Turing Machines

Marcin Sydow

Turing Machines - the Role

Basic tool for complexity theory. An abstract model of computation machine:

- conceptually simple
- can execute any computation possible on “known” computers
- resources consumption models well “real” computers

Properties of TM

First property is very important, since it makes it possible to formulate and prove theorems concerning machines with the use of only the simplest mathematical concepts.

Second property is connected with famous **Church-Turing thesis**: *each function that can be effectively computed can be computed on a Turing Machine* (i.e. TM is a sufficiently powerful computational model)

Third property is particularly important for complexity theory: time and space used by TM do not differ *significantly* (in asymptotic sense) from those used by "real" computers.

Computation Model

Description, time and space consumption of computation is dependent on the particular model.

Real computers are quite complex, thus there is a need to define something simpler, yet actually not very different.

There exist many models of computation, including:

- Turing Machine
- RAM machine
- logical circuits

Turing Machine

Alan Turing (1912-1954)

Turing machine was proposed as the model of "any possible computation" in 1936, mainly as a notation for expressing algorithms and for logicians to determine which problems are possible to be algorithmically computed.

Simple computation model:

- unlimited tape divided into cells for storing symbols
- head that can read/write symbols from/to the tape
- control module that can be in one of (finitely) many states and, based on the current state and current symbol, decides on the next steps:
  - overwrite current symbol with some other symbol
  - move the head to left or right
  - change current state

Formal Definition of Turing Machine

$M = (Q, \Sigma, \Gamma, \#, \delta, q_0, F)$, where:

- Q is a finite set of states
- Σ is a finite set of input symbols that represent the input (written on the tape)
- Γ is a finite set of possible symbols to be written on the tape $\Sigma \subseteq \Gamma$
- empty symbol # (not an input symbol) that fills almost all cells of the tape
- δ transition function. It is a partial function $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$.

  By $\delta(A, x) = (B, y, K)$ we denote that TM in state A and reading symbol x gets to state B, writes symbol x and moves the head to left or right: $(K = \leftarrow)$, $(K = \rightarrow)$.
- q_0 initial state
- F is a set of accepting states (subset of Q). TM terminates when any of accepting states is reached

How does TM work?

We make the following assumptions about the initialisation of TM:

- the tape is filled with infinitely many empty symblos except some finite contiguous region filled with input symbols (input word).
- head is over the leftmost input symbol
- state is set to q_0

Then, the machine works accordingly to the δ function. It will change the state, move the head, read and write symbols. until any of the conditions happen:

- TM reaches one of the accepting states (we say "TM accepts the input" or simply "accepts")
- TM in state A reads x and $\delta(A, x)$ is not defined (we say "TM halts with an error")

It may also happen, that TM never reaches any of the conditions - in such case TM *never halts*

Representation

How to represent a particular TM?

- a tabular representation of the δ function. Each row contains sequence: A,x, B,y,K, that represents $\delta(A, x) = (B, y, K)$. Anything else, except the initial state and accepting states can be derived from the table.
- graph (state diagram): vertices are states, direct edges from A to B represents transition with label of the form x,y,K (meaning $\delta(A, x) = (B, y, K)$).

Exercises

Define a TM that solves the following task:

- recognises palindromes $ww^{\leftarrow} : w \in \{0,1\}^*$
- multiplies 2 given numbers (unary encoded) separated by 0
- resets the head to the leftmost symbol
- terminates with clear tape (erases the input word)

TM and Languages. Partially Decidable Languages

Definition

The language of M denoted as $L(M) \subseteq \Sigma^*$, is the set of those input words for which M accepts. We also say that L(M) recognises (or accepts) the language L(M)

Definition

A language $L \subseteq \Sigma^*$ is called recursively enumerable (or partially decidable) iff there exists machine M such that L = L(M).

Comment: why "partially?". Since for $w \in L$, the machine halts (accepts). However for $w \notin L$, the machine can loop (work endlessly).

Decidable Languages

Definition

Machine M has stop property if for any word $w \in \Sigma^*$ M always halts. Language $L \subseteq \Sigma^*$ is *recursive* (or decidable) iff there exists TM M with stop property so that L = L(M)

Decidable Languages

Definition

Machine M has stop property if for any word $w \in \Sigma^*$ M always halts. Language $L \subseteq \Sigma^*$ is *recursive* (or decidable) iff there exists TM M with stop property so that L = L(M)

Remark: There exists language that is not decidable.

Decidable Languages

Definition

Machine M has stop property if for any word $w \in \Sigma^*$ M always halts. Language $L \subseteq \Sigma^*$ is *recursive* (or decidable) iff there exists TM M with stop property so that L = L(M)

Remark: There exists language that is not decidable.

Exercise: Show that if $L_1 \in \Sigma^*$ and $L_2 \in \Sigma^*$ are decidable then $L_1 \cap L_2$ and $L_1 \cup L_2$ are decidable.

Encoding of Turing Machines

Importantly, any Turing Machine can be *encoded* as a sequence of symbols so that it can be written as input to some other Turing Machine.

