Introduction to Computer Science

Géza Horváth

Ninth Lecture

Turing machine – introduction

Turing machines play a fundamental role in the algorithms and computational theory. The concept of Turing machine was invented by Alan Turing in 1936-1937. This simple hypothetical device is able to compute all the functions which are algorithmically computable.

Before we deal with the Turing machine as a universal tool for describing algorithms, we introduce the Turing machine as a universal language definition device.

Turing machine – introduction

The basic concept is that the Turing machine manipulates a string on a two-way infinite tape according to transition rules.The tape contains an infinite number of cells, and each cell contains one letter. At the beginning, the tape contains the input string, and the rest of the cells contain a special tape symbol called a blank symbol. There is a head, which can read and write the content of the current cell of the tape, and can move both to the left and to the right. At the beginning, the head is over the first letter of the input string. The Turing machine also has its own inner state, which can be changed in each step. At the beginning, the inner state of the Turing machine is the initial state. The transition rules are the "program" of the Turing machine.

Turing machine – introduction

In each step the machine reads the letter contained by the current cell of the tape, and also reads its own inner state, then writes a letter into the current cell, changes its inner state and moves the head to the left or to the right, or stays in the same position. Sometimes, it does not change its inner state, and sometimes it does not change the content of the current cell. The operations of the Turing machine are based on the transition rules.

Turing machine – definition

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,* $F \subseteq Q$*.*

Turing machine – accepting languages

In each step, the Turing machine has its configuration (u, q, av), where $q \in Q$ is the current state, $a \in V$ is the letter contained by the current cell, and $u, v \in V^*$ are the words before and after the current letter, respectively. The first letter of the word u and the last letter of the word v cannot be the blank symbol, and the word uav is the "important" part of the tape, the rest of the tape contains only blank symbols. At the beginning, the Turing machine has its initial configuration: (λ, q_0, av), where av is the whole input word. In each step, the Turing machine changes its configuration according to the transition function. There are three possibilities, depending on the movement part of the applied rule.

Turing machine – accepting languages

1. The simplest case is when the applied transition rule has a form $(q_2, a_2, Stay) \in \delta(q_1, a_1)$. In this case, we just change the state and the symbol contained by the current cell of the tape according to the current rule. Formally, we say the TM can change its configuration from $(u, q_1, a_1 v)$ to $(u, q_2, a_2 v)$ in one step, if it has a transition rule $(q_2, a_2, Stay) \in \delta(q_1, a_1)$, where $q_1, q_2 \in Q$, $a_1, a_2 \in V$, and $u, v \in V^*$.

   It is denoted by $(u, q_1, a_1 v) \vdash_{TM} (u, q_2, a_2 v)$.

2. The next possibility is when the applied transition rule has a form $(q_2, a_2, Right) \in \delta(q_1, a_1)$. In this case, we change the state and the symbol contained by the current cell of the tape according to the current rule, and move the head to the right. Formally, we say the TM can change its configuration from $(u, q_1, a_1 v)$ to $(u a_2, q_2, v)$ in one step, if it has a transition rule $(q_2, a_2, Right) \in \delta(q_1, a_1)$, where $q_1, q_2 \in Q$, $a_1, a_2 \in V$, and $u, v \in V^*$.

   It is denoted by $(u, q_1, a_1 v) \vdash_{TM} (u a_2, q_2, v)$.

   Here, we have a special case, namely, if $a_2 = \#$ and $u = \lambda$, then $(u, q_1, a_1 v) \vdash_{TM} (\lambda, q_2, v)$.

Turing machine – accepting languages

3. The last possibility is when the applied transition rule has a form $(q_2, a_2, Left) \in \delta(q_1, a_1)$. In this case, we change the state and the symbol contained by the current cell of the tape according to the current rule, and move the head to the left. To formalize this case, we have to write the word u in a form $u = wb$, where b is the last letter of the word u. We say that the TM can change its configuration from (wb, q_1, a_1v) to (w, q_2, ba_2v) in one step, if it has a transition rule $(q_2, a_2, Left) \in \delta(q_1, a_1)$, where $q_1, q_2 \in Q$, $a_1, a_2, b \in V$, and $w, v \in V^*$.
   It is denoted by $(wb, q_1, a_1v) \vdash_{TM} (w, q_2, ba_2v)$.
   Here, we also have a special case, namely, if $a_2 = \#$ and $v = \lambda$, then $(wb, q_1, a_1v) \vdash_{TM} (w, q_2, b)$.

