Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 1

Introduction

slides by Graham Farr

Lecture overview

- General information
- Languages
- Terminology
- Definitions, Theorems, Proofs

People

Lecturers (Clayton campus)

- Prof. Graham Farr, Dr Rebecca Robinson

Admin Tutors (Clayton)

- Michael Gill
- Mathew Baker
- Roger Lim

} FIT2014-AdminTutor-x@monash.edu

Lecturers (Malaysia campus)

- Assoc. Prof. KokSheik Wong
- Assoc. Prof. Anuja Dharmaratne
- Dr Vee Voon Yee

Tutors (Clayton)

- Mathew Baker
- Dr Harald Bögeholz
- Luhan Cheng
- Nathan Companez
- Madison Geeson
- Michael Gill
- Jackson Goerner
- Thomas Hendrey
- Dr Nhan Bao Ho
- Ethan Hunt
- Ben Jones
- Stephen Krol
- Roger Lim
- Isobel Nixon
- Dr Han Duy Phan
- Pooja Pancholi
- Sachinthana Pathiranage
- Leo Pham
- James Sherwood
- Grant Sinclair
- Zhi Hao Tan
- Jared Turnley
- Carl Vu
- Rebecca Young

Textbooks

Recommended Text:

- Michael Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, 2006.

Also useful:

- Daniel I. A. Cohen, *Introduction to Computer Theory (2nd Edition)*, Wiley, New York, 1997.

For some cultural and historical context, and lots of fun:

- Sydney Padua, *The Thrilling Adventures of Lovelace and Babbage*, Penguin, 2015.

Classes

Lectures: on campus, live-streamed and recorded (see Moodle).

Prac classes (2 hours): start week 1 — *this week* (Lab 0, on Linux)
Three types:

- Labs 0,1,2: Weeks 1, 4, 10,
- Tutorials (2 hours): Weeks 2, 3, 6, 7, 8, 9, 11, 12
- Interviews: Weeks 5, 13

Timetable: `http://www.monash.edu/timetables/allocate/`

Mid-Semester Test: week 7

- Tutes/labs continue as usual that week

Assessment

- Tutorial preparation (5% total)
  - all tutorials: weeks 2, 3, 6, 7, 8, 9, 11, 12.
  - Each tutorial has a nominated preparation exercise.
  - You must make a serious attempt at this question, although it does not need to be entirely correct.
  - Submission deadline: at the start of your tutorial.
    - *on-campus classes:* Bring it to class. It will be assessed at the start.
    - *online classes:* Submit online, in Moodle, prior to the start of your class, using the submission link for your class in each week.
  - You get 1 mark for a serious and clear attempt; 0 marks otherwise.
  - Maximum 8 marks for the semester.

Assessment (continued)

- Tutorial preparation (5%) (see previous slide)

- Assignment 1 (10%)
  - due Friday 11:55pm in week 4
  - interviews in week 5
  - final mark $=$ provisional mark $\times \underbrace{\text{interview factor}}_{\text{between 0 and 1}}$

- Mid-Semester Test (15%)
  - in week 7

- Assignment 2 (20%)
  - due Friday 11:55pm in week 10
  - interviews sometime in weeks 11–13
  - final mark $=$ provisional mark $\times \underbrace{\text{interview factor}}_{\text{between 0 and 1}}$

- Final Exam (50%)

Assessment (continued)

Hurdles

- Total of non-Final-Exam assessments:

  two Assignments
  + Mid-Semester Test
  + Tutorial Preparation:

  at least 45% of available marks

- Final Exam: at least 45% of available exam marks
- Overall: at least 50%

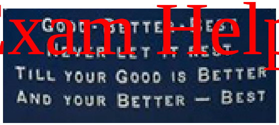
Assessment (continued)

Academic Integrity

- All assessment in this unit is based on your **own individual work**. All your assignments, tests, exams . . . must be **your own work**, no-one else's.
- **No** plagiarism, collusion, cheating, etc.
- For further details, see:
  `https://www.monash.edu/students/admin/policies/academic-integrity`
- In FIT2014, plagiarism and cheating don't work.
- **Every** assignment submission is followed up by an interview to check understanding and authenticity.
- **Any** other assessment may be followed up with an interview to check understanding and authenticity.
- Academic integrity cases are time-consuming for staff.
  We will put in whatever time is needed.
  But we'd rather spend that time teaching you and helping you learn!

