

Math 278 - Geometry and Algebra of Computational Complexity

Taught by David Donghoon Hyeon
Notes by Dongryul Kim

Fall 2018

`~instructor~ ~meetingtimes~ ~textbook~ ~enrolled~ ~grading~
~courseassistants~`

Contents

| | | |
|----------|--------------------------------------------------------|-----------|
| 1 | September 5, 2018 | 3 |
| 1.1 | Turing machines | 3 |
| 2 | September 10, 2018 | 5 |
| 2.1 | Non-deterministic Turing machines | 5 |
| 2.2 | Encoding Turing machines | 6 |
| 3 | September 12, 2018 | 8 |
| 3.1 | Uncomputable functions | 8 |
| 4 | September 17, 2018 | 9 |
| 4.1 | Satisfiability | 9 |
| 4.2 | Hilbert's Nullstellensatz | 10 |
| 4.3 | Hilbert's tenth problem | 11 |
| 5 | September 19, 2018 | 12 |
| 5.1 | Decidable and semi-decidable sets | 12 |
| 5.2 | Register machines | 13 |
| 6 | September 24, 2018 | 15 |
| 6.1 | Register equations and their Diophantization | 15 |
| 7 | September 26, 2018 | 17 |
| 7.1 | Diophantization of the exponential relation | 17 |
| 7.2 | Finishing Hilbert's tenth problem | 18 |

| | | |
|-----------|---------------------------------------------------|-----------|
| 8 | October 10, 2018 | 20 |
| 8.1 | Crash course on quantum mechanics | 20 |
| 9 | October 15, 2018 | 22 |
| 9.1 | Grover search algorithm | 22 |
| 9.2 | Quantum circuits | 23 |
| 10 | October 17, 2018 | 25 |
| 10.1 | Circuit for the Grover search algorithm | 25 |
| 10.2 | Quantum Fourier transform | 26 |

1 September 5, 2018

There are going to be biweekly homeworks, and a final writing project. The goal of the course is to introduce you to the various aspects of computational complexity theory. There will be four parts:

1. Turing machines, deterministic and non-deterministic, probabilistic algorithms, reduction, NP-completeness
2. Undecidable problems, Hilbert's 10th problem of solving diophantine equations
3. Computer models, continuous time systems, Blum–Smale–Shub model, quantum computers
4. Geometric complexity theory, algebro-geometric and representation theoretic approach to $P \neq NP$

We may consider the determinant as a point in $\mathbb{P}(\text{Sym}^n(\mathbb{C}^{n^2}))$. There is this conjecture that there is no constant $c \geq 1$ such that for all large m ,

$$\text{GL}_{m^{2c}}[\ell^{m^c - m} \text{perm}_m] \notin \overline{\text{GL}_{m^{2c}}[\det_{m^2}]}.$$

This implies $P \neq NP$.

When you do any kind of programming at home, you use discrete time and discrete space. At the end, it really looks like

$$x_{k+1} = f(x_k).$$

On the other hand, the continuous time and space analogue will be a differential equation

$$y' = f(y).$$

Differential analyzers and continuous neural networks are like this. On the other hand, states in quantum computers lie in Hilbert spaces, and so they have continuous space but discrete time.

1.1 Turing machines

This is going to be boring. Let Σ be a finite set of alphabets, for instance, $\Sigma = \{0, 1\}$ for modern computers. Σ^* is the set of all words on Σ .

Definition 1.1. A **language** over Σ is a subset of Σ^* . A **decision problem** encoded on Σ is a partition

$$\Sigma^* = (\text{yes}) \amalg (\text{no}) \amalg (\text{non}).$$

(You get a yes or a no or an error.) The language associated to a decision problem Π is the “yes” part, and is denoted by L_Π .

Definition 1.2. A **deterministic Turing machine** has a read-write head, a bi-infinite tape, and a DTM program consisting of

- Σ a finite set of tape symbols, with $b \in \Sigma$ a blank symbol, and $\gamma \subseteq \Sigma$ a set of input symbols with $b \notin \gamma$,
- a finite set Q of states with distinguished q_0, q_Y, q_N of start, yes, no states,
- a transition function

$$\delta : (Q \setminus \{q_Y, q_N\}) \times \Sigma \rightarrow Q \times \Sigma \times \{\pm 1\}.$$

You should think of there being an infinite tape and a state-controller pointing to a certain point on the tape. The state-controller reads the tape symbol at that point, and plugs its own state and the tape symbol to δ . The output will be the new state of the state-controller, the symbol that will be written, and where the read-write head will move next. The program ends when either q_Y or q_N is hit.

On some inputs, a deterministic Turing machine may never halt. In fact, there is no “algorithm” that can determine whether a given deterministic Turing machine halts on a certain input. We will prove this shortly.

Example 1.3. Consider the following Turing machine. Find what this does.

| $q \setminus \sigma$ | 0 | 1 | b |
|----------------------|----------------|----------------|----------------|
| 0 | 0, 0, 1 | 0, 1, 1 | 1, b , -1 |
| 1 | 2, b , -1 | 3, b , -1 | N , b , -1 |
| 2 | Y , b , -1 | N , b , -1 | N , b , -1 |
| 3 | N , b , -1 | N , b , -1 | N , b , -1 |

Definition 1.4. Let M be a deterministic Turing machine. The language recognized by M is

$$L_M = \{x \in \gamma^* : M \text{ accepts } x\}.$$

So M solves the decision problem Π if $L_M = \Pi$.

Definition 1.5. The **time complexity** of M is the function

$$T_M(n) = \max_{|x|=n} (m : M \text{ halts on } x \text{ in } m \text{ steps}),$$

where a step is a movement of the head.

2 September 10, 2018

Today we will talk about non-deterministic Turing machines.

2.1 Non-deterministic Turing machines

I will give two definitions, which are going to be equivalent. Recall that a deterministic Turing machine is just a infinite tape with a read-write head. The program really is the transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{\pm 1\}$. In a **non-deterministic Turing machine**, the picture is the same, but there are two transition functions δ_0 and δ_1 . At each computational step, the machine makes an arbitrary choice between δ_0 and δ_1 .

