The method of proof that we used for showing that HALT is undecidable is called “reduction from $L_{TM}$”:

\[
L_{TM} \leq_M \text{HALT}
\]

can decide $\iff$ can decide

HALT is at least as hard as $L_{TM}$.

**Definition.** Function $f : \Sigma^* \to \Sigma^*$ is a computable function if some Turing machine $M$ on every input $w$ halts with just $f(w)$ on its tape.

**Definition.** Language $A$ is mapping reducible to language $B$, written $A \leq_M B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every $w$

\[
w \in A \iff f(w) \in B.
\]

The function $f$ is called the reduction from $A$ to $B$. 

![Diagram](attachment://diagram.png)
Theorem. If $L_A \leq_M L_B$ and $L_B$ is decidable, then $L_A$ is decidable too.

Proof. Let $M_B$ be a Turing machine that decides $L_B$ and $f$ be a reduction from $L_A$ to $L_B$. We describe a machine $M_A$ that decides $L_A$:

1. On input $w$ compute $f(w)$;
2. Run $M_B$ on $f(w)$ and output what $M_B$ outputs.

$M_A$ decides language $L_A$ indeed:

- If $w \in A$, then $f(w) \in B$, since $f$ is reduction. $M_B$ accepts $f(w)$ – therefore $M_A$ accepts $w$.
- If $w \notin A$, then $f(w) \notin B$. $M_B$ rejects $f(w)$ and therefore $M_A$ rejects $w$. \qed

Polynomiality

Definition. Let $M$ be a deterministic Turing machine that halts on all inputs. The running time (time complexity) of $M$ is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$.

- $f(n)$ is the running time of $M$.
- Usually $n$ is the length of the input.

Definition. Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) = O(g(n))$, if there are $C > 0$ and an integer $n_0$ such that for all $n > n_0$:

$$f(n) < C g(n).$$

Example 1.

- If $f(n) = n + 10$, $g(n) = n^2$, then $f(n) = O(g(n))$, because if $n > 3$, then $n + 10 < n^2$. So we can take $C = 1$ and $n_0 = 3$.
- If $f(n) = 5n^3 + 2n^2 + 7n + 10$ and $g(n) = n^3$, then $f(n) = O(g(n))$. We can take $C = 6$ or $n_0 = 4$, or we can take $C = 100$ and $n_0 = 0$. As you see, the choice of $C$ and $n_0$ is not unique.
- If $f(n) = 10n^2 + 100n + 10$ and $g(n) = 2^n$, then $f(n) = O(g(n))$.
- If $f(n) = \log_2 n$ and $g(n) = \sqrt{n}$, then $f(n) = O(g(n))$. 

2
Since \( \log_a n = \log_b b \cdot \log_b n \), then the basis of the logarithm is not important under \( O \)-notation (except exotic cases) and we simply write \( O(\log n) \).

**Definition.** Let \( f, g : \mathbb{N} \rightarrow \mathbb{R}^+ \). We say that \( f(n) = o(g(n)) \), if
\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.
\]

**Example 2.**
- \( \sqrt{n} = o(n) \)
- \( \log n = o(n^{\frac{1}{100}}) \)
- \( n \log n = o(n^2) \)
- \( n^{100} = o(2^n) \)

**Classes P and NP**

For the following discussion, we do not distinguish between different polynomial complexities. However, we distinguish between polynomial and exponential time complexities. Different reasonable deterministic computational models are polynomially equivalent.

**Definition.** \( P \) is a class of languages, whose time complexity is \( f(n) = O(p(n)) \), where \( p \) is some polynomial in \( n \).

\( P \) is invariant for all models of computation that are polynomially equivalent to the single-tape deterministic Turing machine. \( P \) roughly corresponds to the class of problems that are polynomially solvable on a computer.

**Example 3.** The following languages are in \( P \).

- Any regular language.
- Given an array, find whether it is monotonically non-decreasing.
- 
  \[ L_G = \{ \langle G, s, t, k \rangle \mid \text{shortest path in graph } G \text{ from } s \text{ to } t \text{ has length } k \} \].

**Definition.** \( NP \) is the class of languages which are decided by some non-deterministic polynomial-time Turing machine.
Example 4. A clique in a graph is a sub-graph, where every two nodes are connected by an edge. Then language

\[ \text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with clique of size } k \} \]

Let us show that CLIQUE \( \in \text{NP} \).

On input \( \langle G, k \rangle \), the nondeterministic Turing machine does the following:

1. Nondeterministically selects a subset \( S \) of \( k \) nodes in \( G \).
2. Checks whether \( G \) contains all the edges between pairs of nodes in \( S \).
3. If yes – accepts, if not – rejects.

Trivially, \( \text{P} \subseteq \text{NP} \). CLIQUE is an example of a problem, which is in \( \text{NP} \), but it is not known if it is in \( \text{P} \). It is know at all if \( \text{P} = \text{NP} \), in other words it is not know if there exists problem which is in \( \text{NP} \) but not in \( \text{P} \).

An \( \text{NP} \)-complete problem: if there exists a polynomial-time deterministic algorithm for that problem, then there exists a polynomial-time deterministic algorithm for any problem \( \text{NP} \) (i.e \( \text{P} = \text{NP} \)).
Practise session

1. Let

\[ \text{REGULAR} = \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) \text{ is a regular language} \} \]

Prove that REGULAR is undecidable.

