Module 1

Overview and background

What is it all about?

CS 360: Introduction to the Theory of Computing

Collin Roberts, University of Waterloo

1.1

1 Welcome

Topics of Module 1

- Overview of the course
- Administrivia
- Mathematical background

Ideally this is not all administrivia

1.2

2 Overview of course

What is this course about?

Theory of computing

- What does it mean to compute?
- What kinds of machines compute?
- How much more can computers do as we give them new powers?

Who cares?

1.3

Who cares?

We will keep coming back to that.

Here are a few reasons:

- Philosophical: It is good to have a sense of what computation means
- Mathematical: Mathematical rigour underlying computer science
- Engineering: Many practical problems are about engineering with small resources. How does this limit us?
- Scientific: Some advances will not change the power of computers; others would be enormous. Let's pursue the important ones.

1.4

How we will study theory

We will have a focus on mathematical rigour.

Example: you have already seen a lot of the first few weeks of the course, but without full detail.

- We will prove almost everything we do
- We will work with lots of proof techniques and approaches
- We will work with functions, sets, sets of sets, etc.

1.5

Course myths

This course has several:

- It is too easy
  - You have seen regular expressions in 241.
  - A lot of it sounds reasonable.
  - It is easy to convince yourself you understand the details, even if you do not!
  - This is dangerous!
- It is too hard
  - Some complicated topics in the course come back after you think you are done with them
  - The later part of the course is harder than the beginning

1.6

Other course myths

It is irrelevant

- Problems are like puzzles, not like what happens at your co-op jobs
- Where is the connection between a finite automaton and your desktop machine?
- We are ignoring all the complexity of computation!

But these are myths.

1.7

Only myths

Myths are not always true.
The advantages of our approach:

- We will focus on the important core of a problem, not the implementation details
- Abstraction lets us get at the essence of computing, not the wiring details

And, many CS 360 ideas are used directly in important applications.

- Research in bioinformatics, for example.
- They also come up in parsing, in data compression, and in other areas.

1.8

The real point of 360

We will model computer science problems as mathematical questions. These problems will give rise to formal languages

- Solve a problem = test membership in a language
- Simple languages correspond with simple algorithms

Model of computers: Formal machine models

- We work from simple to complex
- It is fun to see what simple models can do, and what they cannot do

1.9

Complexity of models

We will work from simple to complex.

- Simplest: regular languages, and finite automata
- More complex: Context-free languages, and pushdown automata
- Still more complex: recursive and recursively enumerable languages, and Turing machines

Turing machines are actually more complex than your PC (they have infinite memory).
Actually, your PC is (technically) a finite state machine. It just has lots of states.

1.10

Limitations of computers

For each of these types of models, we will find some language it cannot recognize.
Finite state machines do not have a memory.

- We will find problems that need memory to solve.

A pushdown automaton cannot access its inputs in arbitrary order

- We will find problems that cannot be solved without that ability.

For Turing machines, we will identify problems that cannot be solved, even with infinite memory available.

- This is probably the most important contribution of computer science to Western thought.

1.11

More properties and topics

Families of languages are recognized by a particular computation model

- Regular languages $\leftrightarrow$ finite automata
- Context-free languages $\leftrightarrow$ pushdown automata
- Recursive languages, recursively enumerable languages $\leftrightarrow$ Turing machines

The power of additions to computation models

- Often, small changes have no effect on computing power.
- Other times, perhaps unexpectedly, they do.

1.12

History of the field

This subject mostly dates to the middle third of the 20th century. Perhaps the most interesting person of the many interesting people we will talk about: Alan Turing (1912-1954) He was a

- Cryptographer
- Philosopher
- Mathematician
- Theoretical biologist

Read The Enigma, by Andrew Hodges, or The Man Who Knew Too Much, by David Leavitt, or see the movie The Imitation Game if you want to learn about him.

