Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 2: Turing Machines, Decision Problems, Languages

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Modelling computation
- Turing machines
- Formal languages and decision problems
- Time and space complexity, complexity class P

1 Modelling Computation

- **To discuss what *can* and *cannot* be done with computation, we must define what computation is**
- **The choice of *model* is important and delicate...**
  - We must capture all possible computation: *abacuses*, *sliderules*, *modern computers*, *future computers*, *computation in nature*, ...
  - We must also capture the notion of *computational efficiency* in a robust and universal way
- **... but does not really matter:**
  - **Formally:** all known models are *equivalent*
  - **Informally:** any sensible model can simulate all computation (*Church-Turing thesis*)
  - A lot of computational complexity can be understood without formally discussing the model

History (1/2)

- **The nature of computation was studied already before the existence of the modern computer**
  - Motivation: foundations of mathematics
- ***Computation*** **and** ***algorithm*** **were understood as mechanical rules for manipulating numbers**
  - Muhammad ibn Musa al-Khwarizmi: one of the first published algorithms, origin of the word 'algorithm'
  - Charles Babbage and Ada Lovelace: first mechanical computer and first algorithm intended for a machine

History (2/2)

- **Around 1930s, many models proposed:**
  - Kurt Gödel: *recursive functions*
  - Alonzo Church: *lambda calculus*
  - Emil Post: *rewriting systems*
  - Alan Turing: *Turing machines*
  - All of these are **equivalent** – can simulate each other

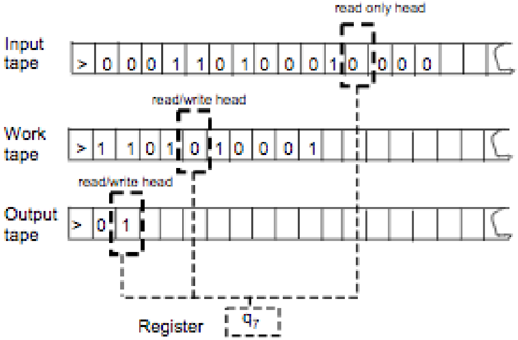
Gödel

Post

Church

Turing

2 Turing Machines



Informal introduction (1/3)

- **A *Turing machine* is an abstract machine:**
  - It has a register with finite number of possible *states*
  - It has k *tapes* that serve as the memory
  - Each tape has infinitely many *cells* that can store a symbol
  - First tape is a read-only input tape, last tape is output tape
  - Each tape has a read/write *head*
  - The head is is positioned over a single cell

Informal Introduction (2/3)

- **Computation is performed step-by-step**
- **First, Turing machine reads its current *configuration*:**
  - The current state
  - The symbols in cells below each head
- **Based on the read information and a *program*, move to a new configuration:**
  - Change to a new state
  - Write a new symbol below each head
  - Move each head left or right (or keep it in place)

Informal Introduction (3/3)

- **Initialising computation:**
  - We write the input on the first tape
  - We initialise other tapes with special blank symbol $\square$
- **Perform a sequence of steps until computation stops**
  - Special *halting state*
- **Read the output from the output tape**

Formal Definition

Definition (Turing Machine, TM)

A Turing machine M is a tuple (Γ, Q, δ), consisting of:

- A finite set Γ of symbols, called the *alphabet* of M. We assume Γ contains symbols 0 and 1 and special symbols
  - $\square$ (*blank*), and
  - $\triangleright$ (*tape end*)
- A finite set Q of *states* of M. We assume Q contains
  - a special state q_0 called the *start state*, and
  - a special state q_h called the *halting state*
- A function $\delta \colon Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$, where $k \geq 2$, called the *transition function* of M

Configurations

- **The contents of each tape is an finite string:**
  - Cells numbered $0, 1, 2, \ldots$ (0 is the leftmost position)
  - Cell number 0 always holds the symbol $\triangleright$
  - Other cells initialised to $\square$: only a finite number of other symbols during a finite execution

- **A *configuration* of Turing machine M is defined by:**
  - The current state $q \in Q$
  - The contents of the tapes (finite strings)
  - Positions of the heads on tapes (natural numbers)
  - **Formally:** configuration is an element $c \in Q \times (\Gamma^*)^k \times \mathbb{N}^k$

Transition Function

- **The transition function is the ‘program’ of TM**

- Let TM be in a configuration with
  - state $q \in Q$
  - symbols $a = (a_1, a_2, \ldots, a_k)$ under the k heads

- For transition function $\delta(q, a) = (p, b, D)$, where
  - $b = (b_1, b_2, \ldots, b_{k-1}) \in \Gamma^{k-1}$, and
  - $D = \{D_1, D_2, \ldots, D_k\} \in \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$

  the machine moves to a new state such that
  - the new state will be $p \in Q$,
  - TM writes symbols $b_1, b_2, \ldots, b_{k-1}$ on tapes $2, 3, \ldots, k$, and
  - TM moves head on tape i according to D_i
    (L = left, S = stay in place, R = right)