Definition 2.1. A **computation path** is the sequence of choices the machine makes. For instance, it looks like

$$\delta_0\delta_1\delta_0\delta_0\delta_1\delta_1\cdots \text{ or } 010011\cdots.$$

The length of the computation path is going to be the length of the computation.

Definition 2.2. M is said to run in time $T(n)$ if for every input x and every computation path, the machine halts within $T(|x|)$ steps. We say that M is a **polynomial time** non-deterministic Turing machine if it runs in some polynomial time.

We say that M accepts x if there exists a computation path that halts with q_Y . Then we define the language accepted by M as

$$L_M = \{x \in \Sigma^* : M \text{ accepts } x\} \subseteq \Sigma^*.$$

Then we define

$$\mathcal{NP} = \{L \subseteq \Sigma^* : \text{exists a polynomial nDTM } M \text{ with } L_M = L\}.$$

It is clear that $\mathcal{P} = \mathcal{NP}$, because a DTM is always a nDTM. (\mathcal{P} is the same thing with DTM instead of nDTM.) Intuitively, \mathcal{NP} means that you can check an answer (computational path) in polynomial time.

Let me give an alternative definition of an nDTM. We now consider a two-tape machine, and we consider a transition function

$$\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \Gamma \times \{\pm 1\} \times \{0, 1\}.$$

It also has a “guessing module”. On an input x on the first tape, the guessing module writes an arbitrary guess y on second tape, of length bounded in polynomial by the length of x . Then the machine proceeds with the computation deterministically.

Definition 2.3. We say that M runs in time $T(n)$ if on an input x and for any guess, M halts in $T(|x|)$ steps.

Using this, we can again define \mathcal{NP} so that L is in \mathcal{NP} if there exists a language R (recognizable by a polynomial DTM) and a polynomial q such that

$$L = \{x : \exists y, |y| \leq q(|x|), (x, y) \in R\}.$$

In this case, we say that y is a “witness” or a “certificate” for x .

Theorem 2.4. *The two definitions are equivalent.*

Proof. Let L be \mathcal{NP} according to the first definition. Then you can use the computation path as the guess. In particular, we can do something like

$$\delta(q, \sigma_1, \sigma_2) = (\sigma_2 \delta_1(\sigma_1, q) + (1 - \sigma_2) \delta_0(\sigma_1, q), 1).$$

The other direction does it similarly. □

You can also define stuff like k -tape machines, but if you think hard enough, you will see that there is no difference.

Definition 2.5. We say that a problem Π is **reduced** to Π' if there is a (polynomially) computable function

$$f : \Sigma^* \rightarrow \Sigma^*$$

such that $x \in L(\Pi)$ if and only if $f(x) \in L(\Pi')$.

What do we mean by a computable function? The easiest way to define it is by using a k -tape machine. This k -tape machine M has a dedicated input tape and an output tape. We say that M computes f if on input x , the content of the output tape is equal to $f(x)$ when the machine halts.

Definition 2.6. A problem or language is said to be **NP-hard** if any NP language can be polynomially reduced to it. It is said to be **NP-complete** if it itself is in NP.

If you search on Wikipedia, you can find hundreds of examples of NP-complete problems, mostly in discrete mathematics.

2.2 Encoding Turing machines

Now we want to encode a Turing machine, i.e., construct a map

$$\epsilon : \{0, 1\}^* \rightarrow \{\text{Turing machines}\}.$$

We are going to make a Turing machine on $\{0, 1, -\}$ and $Q = \{0, 1, 2, \dots, l\}$. We encode l and the transition function from values $\delta(\sigma, q)$ as a binary word. If any binary string does not come from this procedure, map it to some trivial Turing machine. This defines ϵ .

Definition 2.7. There exists a DTM \mathcal{U} such that for every (x, α) ,

$$\mathcal{U}(x, \alpha) = M_\alpha(x).$$

This is called the **universal Turing machine**. If M_α halts on input x within T steps, then \mathcal{U} halts in (x, α) within $CT \log T$ steps.

Our personal computers are all like this. If you write a program, you can run it. You can see at a high level how this will work. I was told that it is very involved to actually construct this machine.

3 September 12, 2018

We will only have 30 minutes of lecture because there is the Ahlfors lectures.

3.1 Uncomputable functions

If you want to show that uncomputable functions exists, this is easy because there are countably many Turing machine, and uncountably many languages. So we want a construction of a function that is not computable by any DTM.

Example 3.1. Recall that we had this encoding of a DTM given by

$$\epsilon : \Sigma^* \rightarrow \{\text{DTMs}\}; \quad \alpha \mapsto M_\alpha.$$

Now define

$$f(\alpha) = \begin{cases} 0 & M_\alpha \text{ accepts } \alpha, \\ 1 & \text{else.} \end{cases}$$

Then we claim that f is not computable. Suppose that $M = M_{\alpha^*}$ computes f . Then

$$M_{\alpha^*}(\alpha^*) = 1 \quad \Leftrightarrow \quad f(\alpha^*) = 1 \quad \Leftrightarrow \quad M_{\alpha^*} \text{ does not accept } \alpha^*.$$

This is contradictory.

Example 3.2. Here is another example. Consider the problem of taking (α, x) and outputting whether M_α halts on input x . Suppose M_ξ solves the Halting problem HALT. We are then going to build a solution to the previous function by using the universal Turing machine. You first plug in (α, α) to M_ξ , and if it says no, just output 1. If it says yes, run \mathcal{U} with α and α , and output the answer. This shows that the halting problem is undecidable.

Example 3.3. Let us look at the Bounded Halting Problem for nDTMs, denoted BHPN. First note that nDTMs can be encoded,

$$\epsilon : \Sigma^* \rightarrow \{\text{nDTMs}\},$$

and also that there is an efficient universal nDTMs. Now the input is (α, x, t) , and the problem is,

Does M_α halt on x on t steps?

This problem is \mathcal{NP} because we can use the universal machine. On the other hand, it is \mathcal{NP} -hard as well. To see this, let $L \in \mathcal{NP}$ and let M be the nDTM that recognizes L . Then we can define

$$f : \Sigma^* \rightarrow \Sigma^*; \quad x \mapsto (\alpha, x, T(|x|)).$$

This reduces L to the Bounded Halting Problem. This shows that BHPN is \mathcal{NP} -complete.