1.13

End of overview

Models of computation

- Formal representations of problems
- Ever more powerful computation models
- Problems that each model cannot solve

Formal mathematical proof of CS ideas

- Lots and lots of induction proofs
- Mathematical rigour

Reduction

- Solving one problem with the answer to another problem.
- Unsolvable problems

I hope you will have fun!

1.14

3 Administrivia

Administrivia

- Course information (assignments, lecture notes, schedules, etc.) will all be on the course web sites
- The grading scheme for the course is documented in the course outline and on the course websites
- Refer to the schedule page of the unsecured website for all due dates, except that for the final assessment (which will be scheduled by the Registrar)

1.15

Staff

Instructor

- Collin Roberts, cd2rober@uwaterloo.ca
- Office hours: TBD (See LEARN)

The course staff will also be using Piazza to answer student questions.
We are here to help.

1.16

Cheating

Please do not cheat.

- There are very clear expectations on the course outline.
- Do not work in groups.
- I hate undergraduate cheating. Please do not make me have to deal with it.

1.17

4 Mathematical Background

Basic mathematical details

You have seen these before.
CS 360 primarily is concerned with finite-length strings over a finite alphabet.

- Alphabet: Finite set of symbols, usually denoted Σ
  Examples:
  - Binary alphabet: $\Sigma = \{0,1\}$
  - Latin alphabet: $\Sigma = \{a,b,\ldots,z\}$
  - Unary alphabet: $\Sigma = \{1\}$.
- A string is an ordered sequence of alphabet symbols. Strings are also called words.
  Examples of words: rover, 1010110
- Denote the length of the string x by $|x|$.
  In CS 360, all of our strings all have finite length.
- Empty string: ε
  Length of the empty string: $|\varepsilon| = 0$.

1.18

Manipulating strings

- Prefixes: The first k characters of a string v, for some $k \geq 0$
- Suffixes: The last k characters of a string v, for some $k \geq 0$
- Substrings: A string z that is found as consecutive characters in a string v. A substring can be fully described by its starting position and length, or by its starting and ending positions.

Examples: Suppose $v = temptation$

- Prefixes: t, $tempt$, ε, etc.
- Suffixes: n, $ation$, $temptation$, ε, etc.
- Substrings: All of these, and also tat, $mptat$, etc.

Question: How many non-empty substrings does a string v of length k have? (N.B. we are only interested in counting the possible choices, not enforcing that the substrings obtained be distinct form one another.) Answer: $\frac{k(k+1)}{2}$. Can you derive this expression for yourself?

1.19

Other string operations

Concatenation:

- One string after another
- Example: if $x = car$, $y = rot$, then $xy = carrot$.
- Not surprising: if $x = \varepsilon$, then $xy = y$.
- Formal definition of prefix:
  x is a prefix of y if $y = xz$, for some string z.
- Length of a concatenation equals sum of lengths: $|xy| = |x| + |y|$.

Power operation:

- x^k: k copies of string x, concatenated.
- Example: if $x = 010$, then $x^3 = 010010010$.
- Convention: $x^0 = \varepsilon$, for ease in recursive definitions.

1.20

More string operators

- Counting members:
  $n_b(x) = \#$ of times alphabet symbol b is found in x.
  Example: $n_b(banana) = 1$, $n_a(banana) = 3$.
  Question: Is $n_\varepsilon(x)$ well-defined, for some string x? Answer: No. Formally, ε is never included in any alphabet. Practically, there is no way to make this expression unambiguous. Asking for $n_\varepsilon(x)$ is analogous to asking how many factors of 1 can be pulled from some integer z.
- Reversing strings:
  $x^R = x$, written in reverse order. Example: if $x = stressed$, then $x^R = desserts$. Not x to the power of R!
- Palindrome: x a palindrome if $x = x^R$.

1.21

5 Languages

Language

Σ^*: All finite strings over alphabet Σ
Language: a subset of Σ^*.

