COMP30026 Models of Computation

Lecture 22: Reducibility, Review

Mak Nazecic-Andrlon and William Umboh

Semester 2, 2024

Some material from Michael Sipser's slides

Where are we?

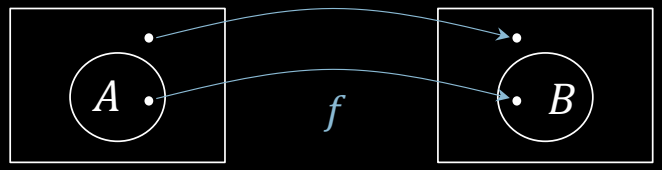
Last time:
- The Reducibility Method for proving undecidability and T-unrecognizability.
- General reducibility
- Mapping reducibility

Today: (Sipser §5.1, §5.3)
- Mapping reducibility (cont.)
- Review

Recall: Mapping vs General Reducibility

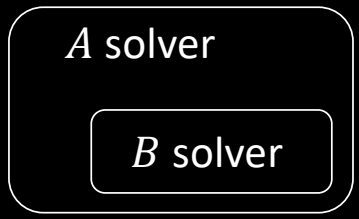
Mapping Reducibility of A to B: Translate A-questions to B-questions.

- A special type of reducibility
- Useful to prove T-unrecognizability



(General) Reducibility of A to B: Use B solver to solve A.

- May be conceptually simpler
- Useful to prove undecidability but NOT T-unrecognizability



Noteworthy difference:

- A is reducible to $\overline{A}$
- BUT A may not be mapping reducible to $\overline{A}$.

  For example $\overline{A_{\mathrm{TM}}} \not\leq_{\mathrm{m}} A_{\mathrm{TM}}$

Example of general reduction that is not mapping reduction: Halting problem

Recall: Reducibility – Templates

To prove B is undecidable:

- Show undecidable A is reducible to B. (often A is A_{TM})
- Template: Assume TM R decides B.
  Construct TM S deciding A. Contradiction.
- This is called a general reduction.

To prove B is T-unrecognizable:

- Show T-unrecognizable A is mapping reducible to B. (often A is $\overline{A_{\mathrm{TM}}}$)
- Template: give computable reduction function f.
- This is called a mapping reduction
- This also works to prove A is undecidable and is useful for undecidability proofs

Note: mapping reduction is special case of general reduction

E_{TM} is undecidable

Let $E_{\text{TM}} = \{\langle M \rangle | \ M \text{ is a TM and } L(M) = \emptyset \}$

Theorem: E_{TM} is undecidable

Proof by contradiction. Show that A_{TM} is reducible to E_{TM}.

Assume that E_{TM} is decidable and show that A_{TM} is decidable (false!).

Let TM R decide E_{TM}.

Construct TM S deciding A_{TM}.

$S =$ "On input $\langle M, w \rangle$

1. Transform M to new TM $M_w =$ "On input x
    1. If $x \neq w$, *reject*.
    2. else run M on w
    3. *Accept* if M accepts."
2. Use R to test whether $L(M_w) = \emptyset$
3. If YES [so M rejects w] then *reject*.
   If NO [so M accepts w] then *accept*.

```
import R

def S(M, w):
    def M_w(x):
        if x ≠ w:
            Reject
        Else:
            return M(w)
    return not R(M_w)
```

M_w works like M except that it always rejects strings x where $x \neq w$.

So $L(M_w) = \begin{cases} \{w\} & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ rejects } w \end{cases}$

E_{TM} is T-unrecognizable

Recall $E_{\text{TM}} = \{\langle M \rangle | \ M \text{ is a TM and } L(M) = \emptyset \ \}$

Theorem: E_{TM} is T-unrecognizable

Proof: Show $\overline{A_{\text{TM}}} \leq_{\text{m}} E_{\text{TM}}$

Recall TM M_w = "On input x
1. If $x \neq w$, *reject*.
2. else run M on w
3. *Accept* if M accepts."