4 September 17, 2018

Last time we constructed an uncomputable function. The point was to give an explicit construction. This was

$$f(\alpha) = \begin{cases} 0 & M_\alpha \text{ accepts } \alpha \\ 1 & \text{else.} \end{cases}$$

Then we showed that **HALT** is uncomputable by reducing it to this function. Then, we showed that **BHNP** is \mathcal{NP} -complete. This problem was defined by

$$\{(\alpha, x, t) : M_\alpha \text{ accepts } x \text{ within } x \text{ steps}\}.$$

Now we want a natural problem that is \mathcal{NP} -complete.

4.1 Satisfiability

Let Γ be a finite set of variables. Then a **literal** is a variable x or a negation of a variable $\neg x$. A **clause** is a finite set of literals. A **truth assignment** is a map $\xi : \Gamma \cup \neg\Gamma \rightarrow \{0, 1\}$ such that $\xi(\neg x) = 1 - \xi(x)$. An instance of the problem **SAT** is a finite set I of clauses, and the problem is,

Does there exist a truth function ξ satisfying all $C \in I$, where ξ satisfies $C = \{U_1, \dots, U_l\}$ means that $\xi(U_i) = 1$ for some i ?

Using the logical “and” \wedge and “or” \vee , we can write it as finding a solution to

$$\bigwedge_{C_i \in I} (U_{i1} \vee U_{i2} \vee \dots \vee U_{ij_i}).$$

Theorem 4.1 (Cook, 1971). *The problem SAT is \mathcal{NP} -complete.*

Proof. It is easy to show that it is \mathcal{NP} , because we can set the guess as the truth function. Now let us show that it is \mathcal{NP} -hard. Suppose $L \in \mathcal{NP}$ is recognized by a nDTM M . Assume that the tape symbols are $\{0, 1, -1 = \text{blank}\}$, and states $\{0 = q_0, 1 = q_Y, 2 = q_N, \dots, l\}$. Let the input be x , with $n = |x|$, and assume the running time is $p(n)$.

Now what we are going to do is the write down everything in the computation and turn it into a single formula. Define the logic variables

$$\begin{aligned} \sigma_{t,i,j} &= \text{at time } t, \text{ the tape content in the } i\text{th square is } j, \\ q_{t,s} &= \text{at time } t, \text{ state is } s, \\ h_{t,i} &= \text{at time } t, \text{ head is at tape square } i. \end{aligned}$$

Here, the number of variables is at most constant times $p(n)^2$. Next we can write down all the relations between the variables that we need for it to accept the input. These are

- $q_{0,0}$,

- $q_{p(n),1}$,
- σ_{0,i,x_i} for $1 \leq i \leq n$,
- $\sigma_{0,i,-1}$ for $i \leq -q(n)$ and $i \geq n+1$, (the squares between $-q(n)$ and 0 is used to store the guess)
- $\bigvee_i h_{t,i}$,
- $\neg h_{t,i} \vee \neg h_{t,j}$ for $i \neq j$,
- $\bigvee_j \sigma_{t,i,j}$,
- $\neg \sigma_{t,i,j} \vee \neg \sigma_{t,i,j'}$ for all $j \neq j'$.
- equations encoding the transition functions like

$$\begin{aligned} \neg \sigma_{t,i,j} \vee \neg h_{t,i} \vee \neg q_{t,s} \vee \sigma_{t+1,i,j'}, \\ \neg \sigma_{t,i,j} \vee \neg h_{t,i} \vee \neg q_{t,s} \vee \sigma_{t+1,i,j'}, \\ \neg \sigma_{t,i,j} \vee \neg h_{t,i} \vee \neg q_{t,s} \vee \sigma_{t+1,i,j'}, \end{aligned}$$

and equations stating that nothing else changes.

You can count the number of variables, and then you are going to see that the number of equations is polynomial in n . \square

4.2 Hilbert's Nullstellensatz

Consider an algebraically closed $k = \bar{k}$. Here is a weak version of Hilbert's Nullstellensatz.

Theorem 4.2. *If $f_1, \dots, f_m \in k[x_1, \dots, x_n]$, then*

$$f_1 = f_2 = \dots = f_m = 0$$

has no solution if and only if there exist $g_i \in k[x]$ such that $\sum f_i g_i \equiv 1$.

Now consider the problem HN_k , which have instances $f_1, \dots, f_m \in k[x]$, and ask,

Does $f_1 = \dots = f_m = 0$ have a common solution?

If we use Hilbert's Nullstellensatz, we get a linear algebra problem by writing down the coefficients. If we write $f_i = \sum_{\alpha} a_{i\alpha} x^{\alpha}$ and $g_i = \sum_{\beta} b_{i\beta} x^{\beta}$, then we are solving

$$\sum_{\alpha+\beta=\gamma} a_{i\alpha} b_{i\beta} = \begin{cases} 1 & \gamma = 0 \\ 0 & \gamma \neq 0. \end{cases}$$

But what is the size of the system?

Theorem 4.3 (Browawell, Kollar). *We can further impose $\deg(g_i) \leq O(d^n)$, where $d = \max\{3, \deg(f_i)\}$.*

In fact, we are going to show that HN is \mathcal{NP} -hard, and \mathcal{NP} -complete over a finite field. This is an important basis for security analysis in cryptography.

Theorem 4.4. HN is \mathcal{NP} hard.

Proof. We will reduce SAT to HN. An instance looks like

$$\bigwedge (u_{i1} \vee \cdots \vee u_{is_i}),$$

and so we consider the system of polynomial equations

$$f_C = \prod f_i$$

for each $C \in I$. □

4.3 Hilbert's tenth problem

This is trying to solve Diophantine equations. A Diophantine equation is,

$$P(x_1, x_2, \dots, x_n) = 0$$

where $P \in \mathbb{Z}[x_1, \dots, x_n]$. Then Hilbert's question was to find an algorithm for determining whether a given $P = 0$ has a solution in rational integers.