Solution. We show reduction:

\[ L_{TM} \leq_M \text{REGULAR}. \]

Assume that REGULAR is decidable, and let \( M_R \) be a Turing machine that decides REGULAR. We construct Turing machine \( M_L \) that decides \( L_{TM} \). On the input \( \langle M, w \rangle \), \( M_L \) does the following:

1. Constructs machine \( M_0 \), which on input \( x \) does the following:
   - (a) if \( x \) is of the form \( 0^n1^n \) – accepts;
   - (b) if \( x \) is not of the form \( 0^n1^n \), run \( M \) on input \( w \) and accept if and only if \( M \) accepts.
2. Runs \( M_R \) on input \( \langle M_0 \rangle \).
3. If \( M_R \) accepts – accept, if \( M_R \) rejects – reject.

What is the language of \( M_0 \)?

- If \( M \) accepts \( w \), then \( L(M_0) = \Sigma^* \). This is regular language.
- If \( M \) does not accept \( w \), then \( L(M_0) = \{ 0^n1^n \mid n \geq 0 \} \). This is non-regular language.

Therefore:

- if \( M \) accepts \( w \) then \( L(M_0) = \Sigma^* \) and \( M_R \) accepts \( \langle M_0 \rangle \) in Step 2. Therefore \( M_L \) accepts.
- If \( M \) does not accept \( w \) then \( L(M_0) \) is irregular and \( M_R \) rejects \( \langle M_0 \rangle \) in Step 2. Therefore, \( M_L \) rejects.

So \( M_L \) accepts \( \langle M, w \rangle \) if and only if \( M \) accepts \( w \).

Note. All steps are computable by the Turing machines. In particular, constructing \( M_0 \) is possible: first \( M_0 \) checks for certain type of input and then simulates \( M \) on \( w \).

Conclusion. We found that if there exists \( M_R \) (the machine that decides REGULAR), then there exists \( M_L \) (the machine that decides \( L_{TM} \)). Not possible. Contradiction!
2. True or false?

(a) \( n = o(2n) \)

(b) \( 3n^5 = O(10n^3 + 20n^2 + 100) \)

(c) \( 2^n = o(3^n) \)

(d) \( n^2 = O(n \log n) \)

Solution.

(a) False: \( \lim_{n \to \infty} \frac{n}{2n} = \frac{1}{2} \neq 0 \).

(b) False: for any \( n_0 \) and \( C > 0 \) there exists arbitrary large \( n \geq \max\{n_0 + 1, 1000C\} \), such that \( n > n_0 \) and \( 3n^5 > C(10n^3 + 20n^2 + 100) \).

(c) True: \( \lim_{n \to \infty} \frac{n}{2^n} = \lim_{n \to \infty} \left(\frac{2}{3}\right)^n = 0 \).

(d) False: since \( \lim_{n \to \infty} \frac{n^2}{n \log n} = \lim_{n \to \infty} \frac{n}{\log n} = \infty \), then for any constant \( C > 0 \) there exists \( n \) such that \( f(n) > C g(n) \).

3. Let \( t(n) \) be a function, where \( t(n) \geq n \). Show that every \( t(n) \)-time multitape Turing machine has an equivalent \( O(t^2(n)) \)-time single-tape machine.

Solution. We saw earlier in the course how to convert multitape machine into an equivalent single-tape machine. Now, we show that simulating each step of the multitape machine takes at most \( O(t(n)) \) steps of a single-tape machine.

Recall that initially the single-tape machine \( M_S \) puts on its tape the content of all tapes of the multi-tape machine \( M_M \). To perform one step, \( M_S \) scans all the tape content to determine the symbols under the heads of \( M_M \). Then, \( M_S \) makes another pass to write the new tape content. If one of the heads of \( M_M \) moves rightwards from the last non-blank symbol, \( M_S \) should shift the content of the tape one position to the right.

- What is the maximal number of steps for one scan? Since \( M_M \) makes \( O(t(n)) \) steps in total, the total length of the active part of the tape of \( M_S \) if of length \( O(t(n)) \). Hence each scan of the tape by \( M_S \) takes \( O(t(n)) \) time.

- To simulate each step of \( M_M \), \( M_S \) performs two scans and possibly one shift to the right. Each such operation (scan/shift) takes at most \( O(t(n)) \).
The total time needed for simulation of $M_M$ by $M_S$:

- initialisation of the tape: $O(t(n))$;
- simulation of each of $t(n)$ steps of $M_M$ by $M_S$: $t(n) \cdot O(t(n)) = O(t^2(n))$.

Total time: $O(t^2(n)) + O(t(n))$.

Since $O(t(n)) \geq O(n)$ (otherwise, $M_M$ cannot even read all its input), we obtain that the total time complexity is $O(t^2(n))$.

4. Define the language:

$$PATH = \{(G, s, t) \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}.$$  

For example:

```
  o---o   o
 |   |   /|
 |   |  / \\
 s   o --- o
     |   |   \
     o---o---o
      |   |   |
      |   |   |
      v   v   v
       t
```

Prove that $PATH \in P$.

**Solution.** Consider the following algorithm $M$ for $PATH$, which on input $(G, s, t)$ does the following:

1. Marks node $s$.
2. Repeats the following:
   - scans the edges $(u, v)$ in graph $G$; if $u$ is marked and $v$ is not marked – mark $v$.
3. If $t$ is marked – accept, otherwise – reject.

**Correctness.** If there is a path $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \ldots \rightarrow v_{l-1} \rightarrow v_l = t$, then by induction on $i$, the node $v_i$ will be marked. If there is no path from $s$ to $t$, the there is no way to mark $t$.

**Complexity.** Step 1 takes polynomial time. Let $m$ be a number of nodes in $G$. Then Step 3 is executed at most $m$ times (because each time we mark at least one node). Each of the steps 1–4 requires polynomial time complexity. Therefore, the total complexity is polynomial.