CSE3305 Formal Methods II

Lecture 4: Introduction to Computational Complexity Theory

Dr. Kevin Korb
School of Info Tech
and Software Eng.
Building 75 (STRIP), Rm 121
kevin.korb@infotech.monash.edu.au
Contents

• The Complexity Hierarchy
• Polynomial Time
• Decision Problems
• P and NP
• Nondeterminism
• Computability and Noncomputability
• Noncomputable Numbers

Recommended Reading:
Dewdney *The New Turing Omnibus*
Neapolitan and Naimipour, Chap 9
Wikipedia
The Complexity Hierarchy

- **Unsolvable, Noncomputable**: infinite-time problems
- **EXP**: exponential-time problems
- **NP-Hard**: problems which are “NP general”
- **NP-Complete**: nondeterministic polynomial-time problems which are “NP general”
- **NP**: nondeterministic polynomial-time problems
- **P**: polynomial-time problems

Formal Methods II
Turing Machines

**Definition 1** A Turing Machine $M$ is a set of quintuples $\{Q_n\} = \{\langle q_i, q_j, s_k, s_l, H \rangle \}$ where

- $q_i, q_j \in \{1, \ldots, m\}$ (the machine states)
- $s_k, s_l \in \{s_0, \ldots, s_r\}$ (the symbols)
- $H \in \{R, L\}$ (tape head direction)

such that no two quintuples have the same first and third elements.

The quintuples with $q_i = 1$ describe what happens in the START state of the machine.

**Definition 2** The size of a Turing machine $M$ is the number of its distinct states, written $|M|$.

This emphasizes that Turing machines are abstract (idealized) objects — identifying them with sets of quintuples of numbers and symbols.

OK, we need a few other things like a read/write head and an infinitely long tape. These are common to all Turing machines; the set $\{Q_n\}$ distinguishes between machines.
Turing Machines

We use the conventions:

- $s_0 = \square = \text{blank}$
- The tape is infinite to the right.
- In the START state the head is located at the leftmost square.
- If there is no quintuple for state and input then HALT (variation: designate a halting state)

$M$ operation:

1. $state \leftarrow 1$;
2. LOOP: $input \leftarrow \text{READ TAPE}$;
3. IF $\exists Q_i \text{ WITH } q_i = state \text{ AND } s_k = input$ THEN
   (A) WRITE $s_l$ ON TAPE;
   (B) MOVE HEAD $H$-WISE;
   (C) $state \leftarrow q_j$;
   (D) GO LOOP;
   ELSE HALT;
Polynomial Time Algorithms

Definition 3 A Turing machine $M$ is polynomially bound iff there is a polynomial function $p(n)$ s.t. for any input $x$ of size $n$ ($|x| = n$) $M$ halts after $p(n)$ or fewer steps ($\|M\| \leq p(n)$).

Note: $\|M\|$ = the number of steps $M$ takes before halting.

We will say a function $f$ is poly bound if a TM which computes it is poly bound.

Examples:

- Addition
- Multiplication
- Sorting

Non-Examples:

- TSP
- Optimizing in dag space
- Many other optimization problems

Formal Methods II
Decision Problems

Definition 4 Decision problems ask questions of the form: is there a TM which, given input $x$, will report back in finite time that $P(x)$ for some predicate $P$, when it is true?

Predicates are expressions of the form: "_ is $\Phi$" E.g.,

- "_ is blue"
- "_ is prime"

So, a decision problem asks the question: Can some TM answer "yes" to $\Phi(x)$ whenever it’s true? A TM which always correctly answers "no" solves the complementary decision problem.

Examples:

- Is $x$ a prime number?
- Is $x$ an odd number?
- Is $x$ a Turing machine which halts when given a null input?
Optimization Problems

Note that optimization problems can be converted into decision problems.

E.g.,

- The **shortest path problem** can be solved using an algorithm to decide the predicate: “Is this path shorter than $D$?”