Reduction function: $f(\langle M, w \rangle) = \langle M_w \rangle$

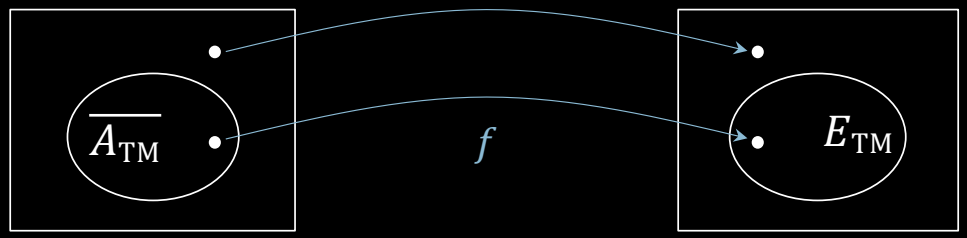
Correctness: Need to show $\langle M, w \rangle \in \overline{A_{\text{TM}}}$ iff $\langle M_w \rangle \in E_{\text{TM}}$

By def of $\overline{A_{\text{TM}}}$, $\langle M, w \rangle \in \overline{A_{\text{TM}}}$ iff M rejects w

By def of M_w, M rejects w iff $L(M_w) = \emptyset$

By def of E_{TM}, $L(M_w) = \emptyset$ iff $\langle M_w \rangle \in E_{\text{TM}}$

Therefore, $\langle M, w \rangle \in \overline{A_{\text{TM}}}$ iff $\langle M_w \rangle \in E_{\text{TM}}$, as desired.



Exercise: $\overline{E_{\text{TM}}}$ is T-recognizable

EQ_{TM} T-unrecognizable

$EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

Theorem: EQ_{TM} is T-unrecognizable

Proof: $\overline{A_{\text{TM}}} \leq_{\text{m}} EQ_{\text{TM}}$

For any w let $T_w =$ "On input x

1. Ignore x.
2. Simulate M on w."

def $T_w(x)$:
  return $M(w)$

T_w acts on all inputs the way M acts on w.

So $L(T_w) = \begin{cases} \Sigma^* & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ rejects } w \end{cases}$

Here we give f which maps $\overline{A_{\text{TM}}}$ problems (of the form $\langle M, w\rangle$) to EQ_{TM} problems (of the form $\langle T_1, T_2\rangle$).

$f(\langle M, w\rangle) = \langle T_w, T_{\text{reject}}\rangle$ $\quad$ T_{reject} is a TM that always rejects: $L(T_{reject}) = \emptyset$.

Correctness: Need to show $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff $\langle T_w, T_{\text{reject}}\rangle \in EQ_{\text{TM}}$

By def of $\overline{A_{\text{TM}}}$, $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff M rejects w.

By def of T_w, M rejects w iff $L(T_w) = \emptyset$

By def of T_{reject} and EQ_{TM}, $L(T_w) = \emptyset$ iff $\langle T_w, T_{\text{reject}}\rangle \in EQ_{\text{TM}}$.

$\overline{EQ_{\text{TM}}}$ T-unrecognizable

$\overline{EQ_{\text{TM}}} = \{\langle M_1, M_2\rangle |\ M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) \neq L(M_2)\ \}$

Theorem: EQ_{TM} is T-unrecognizable

Proof: $\overline{A_{\text{TM}}} \leq_{\text{m}} \overline{EQ_{\text{TM}}}$

For any w let T_w = "On input x

1. Ignore x.
2. Simulate M on w."

def $T_w(x)$:
return $M(w)$

T_w acts on all inputs the way M acts on w.

So $L(T_w) = \begin{cases} \Sigma^* & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ rejects } w \end{cases}$

Here we give f which maps $\overline{A_{\text{TM}}}$ problems (of the form $\langle M, w\rangle$) to $\overline{EQ_{\text{TM}}}$ problems (of the form $\langle T_1, T_2\rangle$).

$f(\langle M, w\rangle) = \langle T_w, T_{\text{accept}}\rangle$ T_{accept} is a TM that always accepts: $L(T_{reject}) = \Sigma^*$.

Correctness: Need to show $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff $\langle T_w, T_{\text{accept}}\rangle \in \overline{EQ_{\text{TM}}}$

By def of $\overline{A_{\text{TM}}}$, $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff M rejects w.

