Freie Universität Berlin

Tutorials on Quantum Information Theory

Winter term 2021/22

Problem Sheet 8
Turing Machines and Complexity (Best Christmas Present Ever!!)

J. Eisert, A. Nietner, F. Arzani, C. Bertoni, R. Suzuki

1. **Turing machines and computability.** (7 points: 3+4)

   Recall from the lecture the definition of a *Turing machine*: A Turing machine comprises a tape, a state register, and a read-write head which moves along the tape. The machine is characterised by a finite set of states S with distinguished start $\perp$ and end states $\top$, a finite set of symbols Σ and a table of instructions (the program).

   At any point in time, the Turing machine with internal state $s \in S$ reads in the symbol $\sigma \in \Sigma$ from the tape at the current position of the read-write head. Given the internal state and the symbol the machine transitions to a new internal state s' and performs an action. An action is given by either a move, one step to the left on the tape L or one to the right R, or by replacing the current symbol σ by a new symbol $\sigma' \in \Sigma$. In the latter case the machine then is in the new internal state s' and sees the current symbol σ'.

   A program is thus specified by a set of 4-tuples (s, σ, s', a), where $s \in S$ is the current internal state of the machine, $\sigma \in \Sigma$ is the current symbol on the tape, $s' \in S$ is the new internal state the machine transists to when reading σ in the state s and $a \in \Sigma \cup \{L, R\}$ is the action of the machine to be performed upon reading σ in state s.

   The machine is initialised in state $\perp$ on the very left of tape. Upon reaching the final state $\top$ the program terminates (we also say the machine accepts or the machine halts).

   We now want to write a few little programs on a Turing machine.

   Assume we are given representations of natural numbers $k \in \mathbb{N}$ represented in unary notation on $k+1$ bits so that 0 is represented by $\overline{0} = 1$, and k by $\overline{k} = \underbrace{11\cdots1}_{k+1}$.

   a) Write a Turing program (i.e. specify a set of tuples (s, σ, s', a) that define the program) that determines the parity of a number k given a tape of the form $0\overline{k}00\cdots$ and writes it on the tape after the input.

      Hint: You may freely choose the set of symbols and internal states.

   b) Now write a program that adds $k, l \in \mathbb{N}$ given a tape of the form $(0\overline{k}0\overline{l}0\cdots0)$ and outputting a tape of the form $(0\overline{(k+l)}00\cdots$.

   Unary encodings are not very efficient as opposed to binary encodings for which only $\log_2 k$ many bits are required.

   Further reading: Turing (1937), a more enjoyable and clearer read than the publication year might suggest!

2. **Complexity classes and complete problems.** (13 points: 3+3+2+3+2)

   In the lecture, you got to know the classical complexity classes P, NP, #P as well as the quantum class BQP. While P, NP and BQP are called 'decision classes', #P is called a 'counting class'. Let us first introduce decision, search and counting problems:

   Decision problems are given by a family of binary functions $f_n : \{0,1\}^n \to \{0,1\}$ and on *input* $x \in \{0,1\}^n$ one is to return $f_n(x)$ (i.e. the corresponding boolean value of

f_n on input x). Often this is phrased in terms of a *language* rather than families of functions: one says $x \in \{0,1\}^*$, where

$$\{0,1\}^* = \bigcup_{i \in \mathbb{N}} \{0,1\}^i \tag{1}$$

is the set of all finite bit strings, is in the language L if $f(x)$ and $x \notin \mathrm{L}$ if $\neg f(x)$. Here the notion of a language is as a set L that contains the *words* of the language, which are the *strings* of arbitrary finite lengths over some *alphabet*, in our case $\{0,1\}$. So $\mathrm{L} \subseteq \{0,1\}^*$.

A search problem is similar in spirit: It is parametrized by some search function

$$S(x,y) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\} \tag{2}$$

and the task is for a given x to find a y such that $S(x,y) = 1$. Clearly a search problem defines a decission problem in the following way: let $x \in \{0,1\}^*$ be some string then $f(x) = \exists y \in \{0,1\}^* : S(x,y)$. I.e. $f(x)$ is the boolean value whether there exists a y such that $S(x,y) = 1$. Further constraints, eg on the resources of y may be added (c.f. the class NP).

Then, a counting problem is defined as counting the number of solutions to a search problem: given a search problem S and some $x \in \{0,1\}^*$, return $|\{y : S(x,y) = 1\}|$. This is, count the number of solutions y that are accepted by the search problem on input x, $S(x,\cdot)$.

Complexity theory deals mainly with the relations between different complexity classes and characterising problems in terms of their complexity. In both cases, the main proof technique is to embed already known problems into novel problems. This is, for example, one constructs a map Λ that maps instances p of some decision problem P to instances $\Lambda(p)$ of another decision problem P', such that $f_{P'}(\Lambda(p))$ is true if and only if $f_P(p)$ is true (i.e. the truth value of the instance is preserved). This then implies that problem P' is at least as hard as Problem P (having an algorithm that solves P' we automatically have an algorithm that solves P via first applying Λ to the given input).

For a complexity class X, we say that a problem is *in* X if it is contained in X. In particular, if a class Y is a subclass of X then every problem in Y is automatically in X (as sets we literally have $Y \subseteq X$). We call a problem X-hard if it is at least as hard as all problems contained in X. We call a problem X-complete if it is both in X and X-hard. Complete problems are therefore those problems that characterise a complexity class, viz., lie at the boundary of the class.

a) What is the difference between Deterministic, Random (or Probabilistic) and Non-deterministic Turing machines? Find definitions for P and NP that only differnetiate in the Turing machines used. Also find definitions of BPP and BQP that only differentiate in terms of the turing machines used. (Feel free to read up in wikipedia, this is mainly to give you an intuition about these different computational models.)

b) Look up and understand two complete problems for each of the complexity classes NP and #P. particular for NP: Understand the equivalence between the verifier based definition and the definition via non-deterministic Turing machines. Use the verifier based definition of NP to argue that P $\subseteq$ NP.

The problem 3-SAT is a paradigmatic problem, which is complete for the class NP. A 3-SAT formula f on input x is a formula in conjunctive normal form, that is, a formula of the form

$$f(x) = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_{42} \vee \neg x_{10} \vee \neg x_7) \wedge \cdots,$$

where $\neg$ denotes negation, that is $\neg 0 = 1, \neg 1 = 0$, $\vee$ denotes a logical OR, and $\wedge$ denotes a logical AND. The 3-SAT decision problem is to decide, given a formula f, whether there exist assignments of the variables $x = (x_1, \ldots, x_n)$ such that $f(x) = 1$.

c) Argue that 3-SAT is in NP. (*Hint*: Use the verifier based definition)

d) Show that a 3-SAT instance on n binary variables can be embedded in the problem of determining, whether the ground state energy of a classical 3-local Ising Hamiltonian is 0 or at least 1.

   *Hint: Use a classical spin-*1/2 *Hamiltonian of the form*

$$H = \sum_{i,j,k \in [n]} \frac{h_{ijk}}{8} (\mathbb{1} \pm Z_i)(\mathbb{1} \pm Z_j)(\mathbb{1} \pm Z_k),$$

   with integer coefficients h_{ijk}. Map the individual clauses onto the individual local terms and "penalize" the violated constraints by a positive energy.

e) How hard is the computation of ground state energies of local Hamiltonians to inverse polynomial precission (the number of particles n is the scaling parameter)? (It is sufficient to give the hardness proven in the previous exercise).