One of such encodings:

- symbols and states can be represented by consecutive natural numbers
- q $= |Q|$, $Q = \{0, 1, \ldots, q-1\}$
- s $= |\Sigma|$
- g $= |\Gamma|$. It is convenient to assume that numbers in $\Sigma \subseteq \Gamma$ are compatible with those in Γ.
- the number representing the empty symbol
- d - the number of defined (argument,value) pairs of δ
- next, the sequence of d 5-tuples of natural numbers A $<$ q, X $<$ g, B $<$ q, Y $<$ g, $K \in \{0, 1\}$ with the following meaning:
  - $\delta(A, X) = (B, Y, \leftarrow)$, for K=0.
  - $\delta(A, X) = (B, Y, \rightarrow)$, for K=1.

Turing machines as input words

We can write an encoding of a Turing machine (a sequence of natural numbers defined on the previous slide) with unary encoding, separated by symbols of 0 and treat such a TM encoding sequence as an input word (for some other machine!) over $\{0, 1\}$ alphabet.

It is very important that it is a *finite* word.

Similarly, we can always encode the *current configuration* of any TM in similar way.

Time complexity of Turing Machine

Time complexity of Turing Machine M on an input word w, denoted as T(M,w) is the number of steps done by the machine before it halts. If it does not, we set $T(M,w) = \infty$

We can also define time complexity of a machine itself as follows:

Function $f : \mathbb{N} \rightarrow \mathbb{N}$ is *time complexity of M* iff
$\forall_{n \in \mathbb{N}} : f(n) = max\{T(M,w) : w \in \Sigma^n\}$ (assuming it halts)

Space complexity of Turing Machine

Space used by TM M on input word w, denoted as S(M,w) is defined as the number of tape cells that were visited by the head before M halted. If M does not halt, it is not defined.

We say that f : $\mathbb{N} \rightarrow \mathbb{N}$ is space complexity of M (with stop property) iff $\forall_{n \in \mathbb{N}} : f(n) = max\{S(M, w) : w \in \Sigma^n\}$

Multi-tape Turing Machine

The basic model of Turing Machine has many variants.

(Example of extension of the basic TM model)
k-tape Turing Machine has k tapes, k independent heads (one for each tape), and the control function depends on all read symbols (and controls all heads independently). We allow that head can stay at the same position in this variant.

The input word is written on the first tape, other are empty, acceptation is defined as previously.

Off-line Turing Machine

(Example of restriction of the k-tape Turing Machine)

The restriction is that it is not possible to write on the first tape (read-only input tape). Usually, in addition, the head on the last (output) tape can move only to the left (write-only output tape). Other tapes are regarded as "working" space. In this case we do not count the space used on input and output tapes. This allows to analyse interesting machines that have sub-linear space complexity.

Some definitions of the basic Turing Machine assume that the tape is bounded on one (left) end. This model does not restrict the power significantly (e.g. odd and even cells can represent the "left" and "right" part, etc.)

Simulation of a k-tape TM on a basic TM

Allowing for multiple tapes does not make the computation model much more powerful.

Lemma

For any language recognised by a k-tape TM $M_{(k)}$ in time complexity $\mathcal{O}(f(n))$ it is possible to construct a basic TM that simulates it in time complexity $\mathcal{O}(f(n)^2)$.

Comment: k-tape TM can be also simulated on 2-tape off-line TM

Non-deterministic Turing Machine

A very important extension of the basic model of TM that introduces non-determinism into the transition function of TM.

The idea is that, given any particular configuration of non-deterministic TM, it can "choose" one of many steps to follows.

More precisely, the transition function of non-deterministic TM is defined as a partial, multi-valued function as follows:
$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{\leftarrow, \rightarrow\})$

Thus, a non-deterministic TM could be viewed as "random", but this there is a better interpretation especially in the context of the definition of acceptation of non-deterministic TM.

Definition of acceptation of a non-deterministic TM

Definition

Non-deterministic TM accepts a word m if there exists at least one sequence of transitions that reaches the accepting state.

Thus, instead of viewing non-deterministic TM as a "random" device it is more precise to view it as a machine that checks *all possibilities* during the computation. This view better demonstrates the power of non-determinism.

The question of "how big power does non-determinism give to TM" is represented by the famous problem P vs NP.

Complexity of non-deterministic TM

We say that non-deterministic TM halts on word w iff no computation scenario loops endlessly.

Time and space used by non-deterministic TM can be defined as:

- computation time NT(M,w) as the number of steps of the longest possible computation
- computation space NS(M,w) as the maximum number of cells needed in a possible computation scenario

Comment: time is represented by the height of the tree of possible computations

The definitions of time and space complexity are analogous to those of the deterministic one.

Simulation of non-deterministic TM on a deterministic one

Lemma

If language L is recognised by a non-deterministic TM in time $\mathcal{O}(f(n))$, then there exists a deterministic TM recognising L in time $\mathcal{O}(c^{f(n)})$ for some constant c.

Lemma

Any language recognised by a non-deterministic k-tape TM in time $\mathcal{O}(f(n))$ is also recognised by some 2-tape non-deterministic TM in time $\mathcal{O}(f(n))$.

Literature

- Complexity Theory:
  - Papadimitriou "Computational Complexity", Addison Wesley Longman, 1994
  - Garey, Johnson "Computers and Intractability" (issued in 1979, not easy to get nowadays)
- Introductory Textbooks:
  - Cormen et al. "Introduction to Algorithms" 3rd edition, MIT Press
    chapters 34,35
  - Kleinberg, Tardos "Algorithm Design", Addison Wesley, 2006
    chapters 8,10,11

Thank you for attention