COMP3153

Theory of Computation

Lecture 1: Course Introduction

Outline

Acknowledgement of Country

I would like to acknowledge and pay my respect to the Bedegal people who are the Traditional Custodians of the land on which UNSW is built, and of Elders past and present.

COMP4141 22T1 Staff

Lecturer:	Paul
Email:	paul.[redacted]nsw.edu.au
Lectures:	Mondays 2pm-4pm and Thursdays 12-2pm
Tutorial:	Tuesday (online) 4-6pm
Consults:	Thursdays 8-9pm
Research:	Theoretical CS: Algorithms, Formal verification
Tutors:	Ian Thorvaldson (Face-to-face)
	Ethan Brown (Wed online)

Teaching arrangements

http://www.cse.unsw.edu.au/~cs4141/

- Online lecture
- ed forum
  - No email? Join link
- Lecture recordings
- Tutorials (TBP)
- Online consultation: (TBP), Thursdays 8pm
- email: `paul.hunter@unsw.edu.au`
- Course textbook: Michael Sipser, *Introduction to the Theory of Computation*

Tutorials

- Tuesday 2pm (Ia[illegible] link
- Tuesday online 4pm (Paul): Zoom link
- Tuesday F2F 4pm (Ian): Quad G042
- Wednesday online 4pm (Ethan): Zoom link
- Wednesday F2F 4pm (Ian): Quad G042

Note: No F2F Tutorial on Tuesday @ 2pm

Organisation

Classes

- Lectures: slides re notes
- Tutorials: proble

Homework

- Four assignments
- due Fridays (odd weeks) 12 noon (Sydney time)
- Individual submissions through webCMS/give
- high-level discussions with others ok

Assessment

- 50% homework
- 50% exams:
  a final take-home exam (date tba) (24 hours) worth 50%

Lateness policy

- 1 hour grace period
- 10% off raw mark per 12 hours or part thereof
- If you cannot meet a deadline through illness or misadventure you need to apply for Special Consideration.

Outline

Why do COMP4141?

Models of Computation

What is comput

How can we mo utation?

Why should we

Computational Complexity

What sorts of things can (and cannot) be computed?

What happens when we limit resources (e.g. time or memory)?

How to win $1 million...

Set theory, developed as a foundation for all of mathematics, provides a very useful formal framework in which to express the answers to such questions.

Value of the course

In 20 years, computers and programming will be vastly different.
But this material will be very much the same—and will still be useful.
Provides insight into fundamental questions

- defines the questions
- answers some
- many are open!
- very close connection with logic, algorithms, linguistics, others.

Provides advanced problem-solving tools.

- springboard for more advanced courses
- research
- applications

Practice with mathematics and proofs.

Course timeline (roughly)

Week 1	Introduc Theory, Finite automata
Week 2	Regular
Week 3	Context- uages and Pushdown automata
Week 4	Recursively enumerable languages and Turing Machines
Week 5	Decidability and reductions
Week 6	Flex week
Week 7	Time and space complexity, P and NP
Week 8	NP-completeness, SAT, P TIME reductions
Week 9	PSPACE, LogSPACE, Alternation
Week 10	Probabilistic computation, Approximation

Outline

Set Theory

- union: $S \cup T$
- intersection: $S \cap$
- empty set: $\emptyset$
- set difference: $S \setminus T$ or $S - T$
- complement: $\overline{S}$
- distributivity: $S \cup (T \cap U) = (S \cup T) \cap (S \cup U)$

  $S \cap (T \cup U) = (S \cap T) \cup (S \cap U)$
- subset: $S \subseteq T$
- element of: $x \in S$
- comprehension: $\{x \in S \mid \varphi(x)\}$ or $\{x \in S : \varphi(x)\}$
  the set of elements of S satisfying φ

Representing Sets (discussion)

Suppose a programmer needs to represent a small, *finite*, set S.

- What does "*rep*[illegible]ean?
  Answer: You ca[illegible]questions about it.
  Simple common [illegible] Is $x \in S$?
  Other questions: Is $S = \emptyset$? Is $S \cap T = \emptyset$? Etc.
- What representations would be appropriate?

Suppose you want to represent *infinite* sets. How do you do it?

- Same question: What does "represent" mean?
  Same answer: You can answer questions about it.
  Same simple common question: Is $x \in S$?
- What representations would be appropriate?
  That's what this part of the course is about.

What is a representation?

Suppose you have de[illegible]otation for sets, that is a *representation* that can be stored in a computer.
Can all sets be represented?

This raises profound questions: Which sets can be represented on a computer and which can't?

One view of formal language theory

Automata and complexity theory is concerned with properties of *formal languages*.

In formal language, automata and complexity theory, a *language* is just a set of strings.

(Like many mathematical definitions, this leaves behind most of what we think of as "languages," but can be made precise. And it leads to very profound results.)

Basically, any object or value that is of interest to computer science can be represented as a string.

So a set of anything can be considered a language.

Questions from formal language theory

What (infinite) sets are representable?

What can a computer do with the representations, in theory?

What cannot be done with the representations, in theory?

What problems are easy, hard, or impossible to solve computationally?

Another view of formal language theory

For practical purposes, a language is the same thing as a Boolean function. Such a function is also called a *property* or a *predicate*.

