Today's lecture: Course Overview

I. What is the "Theory of Computing"?
II. Basic course information
III. Machines and Languages

Textbook

Introduction to the Theory of Computation by Michael Sipser.

Schedule:

September 5: Mathematical Background – Read: *Sipser* chapter 0
Homework 0 goes out (due Sept. 12).

September 8: Finite Automata – Read: *Sipser* 1.1.
Lecture will cover through Example 1.15 (i.e. pages 31–40).

September 10: Regular Languages.
The rest of *Sipser* 1.1 (i.e. pages 40–47).

September 12: Non-Determinism – Read: *Sipser* 1.2.
Lecture will cover through Example 1.35 (i.e. pages 47–52).
Homework 0 due. Homework 1 goes out (due Sept. 19).

September 15: NFAs
Lecture will cover through Example 1.38 (i.e. pages 53-54).
Homework 0 due.

September 17: Equivalence of DFAs and NFAs.
The rest of *Sipser* 1.2 (i.e. pages 54–63).

September 19: Regular Expressions – Read: *Sipser* 1.3.
Lecture will cover through Example 1.58 (i.e. pages 63-59).
Homework 1 due. Homework 2 goes out (due Sept. 26).

September 22: Equivalence of DFAa and Regular Expressions.
The rest of *Sipser* 1.3 (i.e. pages 63–76).
Homework 1 due.

September 24: Nonregular Languages – Read *Sipser* 1.4.
Lecture will cover through Example 1.73 (i.e. pages 77-80).

September 26: Pumping Lemma Examples.
The rest of *Sipser* 1.4 (i.e. pages 80–82).
Homework 2 due. Homework 3 goes out (due Oct. 3).

September 29: Introduction to Context Free Languages – *Sipser* 2.1.
Lecture will cover through "Designing Context-Free Grammars" (i.e. pages 99-105).

October 1: Chomsky Normal Form.
The rest of *Sipser* 2.1 (i.e. pages 105–109).

October 3: Push-Down Automata – Read *Sipser* 2.2.
Lecture will cover through page 114, "Examples of PushDown Automata."
Homework 3 due. Homework 4 goes out (extra credit).

October 6: Midterm Review.

October 8: Midterm 1: Regular Languages.
This midterm will cover anything we've covered about regular languages. Context-free languages will not be included on this exam.

October 10: Equivalance of PDAs and CFGs.
The rest of *Sipser* 2.2 (i.e. pages 115–122).
Homework 4 due. Homework 5 goes out.

October 15: Non-context free languages – Read *Sipser* 2.3.

October 17: Everything else about CFLs.
Homework 5 due. Homework 6 goes out.

October 20: Introduction to Turing Machines – Read *Sipser* 3.1.

October 22: Turing machine variants. Read *Sipser* 3.2.

October 24: Algorithms and Turing Machines Read *Sipser* 3.3.
Homework 6 due. Homework 7 goes out.

October 27: Decidability – Read *Sipser* 4.1.

October 29: The halting problem – Read *Sipser* 4.2.
Lecture will cover through Figure 4.21.

October 31: More on decidability.
Homework 7 due. Homework 8 goes out (extra credit).

November 3: Midterm Review.

November 5: Midterm 2.
This midterm will cover anything we've covered about context-free languages and the basics of Turing machines (i.e. *Sipser* chapter 3). A knowledge of regular languages will be assumed, but will not be the focus of the exam (I won't promise whether there will, or will not be any questions about regular languages on the exam). Decidability and the halting problem will not be on the midterm.

November 7 – 14: Reducibility – *Sipser* chapter 5.
Nov. 7: Homework 8 due. Homework 9 goes out.
Nov. 14: Homework 9 due. Homework 10 goes out.

November 17 – November 28: To be decided.
Possible topics include: cryptography, NP completeness, proving software correct, quantum computation, molecular computation, catching up because some topics took longer than I planned, . . .
Nov. 21: Homework 10 due. Homework 11 goes out.
Nov. 28: Homework 11 due.

I. Course Mechanics

A. Web page: http:www.ugrad.cs.ubc.ca/~cs421
Newsgroup: ubc.courses.cpsc.421

B. People:

1. Instructor: Mark Greenstreet
Contact information:

office:	CICSR/CS 323
e-mail:	mrg@cs.ubc.ca
office hours:	in my office
	Monday 9am-10am
	Thursdays 10am-11am
	Or by appointment

What I like to do:
My research is in how to design the integrated circuits that are used in today's computers. I work with designers at IBM, Intel, Rambus and SUN Microsystems. I'm particularly interested in how to make mathematical proofs that hardware works right and in designing very fast circuits. Recently, I've started some research into parallel computer architecture. I'll be teaching UBC's first undergraduate course on parallel architectures and programming (listed as CpSc 448B) next term.

2. TA's: Brad Bingham

Who	Office Hours	e-mail
Brad Bingham	You decide – we'll vote in class.	binghamb@cs.ubc.ca

Office hours: Jan: Tuesdays 11am-12noon.

