Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 3: Representations, Universality, Undecidability

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Decision problems
- Instance (= input) representations
- Turing machine representations
- The universal Turing machine
- Undecidability

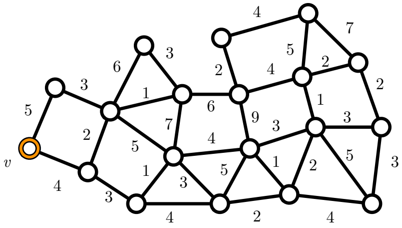
Decision Problems

- **Recall our definition of *decision problems*:**
  - Decision problem $\sim$ language $L \subseteq \{0,1\}^*$
- **We model all computational tasks as decision problems:**
  - How to handle *optimisation problems*?
  - How to handle non-binary string inputs, like graphs?

Decision Problems: Example

Travelling Salesman Problem (Decision Version)

- **Instance:** Graph $G = (V, E)$ with positive edge weights, integer $W \geq 0$, a vertex $v \in V$.
- **Question:** Is there a tour starting from vertex v that visits all other vertices exactly once and then returns to v with weight at most W?



Representations

- **For general inputs:**
  - Encode all inputs as binary
  - Just like we actually do with computers
- **More formally:**
  - Define an encoding function that maps instance x into a binary string $\llcorner x \lrcorner$

Representations: Numbers

- **Numbers are represented in binary**
  - $\llcorner n \lrcorner$ is the binary representation of n
  - Leading zeros can be ignored

$$\llcorner 1 \lrcorner = 1$$

$$\llcorner 2 \lrcorner = 10$$

$$\llcorner 3 \lrcorner = 11$$

$$\llcorner 10 \lrcorner = 1010$$

$$\llcorner 1203 \lrcorner = 10010110011$$

Representations: Non-binary strings

- **Encoding strings over non-binary alphabet Γ:**
  - Encode each symbol using $\lceil \log_2 |\Gamma| \rceil$ bits
  - Encode strings by concatenating the binary representations

- **Example:** $\Gamma = \{a, b, c, d\}$, $\lceil \log_2 |\Gamma| \rceil = 2$

$$\llcorner a \lrcorner = 00 \qquad \llcorner b \lrcorner = 01$$
$$\llcorner c \lrcorner = 10 \qquad \llcorner d \lrcorner = 11$$

$$\llcorner ababcd \lrcorner = 00\,01\,00\,01\,10\,11 = 000100011011$$

Representations: Pairs and tuples

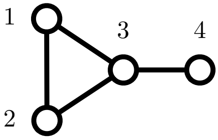
- **Encoding pairs of objects:**
  - Assume we already have a encoding function $\llcorner \cdot \lrcorner$ for objects x and y using alphabet Γ
  - Let # be a symbol not in Γ
  - **Pairs:** encode (x, y) as $\llcorner x \lrcorner \# \llcorner y \lrcorner$
  - **Tuples:** encode $(x_1, x_2, \ldots, x_k)$ as $\llcorner x_1 \lrcorner \# \llcorner x_2 \lrcorner \# \cdots \# \llcorner x_k \lrcorner$
  - Encode the resulting string in binary
- Apply recursively for nested pairs and tuples

Representations: Graphs

- **Convenient to assume:** vertex set is $V = \{1, 2, \ldots, n\}$
- **Two common encoding schemes for graphs:**
  - *Adjacency lists*
  - *Adjacency matrices*

Representations: Adjacency Lists

- **Adjacency list representation:**
  - For each v, list the neighbours of v
  - List all the adjacency lists
  - Encode using the tuple encoding



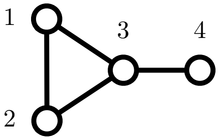
$$\begin{aligned}(\quad &(1,(2,3)),\\ &(2,(1,3)),\\ &(3,(1,2,4)),\\ &(4,(3)) \quad)\end{aligned}$$

Representations: Adjacency Matrices

- **Adjacency matrix representation of $G = (V, E)$:**
  - Matrix M_G such that

$$M_G(v,u) = \begin{cases} 1 & \text{if } v \neq u \text{ and } v \text{ and } u \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

- **Encode the matrix as a string:**
  - Example: $\llcorner G \lrcorner = 0110\#1010\#1101\#0010$



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency Lists vs. Adjacency Matrices

- **Graph $G = (V, E)$ with n vertices and m edges**
  - **Adjacency list encoding:** $O(n + m \log n)$ bits
  - **Adjacency matrix encoding:** $O(n^2)$ bits

- **Representations can be extended to handle directed graphs and weighted graphs**

- **Equivalent in terms of polynomial-time algorithms**
  - Can convert from one to the others in polynomial time
  - However, can matter in other settings for *sparse graphs* (meaning $m = o(n^2)$)

Representations in Practice

- **We assume that representations are ‘reasonable’:**
  - Encoding is injective, i.e. one-to-one
  - Conversion between two reasonable representations can be done in polynomial time
  - We can decide in polynomial time if a given string $x \in \{0,1\}^*$ represents a valid object
- **We assume encoding happens in the background:**
  - We don’t distinguish between the input and its encoding
  - For non-encoding strings, output 0

Decision Problems: Example

Travelling Salesman Problem (Decision Version)

- **Instance:** Graph $G = (V, E)$ with positive edge weights, a vertex $v \in V$, and an integer $W \geq 0$, .
- **Question:** Is there there a tour starting from vertex v that visits all other vertices exactly once and then returns to v with weight at most W?

