

CS-E4530

Computational Complexity Theory

Janne H. Korhonen

Aalto University

Spring 2018

Lecture 2:

Turing Machines

Modelling Computation

What Is Computation?

- To discuss what **can** and **cannot** be done with computation, we must define what computation is
- The choice of **model** 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

What Is Computation?

- The choice of **model** 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

- **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

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



Post



Church



Turing

Turing Machines

Informal Introduction

- **Want to compute a function** $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$
- **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 positioned over a single cell

Informal Introduction

- **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

- **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 $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, where $k \geq 2$, called the **transition function** of M .

Configurations

- **The contents of each tape is an finite string:**
 - Cells numbered $0, 1, 2, \dots$ (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 $a = (a_1, a_2, \dots, a_k)$ under the k heads
- For transition function $\delta(q, a) = (p, b, D)$, where
 - $b = (b_1, b_2, \dots, b_{k-1}) \in \Gamma^{k-1}$, and
 - $D = \{D_1, D_2, \dots, D_k\} \in \{L, S, R\}^k$

the machine moves to a new state such that

- the new state will be $p \in Q$,
- TM writes symbols b_1, b_2, \dots, b_{k-1} on tapes 2, 3, \dots , 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** $x = x_1x_2 \dots x_k \in \{0, 1\}^*$, **a TM M starts:**
 - With initial state $q_0 \in Q$
 - With tape 1 initialised with $\triangleright x_1x_2 \dots x_k$
 - With other tapes empty (i.e., initialised with $\triangleright \square\square \dots$)
 - With all heads in position 1
- **TM M halts with output** $y = y_1y_2 \dots y_k \in \{0, 1\}^*$ **if;**
 - M is in state q_h
 - The content of output tape is $\triangleright y_1y_2 \dots y_k$

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 $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ 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 $x \in \{0, 1\}^*$.

Example: Palindromes

- **Define a function $P: \{0, 1\}^* \rightarrow \{0, 1\}$ 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

Example: Palindromes

- **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.

Time and Space

Time and Space

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 $T: \mathbb{N} \rightarrow \mathbb{N}$ 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 $x \mapsto \lfloor T(|x|) \rfloor$ in time $T(n)$, where $\lfloor T(n) \rfloor$ denotes the binary representation of the number $T(n)$.

- **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 $f: \{0, 1\}^ \rightarrow \{0, 1\}^*$, and let $T: \mathbb{N} \rightarrow \mathbb{N}$ 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

Number of Tapes Does Not Matter

Theorem

Let $f: \{0, 1\}^ \rightarrow \{0, 1\}^*$, and let $T: \mathbb{N} \rightarrow \mathbb{N}$ 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 i th tape in position $i, i + k, i + 2k, \dots$
- Use special symbols to denote head positions
- Each pass: read the whole tape, update heads and writes

Turing Machines and Languages

Decision Problems

- **For technical convenience, are going to focus on studying decision problems or languages**
 - Decision problem: a function $f_L: \{0, 1\}^* \rightarrow \{0, 1\}$
 - Equivalently, a subset $L \subseteq \{0, 1\}^*$ via the correspondence

$$f_L(x) = 1 \Leftrightarrow x \in L$$

- **Think as yes-no questions**

Accepting and Deciding

Definition (Accepting a language)

Turing machine M **accepts** a language L if:

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

Definition (Deciding a language)

Turing machine M **decides** a language L if:

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

Time Complexity Classes

Definition (Class DTIME)

Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be a function. The class $\text{DTIME}(T(n))$ 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 $c \cdot T(n)$.

Polynomial Time

Definition (Class P)

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

- **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!
 - $O(2^{n/100})$ 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
- Running time and space usage
- Decision problems
- Classes $\text{DTIME}(T(n))$ and P