Transition Function

- **Convenient to add additional requirements for the transition function δ**
- **Does not move from halting state**
  - If TM is in state q_{h}, don't change configuration
- **Always moves right on $\triangleright$**
  - If there is symbol $\triangleright$ on tape i, then the move command for tape i is R
- For convenience, we may assume that the transition function only writes symbols 0 and 1 on the output tape

Starting and Halting

- **On input $x = x_1 x_2 \dots x_k \in \{0,1\}^*$, a TM M *starts*:**
  - With initial state $q_0 \in Q$
  - With tape 1 initialised with $\triangleright x_1 x_2 \dots x_k$
  - With other tapes empty (i.e., initialised with $\triangleright \Box\Box \dots$)
  - With all heads in position 1
- **TM M *halts* with output $y = y_1 y_2 \dots y_k \in \{0,1\}^*$ if;**
  - M is in state q_{h}
  - The content of output tape is $\triangleright y_1 y_2 \dots y_k$

Execution

- ***Execution*** **of a TM M on input x is the sequence of configurations:**
  - Begins with the starting configuration on input x
  - Subsequent configurations obtained via the transition function
  - Execution ends when the machine reaches a configuration with the halting state q_{h} (execution *halts*), or the execution is infinite (execution does not halt or *diverges*)

- **Output of machine M:**
  - If M halts on input x with output y, we write $M(x) = y$ to denote the *output* of M in input x
  - Alternatively, the machine may not halt

Computing a Function

Definition (Computing a function)

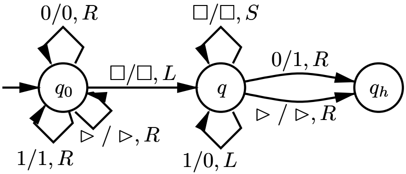
Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a function. We say that a Turing machine M *computes the function* f if M halts on all inputs and $M(x) = f(x)$ for all $x \in \{0,1\}^*$.

Diagram Notation for Transition Function

Consider a *single-tape* Turing machine $M_{inc} = (\Gamma, Q, \delta)$ with $\Gamma = \{0, 1, \square, \triangleright\}$ and $Q = \{q_0, q, q_h\}$, and the following transition function δ:

$t \in Q$	$a \in \Gamma$	$\delta(q, a)$
q_0	0	$(q_0, 0, R)$
q_0	1	$(q_0, 1, R)$
q_0	$\square$	$(q, \square, L)$
q_0	$\triangleright$	$(q_0, \triangleright, R)$
q	0	$(q_h, 1, S)$
q	1	$(q, 0, L)$
q	$\square$	$(q, \square, S)$
q	$\triangleright$	$(q_h, \triangleright, R)$

Diagram representation:



The machine is designed to compute the successor $n + 1$ of a natural number n given *in binary*.[1]

[1] Actually, the machine design is not quite correct. Can you spot the error?

Example: Palindromes

- **Define a function $P\colon \{0,1\}^* \to \{0,1\}$ as follows:**
  - $P(x) = 1$ if x is a *palindrome*, that is, if x read from the right is the same as x read from the left
  - $P(x) = 0$ otherwise
- **Basic idea for a Turing machine that computes P:**
  - Copy input to a working tape
  - Move the input tape head to the beginning of the input
  - Move input tape head to the right and working tape head to the left. If the heads see different symbols at any point, write 0 on the output tape and halt.
  - If working tape head reaches $\triangleright$, write 1 on the output tape and halt.

3 Formal Languages and Decision Problems

Basic concepts and notation:

- **Alphabet:** a finite set of symbols S
- **String:** a finite sequence $x = x_1x_2 \dots x_k$ of symbols from S
  - Example: $S = \{0,1\}$, $x = 0101101$
  - ε = empty string of length 0
- **(Formal) Language:** a set of strings over some alphabet S

- $|x|$ = length of string x
- x^k = x concatenated with itself k times
  - Example: $1^7 = 1111111$, $(01)^3 = 010101$

- S^k = all strings of length k over alphabet S
- S^* = all strings over alphabet S
  - Example: $\{0,1\}^*$ = all binary strings

Representing Decision Problems

- **For technical convenience, we focus on *decision problems***
- A decision problem for property P asks if for any given instance x, instance x has property P.
- If the instances of a decision problem P are encoded as strings over some alphabet S, then P can be represented by its characteristic funtion $f_P \colon S^* \to \{0,1\}$ (1 $\sim$ "yes", 0 $\sim$ "no")
- Equivalently, the problem can be represented by its set of "yes"-instances, i.e. the language

$$L_P = \{x \in S^* \mid f_P(x) = 1\}.$$

- Because of this equivalence, the terms "language" and "decision problem" are used interchangeably.