For example, the predicate **even**(x), which returns "true" iff x is (string representation of) an even number, can be considered to represent the set of even numbers (the set of x such that "implicit set lookup").

So, if we can answer questions about languages, we are also answering questions about properties of objects.

Application: Computer languages

Basis for tools and p[illegible]ng techniques.

- Lexical analysis
- Parsing
- Program analysis

Many interesting problems in programming language implementations are hard or impossible to solve in general.

Examples:

- Equivalence of grammars.

- Almost any exact analysis.

Application: Formal Verification

Formal verification a[illegible] prove system designs (e.g. programs) correct, o[illegible]ugs.

Methods are generally from logic and automata theory. Many of the constructions in this course are used in practical tools.

- Automata constructs (e.g. product construction)
- Reductions to SAT (an NP-completeness proof technique).
- "Bounded model checking"—the idea is from Cook's theorem

It is also important to know a little about complexity theory, since many problems in this area are hard or impossible to solve, in general.

Outline

Basic concept

Definition

An *alphabet* is a non[illegible]ite set. The members of the alphabet are called *symbols*.

Examples

Binary alphabet $\{0, 1\}$

ASCII character set—the first 128 numbers, many of which are printed as special characters. Also, any other finite character set.

The capital Greek sigma (Σ) is often used to represent an alphabet.

Strings

Informally: A string sequence of symbols from some alphabet.

Examples

- ϵ—the empty string (the same for every alphabet). (Leaving a blank space for the empty string is confusing, so we use the Greek letter "epsilon".) ϵ is not a symbol. It is the string with no symbols; the string of zero length.
- 000, 01101 are strings over the binary alphabet
- "String" is a string over the ASCII character set, or the English alphabet.

Strings cont.

Definition (strings over alphabet Σ)

Base: ϵ is a string over Σ.

Induction: If x is a string over Σ and a is a symbol from Σ, then ax is a string over Σ.

(Think of ax as appending a symbol to the front of an existing string.)

Notation: The set of all strings over an alphabet Σ is written Σ^*.

Length of a string

Many functions are defined recursively on the structure of strings, and many proofs are by induction on strings.

Informally: The *length* of a string is the number of occurrences of symbols in the string (the number of different positions at which symbols occur).

The length of string x is written $|x|$.

Definition (length)

Base: $|\epsilon| = 0$

Induction: $|ax| = 1 + |x|$

Concatenation of strings

Informally: The concatenation of strings x and y over alphabet Σ is the string formed by following x by y. It is written $x \cdot y$, or (more often) xy.

Examples

- $abc \cdot def = abcdef$
- $\epsilon \cdot abc = abc$

Definition (concatenation)

The definition is recursive on the structure of the first string:

Base: $\epsilon \cdot x = x$ if x is a string over Σ.

Induction: If x and y are strings over Σ and $a \in \Sigma$ then

$$(ax) \cdot y = a(x \cdot y)$$

Note: The parentheses are not symbols, they are for grouping, so $(ax) \cdot y$ is ax concatenated with y.

Proof by Induction

- Show that for all strings x, y, z over Σ concatenation is associative, i.e.,

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

Sidenote: Proof Expectations

We don't want to lose sight of the forest because of the trees. Here are the "forest-level" points with proofs.

- What is the proof technique?
  - Induction on [illegible]. *What are the base and induction steps?*
  - Induction on [illegible]s. *What are the base and induction steps?*
  - Diagonalization
  - Reduction from another problem. *Which direction is the reduction?*
- What are the key insights in the proof?
- Often this is a construction (often something that can be implemented as a computer program)
  - Translation between regular expressions, various finite automata.
  - Translation from one problem to another.

Explain these things clearly in your proofs. If we can see *quickly* that you did the right kind of proof and got the major points right, you may get nearly full marks.

Sidenote: Proof Guidelines

1. State *what* is being proved precisely and clearly.
2. Start proof with [illegible]nation of the *strategy* (e.g. "induction on y")
3. Provide guidepo[illegible]*Base*, *Induction*)
4. Highlight the interesting key parts of the proof (where did you have to be clever?)
5. Make it easy for the graders to see these things.

NB

Use Sipser's proofs as blueprints. As beginners, you need to provide more detail than he typically does. The license to be brief has to be earned by repeatedly demonstrating the capability of filling in all omitted detail. Do not omit detail your average reader/fellow student cannot be expected to fill in.

Languages

Definition

A *language* over Σ is a subset of Σ^*.

NB

Of course, this omits almost everything that one intuitively thinks is important about a language, such as meaning. But this definition nevertheless leads to incredibly useful and important results.

Examples

- $\emptyset$ (the empty language).
- $\{\epsilon\}$ (the language consisting of a single empty string).
- The set of all strings with the same number of as as bs.
- The set of all prime numbers, written as binary strings.
- The set of all strings representing C programs that compile without errors or warnings.
- The set of all first-order logic formulas.
- The set of all theorems of number theory, in an appropriate logical notation.
- The set of all input strings for which a given Boolean C function returns "true."

Outline

Maze navigation

Task: "Program" a robot to navigate a maze

Every "tick" the robot receives inputs from its sensors and can send outputs to its peripherals.

Inputs	Outputs
Obstacle in front	Move forward 1 step
Moving forward	Turn left 90°
Turning	Turn right 90°

Write a program to drive the robot out of an (unknown) maze