Definition 4.5. A set $S \subseteq \mathbb{N}^n$ is said to be **Diophantine** if there exists a (integer coefficient) polynomial P such that

$$S = \{a \in \mathbb{N}^n : \text{there exists } \underline{x} \in \mathbb{N}^m \text{ such that } P(a, \underline{x}) = 0\}.$$

For instance, $\{(a, b) : a \geq b\}$ is $\{(a, b) : \exists x, a = b + x\}$, and so Diophantine. The set of composites is

$$\{a : \exists x, y, (a = (x + 2)(y + 2))\}.$$

The set of primes also happens to be prime, and this is a consequence of the Hilbert's tenth problem.

5 September 19, 2018

To show the \mathcal{NP} -completeness of SAT, we assigned a bunch of variables to decide the “configuration”. Then we encoded what it means to compute, as relations between these variables. This gave a polynomial reduction of any \mathcal{NP} problem to SAT.

5.1 Decidable and semi-decidable sets

Then we defined Diophantine sets as sets S that can be expressed as

$$S = \{a \in \mathbb{N}^m : \text{there exists } x \in \mathbb{N}^n \text{ such that } P(a, x) = 0\}$$

for some polynomial $P(a, x)$. We saw the examples $\{(a, b) : a \geq b\}$ and $\{\text{composites}\}$. A more interesting example is $\{(x, y, n) : x^n + y^n = z^n\}$. In fact, we are going to see that all sets that are algorithmically determinable are Diophantine.

We say that Hilbert’s 10th problem is decidable (resp. undecidable) over R if there is (resp. is not) an algorithm for deciding whether a given Diophantine equation has a solution in R . Also, let us denote Hilbert’s 10th problem by H10. Hilbert’s hope was that H10 is decidable over \mathbb{Z} . Then it is also decidable over \mathbb{Q} .

Theorem 5.1 (Davis–Putnam–Robinson–Matiyasevich). *The problem H10 is undecidable over \mathbb{Z} .*

Definition 5.2. A set S is **decidable** if there is a deterministic Turing machine that computes χ_S .

For example, $L(\text{HALT})$ is not a decidable set. But we can extend this a bit further.

Definition 5.3. A set S is **semi-decidable** if it is the halting set of a deterministic Turing machine.

Because $L(\text{HALT})$ is the halting set of the universal DTM, it is semi-decidable. This is a really important ingredient in the proof of Hilbert’s 10th problem.

Definition 5.4. We say that S is **recursively enumerable** if there exists a deterministic Turing machine M that outputs $x_1 \# x_2 \# x_3 \# \dots$ where S is precisely the set $S = \{x_1, x_2, \dots\}$. In other words, S is the range of a computable function.

Proposition 5.5 (homework). *Recursive enumerability is equivalent to semi-decidability.*

Theorem 5.6 (Davis–Putnam–Robinson–Matiyasevich). *A set is Diophantine if and only if it is recursively enumerable.*

Proof. A Diophantine set is recursively enumerable, because we can try all the possible solutions and test them in order. The other direction is hard, but here is an overview. Let S be a recursively enumerable set. This means that S can be enumerated by a deterministic Turing machine. Now I want to write down a Diophantine equation such that it a tuple is being outputted if and only if it is a solution.

- We first arithmetize register machines. A register machine is a machine that is equivalent to a Turing machine. It has a register (which is like the tape in a Turing machine) and command lines (which is like the transition function in a Turing machine). We assign variables for each register and line, and then write down the relations.
- Then we Diophantize these relations. Many of the relations are going to be of the form

$$r \preceq s$$

which are called **bit maskings**. Here, r and s are binary numbers, and we define $r \preceq s$ if $r_i \leq s_i$ for all i . We are going to turn this into an exponential relation, using Lucas's theorem. (If you have done enough problem solving in high school, this is a standard trick.) Then we are going to show that this is a Diophantine relation.

So we turn a Turing machine into a Diophantine equation. □

5.2 Register machines

So let me define a register machine. There are finitely many registers, R_1, \dots, R_r , and they can store nonnegative integers, of arbitrary size. It comes with a finite (command) lines L_1, L_2, \dots, L_l , where each L_i looks like

$$\begin{aligned} L_i : R_j &\leftarrow R_j \pm 1 \quad \text{or} \\ L_i : \text{GOTO } L_k &\quad \text{or} \\ L_i : \text{IF } R_j > 0 (\text{or } = 0) &\text{ GOTO } L_k. \end{aligned}$$

We say that M computes $y = f(x)$ if we have $x = (x_1, \dots, x_n)$ in the registers at time $t = 0$, and when the program ends, the values stored at the register are $f(x) = (f_1(x), \dots, f_n(x))$.

So let us try to arithmetize this register machine. Let us say that R_1, \dots, R_n are our registers, L_1, \dots, L_l are the lines, and $x \in \mathbb{N}^n$ is the input, with s the running time. First choose $Q = 2^{\text{big}}$ really big so that

$$x + s < \frac{Q}{2}, \quad l < \frac{Q}{2}, \quad r_{j,t} < \frac{Q}{2}.$$

This is going to be the possible range of the registers. Define the variables

$$\begin{aligned} r_{j,t} &= \text{register value of } R_j \text{ at time } t, \\ l_{i,t} &= \begin{cases} 1 & \text{machine carries out } L_i \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Then we define

$$R[j] = \sum_{t=0}^s r_{j,t} Q^t, \quad L[i] = \sum_{t=0}^s l_{i,t} Q^t$$

to make the data into a single number. Now we have the parameters x, y and variables $s, Q, R[1], \dots, R[n], L[1], \dots, L[l]$.

What are now the relations?

- start and end: $L_1 \succeq 1$ and $L_l = Q^s$,
- $Q = 2^t$,
- $x + s < Q/2, l < Q/2, R_j \preceq (Q/2 - 1)I$ (this enforces $r_{j,t} < Q/2$ because $r_{j,t}$ moves by at most 1),
- $I = (Q^{s+1} - 1)/(Q - 1)$,
- $L_i \preceq I$ and $\sum_{i=1}^l L_i = I$,
- execution commands: if $L_i : R_j \leftarrow R_j \pm 1$, then $QL_i \preceq L_{i+1}$, and other commands.

6 September 24, 2018

Last time we looked at register machines, which had registers R_1, \dots, R_r that can store arbitrarily large integers, and lines L_1, \dots, L_l that can change the value of a register by 1 or transfer to another line.