Come to your classes!

Good:	watching lecture recordings
Better:	attending live-streamed lectures online
Best:	attending lectures in person


Good-Better-Best
Never let it rest
Till your Good is Better
And your Better – Best

https://furphyfoundry.com.au/

Lecture attendance is especially important in theoretical subjects.

- S. Trenholm, B. Hajek, C.L. Robinson, M. Chinnappan, A. Albrecht & H. Ashman, Investigating undergraduate mathematics learners' cognitive engagement with recorded lecture videos, *International Journal of Mathematical Education in Science and Technology* **50** (2019) 3–24. https://doi.org/10.1080/0020739X.2018.1458339

Come to your classes!

When attending online tutorials/labs:

- Turn cameras on!
  - This makes it more real, more active, more effective, for everyone.
  - It's the online version of coming into the classroom.
- This is now a **requirement** in this unit.
  - For exemptions: email the lecturer on your campus.
  - Camera off, without an exemption, may result in removal from online class.
- Learn to use Zoom functions:
  - you can blur your background (via pop-up menu ^ next to Start Video button)
  - you can use an image as your background (ditto);
  - you can Hide Self View if you wish:
    - hover over video, click on the short row of three dots • • • , click on Hide Self View
    - `https://support.zoom.us/hc/en-us/articles/115001077226-See-or-Hide-My-Video`
- Human connection is fundamental to learning and teaching.

Further information

- Moodle (including Forums → Ed discussion)
- Tutor

- Admin Tutors at Clayton campus: ✉ `FIT2014-AdminTutor-x@monash.edu`
- Lecturer

Why study Theory of Computation?

- To understand properly the power and limits of computation;
- To identify whether a problem is tractable or intractable;
- To understand why a particular problem seems hard: is it because of the limitations of your computer, or because of some intrinsic feature of the problem?
- To be able to identify problems from different fields that have the same underlying structure.

Some applications

- Pattern recognition (in text, DNA, proteins, financial data, . . .)
- Modelling of natural languages
- Compilers and interpreters for programming languages
- Information security
- Communications: codes, protocols
- Verification of complex systems

Languages

Computation is done with strings of symbols, so . . .

Alphabets

An **alphabet** is a finite set of symbols.

Its members are called **letters** or **characters**.

We often denote it by Σ.

Examples of alphabets:

- $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$
- $\{\mathtt{a}, \mathtt{b}\}$
- $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}\}$
- $\{\mathtt{0}, \mathtt{1}\}$

Languages

Words

A **word** is a finite string of symbols.

A **word over alphabet** Σ is a finite string of symbols all taken from Σ.

The **empty word** is the word of length 0, denoted by ε (or sometimes Λ).

Languages

A **language over alphabet** Σ is a set of words over Σ.

Special languages:

- **empty language**: $\emptyset = \{\}$
  - not to be confused with the language containing just the *empty word*: $\{\varepsilon\}$
- $\Sigma^k := \{$ all words over Σ of length $k\,\}$
  - E.g.: $\{\mathtt{a}, \mathtt{b}\}^2 = \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$
- **universal language**: $\Sigma^* := \{$ all finite words over $\Sigma\,\}$

Languages

Natural languages

- English, Australian, Woiwurrung, Chinese, Auslan, . . . , Hebrew, Indonesian, Kannada, Hokkien, Croatian, Romanian, Russian, Tamil, Klingon, Spanish, Turkish, Tagalog, Vietnamese, French, Hungarian, Swedish, Elvish, Esperanto, Greek, Latin, Korean, Urdu, Konkani, Bengali, Singlish, Dutch, Ukrainian, Persian, Malay, German, American Sign Language (ASL), Hindi, . . .

Programming languages

- Python, Java, Haskell, awk, C, MIPS Assembler, Smalltalk, Prolog, Simula67, Interprogram, Algol 60, Cobol, Fortran, . . . , Rust, C++, MATLAB, Racket, R, Lisp, UwU, Go, Lua, C♯, APL, Malbolge, Scratch, Verilog, Phi, Julia, . . .

Languages also appear in:

- mathematics, music, knitting, games, . . .