By def of T_w, M rejects w iff $L(T_w) = \emptyset$

By def of T_{reject} and $\overline{EQ_{\text{TM}}}$, $L(T_w) = \emptyset$ iff $\langle T_w, T_{\text{accept}}\rangle \in \overline{EQ_{\text{TM}}}$

Reducibility terminology

Why do we use the term "reduce"?

When we reduce A to B, we show how to solve A by using B
and conclude that A is no harder than B. (suggests the $\leq_{\mathrm{m}}$ notation)

Possibility 1: We bring A's difficulty down to B's difficulty. (A no harder than B)

Possibility 2: We bring B's difficulty up to A's difficulty. (B no easier than A)

Defn. When $A \leq_{\mathrm{m}} B$ and $B \leq_{\mathrm{m}} A$, then $A =_{\mathrm{m}} B$. The two problems are equivalent, i.e. A is recognizable/decidable iff B is recognizable/decidable.

Defn. When $A \leq_{\mathrm{m}} B$ and $B \leq_{\mathrm{m}} C$, then $A \leq_{\mathrm{m}} C$.

For example, if we showed $A_{TM} \leq_{\mathrm{m}} B$, then to show C is undecidable, we can also show $B \leq_{\mathrm{m}} C$. Moral: We don't always have to reduce from A_{TM}!

Student Learning Survey

https://unimelb.bluera.com/unimelb/

- What was good? What can be improved?
- Mention tutor/lecturer name specifically
- Mid-Sem Survey Feedback not yet received
- We take feedback seriously. Based on student feedback, we have
  - Removed Haskell
  - Included more resources for discrete math and informal proofs
  - Submitted a handbook change proposal to increase tutorials from 1-hr to 2-hrs
  - Added weekly quizzes to help you catch misunderstandings/misconceptions early
- William:
  - Added illustrations, examples and Python equivalents to make arguments more concrete
  - Added motivation
  - Alternative proof of undecidability

Theory and Practice Again

There is nothing more practical than a good theory.”

Kurt Lewin

“The theory I do gives me the vocabulary and the ways to do practical things that make giant steps instead of small steps when I’m doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they’re just purely inspired by other theories. There’s this symbiotic relationship . . . ”

Donald Knuth

Formalisation and Abstraction

Computing systems are incredibly complex.

The best chance we have of understanding a very complex system is to isolate the essence of the machinery or system, that is, to abstract away from irrelevant detail.

The role of theory is to get to the core of the subject or phenomenon under study.

Mathematics, and mathematical notation, is the result of centuries of work on this sort of activity: extracting the essence of a phenomenon so as to focus the thinking on what matters.

Propositional Logic

Propositional formulas: Syntax and semantics.

Semantics is simple, in principle—just a matter of constructing truth tables. However, it is useful to develop an understanding of the algebraic rules, how to rewrite formulas to equivalent formulas in normal form, and so on.

Important concepts: Satisfiability and validity of formulas, logical consequence and equivalence.

Normal forms: CNF

Mechanical proof: Propositional resolution.

First-Order Predicate Logic

Syntax and semantics.

Important semantic tools: the concepts of interpretation, and of model of a formula (making it true).

Components of an interpretation: Universe of discourse and mappings giving meaning to predicate symbols, function symbols, constant symbols.

To define the meaning of quantifiers we also need to consider variable assignments.

Important concepts: Models and counter-models, satisfiability and validity, semantic consequence and equivalence.

First-Order Predicate Logic

Useful to develop an understanding of how formulas can be rewritten, rules of passage for the quantifiers and so on.

Normal forms: Clausal form.

Obtaining equi-satisfiable formulas in clausal form: Skolemization.

Mechanical proof: Resolution, including unification.

Relevance of Logic

Propositional Logic

- Historically: Use in hardware design, fault finding and verification (model checking).
- Boolean modelling is increasingly important because of the availability of powerful SAT solvers.

First-Order Predicate Logic

- Historically: Use in artificial intelligence, proof assistants, automated theorem proving.
- First-order predicate logic is a computer scientists' lingua franca.
- Constraint solvers for various theories play central roles in tools for software verification, vulnerability detection, test data generation, planning and scheduling, ...

Proof

There is an expectation that you can provide readable and valid proofs for simple assertions (about the material covered in the subject).

Relevance:

A formal notation is the basis for precise and unambiguous expression.