Example 6.1. Suppose you want to compute $f(x) = 2x$, and let's say that we start with x in R_2 and 0 in R_1 . Then the register machine

```
L1 If  $R_2 = 0$  Goto  $L_6$ 
L2  $R_2 \leftarrow R_2 - 1$ 
L3  $R_1 \leftarrow R_1 + 1$ 
L4  $R_1 \leftarrow R_1 + 1$ 
L5 Goto  $L_1$ 
L6 End
```

computes this.

Consider the function

$$G(l) = \max\{\text{output of a } l\text{-line machine with input } R_1 = 0\}.$$

This is well-defined, because there are only finitely many machines with l lines, up to equivalence. Suppose M is a c -line machine that computes $f(x)$. Then if we put x lines saying $R_1 \leftarrow R_1 + 1$ and then 5 lines $x \mapsto 2x$ and then c lines for M , we can compute $f(2x)$. So we get

$$f(2x) \leq G(x + 5 + c).$$

In particular, we can never compute G , because then $G(2x) \leq G(x + 5 + c)$ is a contradiction.

6.1 Register equations and their Diophantization

Given a register machine M , we want to find a polynomial $P(x, y, \dots) = 0$ which has a solutions if and only if $y = M(x)$. We started with these variables

$$s, \quad r_{jt}, \quad l_{it}$$

as in the case of SAT. But then, the problem is that the number of variable depends on s . So instead, we defined $Q = 2^N$ and

$$R_j = \sum r_{jt} Q^t, \quad L_i = \sum l_{it} Q^t.$$

Then we had all these relations between $R_j, L_i, s, x, y, Q, I = \sum Q^t$. We could also recover r_{jt} and l_{it} by looking at the Q -ary expansion of R_j and L_i .

There were the universal equations, and the execution commands are the following:

- $QL_i \preceq L_{i+1}$ for L_i not containing Goto,
- $QL_i \preceq L_{i+1} + L_k$ and $QL_i \preceq L_k + (IQ - 2R_j)$ (this requires some analysis), if L_i contains “If $R_j > 0$ goto L_k ”,
- something like $R_j = QR_j + \sum_i L_i - \sum_i L_i$ and $R_1 + yQ^s = R_1Q + \sum_i L_i - \sum_i L_i + x$ that encodes how the register values transform.

So the point is that all of them are of the form (up to Diophantine relations)

$$a = b^c \text{ or } a \preceq b.$$

For the bit masking relation, we use the following theorem.

Theorem 6.2 (Lucas). *If p is a prime, we have*

$$\binom{r}{s} \equiv \prod_i \binom{r_i}{s_i} \pmod{p}$$

where $r = \sum r_i p^i$ and $s = \sum s_i p^i$ are the p -ary expansions.

As a consequence, $s \preceq r$ is equivalent to $\binom{r}{s}$ being odd. Then this relation will be Diophantine if and only if I can encode $u = \binom{r}{s}$ as a Diophantine equation.

Theorem 6.3. *The set $\{(u, r, s) : u = \binom{r}{s}\}$ is Diophantine.*

Proof. We note that

$$\frac{(a+1)^r}{a^s} = a^{r-s} + \binom{n}{n-1} a^{r-s-1} + \cdots + \binom{r}{s} + \binom{r}{s-1} \frac{1}{a} + \cdots + \frac{1}{a^s}.$$

But we note that if $a > 2^r$, then the terms involving $\frac{1}{a}$ will sum to a number smaller than 1. This shows that for any $a > 2^r$, then

$$\text{Rem}\left(\left\lfloor \frac{(a+1)^r}{a^s} \right\rfloor, a\right) = \binom{r}{s}.$$

Note that the relation $\text{Rem}(b, a) = r$ is Diophantine, and similarly the integer part is also Diophantine. So we prove this theorem if we can encode the relation $a^b = c$. \square

So everything reduces to the exponential relation.

Theorem 6.4. *The set $\{(a, b, c) : a = b^c\}$ is Diophantine.*

This uses Pell's equations, and is rather involved. I will only give an overview of how this works next time.

7 September 26, 2018

Last time wrote down the register relations, universal ones and program-specific ones. Many of them were bit-masking relations, and we reduced these to exponential relations. So we needed to know how we can encode the exponential relation

$$\{(a, b, c) : a^b = c\}.$$

This is what we are going to do today.

7.1 Diophantization of the exponential relation

Definition 7.1. For $d = a^2 - 1$ and a an integer, **Pell's equation** is the equation

$$x^2 - dy^2 = 1.$$

The equation admit solutions of the form

$$x_a(n) + y_a(n)\sqrt{a^2 - 1} = (a + \sqrt{a^2 - 1})^n.$$

Using this, we can prove that

$$\{(a, b, n) : b = x_a(n)\}$$

is Diophantine. In fact, the relation $c = y_a(b)$ can be encoded by

- $d^2 - (a^2 - 1)c^2 = 1$,
- $f^2 - (a^2 - 1)e^2 = 1$,
- $i^2 - (g^2 - 1)h^2 = 1$,
- $e = (i + 1)2c^2$,
- $g \equiv a \pmod{f}$,
- $g \equiv 1 \pmod{2c}$,
- $k \equiv c \pmod{f}$,
- $k \equiv b \pmod{2c}$,
- $b \leq 2c$.

To show this, let $h = y_g(r)$ for some r . Then show $b \equiv r \pmod{2c}$ and $r \equiv \pm p \pmod{2c}$, where $c = y_a(p)$. Then we can show that $b = p$ by using $b \leq 2c$.

Note that $x_a(n)$ and $y_a(n)$ grows exponentially in n . One can show that we have

$$(2a - 1)^n \leq y_a(n + 1) \leq (2a)^n.$$

Theorem 7.2 (Robinson). *For all $n \geq 0$ and $b \geq 0$, we have*

$$x_a(n) - (a - b)y_a(n) \equiv b^n \pmod{2ab - b^2 - 1}.$$

Proof. I don't have any intuition for this, but you can play around with numbers. \square

So if $a > y_b(n+1)$ then we have

$$b^n = \text{Rem}(x_a(n) - (a-b)y_a(n), 2ab - b^2 - 1).$$

This is because $b^n < 2ab - b^2 - 1$ since a is really big. This finally shows that the exponential relation is Diophantine.