- That is, a set of strings over Σ.
- Some languages are finite; others are not.

Examples:

- $\{1, 01, 001, 001, 0001, \ldots\}$
- $\{cat, dog, hamster\}$
- $\{aa, abab, abbabb, abbbabbb, \ldots\}$
- $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$
- $\emptyset$

The second last of these is just $\{0,1\}^*$.
We can describe languages by rules, or by listing all their elements if they are finite.

1.22

Classes of languages

Class of languages: A set of languages.

- Note: a set of sets of words.

Example:

$$\begin{aligned} \mathscr{C} = \{&\{1, 11, 111, 1111, 11111, \ldots\}, \\ &\{0, 00, 000, 0000, 00000, \ldots\}, \\ &\{0, 1, 00, 11, 000, 111, \ldots\}, \\ &\{\varepsilon\}\} \end{aligned}$$

We will care about classes with something in common. (Example: $\mathscr{C} = \{$all languages with ten words$\}$.)

(This would be hard to argue about the class $\mathscr{C}$ that contains the four languages above!)

Remark: Languages in a class need not be disjoint. The above example demonstrates this. 1.23

Important details about classes of languages

Important: A string is not a member of a class of languages.

- Strings are members of languages.
- Languages are members of classes.
- The only class that is also a language: $\emptyset$

Also, note:

- ε is a word.
- $L = \{\varepsilon\}$ is a language.
- The language L can be part of a class of languages (it is part of $\mathscr{C}$).
- ε cannot be part of a class of languages. (It is a word.)
- Letting $L_1 = \{\varepsilon\}$ and $L_2 = \emptyset$, we have that $L_1 \neq L_2$, as $|L_1| = 1 \neq 0 = |L_2|$. 1.24

Language operations

Language: set of strings

- We can take unions and intersections of languages.
- Also: complement: $L' = \Sigma^* \setminus L$.
  - L' depends on what Σ is!
  - Recall, $L \subseteq \Sigma^*$ by definition.
  - Hence $L' = \emptyset$ if and only if $L = \Sigma^*$.
- Concatenation: If L and M are languages, then $LM = \{xy | x \in L, y \in M\}$.
  - Example: $L = \{love, true\}$, $M = \{\varepsilon, ly, r\}$ $LM = \{love, true, truely, lovely, lover, truer\}$ 1.25

Powers of languages

- Finite powers:
  - L^k is k copies of L, concatenated.
  - $L^0 = \{\varepsilon\}$.
  - $L^1 = L$.
  - If $L = \{sorta, kinda\}$, then $L^2 = \{sortasorta, kindakinda, sortakinda, kindasorta\}$
- Kleene star operator (so named after S.C. Kleene):
  - $L^* = \bigcup_{i=0}^{\infty} L^i$.
  - $L^+ = \bigcup_{i=1}^{\infty} L^i$.
  - If $L = \{sorta, very\}$, then
    - $L^* = \{\varepsilon, sorta, very, sortavery, verysorta, \ldots\}$, and
    - $L^+ = \{sorta, very, sortavery, verysorta, \ldots\}$.
  - If $M = \{1\}$, then
    - $M^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$, and
    - $M^+ = \{1, 11, 111, 1111, \ldots\}$.
  - If $N = \{\varepsilon, 1\}$, then
    - $N^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$, and
    - $N^+ = \{\varepsilon, 1, 11, 111, 1111, \ldots\}$.
    - Note, $N^* = N^+$ for this choice of N.
  - It is generally true that $L^+ = LL^*$.
  - The proof is left as an exercise. 1.26

How to define languages

Can describe languages using these sorts of operations:
Example:

- $M = (L \cap N)^* OP$ is a language.
- Each word in M consists of:
  - A concatenation of a finite number of strings from $L \cap N$,
  - followed by a string from O,
  - followed by a string from P.

Very important: Keep your set notation clear, so that you know when you are dealing with sets, strings or symbols.

