Freie Universität Berlin
Tutorials on Quantum Information Theory
Winter term 2020/21

Problem Sheet 8
Turing Machines and Complexity

J. Eisert, J. Haferkamp, J. C. Magdalena De La Fuente

---

1. **Turing machines and computability.**

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 and end states S and T, 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, which may either be 'move left' (L) or 'move right' (R) or 'erase σ and write σ'' with $\sigma' \in \Sigma$.

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 and $a \in \Sigma \cup \{L, R\}$ is the action of the machine. The machine is initialised in state A on the very left of tape. Upon reaching the state T the program terminates.

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 $\overline{k} = \underbrace{11 \cdots 1}_{k+1}$.

a) Write a Turing 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.

Solution: For a better readability we denote the set of tuple in $S \times \Sigma \times S \times \Sigma \cup \{L, R\}$ defining the program as a mapping $S \times \Sigma \to S \times \Sigma \cup \{L, R\}$.

Let $\Sigma = \{\text{‘E’}, \text{‘O’}, \text{‘0’}, \text{‘1’}\}$ and $S = \{S, E, O, T\}$, the program is given by

$$\{$$
$$(S, \text{‘0’}) \mapsto (S, R) \quad (1)$$
$$(S, \text{‘1’}) \mapsto (E, R) \quad (2)$$
$$(E, \text{‘1’}) \mapsto (O, R) \quad (3)$$
$$(O, \text{‘1’}) \mapsto (E, R) \quad (4)$$
$$(E, \text{‘0’}) \mapsto (T, \text{‘E’}) \quad (5)$$
$$(O, \text{‘0’}) \mapsto (T, \text{‘O’}) \quad (6)$$
$$\}.$$

The program writes ‘E’ if the given number is even and ‘O’ otherwise.

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 $(0\overline{(k+l)}00\cdots$.

Solution: The strategy we want to implement is to concatenate the two numbers by replacing the '0' in between the two numbers by a '1' and deleting two '1's at the end. Let $\Sigma = \{\text{'0'}, \text{'1'}\}$ and $S = \{S, K, L, D1, D2, T\}$, the program is

$$\{$$

$$(S, \text{'0'}) \mapsto (S, R) \tag{7}$$
$$(S, \text{'1'}) \mapsto (K, R) \tag{8}$$
$$(K, \text{'1'}) \mapsto (K, R) \tag{9}$$
$$(K, \text{'0'}) \mapsto (L, \text{'1'}) \tag{10}$$
$$(L, \text{'1'}) \mapsto (L, R) \tag{11}$$
$$(L, \text{'0'}) \mapsto (D1, L) \tag{12}$$
$$(D1, \text{'1'}) \mapsto (D1, \text{'0'}) \tag{13}$$
$$(D1, \text{'0'}) \mapsto (D2, L) \tag{14}$$
$$(D2, \text{'1'}) \mapsto (D2, \text{'0'}) \tag{15}$$
$$(D2, \text{'0'}) \mapsto (T, R) \tag{16}$$

$$\}.$$

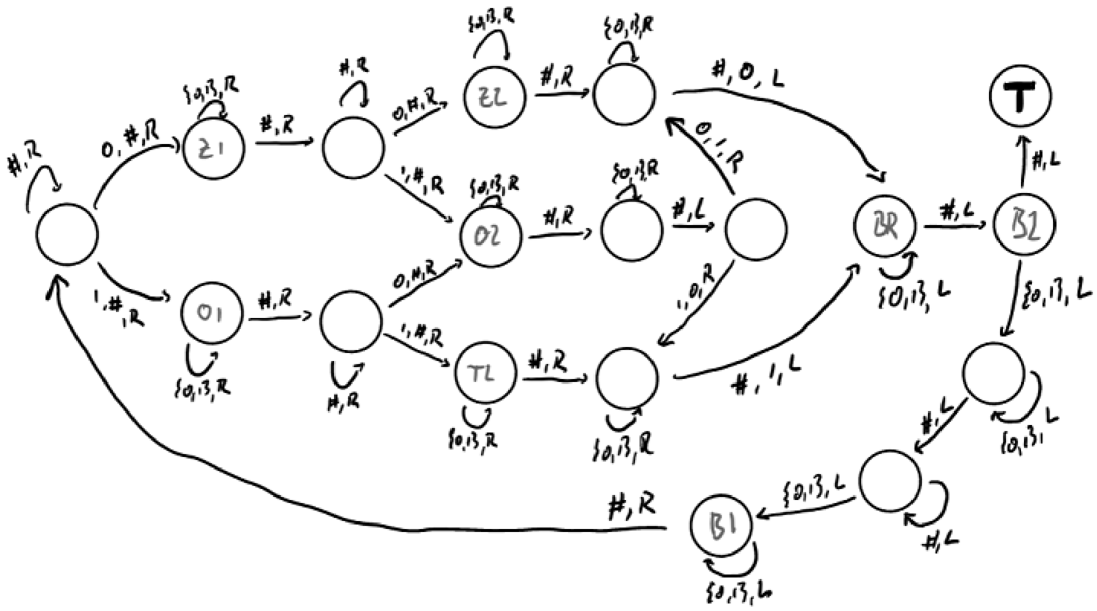
Unary encodings are not very efficient as opposed to binary encodings for which only $\log_2 k$ many bits are required.