Now, let X and Y be configurations of the same Turing machine. Then, we say that the Turing machine can change its configuration from X to Y in finite number of steps, if $X = Y$, or there are configurations $C_0, C_1, ..., C_n$ such that $C_0 = X$, $C_n = Y$, and $C_i \vdash_{TM} C_{i+1}$ holds for each integer $0 \leq i < n$.
It is denoted by $X \vdash^*_{TM} Y$.

Language accepted by Turing machine

A configuration is called a final configuration, if the Turing machine is in a final state. Now, we can define the language accepted by the Turing machine. The input word is accepted, if the Turing machine can change its configuration from the initial configuration to a final configuration in finite number of steps.

Definition

$L(TM) = \{p \mid p \in T^*, (\lambda, q_0, p) \vdash^*_{TM} (u, q_f, av), q_f \in F, a \in V, u, v \in V^*\}$.

Turing machine example – accepting palindromes over $\{a, b\}$

Example

$TM = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}, \{a, b\}, \{a, b, \#\}, q_0, \#, \delta, \{q_f\})$
$\delta(q_0, \#) = \{(q_f, \#, Stay)\},$
$\delta(q_0, a) = \{(q_1, \#, Right)\},$
$\delta(q_0, b) = \{(q_2, \#, Right)\},$
$\delta(q_1, a) = \{(q_1, a, Right)\},$
$\delta(q_1, b) = \{(q_1, b, Right)\},$
$\delta(q_1, \#) = \{(q_3, \#, Left)\},$
$\delta(q_3, a) = \{(q_5, \#, Left)\},$
$\delta(q_3, \#) = \{(q_f, \#, Stay)\},$
$\delta(q_2, a) = \{(q_2, a, Right)\},$
$\delta(q_2, b) = \{(q_2, b, Right)\},$
$\delta(q_2, \#) = \{(q_4, \#, Left)\},$
$\delta(q_4, \#) = \{(q_f, \#, Stay)\},$
$\delta(q_4, b) = \{(q_5, \#, Left)\},$
$\delta(q_5, a) = \{(q_5, a, Left)\},$
$\delta(q_5, b) = \{(q_5, b, Left)\},$
$\delta(q_5, \#) = \{(q_0, \#, Right)\}.$

Turing machine – computing functions

The Turing machine is not just a language definition tool, it is also a computing device.

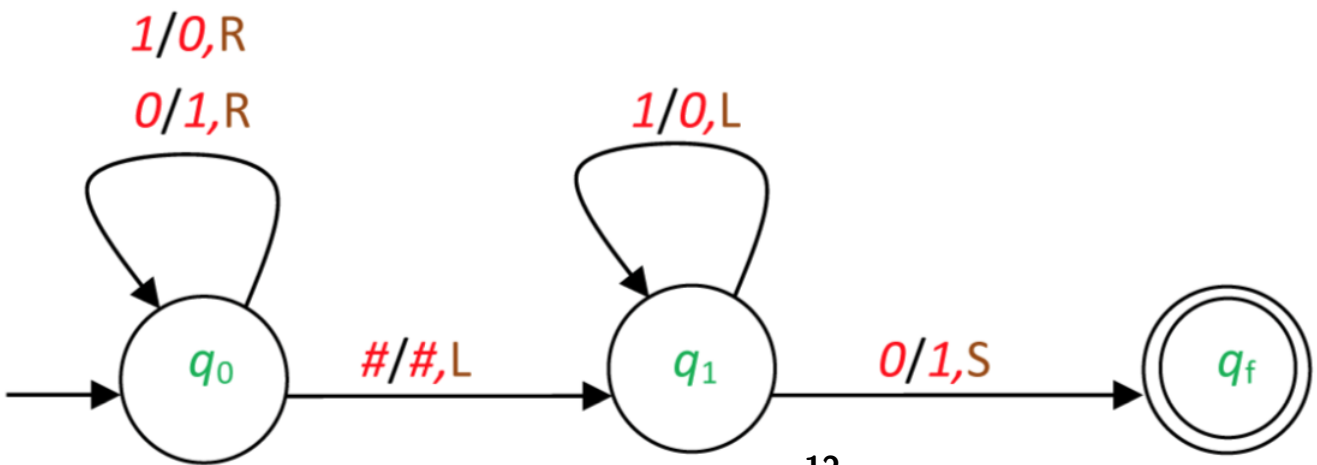
When we use the Turing machine to compute/calculate a function, we use it the same way as before. At the beginning, the input word is on the tape, and when the Turing machine reaches a final configuration (u, q_f, av), the result of the computation/calculation is the word uav, which is the significant part of the tape.

