Deterministic Turing machines

There are several equivalent definitions for the Turing machine. We are going to introduce some of them.
Our first definition is the deterministic Turing machine, which has the same language definition power as the nondeterministic Turing machine. A Turing machine is called deterministic, if from each configuration it can reach at most one other configuration in one step. The formal definition is the following:

Definition

The Turing machine $TM_d = (Q, T, V, q_0, \#, \delta, F)$ is deterministic, if the transition function $\delta(q, a)$ has at most one element for each pair (q, a), where $q \in Q$ and $a \in V$.

In other words, the Turing machine $TM_d = (Q, T, V, q_0, \#, \delta, F)$ is deterministic, if the form of the transition function δ is $Q \times V \to Q \times V \times \{Left, Right, Stay\}$.

Theorem

For each Turing machine TM there exists a deterministic Turing machine TM_d such that $L(TM) = L(TM_d)$.

Definition

The (nondeterministic) Turing machine (TM) is the following 7-tuple:

$$TM = (Q, T, V, q_0, \#, \delta, F)$$

where

- *Q is the finite nonempty set of the states,*
- *T is the set of the input letters, (finite nonempty alphabet), $T \subseteq V$,*
- *V is the set of the tape symbols, (finite nonempty alphabet),*
- *q_0 is the initial state, $q_0 \in Q$,*
- *$\#$ is the blank symbol, $\# \in V$,*
- *δ is the transition function having a form $Q \times V \to 2^{Q \times V \times \{Left, Right, Stay\}}$, and*
- *F is the set of the final states, $F \subseteq Q$.*

Linear bounded automata

Here we present a special, bounded version of the Turing machines, by which the class of context-sensitive languages can be characterized. This version of the Turing machine can work only on the part of the tape where the input is/was. These automata are called linear bounded automata (LBA).

Definition

Let $LBA = (Q, T, V, q_0, \#, \delta, F)$ *be a Turing machine, where*

$$\delta : Q \times (V \setminus \{\#\}) \to 2^{Q \times V \times \{Left, Right, Stay\}}$$

and

$$\delta : Q \times \{\#\} \to 2^{Q \times \{\#\} \times \{Left, Right, Stay\}}.$$

Then LBA is a (nondeterministic) linear bounded automaton.

One can observe that # signs cannot be rewritten to any other symbol, therefore, the automaton can store some results of its subcomputations only in the space provided by the input, i.e., in fact, the length of the input can be used during the computation, only.

To establish a connection between the classes of context-sensitive languages and linear bounded automata we present the following theorem.

Theorem

The class of languages accepted by linear bounded automata and the class of context-sensitive languages coincide.

Unsolvable problems

There is a well known example: **the halting problem**. There is a computer program given with an input, decide whether the program stops running after a while, or goes into an infinite

loop. The same problem with Turing machine appears to be the following: given a description of a Turing machine and the input word, decide whether the Turing machine stops running or keeps on running forever. Let us denote the description of a Turing machine with d_{TM} and the input word with w. The problem is to create a Turing machine which decides about each input word d_{TM}#w whether or not the Turing machine TM goes into an infinite loop with the input word w. It has been shown that this problem cannot be decided by a Turing machine, so there is no universal algorithm to decide if a given computer program goes into an infinite loop or not. The equivalent problem is to create a Turing machine which accepts an input word d_{TM}#w, if the Turing machine TM stops with the input word w. As one can see, computing/calculating a function or accepting a language is not so distant from each other. We also have to point out that the halting problem cannot be solved, because we suppose that the Turing machine has an infinite tape. The problem can be solved in a computer with a finite memory, however, the algorithm is not efficient in practice.

https://en.wikipedia.org/wiki/List_of_undecidable_problems#Problems_in_logic

1. The halting problem (determining whether a Turing machine halts on a given input) and the mortality problem (determining whether it halts for every starting configuration).
2. Determining whether a Turing machine is a busy beaver champion (i.e., is the longest-running among halting Turing machines with the same number of states and symbols).
3. Rice's theorem states that for all nontrivial properties of partial functions, it is undecidable whether a given machine computes a partial function with that property.
4. The halting problem for a Minsky machine: a finite-state automaton with no inputs and two counters that can be incremented, decremented, and tested for zero.
5. Universality of a Nondeterministic Pushdown automaton: determining whether all words are accepted.
6. The problem whether a tag system halts.

Time and space complexity

We have already shown that there are functions which cannot be algorithmically computed, consequently there are problems which cannot be solved. However, there are many problems, which can be solved, and we would like to know the complexity of the algorithms computing the solutions. For this reason, scientists have introduced the time complexity of the algorithms. The time complexity of an algorithm is not a constant number. For a longer input the Turing machine needs longer time to compute, so the time complexity of an algorithm is a function for each algorithm, and the parameter of the function is the length of the input word w. This length is commonly denoted by n, so $n = |w|$, and the time complexity of the Turing machine TM is denoted by T(n). We can investigate the space complexity of an algorithm as well. Let us denote by S(n) the function which shows how many cells we use on the tape of the Turing machine TM for an input word of length n. Of course, the time complexity and the space complexity are not independent from each other. If the time is limited, we have a limitation on the number of steps, so we can go to a limited distance from the initial position on the tape, which means that the space is also limited. The most important time complexity classes are the followings:

1. constant time, when the calculating time is fixed, does not depend on the length of the input, denoted by O(1),
2. logarithmic time, when the calculating time is not more than a logarithmic function of the length of the input word, denoted by O(log n),
3. linear time, when the calculating time is not more than a linear function of the length of the input word, denoted by O(n),
4. polynomial time, when the calculating time is not more than a polynomial function of the length of the input word, denoted by $O(n^k)$,
5. exponential time, when the calculating time is not more than an exponential function of the length of the input word, denoted by $O(2^{n^k})$.

Nondeterministic Turing machines

Definition

The (nondeterministic) Turing machine (TM) is the following 7-tuple:

$$TM = (Q, T, V, q_0, \#, \delta, F)$$

where

- Q *is the finite nonempty set of the states,*
- T *is the set of the input letters, (finite nonempty alphabet),* $T \subseteq V$,
- V *is the set of the tape symbols, (finite nonempty alphabet),*
- q_0 *is the initial state,* $q_0 \in Q$,
- $\#$ *is the blank symbol,* $\# \in V$,
- δ *is the transition function having a form* $Q \times V \to 2^{Q \times V \times \{Left, Right, Stay\}}$, *and*
- F *is the set of the final states,* $F \subseteq Q$.

5 Debrecen, 2019

Famous languages classes: P and NP.

P is the set of decision problems solvable in polynomial time by a deterministic Turing machine.
NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.