- This is especially a pain for ε and $\{\varepsilon\}$, and $\emptyset$.

1.27

Proofs about languages

Three important proof ideas:

- To prove set A equals set B, show $A \subseteq B$ and $A \supseteq B$.
- To prove a statement is false, give a counterexample.
- To prove a statement is true for all members of a set A, use induction. Start with simple objects of A, move to complicated ones.
  Quote from a TA: "CS 360 is just one long induction proof." (This will not be true later in the course.)

1.28

Technique 1: equality by subsets

Given: languages L_1, L_2 and L_3.
Is $L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3$ true for all choices of L_1, L_2 and L_3?
Yes.
Proof: We show each language is a subset of the other, so they must be equal.

- Let x be a word in $L_1(L_2 \cup L_3)$.
- We show it is also in $L_1L_2 \cup L_1L_3$; thus, $L_1(L_2 \cup L_3) \subseteq L_1L_2 \cup L_1L_3$.
- Since $x \in L_1(L_2 \cup L_3)$, we know $x = yz$, for some $y \in L_1$ and $z \in L_2 \cup L_3$.
- If $z \in L_2$, then $x = yz \in L_1L_2$, while if $z \in L_3$, then $x = yz \in L_1L_3$.
- In both cases, $x \in L_1L_2 \cup L_1L_3$.

(The other direction is similarly easy and is left as an exercise.)

1.29

Technique 2: Proof by counterexample

Given: Languages L_1, L_2, L_3. Is $L_1(L_2 \cap L_3) = L_1L_2 \cap L_1L_3$ always true for all choices of L_1, L_2, L_3?
Answer: No.

- Counterexample:
  - $L_1 = \{a, ab\}, L_2 = \{b\}, L_3 = \{\varepsilon\}$.
    Since $L_2 \cap L_3 = \emptyset$, $L_1(L_2 \cap L_3) = \emptyset$.
    (Note: $L\{\varepsilon\}$ is different from $L\emptyset$!)
    But $L_1L_2 = \{ab, abb\}$, and $L_1L_3 = \{a, ab\}$, so $L_1L_2 \cap L_1L_3 = \{ab\}$.
    And $L_1(L_2 \cap L_3) \neq L_1L_2 \cap L_1L_3$
- It is not true for all L_1, L_2, and L_3.
- It is not a theorem, period.

Remember: A single case that is not true proves a conjecture false.

1.30

Technique 3: induction

Three kinds of induction:

- Weak induction If hypothesis is true for k, and if being true for ℓ implies being true for $\ell+1$, then it is true for all $\ell > k$.
- Strong induction If hypothesis is true for k, and if being true for all m between k and ℓ implies being true for $\ell+1$, then it is true for all $\ell > k$
- Structural If hypothesis is true for simple objects of a given type, and it is true for how we combine them to make new objects of the type, then it is true for all objects of a given type.

1.31

Induction examples

We will use induction over and over. Here are some first theorems.
Theorem: If $L^2 \subseteq L$, then $L^+ \subseteq L$. (Remember: $L^+ = LL^*$.)
Proof:

- We want to show that $LL^i \subseteq L$ for all $i \geq 0$.
- The proof is by induction on i.
- Base cases:
  - $i = 0$. This is just $L = L$.
  - $i = 1$. That is the theorem's hypothesis, that $L^2 \subseteq L$.
- Induction ($i > 1$):
- Induction hypothesis: $L^2 \subseteq L$ implies $LL^k \subseteq L$, for $k = i-1$
- Need to show: If the induction hypothesis is true, then $LL^i \subseteq L$.