7.2 Finishing Hilbert's tenth problem

Theorem 7.3. *Hilbert's tenth problem is undecidable.*

Proof. Consider $S = L(\text{HALT})$, which is undecidable but semidecidable. (This means that there is a register machine M such that $S = \{M(1), M(2), \dots\}$.) Suppose the problem is decidable. Then associated to M , there is a Diophantine equation such that

$$y = M(n) \iff \exists \vec{x}, P(y, n, \vec{x}) = 0.$$

So given y , we can test if $P(y, -, -) = 0$ has a solution by a register machine. This determines whether $y \in S$ or not. This contradicts that S is not decidable. \square

Actually, we have a stronger statement. There exists a single (family of) Diophantine equation whose solvability cannot be algorithmically decided.

This whole proof implies that all computable functions are polynomials. Let me be more precise.

Proposition 7.4. *Let $y = f(x)$ be computable. Then there exists a polynomial $P(x, x_0, x_1, \dots, x_n)$ such that*

$$\{(x, y) : y = f(x)\} = \{(x, y) : \exists x_0, \dots, x_n, y = P(x, x_0, \dots, x_n)\}.$$

Proof. Because $\{y = f(x)\}$ is Diophantine, there exists a polynomial $Q(x, y, x_1, \dots, x_n)$ such that $y = f(x)$ if and only if $Q(x, y, x_1, \dots, x_n) = 0$ for some x_i . This is then equivalent to existence of x_0, \dots, x_n such that

$$(x_0 + 1)(1 - Q(x, x_0, x_1, \dots, x_n)^2) = y + 1.$$

This is called Putnam's trick. \square

Also, we see that there exists a **universal Diophantine equation**.

Theorem 7.5. *Fix $n \in \mathbb{N}$. Then there exists a polynomial*

$$U_n(a_1, \dots, a_n, k, y)$$

such that for any polynomial $D(a_1, \dots, a_n, y)$, there exists a k_D such that

$$\{a : \exists \underline{x}, D(a, \underline{x}) = 0\} = \{a : \exists \underline{y}, U(a, k_D, \underline{y}) = 0\}.$$

Proof. We note that the Diophantine sets are enumerable, so let S_1, S_2, \dots be the sets. Let M_1, M_2, \dots be the machines enumerating the solutions, i.e., $S_i = \{M_i(1), M_i(2), \dots\}$. Then we can construct a machine that enumerates

$$\{(a, k) : a \in S_k\},$$

by using the machines. So this is semi-decidable. The Diophantine equation associated to this is going to be the universal equation. \square

8 October 10, 2018

This was a guest lecture by Matthias Christandl. I will talk about quantum mechanics, applied to computer science. This was developed in the early 20th century, in order to overcome the difficulty of describing small particles. It is a mathematical theory that was hugely successful in both predicting and explaining new phenomena.

8.1 Crash course on quantum mechanics

There are some axioms.

- There is a complex Hilbert space \mathcal{H} , with a Hermitian metric, the system of the physics. Two systems \mathcal{H}_A and \mathcal{H}_B combine to $\mathcal{H}_A \otimes \mathcal{H}_B$.
- The state of a system is given by a vector $\psi \in \mathcal{H}$, normalized so that $\|\psi\| = 1$.
- The time evolution is given by the Schrödinger equation

$$i \frac{\partial}{\partial t} \psi(t) = H(t) \psi(t),$$

where $H(t)$ is the Hamiltonian (or energy) that is a Hermitian operator on \mathcal{H} .

- Measurement is given by $\{P_i\}_{i \in I}$, a family of projection, such that $\sum_{i \in I} P_i = \mathbf{1}$ and $P_i P_j = 0$ for $i \neq j$. If a measurement is carried out, you get the outcome “ i ” with probability

$$p_i = \langle \psi, P_i \psi \rangle.$$

After the measurement, the state becomes $\psi_i = \frac{1}{\sqrt{p_i}} P_i \psi$.

Example 8.1. There is the qubit, or the spin- $\frac{1}{2}$ system. This is the simplest possible non-trivial example, $\mathcal{H} = \mathbb{C}^2$. We put the inner product

$$\langle \psi, \phi \rangle = \bar{\psi}_0 \phi_0 + \bar{\psi}_1 \phi_1.$$

Let us look at the Hamiltonian

$$H(t) = H = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Then the time evolution of an arbitrary ψ will be

$$\psi \mapsto \psi_0 e^{-it} e_0 + \psi_1 e^{it} e_1.$$

There is a resolution of the identity,

$$P_0 = e_0 e_0^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad P_1 = e_1 e_1^\dagger = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Now let us imagine that we have a different apparatus, given by a different resolution of the identity

$$Q_0 = \frac{1}{2}(e_0 + e_1)(e_0 + e_1)^\dagger, \quad Q_1 = \frac{1}{2}(e_0 - e_1)(e_0 - e_1)^\dagger.$$

If you take this state $\frac{1}{\sqrt{2}}(e_0 + e_1)$, then you always get $Q = 0$ with probability 1. But if you measure it with P and then measure it with Q , the result will be $Q = 0$ with probability $\frac{1}{2}$ and $Q = 1$ with probability $\frac{1}{2}$.

This really is a generalization of the classical bit. You can also change the first and second components by unitary operators.

Example 8.2. For the harmonic oscillator, we have

$$\mathcal{H} = L^2(\mathbb{R}), \quad H = \frac{\omega^2 \hat{x}^2}{2} + \frac{\hat{p}^2}{2m},$$

where $\hat{x}\psi(x) = x\psi(x)$ and $\hat{p}\psi(x) = -i\frac{\partial}{\partial x}\psi(x)$. Then

$$H = \sum_n \left(n + \frac{1}{2}\right) f_n f_n^\dagger.$$

If we now have n qubits, the Hilbert space is

$$\mathcal{H} = \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 \cong \mathbb{C}^{2^n}.$$

The basis of given by

$$e_{i_1} \otimes e_{i_2} \otimes \cdots \otimes e_{i_n}.$$

If we write $|e_0\rangle = |0\rangle$ and $|e_1\rangle = |1\rangle$, we can write this vector as $|i_1 i_2 \cdots i_n\rangle$. Then we can have states like

$$\psi = \frac{1}{\sqrt{2}}(|00 \cdots 0\rangle + |11 \cdots 1\rangle).$$

These are pretty hard to create in labs, for n pretty large. A quantum circuit is basically applying unitary operations one after another.