- Input is an encoding of a tuple (G, v, W), where G is a weighted graph, v is an integer (i.e. a vertex), and W is an integer
- If the encoding is not valid, output 0
- Otherwise, output is 1 or 0 depending on the instance

Representations: Turing Machines

- **Turing machines are finite objects, and we can obviously represent them as binary strings**

- **Concretely:**
  - Map the alphabet and the state space to integers
  - Turing machine is a tuple $M = (\Gamma, Q, \delta)$
  - Γ can be interpreted as a tuple of integers
  - Q can be interpreted as a tuple of integers
  - Each entry in δ can be interpreted as a tuple, and δ itself can be interpreted as a tuple

- **Apply encoding for tuples**

Representations: Turing Machines

- **Convenient to tweak the semantics so that we have certain nice properties**
- **Each TM is represented by *infinitely many strings***
  - Allow ‘empty symbols’ at the end of the representation
- **Each string represents *some Turing machine***
  - Non-valid encodings are mapped to a single TM
  - E.g. a TM that always halts immediately
- **Notation:** M_α = Turing machine represented by string $\alpha \in \{0,1\}^*$

Turing Machines as Data

- **Simple, yet important consequences of previous:**
  - Turing machines ($\sim$ programs) can be treated as data
  - One can define computational problems that refer to Turing machines
  - The set $\mathcal{M}$ of all Turing machines can be *enumerated*:
    - $\mathcal{M} = \{M_\alpha \mid \alpha \in \{0,1\}^*\}$, or
    - $\mathcal{M} = \{M_1, M_2, \dots\}$, via the correspondence $\alpha \sim$ number represented by binary string 1α

Universal Turing Machine: The Idea

- Since Turing machines can be treated as data, one can have Turing machines simulate other Turing machines provided as input

- **Actually, there is a *universal Turing machine* $\mathcal{U}$:**
  - Input: an encoding α of a Turing machine $M = M_\alpha$ and a string x
  - $\mathcal{U}$ simulates M on input x and produces output $M(x)$
  - Moreover, one can make this simulation *efficient*

- **Hence, a single Turing machine captures *all computation***

- In modern terms, $\mathcal{U}$ is an *interpreter* for the TM programming language, written in the same language

Universal Turing Machine: The Theorem

Theorem

There is a Turing machine $\mathcal{U}$ such that for every $\alpha, x \in \{0,1\}^*$,

- *if M_α halts on input x, then $\mathcal{U}\big((\alpha,x)\big) = M_\alpha(x)$, and*
- *if M_α does not halt on input x, then $\mathcal{U}$ does not halt on (α,x).*

Moreover, if M_α halts on input x in T steps, then $\mathcal{U}$ halts on input (α,x) in CT^2 steps, where C is a constant that only depends on M_α.

Universal Turing Machine: Proof Idea

- **Turing machine $\mathcal{U}$ has as inputs:**
  - string $\alpha \in \{0,1\}^*$, representing a k-tape TM M_α
  - string $x \in \{0,1\}^*$, the intended input for M_α

- **Basic construction for $\mathcal{U}$:**
  - **Simulated input tape:** simulates the input tape of M_α
  - **Machine tape:** stores the representation of M_α
  - **State tape:** stores the current state of M_α
  - **Simulation tape:** simulates *all* worktapes of M_α
  - Output tape of $\mathcal{U}$ simulates the output tape of M_α

Universal Turing Machine: Proof Idea

- **Simulation of the working tapes:**
  - Using the same tricks as last lecture
  - In interleaved positions, store full contents of all working tapes of M_α in binary
  - Use special marking characters to indicate which positions hold the heads of M_α

Universal Turing Machine: Proof Idea

- **Setup:**
  - Copy the representation of M_α and x to the corresponding tapes
  - Set the current state of M_α to starting state
- **Simulation step:**
  - Scan the simulation tape and store the symbols under head to the state tape
  - Scan the representation of M_α to find a transition corresponding to the current configuration of M_α, write down the written symbols and head movements
  - Pass over simulation tape, apply changes

Universal Turing Machine: Proof Idea

- **Time complexity:**
  - Assume M_α runs for T steps on input x
  - Any tape of M_α can have at most T symbols on it
  - Each simulation step takes at most CT steps for some constant C
  - At most T simulation steps
  - Total CT^2, C subsumes constant factors from setup

Universal Turing Machine (Strong Version)

Theorem

There is a TM $\mathcal{U}$ such that for every $\alpha, x \in \{0,1\}^*$,

- *if M_α halts on input x, then $\mathcal{U}\big((\alpha, x)\big) = M_\alpha(x)$, and*
- *if M_α does not halt on input x, then $\mathcal{U}$ does not halt on (α, x).*

Moreover, if M_α halts on input x in T steps, then $\mathcal{U}$ halts on input (α, x) in $CT \log T$ steps, where C is a constant that only depends on M_α.