Assumptions and notation

Unless otherwise stated, we use alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

Repetition: $\mathtt{a}^2 = \mathtt{aa}$, $\quad \mathtt{ab}^3 = \mathtt{abbb}$, $\quad (\mathtt{ab})^3 = \mathtt{ababab}$

If x is a word, then x^k is the string obtained by concatenating k copies of x together:

$$x^k = \underbrace{xxx \cdots\cdots xx}_{k \text{ copies of } x}$$

$(\mathtt{baa})^0 = \ldots ?$

Some simple languages

EVEN-EVEN $:=$ {all strings in which *each* of a,b occurs an <u>even</u> number of times}

$= \{\varepsilon, \mathtt{aa}, \mathtt{bb}, \mathtt{aaaa}, \mathtt{aabb}, \mathtt{abab}, \mathtt{abba}, \mathtt{baab}, \ldots\}$

Remember, 0 is even!

DOUBLEWORD $:= \{xx : x \in \Sigma^*\}$

$=$ {all strings obtained by concatenating some string with itself}

$= \{\varepsilon, \mathtt{aa}, \mathtt{bb}, \mathtt{aaaa}, , \mathtt{abab}, \mathtt{baba}, \mathtt{bbbb}, \mathtt{aaaaaa}, \mathtt{aabaab}, \ldots\}$

Note, $\varepsilon\varepsilon = \varepsilon$.

PALINDROMES $:=$ {all strings that are the same forwards and backwards}

$= \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{bb}, \mathtt{aaa}, \mathtt{aba}, \mathtt{bab}, \mathtt{bbb}, \mathtt{aaaa}, , \mathtt{abba}, \mathtt{baab}, \ldots\}$

Definitions, Theorems, Proofs

Definition

- specifies the precise meaning of something.

Theorem

- a mathematical statement that has been proved to be true.
- has some close but "less significant" relatives: Proposition, Lemma

Proof

- A step-by-step argument that establishes, logically and with certainty, that something is true.
- Should be verifiable.

Examples of theorems and proofs

Theorem.
English has a palindrome.

Proof. 'rotator' is an English word and also a palindrome. □

An *existential statement* . . .

There exists a palindrome in English

. . . just requires one suitable example for a proof.

Most proofs are not this short . . .

Examples of theorems and proofs

Theorem.
Every English word has a vowel or a ‘y’.

Proof.
‘aardvark’ has a vowel.
‘aardwolf’ has a vowel.
‘aasvogel’ has a vowel.
. . .

. . .
‘syzygy’ has a ‘y’.

. . .
‘zygote’ has a vowel. □

Theorems and proofs

To prove a *universal* statement . . .

For every English word, it has a vowel or a ‘y’

. . . you need to cover every possible case.

One way is to go through all possibilities, in turn, and check each one.
But the number of things to check may be huge, or infinite.
So usually we want to reason in a way that can apply to many different possibilities at once.

Another example theorem

Theorem
DOUBLEWORD $\subseteq$ EVEN-EVEN

Non-proof:
The examples on the previous slide show that every member of DOUBLEWORD has an even number of as and an even number of bs.
So every member of DOUBLEWORD is also a member of EVEN-EVEN.

"Proof by example" is not a proof.

. . . except where the Theorem just asserts the existence of a specific example!

Definitions, Theorems, Proofs

To prove a subset relationship, $A \subseteq B$:

- Prove that every element of A is also an element of B.
  - Start with a general member of A.
  - Work towards proving that it also belongs to B.
- Give names to things.
- Use the definitions of the sets.

Theorem.
DOUBLEWORD $\subseteq$ EVEN-EVEN

Proof. Let $w \in$ DOUBLEWORD.
Then $w = xx$ for some word x.
So, # a's in w $=$ $2 \times$ (# a's in x), so it's even.
Also, # b's in w $=$ $2 \times$ (# b's in x), so it's even too.
So $w \in$ EVEN-EVEN. $\square$

Other topics

- Propositional logic
- Predicates, quantifiers
- Linux
- Regular languages, finite automata
- Grammars, context-free languages, pushdown automata
- Lexical Analysis
- Introduction to parsing
- Turing Machines
- Computability, decidability
- Computational complexity
- Classes P and NP
- NP-completeness

Reading

- Sipser, pp 13–14
  - strings and languages

- Sipser, §0.3, pp 17–20
  - definitions, theorems, proofs