9 October 15, 2018

Today we will continue with classical boolean circuits. Then we will talk about quantum circuits and the Grover search algorithm.

Let us fix notation

$$\mathbb{B} = \{0, 1\}, \quad \mathcal{B} = \mathbb{C}^2,$$

the state spaces for the classical and quantum bits. The two vector $|0\rangle, |1\rangle$ are going to be orthonormal bases of the space \mathcal{B} . Then in

$$\mathcal{B}^{\otimes n} = \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2,$$

we can declare the vectors

$$|j\rangle = |j_1 j_2 \cdots j_n\rangle = |j_1\rangle \otimes \cdots \otimes |j_n\rangle$$

for $j_i \in \mathbb{B}$ to be orthonormal. Each “tensor factor” of \mathcal{B}^n is called a single qubit.

Roughly, what a quantum computer can do is to

- perform unitary transformation operations, and
- perform measurements.

How can we use this to compute a boolean function, say something that an ordinary computer can do? There is the **Hadamard gate**

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

and then we can take

$$H^{\otimes n} |0^n\rangle = \frac{1}{2^{n/2}} \sum_{x \in \mathbb{B}^n} |x\rangle.$$

Then when we measure, we get each $|x\rangle$ with probability $1/2^n$.

To compute $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, we need to devise a unitary operator U such that given $x \in \mathbb{B}^n$,

- (1) $U|x\rangle = \sum c_y |y\rangle$,
- (2) $|c_y|^2$ is largest at $y = f(x)$.

9.1 Grover search algorithm

Here is the problem. There is a hidden $y_0 \in \mathbb{B}^n$, and we need to find y_0 by asking some “oracle” the question “Is y the hidden y_0 ?”. Classically, we need to just guess $N = 2^n$ times and ask the queries. But in the context of quantum computation, we can do this in \sqrt{N} queries with high probability.

In the quantum world, this is a unitary matrix U , given by

$$U|y_0\rangle = |y_0\rangle, \quad U|x\rangle = -|x\rangle \text{ for } x \neq y_0.$$

Define

$$V = I - 2|\xi\rangle\langle\xi|, \quad |\xi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle,$$

which is reflection with respect to hyperplane perpendicular to $|\xi\rangle$.

If you think about it, U is a linear combination of I and $|y_0\rangle\langle y_0|$. So the operator VU is a rotation in the plane spanned by y_0 and ξ . But here, note that

$$\sin \varphi = \langle \xi | y_0 \rangle = \frac{1}{\sqrt{N}}.$$

Here, you have to do some analysis. Each time we apply VU , the rotation is by 2φ . So when we do the rotation m times, to get

$$(VU)^m |\xi\rangle,$$

the rotation angle is $m2/\sqrt{N} \approx \frac{\pi}{2}$.

But how would you implement this algorithm? We certainly can't build a machine for each unitary operator. So we want a relatively small set of unitary operators, and simulate other operators using the basic ones. We will show that any unitary operator can be approximated by a "circuit" over a finite set of unitary operators.

9.2 Quantum circuits

Definition 9.1. A Boolean function is a map $f : \mathbb{B}^n \rightarrow \mathbb{B}$, and a **boolean circuit** (over \mathcal{A}) is a representation of a boolean function as a composition of other boolean functions. It consists of

- variables x_1, x_2, \dots, x_n ,
- auxiliary variables y_1, y_2, \dots, y_m ,
- assignments $y_i = f_j(x, y_1, \dots, y_{j-1})$, where $f_j \in \mathcal{A}$.

You can also think of it as an acyclic directed graph with input vertices with in-degree 0 and output vertices with out-degree 0. There are gates G each with out-degree 1 and a map $G \rightarrow \mathcal{A}$.

Theorem 9.2. Any boolean function can be computed by a circuit over

$$\mathcal{A} = \{\wedge, \vee, \neg\}.$$

A **quantum circuit** is the same thing, but with input qubits, and gates unitary operators.

Lemma 9.3. Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be computed by a (boolean) circuit of size L over \mathcal{A} . Then a map of the form

$$(x, 0^l) \mapsto (f(x), g(x))$$

can be computed by a circuit of size $O(L)$ over the set $\{h_\oplus : h \in \mathcal{A}\} \cup \{\text{id}_\oplus\}$, where this \oplus are the invertible maps

$$h_\oplus(x, y) = (x, y \oplus h(x)).$$

So we are introducing these ancilla qubits.

Theorem 9.4. *Any unitary operator admits an exact realization over $\mathcal{G}_1 \amalg \mathcal{G}_2$, where \mathcal{G}_i is the set of all quantum gates with i qubit inputs.*

This only says that any unitary matrix can be decomposed into 2×2 unitary matrices.

Theorem 9.5. *There is the approximate basis, called the **standard quantum basis***

$$Q = \{H, \sigma^x, K^{\pm 1}, \Lambda \sigma^x, \Lambda^2 \sigma^x\},$$

that generates a dense subgroup of $\mathcal{U}(\mathcal{B}^{\otimes 3})$.

10 October 17, 2018

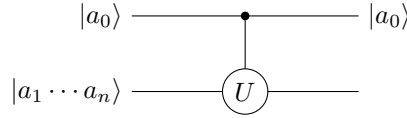
Recall that we set $\mathbb{B} = \{0, 1\}$ and $\mathcal{B} = \mathbb{C}^2$. A quantum gate is just a unitary transformation, and a quantum circuit is just the same as a boolean circuit but with qubit input and quantum gates. We saw the example of the Grover search algorithm, which is just finding an answer by quering the oracle. It turned out that only $\approx 2^{n/2}$ queries are needed to find the answer.