- The **TSP** can be solved using an algorithm to decide the predicate: “Is this circuit (cycle) shorter than $D$?”
P and NP

**Definition 5** A decision problem $P$ is polynomially decidable ($P \in \mathbf{P}$) iff there is a polynomially bound TM which decides it.

**Definition 6** A problem $P$ is nondeterministic polynomial time ($P \in \mathbf{NP}$) iff a possible solution can be checked in polynomial time.

**Examples:**

- Addition and multiplication are NP (Why?)
- Factorizing numbers is NP
- Shortest path length and TSP are NP
Nondeterminism

**Warning:** nondeterminism ≠ indeterminism. They are both distinct from deterministic, but in different ways!

**Definition 7** A computation (algorithm, FSA, etc.) is **nondeterministic** iff the answer involves simultaneously traversing multiple computational paths.

E.g., the language accepted by a nondeterministic FSA is the union of all the sentences accepted by distinct paths.

This is strictly computer science jargon! There are proofs that nondeterministic TMs can compute no function beyond what deterministic TMs can compute (i.e., the computable functions).

**Definition 8** A computation (heuristic, FSA, etc.) is **indeterministic** iff the computational path incorporates steps which are not deterministic (i.e., probabilistic, stochastic).

There is no corresponding proof for indeterministic TMs.
Computability

Definition 9 (Computability)
A function \( f \) is computable iff there is a (deterministic) Turing machine which computes it.

Definition 10 (Computability of Numbers)
A number \( x \) is computable iff there is a Turing machine which, given an index \( i \in \mathbb{N} \) of the digital representation \( \langle d_0, \ldots, d_i, \ldots \rangle \) of \( x \), returns \( d_i \).

Theorem 1  All integers are computable.

Proof.
This is obvious.
Noncomputable Numbers

Noncomputable numbers exist:

- There are at least $\aleph_1 = |\mathbb{R}|$ numbers, by Cantor’s diagonal argument.
- There are no more than $\aleph_0 = |\mathbb{N}|$ Turing machines, since we can enumerate them.
- Therefore, Turing machines cannot compute all of the reals.

Exercise 1 This argument is incomplete, since Turing machines can compute more than one number. But it’s easy to prove that no Turing machine can compute more than $\aleph_0$ numbers. You can fill in this argument, using this idea.
Noncomputable Numbers

Here’s an argument: $\pi$ is noncomputable. Why?

- Because its digital representation is infinite, so a Turing machine which computed it could never halt. But algorithms have to halt; therefore, its noncomputable.

What’s wrong with this argument?
Noncomputable Numbers

Here’s an argument:

• $\pi$ is noncomputable. Why?

• Because its digital representation is infinite, so a Turing machine which computed it could never halt. But algorithms have to halt; therefore, its noncomputable.

What’s wrong with this argument?

*Answer 1 (Reductio)*: the digital representations of all reals are infinite, but they are not all noncomputable.

Rebuttal: but many reals are algebraic numbers, solutions to polynomial equations, which have a finite expression. $\pi$ is transcendental: it has no finite expression.
Noncomputable Numbers

Here’s an argument:

• \( \pi \) is noncomputable. Why?

• Because its digital representation is infinite, so a Turing machine which computed it could never halt. But algorithms have to halt; therefore, its noncomputable.

What’s wrong with this argument?

*Answer 1 (Reductio):* the digital representations of all reals are infinite, but they are not all noncomputable.

Rebuttal: but many reals are algebraic numbers, solutions to polynomial equations, which have a finite expression. \( \pi \) is transcendental: it has no finite expression.

Counter-rebutal: The rebuttal is both irrelevant and false (\( \pi = c/d \)). What matters is whether each digit of \( \pi \) can be computed in finite time (see Def 10), which is true.
Noncomputable Numbers

Here’s a real noncomputable number:

- Consider any enumeration of TMs. This will give us indices for all the TMs: $M_0, M_1, \ldots, M_n, \ldots$

- Consider the real number which has as its representation the infinite bit string such that $b_i = 1$ iff $M_i$ halts with a null input.

- This number cannot be computable. Why?