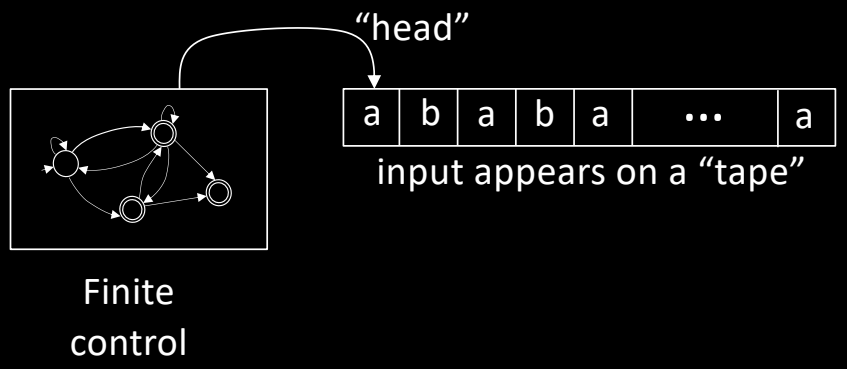
Proof is at the core of clear and rigorous thinking.

Proof is how you conduct the ultimate persuasive argument.

Regular Languages

Finite-state automata: DFAs and NFAs.

Finite-state automata as recognisers.



The regular operations.
Regular expressions.
Closure properties of regular languages.

$(0 \cup 1)^*$

Important techniques: Translating NFAs to DFAs, regular expressions to NFAs, and vice versa.

Using the pumping lemma for regular languages to prove non-regularity.

Useful for: e.g. string search, string processing

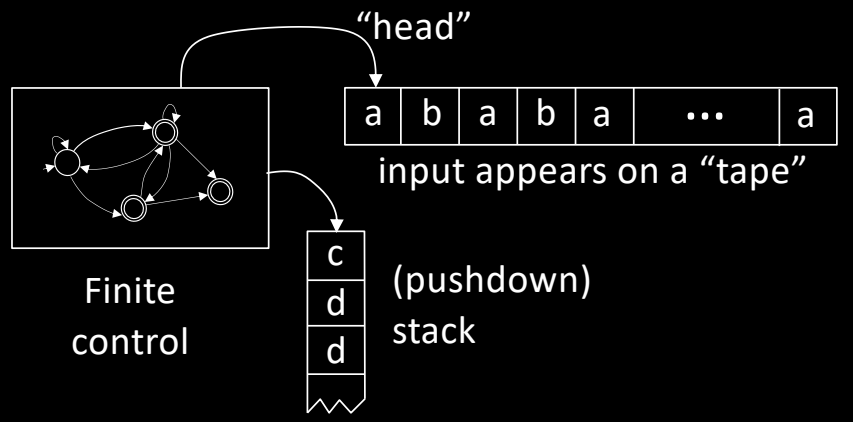
Context-Free Languages

Context-free grammars, derivations of sentences.

Parse trees, ambiguity.

Push-down automata.

(Lack of) closure properties of context-free languages.



$E \rightarrow E+T \mid T$

$T \rightarrow T \times F \mid F$

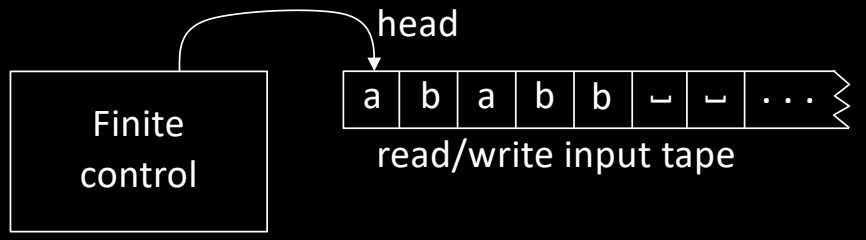
$F \rightarrow (E) \mid a$

Important techniques:

- Translating a CFG to an equivalent PDA.
- Using CFL pumping lemma to prove languages non-context-free.

Applications: Parsers, compilers, machine translation

Computability, Turing Machines



The Church-Turing thesis:
Computable is what a Turing machine can compute.

Decidable languages: Those that are recognised by some Turing machine which halts for all input.