c) Write a program that performs binary addition and writes the solution behind the input on the tape.

Hint: Think about the following questions: How many symbols are required? What is a sensible choice of input representation on the tape?

Solution: $\Sigma = \{0, 1, \#\}$ and let m and n be the bit strings *starting with the least-significant bit on the left* representing two integers. The input should be of the form '$\# \cdots \# m \# n \# 0 \#\#\# \cdots$', where we assume that the bit strings for m and n are zero-padded to have exactly the same length.

Let $A, B \in S$, $i, o \in \Sigma$ and $m \in \{L, R\}$. We use the short hand notation $A \xrightarrow{i,o,m} B = A \xrightarrow{i,o} A' \xrightarrow{o,m} B$ to spare writing out the trivial intermediate states after a writing operation. The program is given by the following diagram:



After having executed the program the tape should look like ‘$\# \cdots \# k \# \cdots$’ with k is the bit-string representing $m+n$.

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

2. **Complexity classes and complete problems.**

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’. Decision problems are problems of the form: Given $n \in \mathbb{N}$, decide whether $\exists x : f(x)$ or $\neg f(x)$, where $f : \{0,1\}^n \to \{0,1\}$ is some binary function. Conversely, in a counting problem, given x and f, we are supposed to return $|\{x : f(x) = 1\}|$ that is, the number of solutions that are accepted by f.

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, that is, to prove statements of the form: Assuming we could solve problem $P \in X$, then we could also solve P'. Hence P' is at least as hard as all problems in X.

For a complexity class X, we say that a problem is *in* X if it is contained in 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) Look up and understand two complete problems for each of the complexity classes NP and #P.

Solution: Examples of NP-complete problems: 3-SAT, Travelling Salesman, Clique, Graph coloring . . .

Examples of #P-complete problems can be defined by taking the counting versions of the above problems, i.e. #SAT, #Travelling Salesman, #Clique, . . .

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 $f(x) = 1$.

b) Argue that 3-SAT is in NP.

Solution: By definition a problem is in NP if we have a polynomial algorithm to check a solution. For 3-SAT given an assignment, one can evaluate the compatibility in at most $3m$ many computational steps.

c) 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*

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

with integer coefficients h_{ijk}.

Solution: We begin by noticing that a 3-SAT clause accepts all but a single assignment of the three variables using that for the clause on x_i, x_j, x_k given by (e.g.) $C_{ijk} = (x_i \vee \neg x_j \vee x_k) = \neg(\neg x_i \wedge x_j \wedge \neg x_k)$ so that $C_{ijk} = 0$ if and only if $x_i = \neg x_j = x_k = 0$.

The idea is now to use Hamiltonian terms that penalize this configuration with an energy penalty. So for the clause C_{ijk} we include a term in the Hamiltonian proportional to the projector onto the subspace spanned by the rejected assignment, i.e.

$$|010\rangle\langle 010|_{ijk} = \frac{1}{8}(1 + Z_i)(1 - Z_j)(1 + Z_k) \tag{17}$$

and likewise for all other clauses. Since all such terms commute, after taking the sum, the ground state of $H = \sum_{ijk} h_{ijk}$ has energy 0 if and only if $f(x)$ has a satisfying assignment.

d) What is a natural quantum equivalent of this problem?

Solution: The quantum equivalent is the so-called local Hamiltonian problem. In the k-local Hamiltonian problem we allow for an arbitrary Spin-Hamiltonian consisting of Pauli operators that have non-trivial support on at most k sites. Compared to our 3-SAT embedding we do not restrict ourself to diagonal projectors, but allow for arbitrary k-local terms, e.g. $X_i \otimes X_j \otimes Y_k$. This actually 'defines' the class QMA - Quantum NP.