10.1 Circuit for the Grover search algorithm

Definition 10.1. Let U be a quantum gate, i.e., $U : \mathcal{B}^{\otimes n} \rightarrow \mathcal{B}^{\otimes n}$. Then we can define

$$\Lambda U : \mathcal{B}^{\otimes(n+1)} \rightarrow \mathcal{B}^{\otimes(n+1)}; \quad |a_0 a_1 \cdots a_n\rangle \mapsto \begin{cases} |a_0 a_1 \cdots a_n\rangle & \text{if } a_0 = 0, \\ |a_0\rangle \otimes U|a_1 \cdots a_n\rangle & \text{if } a_0 = 1. \end{cases}$$

We are going to use the circuit diagram



to represent this.

Now we can use this to build a quantum circuit for the Grover search algorithm. It suffices to build a quantum circuit for $V = I - 2|\xi\rangle\langle\xi|$, because the gate U is already given to us. But then, we just have $|\xi\rangle = H^{\otimes n}$. So it really suffices to build the circuit for

$$I - 2|0^n\rangle\langle 0^n|.$$

Define the function

$$F : \mathbb{B}^{n+1} \rightarrow \mathbb{B}^{n+1}; \quad (a_0, \dots, a_n) \mapsto (a_0 \oplus (\neg(a_1 \vee \cdots \vee a_n)), a_1, \dots, a_n).$$

Note that F is a permutation.

Proposition 10.2. Any permutation can be realized as a circuit over $\{\neg, \wedge_{\oplus}\}$ using ancillas.

Corollary 10.3. For any permutation F , its associated quantum gate \hat{F} can be realized as a circuit over $\{\hat{\neg}, \hat{\Lambda}_{\oplus}\}$ using ancillas.

Here, the quantum analogues $\hat{\neg}$ and $\hat{\Lambda}_{\oplus}$ are just σ_x and $\Lambda^2 \sigma_x$. So they are in our set of standard quantum gates.

Let this circuit (with ancillas) be Z . Now consider the circuit in Figure 1. Then if we feed in $|0, 0^n, 0^l\rangle$, we get

$$Z|0, 0^n, 0^l\rangle = |F(0, 0^n), G\rangle = |1, 0^n, G\rangle,$$

and then σ_z will turn it into

$$-|1, 0^n, G\rangle,$$

and so applying Z again will give

$$Z(-|1, 0^n, G\rangle) = -|0, 0^n, 0^l\rangle.$$

So this does exactly what we wanted it to do.

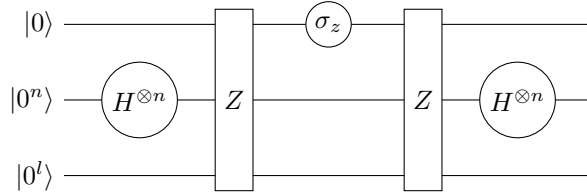


Figure 1: The quantum circuit for Grover's search algorithm

10.2 Quantum Fourier transform

Classical fast Fourier transform takes $O(n2^n)$, but the quantum Fourier transform only takes $O(n^2)$ time.

Definition 10.4. The **discrete Fourier transform** is the linear map

$$(x_0, \dots, x_{N-1}) \in \mathbb{C}^N \mapsto (y_0, \dots, y_{N-1}); \quad y_k = \frac{1}{\sqrt{N}} e^{2\pi i j k / N} x_j.$$

If we do this naïvely, we are doing multiplication and addition $O(N^2)$ times. In fast Fourier transform, we employ divide and conquer to bring it down to $O(N \log N)$ operations. If N is even, we divide

$$\sum e^{2\pi i j k / N} x_j = \sum e^{2\pi i (2m) k / N} x_{2m} + \sum e^{2\pi i (2m+1) k / N} x_{2m+1}.$$

So we don't have to compute them twice.

Definition 10.5. The **quantum Fourier transform** is defined as

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |k\rangle.$$

(Note that this is a unitary operator!) We interpret

$$|j\rangle = |j_1 j_2 \dots j_n\rangle$$

with the encryption $j = j_1 2^{n-1} + \dots + j_{n-2} 2 + j_n$. We can also write $0.j_1 j_2 \dots j_n = 2^{-n} j$.

We know that $\mathcal{B}^{\otimes n}$ has a standard orthonormal basis

$$\{|0\rangle, |1\rangle, \dots, |N-1\rangle\}$$

where $N = 2^n$.

Proposition 10.6. *Actually, the quantum Fourier transform is equivalent to*

$$|j\rangle \mapsto (|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n} |1\rangle) \otimes \dots \otimes (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle) \frac{1}{2^{n/2}}.$$

Proof. Note that $|j\rangle \mapsto \sum_k \omega^k |k\rangle$. But here,

$$\omega^k |k\rangle = \omega^{k_1 2^{n-1}} |k_1\rangle \otimes \dots \otimes \omega^{k_n} |k_n\rangle$$

and note that

$$\omega^{k_s 2^{n-s}} = \begin{cases} |0\rangle & k_s = 0 \\ (e^{2\pi i 0 \cdot j_{n-s+1} \dots j_n} |1\rangle) & k_s = 1. \end{cases}$$

Then the formula follows. \square

So how will we make this into a circuit? We have H , and we will use the gates

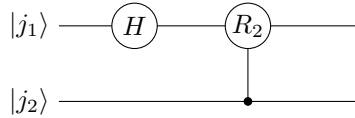
$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{pmatrix}$$

for each k . (If you really want to do over the standard gates, you approximate.)

First we note that we can conveniently write

$$H|j_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0 \cdot j_1} |1\rangle).$$

This is nice, because it is what we needed. For the next thing, we look at the following circuit:



Then we first get

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0 \cdot j_1} |1\rangle)|j_2\rangle$$

after H , and then we will get

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2} |1\rangle)|j_2\rangle.$$

Index

- bit masking, 13
- boolean circuit, 23
- clause, 9
- computation path, 5
- decidable set, 12
- decision problem, 3
- Diophantine, 11
- discrete Fourier transform, 26
- Hadamard gate, 22
- language, 3
- literal, 9
- non-deterministic Turing machine,
5
- NP-complete, 6
- NP-hard, 6
- Pell's equation, 17
- polynomial time, 5
- quantum circuit, 23
- quantum Fourier transform, 26
- recursively enumerable set, 12
- reduction, 6
- semi-decidable, 12
- standard quantum basis, 24
- time complexity, 4, 5
- truth assignment, 9
- Turing machine, 3
- universal Diophantine equation, 18
- universal Turing machine, 7