Turing recognisable languages: Those that have a Turing machine that acts as a recogniser (and does not necessarily halt).

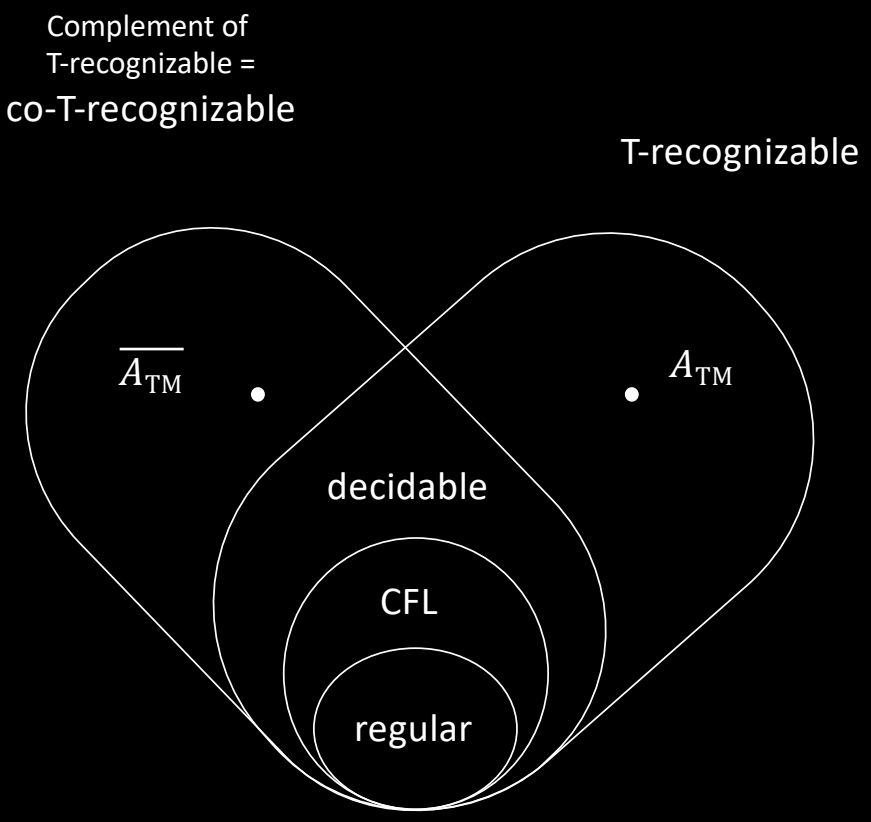
Proof Techniques for (Un)decidability

- Simulation
- Reduction (mapping reducibility and general reducibility)
- Exploitation of closure properties
- Diagonalisation

Not covered (see Sipser):

- Undecidability of combinatorial problems (e.g. Post Correspondence)
- NP-hardness (poly-time mapping reductions)

Models of Computation



Relevance of Computability Theory

Knowing the limits of what can be done allows us to focus on decidable problems and functions that can be captured as algorithms.

It tells us to settle for the less-than-perfect when we are up against undecidable properties.

For example, tools for software and protocol verification, optimizing compilers, and program repair are all based on reasoning with safe approximations of programs' runtime states.

Closure Properties

	$\cup$	$\circ$	*	$R\cap$	$\cap$	compl
Reg	Y	Y	Y	Y	Y	Y
DCFL	N	N	N	Y	N	Y
CFL	Y	Y	Y	Y	N	N
Decidable	Y	Y	Y	Y	Y	Y
Recognisable	Y	Y	Y	Y	Y	N

Here '$\circ$' means concatenation, '*' means Kleene star, and '$R\cap$' means "intersection with a regular language".

DCFL is the class of languages that can be recognised by deterministic PDAs (DPDAs).

Decidability of Language Properties

Question	Reg	DCFL	CFL	Decidable	Recognisable
$w \in L$	D	D	D	D	U
$L = \emptyset$	D	D	D	U	U
$L = \Sigma^*$	D	D	U	U	U
$L_1 = L_2$	D	D	U	U	U
$L =$ given R	D	D	U	U	U
L regular	D	D	U	U	U
$L_1 \subseteq L_2$	D	U	U	U	U

Here 'D' = decidable; 'U' = undecidable.