- **Proof:** complicated.

Undecidability: A Simple Counting Argument

- **For any language L, is there a Turing machine that *decides*, or more weakly *accepts* L?**
  - For definiteness, let us consider languages and Turing machines over the binary alphabet $\{0,1\}$
  - Let $M_1, M_2, \ldots$ be the enumeration of all Turing machines described earlier
  - Denote $L_i =$ language accepted by machine M_i
  - This gives an enumeration of all TM-acceptable (binary) languages $L_1, L_2, \ldots$
  - However we know that the family $\mathcal{L}$ of *all* (binary) languages cannot be thus enumerated (cf. tutorial problem T1.2)
  - Hence there exists a language $L \in \mathcal{L}$ that does not appear in the enumeration $L_1, L_2, \ldots$
  - In summary: there are only countably many Turing machines, but uncountably many languages; thus, there are not enough Turing machines for even *accepting* every language
- **What about *concrete examples* of undecidable languages?**

The Diagonal Language

Definition

The *diagonal function* $f_D \colon \{0,1\}^* \to \{0,1\}$ is defined as

$$f_D(\alpha) = \begin{cases} 0 & \text{if } M_\alpha(\alpha) = 1 \text{, and} \\ 1 & \text{otherwise.} \end{cases}$$

- **The corresponding language is the *diagonal language***

$$D = \{\alpha \mid f_D(\alpha) = 1\} = \{\alpha \mid M_\alpha(\alpha) \neq 1\}$$

- Note that here the condition $M_\alpha(\alpha) \neq 1$ includes the possibility that M_α does not halt on input α, denoted $M_\alpha(\alpha)\uparrow$.

Undecidability of D

Theorem

The diagonal language D is undecidable.

- **Proof:**
  - Assume D is decidable
  - Then there exists a TM M such that for all $\alpha \in \{0,1\}^*$, $M(\alpha) = f_D(\alpha)$
  - In particular, $M(\llcorner M \lrcorner) = f_D(\llcorner M \lrcorner)$
  - This is a *contradiction*: by definition of D,
    - $M(\llcorner M \lrcorner) = 1$ implies $f_D(\llcorner M \lrcorner) = 0$,
    - $M(\llcorner M \lrcorner) = 0$ implies $f_D(\llcorner M \lrcorner) = 1$

The Halting Problem

Definition

The *halting function* f_{HALT} is defined as

$$f_{\mathsf{HALT}}((\alpha, x)) = \begin{cases} 1 & \text{if } M_\alpha \text{ halts on input } x \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

- **The corresponding language is the *halting problem***

$$\mathsf{HALT} = \{(\alpha, x) \mid M_\alpha \text{ halts on input } x\}$$

The Halting Problem

Theorem

The halting problem is undecidable.

- **The proof is by a *reduction* argument:**
  - We show how to effectively transform any instance of the diagonal problem into a "corresponding" instance of the halting problem
  - Then, if we could decide the halting problem, we could also decide the diagonal language, which we know is impossible
  - This shows that in some sense the halting problem is *more difficult* than the diagonal problem

Proof: Halting Problem Is Undecidable

- **Recall that $\alpha \in D$ iff either $M_\alpha(\alpha) \neq 1$ (properly) or $M_\alpha(\alpha)\uparrow$**
- **Assume there is a Turing machine M_H that decides the halting problem**
- **Then we can decide the diagonal language as follows:**
  - On input $\alpha \in \{0,1\}^*$, simulate M_H on instance (α, α)
  - If $M_H(\alpha,\alpha) = 0$, i.e. $M_\alpha(\alpha)\uparrow$:
    - Output 1
  - If $M_H(\alpha,\alpha) = 1$, i.e. $M_\alpha(\alpha)\downarrow$:
    - Use the UTM $\mathcal{U}$ to compute $M_\alpha(\alpha)$
    - If $M_\alpha(\alpha) = 1$ then output 0, otherwise output 1

Implications of Undecidability

- **Halting problem is relevant in practice**
  - Implication: one cannot check programmatically that programs function correctly
  - Specifically, one cannot check for *infinite loops*

- **More generally: *Rice's theorem***
  - All *semantic properties* ot Turing machines, i.e. properties that concern only their input/output characteristics, are undecidable

- **For example:**
  - Does TM M on input x produce output y?
  - Does TM M on some input produce output 0?
  - Does TM M halt on all inputs?
  - Does TM M halt on some input?

Lecture 3: Summary

- Encoding objects as binary strings
- Encoding Turing machines as binary strings
- The universal Turing machine
- Existence of undecidable problems
- Halting problem is undecidable