Deciding Languages by Turing Machines

Definition

Turing machine M *decides* a language L if:

- for any $x \in L$, $M(x) = 1$
- for any $x \notin L$, $M(x) = 0$

Turing machine M *accepts* (*semidecides*) a language L if:

- for any $x \in L$, $M(x) = 1$
- for any $x \notin L$, M does not halt on input x

- If a language (problem) is decided by some Turing machine, it is called *decidable*. (Or historically “recursive”.)
- If a language (problem) is accepted by some Turing machine, it is called *semidecidable*. (Or historically “recursively enumerable”.)
- A language (problem) which is not decidable is *undecidable*. An undecidable language (problem) may still be semidecidable.

4 Time and Space Complexity, Complexity Class P

Definition (Running time)

Let M be a Turing machine that halts on all inputs. We say that M runs in time $T(n)$ if for all inputs $\{0,1\}^*$, the machine M halts after at most $T(|x|)$ steps.

Definition (Space usage)

Let M be a Turing machine that halts on all inputs. We say that M uses $S(n)$ space if for all inputs $\{0,1\}^*$, the machine M visits at most $S(|x|)$ cells on the non-input tapes of M.

Robustness of the Definitions

- **Time and space complexity should not significantly depend on:**
  - Number of tapes
  - Size of the alphabet
- **We will outline proofs that this is the case**
  - Focus on *time*
  - Similar things hold for *space*

Time-Constructible Functions

Definition (Time-constructible function)

Let $T\colon \mathbb{N} \to \mathbb{N}$ be a function. We say that T is *time-constructible* if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$, where $\llcorner n \lrcorner$ denotes the binary representation of the number n.

- **Roughly:** $T(n)$ can be computed in time $T(n)$
- Essentially all sensible functions we care about are time-constructible, so this is mostly a technicality
- Needed for certain proofs to go through

Alphabet Size Does Not Matter

Theorem

Let $f : \{0,1\}^* \to \{0,1\}^*$*, and let* $T : \mathbb{N} \to \mathbb{N}$ *be time-constructible. If* f *can be computed by a TM* M *with alphabet* Γ *in time* $T(n)$*, then there is a TM* M' *that computes* f *using alphabet* $\{0,1,\triangleright,\square\}$ *in time* $4\log_2|\Gamma|T(n)$.

- **Basic idea:** encode the alphabet Γ in binary using $\log_2|\Gamma|$ bits

Number of Tapes Does Not Matter

- **Alternative definition:** ***single-tape Turing machines***
  - Only single read/write tape
  - Input is written on the tape at start
  - End computation with output written on the tape

Theorem

Let $f: \{0,1\}^* \to \{0,1\}^*$, and let $T: \mathbb{N} \to \mathbb{N}$ be time-constructible. If f can be computed by a TM M with k tapes in time $T(n)$, then there is a single-tape TM M' that computes f in time $5kT(n)^2$.

- **Basic idea:**
  - Encode ith tape in position $i, i+k, i+2k, \ldots$
  - Use special symbols to denote head positions
  - Each pass: read the whole tape, update heads and writes

Time Complexity Classes

Definition (Class DTIME)

Let $T\colon \mathbb{N} \to \mathbb{N}$ be a function. The class DTIME$(T(n))$ is the set of languages L for which there exists a Turing machine M and a constant $c > 0$ such that M decides L and runs in time $c \cdot T(n)$.

- Note the constant "slack factor" c. This is to simplify proofs, and also because the multi-tape Turing machine model is robust w.r.t. almost all changes in detail up to a constant factor. (However going from $k \geq 2$ tapes to a single tape induces a quadratic overhead.)

Polynomial Time

Definition (Class P)

$$\mathsf{P} = \bigcup_{d=1}^{\infty} \mathsf{DTIME}(n^d)$$

- **In other words,** P **is the class of all languages that can be decided by a polynomial-time Turing machine**
  - Alphabet size and number of tapes do not matter

Polynomial Time: Discussion

- **Strong Church-Turing Thesis:** any physically realisable system can be simulated by a Turing machine with polynomial overhead
  - Implies that P captures everything computable in polynomial time
  - **Not entirely uncontroversial:** randomised algorithms, quantum algorithms, exotic physics...

Polynomial Time: Discussion

- **Class** P **tries to model *tractable* problems**
  - $O(n^{100})$ is polynomial, but not practical
  - $O(n^3)$ is not really practical either!
  - $O\left(2^{n/100}\right)$ is often practical, but not polynomial
- What about average-case complexity, approximations and randomisation?

Polynomial Time: Discussion

- **Turing machines not great for discussing differences between polynomial-time problems**
  - e.g. $\Theta(n^2)$ vs. $\Theta(n^3)$
  - Polynomial overheads from moving heads, simulation
  - Need more *fine-grained* models for this!
- **Still makes sense to study** P**:**
  - Many real-world problems are outside P
  - Understanding this source of difficulty is useful

Lecture 2: Summary

- Turing machines
- Decision problems and (un)decidability
- Running time and space usage
- Classes DTIME$(T(n))$ and P

`https://www.youtube.com/watch?v=cYw2ewoO6c4`