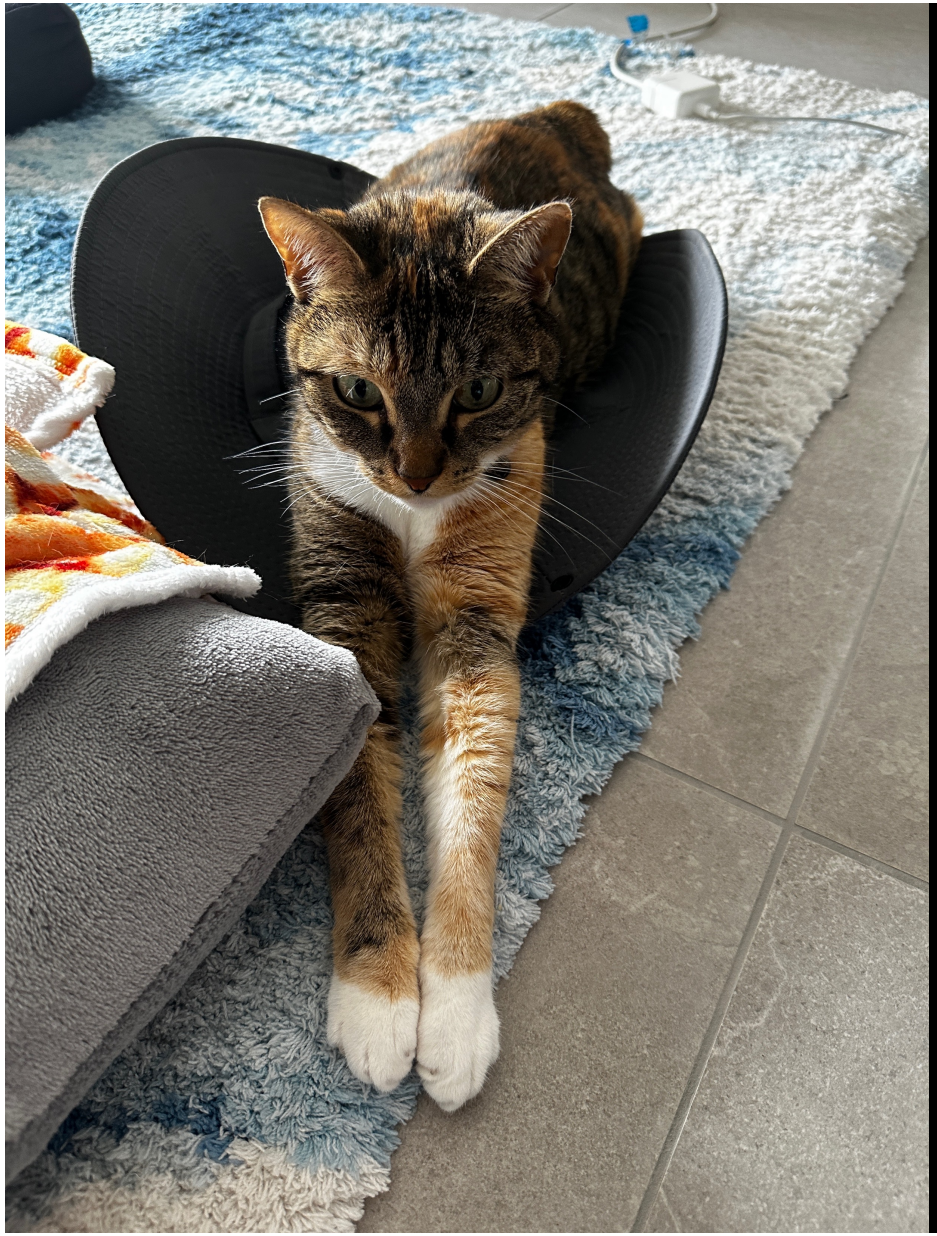
More Theory?

- COMP90077: Advanced Algorithms and Data Structures
- Nature of Computation by Cristopher Moore and Stephan Mertens
- Mathematics and Computation by Avi Wigderson

Preparing for the Exam

Practice on exercises from tutes, books, etc. Post your own solutions on Ed for feedback.

Get a good night’s sleep before the exam.



TACOCAT:
Thank you and good luck!