1.32

First induction example, continued

- Induction hypothesis: $L^2 \subseteq L$ implies $LL^{i-1} \subseteq L$.
- Need to show: If induction hypothesis is true, then $L^2 \subseteq L$ implies $LL^i \subseteq L$.
- Let $x \in LL^i$ be arbitrary.
  - Then $x = yz$, where $y \in L$ and $z \in L^i = LL^{i-1}$.
  - By the induction hypothesis, $z \in LL^{i-1} \subseteq L$, in other words $z \in L$.
  - So $yz \in L^2$.
  - But the hypothesis of the Theorem is that $L^2 \subseteq L$.
  - So $x = yz \in L$, and this proves that $LL^i \subseteq L$.
- Remember: we have to formally state what we are trying to prove, and how.

1.33

Another example with reversals

Recall: x^R is x, written backwards.
Theorem: $(xy)^R = y^R x^R$.
Proof: By induction on $|y|$:

- Base case ($|y| = 0$): Then $y = \varepsilon$, so that

$$(xy)^R = (x\varepsilon)^R = x^R = \varepsilon x^R = \varepsilon^R x^R = y^R x^R.$$

- Induction step ($|y| = k > 0$): The induction hypothesis is that $(xz)^R = z^R x^R$ holds for all z of length $< k$. We need to argue that this implies $(xy)^R = y^R x^R$.
- Write $y = za$, for a single letter a. (We can do this, because $|y| > 0$.)
  Then $xy = xza$, and

$$(xy)^R = (xza)^R \underbrace{=}_{\text{reversal}} a(xz)^R \underbrace{=}_{\text{I.H. for } z} a(z^R x^R) \underbrace{=}_{\text{reversal}} (za)^R x^R = y^R x^R.$$

1.34

Recursive structure, induction

Many objects are defined recursively.

Examples:

- Fibonacci numbers
- Sequences of letters (either empty, or a sequence of letters followed by another letter)
- Binary trees (either empty, a single leaf, or the result of combining two binary trees with a new root)
- And so on. (Regular languages, etc.)

Structural induction: Show a theorem true for simple objects, and true for however objects are combined.

You have seen a lot of this in CS 135/136/245/... 1.35

Example of structural induction

(We will actually come back to structural induction later.)

Let L be the language over $\Sigma = \{+, a, (,)\}$ that constructed according to these rules:

- $a \in L$
- If w_1 and w_2 are in L, so is $(w_1 + w_2)$ (including the parentheses).

Examples of words in L: $a, (a+a), (a+(a+a)), ((a+a)+(a+a)), \ldots$

Theorem: No string in L contains)(. Proof: by structural induction. Let $w \in L$ be arbitrary.

- Base case: $w = a$ does not contain)(, so the base case holds.
- Induction step: As we are no longer in the base case, we have that $w = (w_1 + w_2)$ for some $w_1, w_2 \in L$, each constructed in strictly fewer steps than w.
- The induction hypothesis applied to both of w_1, w_2 then says that neither w_1 nor w_2 contains)(.
- Now we must use this assumption to argue that $(w_1 + w_2)$ contains no)(. 1.36

Proof by structural induction

Proof of induction:

- To show: no) in $(w_1 + w_2)$ comes right before a (.
- Certainly, the last) cannot; it is the last letter.
- And no) in w_2 comes directly before a (, by the induction hypothesis.
- This is also true of all the)'s in w_1, by the induction hypothesis. The last) in w_1 now may come before a +, but not before a (.
- And that is all of the)'s that can occur in $(w_1 + w_2)$.
- This completes the induction step.
- So no member of L contains)(. 1.37

Structural induction: main idea

- Show that a theorem is true for simple objects, and that complicated objects are just simple objects put together.
- It is sometimes easy to forget:We do not need to count steps in a recursive definition; instead just put "simpler" objects together.

We will use structural induction over and over in the first two thirds of the course:

- Regular languages are defined recursively!
- Context-free grammars are inherently recursive. 1.38

Wrap-up

Module 1 goals:

- Overview of the basic goals of the course
- Boring administrivia
- Background: alphabets, strings, languages, proof techniques

Next module:

- Finite automata, regular languages and regular expressions 1.39