blank symbol on this tape, and shifts the tape content from this place until the right-most #. Then it continues the simulation as before.

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

![Directed Graph](image-url)
could be encoded as follows:

5#5#1#2#2#3#3#4#4#5#5#1\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 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.

**Practise session**

1. Define $L = \{ 0^{2^n} \mid n \in \mathbb{N} \}$, i.e. collection of all strings of zeros whose length is a power of 2. We want to construct TM $M$ that decides $L$.

We start with implementation-level description. On input $w$, the machine $M$:

1. Moves from left to right by crossing out every second zero.

2. If in step 1 the tape contains a single zero – accept.

3. If the tape contains add number of zeros, which is $> 1$, rejects.

4. Return the head to the beginning of the tape and goto 1.

\[2\text{In other words, some string unambiguously describing the object.}\]
Note that if the number of zeros is \(2^n\), then on each iteration it decreases by a factor of 2, and after \(n\) iterations \(M\) accepts.

If it is not a power of 2, then at some iteration condition of the step 3 holds and we reject.

Now we give a formal description. We build TM \(M = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{acc}}, q_{\text{rej}})\), where

- \(Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{acc}}, q_{\text{rej}}\}\);
- \(\Sigma = \{0\}\);
- \(\Gamma = \{0, \times, \omega\}\);
- start state is \(q_1\);
- accept state is \(q_{\text{acc}}\);
- reject state is \(q_{\text{rej}}\);
- transition function \(\delta\) is as follows:

Here notation \(0 \rightarrow \omega, R\) means that by reading 0 \(M\) writes \(\omega\) and moves the head to the right (in transition from \(q_1\) to \(q_2\)). Hence, \(\delta(q_1, 0) = (q_2, \omega, R)\).

Note. \(M\) writes \(\omega\) over the left-most 0 on the tape to mark the left-hand end of the tape.
Let us see the run of $M$ on input 0000:

\[
\begin{array}{c}
q_1 0 0 0 0 \\
\times q_3 0 0 0 \\
\times 0 q_4 0 \\
\times 0 \times q_3 \\
\times 0 q_5 \times \\
\times q_5 0 \times \\
q_5 \times 0 \times \\
q_5 \times 0 \times \\
q_2 \times 0 \times \\
\times q_2 0 \times \\
\times q_2 \times \times q_{acc}
\end{array}
\]

Non-deterministic TM (NTM)

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

\[
\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).
\]

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

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 non-deterministic 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.

\[ \square \]