Turing machine – computing functions

In this example we show a Turing machine, which computes the two's complement of the input binary number.

Example

$TM = (\{q_0, q_1, q_f\}, \{0, 1\}, \{0, 1, \#\}, q_0, \#, \delta, \{q_f\})$
$\delta(q_0, 1) = \{(q_0, 0, Right)\},$
$\delta(q_0, 0) = \{(q_0, 1, Right)\},$
$\delta(q_0, \#) = \{(q_1, \#, Left)\},$
$\delta(q_1, 1) = \{(q_1, 0, Left)\},$
$\delta(q_1, 0) = \{(q_f, 1, Stay)\}.$



Deterministic Turing machine

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* $L(TM) = L(TM_d)$*.*

Multitape Turing machine

Now, we are going to introduce the multitape Turing machine, which looks like a more powerful tool compared to the original Turing machine, but in reality it has the same language definition power.

In this case, we have more than one tape, and we work on each tape in each step. At the beginning, the input word is written on the first tape, and the other tapes are empty. (Contains blank symbols in each position.) The multitape Turing machine is in initial state, and the head is over the first letter of the first tape. In each step the multitape Turing machine reads its own state and the symbols from the cells of each tape, then changes its state; it writes symbols into the current cell of each tape and moves the head to the left or to the right, or stays in a same position over each tape separately.

Multitape Turing machine

Observing the formal definition, its only difference compared to the deterministic Turing machine is that it has more tapes, and as a result, it has a different transition function.

Definition

The multitape Turing machine is the following octuple $TM_m = (k, Q, T, V, q_0, \#, \delta, F)$*, where* $Q, T, V, q_0, \#$ *and* F *are the same as before, the integer* $k \geq 0$ *is the number of the tapes, and the form of the transition function* δ *is* $Q \times V^k \to Q \times (V \times \{Left, Right, Stay\})^k$.

As we have noted before, multitape Turing machines accept recursively enumerable languages as well.

Theorem

For each multitape Turing machine TM_m *there exists (a one-tape) Turing machine* TM *such that* $L(TM) = L(TM_m)$.

Turing machine – equivalent definitions

The reason for using the multitape Turing machine or the deterministic Turing machine instead of the original Turing machine is very simple. Sometimes it is more comfortable to use these alternative Turing machines for calculating or proving theorems.

For the same purpose, sometimes we use a Turing machine which has one tape, and this tape is infinite in one direction, and the other direction is "blocked". This version has a special tape symbol in the first cell, and when the Turing machine reads this symbol, the head moves to the right and does not change this special symbol. There is yet another possibility, when the Turing machine must not stay in the same position, in each step the head must move to the right or to the left. Sometimes we use only one final state, and sometimes we use a rejecting state as well, but all of these versions are equivalent to each other.

Theorem

Each of the Turing machine described above accepts recursively enumerable languages, and each recursively enumerable language can be accepted by each type of the above mention Turing machines.

The Church-Turing thesis

The original reason for introducing the Turing machine was to simulate mathematical algorithms which can calculate complicated functions. Later, it has been recognized that all algorithmically computable functions can be calculated by the Turing machine, as well.

Thesis

The statement which claims that **a function is algorithmically computable if and only if it can be computed by the Turing machine** *is called* **Church-Turing thesis** *or simply* **Church's thesis**.

Church's thesis is not a theorem, it cannot be proven, because "algorithmically computable" is not a well defined expression in the thesis. In spite of the fact that Church's thesis is not a proven theorem, the thesis is accepted among scientists.

The Church-Turing thesis

The most important consequence of the thesis is that even the latest, and the strongest computer with a highly improved computer program can only compute the same things as a very simple Turing machine. Therefore we can use the Turing machine to show if something can be computed or not; and this is why the Turing machine plays a fundamental role in the algorithms and computational theory.

Alternative definition of complexity classes P and NP

Definition

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

Definition

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

References

This work was supported by the construction EFOP-3.4.3-16-2016-00021. The project was supported by the European Union, co-financed by the European Social Fund.