3. Grading

Homework	25%
Midterm	30%
Final Exam	45%

a. How many homework assignments will there be?
My plan is to have out one assignment every week. The homework for the two weeks with midterms will be entirely extra credit. This means that there will be 10 normal assignments and two extra-credit ones.

b. When is homework due?
At the beginning of class on the due date. Homework is due on Fridays, and late homework is loses credit as follows:

Turned-in in class	Full credit
In my mailbox or e-mail by 8am, Friday morning	Full credit
Turned in by 4:30pm on Friday:	90% credit
Turned in by 12pm (noon) on Monday:	80% credit
Turned in by 12pm (noon) on Tuesday:	70% credit
Turned in later than noon on Tuesday:	0% credit

If you are turning in late homework, please take it to the computer science main office, get the receptionist to stamp it and write down the time and date when it was turned it, and put it in my mailbox.
Note: no late homework will be accepted for assignments 3 or 7 so I can post solutions for studying for the midterms.

4. Midterm:
There will be two midterms. The first will be on October 8, and the second will be on November 5. Both will be in class (i.e. 8-9am). Contact me by September 17 if you cannot be present on one or both of these dates.

5. Academic Misconduct
I have a very simple criterion for plagiarism: submitting the work of another person, whether that be another

student, something from a book, or something off the web and representing it as your own is plagiarism and constitutes academic misconduct. If the source is clearly cited, then it is not academic misconduct.
If you tell me "This is copied word for word from Jane Foo's solution" that is not academic misconduct. It will be graded as one solution for two people and each will get half credit. I guess that you could try telling me how much credit each of you should get, but I've never had anyone try this before.
I encourage you to discuss the homework problems with each other outside of class. You can help each other understand the concepts and techniques needed to solve the problems, and you can brainstorm together to figure out ideas for solving the problem. If you come up with the actual solution yourself, that's fine. If you're brainstorming with some friends and the key idea comes up, that's OK too – in this case, just add a note to your solution that says something along the lines of "{*Names of people in brainstorm session*} and I were discussing this problem together and came up with the idea of {*state the key idea from the brainstorm here*}".
If you say that you got it off of the web or from another text, you'll be graded by the extent to which your solution shows that you actually understood the solution that you found and were able to reformulate it using your own reasoning.
If you get a solution by any means other than working it out yourself and don't disclose it, then I will follow the university procedures for academic misconduct.
See also: http://members.aol.com/quentncree/lehrer/lobachev.htm.

6. Bug Bounties

I have a simple bug bounty policy: if I make a mistake, you get extra credit. If you find an error in a homework assignment or the lecture notes, post it to the course newsgroup. The first person to post the bug gets the bounty. If the error would prevent you from solving the problem, you get extra credit equal to the value of the problem. If it is a more minor error (or an error in the notes), I'll determine the number of extra credit points according to my sense of the severity of the error.

II. Languages, Machines and the Theory of Computation

A. Languages:

1. Alphabets, strings, etc.

a. An *alphabet* is a finite set of symbols. The symbols can be arbitrary – there's nothing special about them. We'll often write Σ to represent this set.

b. A *string* is a sequence of zero or more symbols. At this point it's reasonable to ask why we work with strings, rather than arrays – wouldn't it be more convenient if we had arrays where we could index into them and access whatever part we wanted?

c. We write Σ^* to denote the set of all strings composed of zero or more elements from alphabet Σ. Note that Σ^* is a set with an infinite number of elements (if Σ has at least one element.

d. A *language* is a subset of Σ^*. It is the set of all strings that have some property.

e. **Note:** For a programmer, arrays are very convenient. On the other hand, they create some additional difficulties for our simple models. How big can an index be? Up to 2^8? Up to 2^{64}? Up to 10^{1000}? If we pick any finite limit, then we end up with a maximum length string in our language. What we can figure out about machines and languages will depend on this limit. Rather than picking some arbitrary limit, we will allow strings to be arbitrarily long. If we use array that can be arbitrarily large, then we must allow indices to be arbitrarily large as well. If the machine can perform operations on indices, then it can do an arbitrarily large amount of work in a single operation (think of multiplying two $1,000,000,000,000$ digit numbers by each other). This seems unrealistic. So, we work with strings. The machine can access the next symbol of the string. Some machines that we study can only look at the string once, a single, left-to-right pass. Other machines will allow the tape head to move back and forth, one tape square at a time. Either way, the machine does a fixed amount of work in each step regardless of the length of the input.
As we'll see later in the term, even with these restrictions, there are machines with strings that can do anything we can do with arrays; it's just takes more steps.

2. Examples of languages:

a. Natural language: Arabic, Bengali, Chinese, Danish, English, French, Greek, Hindi, Italian, Japanese, ...

Properties we might look for in a string

i.. The string is a word in the language.

ii.. The string is a grammatically valid sentence in the language.

iii.. The string is a true statement.

iv.. The last one is in some ways the most interesting, but it's also the hardest. A machine would have to everything about anything that can be described in the language.

b. Mathematics: we can write formal, mathematical propositions or proofs. Properties we might look for:

i.. The string is syntactically correct.

ii.. The string is a true statement.

iii.. The string is a valid proof.

c. Even simpler languages:

i.. Strings that end with `0`.

ii.. Strings with an equal number of `1`'s and `0`'s.

iii.. Strings that are the binary representation of a prime number.

iv.. Strings that are syntactically correct Java programs.

B. Machines

1. A typical computer

a. Lots of I/O devices

b. Complicated hardware

c. Changes with hardware and software advancements

d. All of these make it unsuitable for a formal model

2. Simple models

a. A two tape machine

i.. A single input device: a *tape* that holds a sequence of symbols from a fixed alphabet.

ii.. A single output device: another "tape".

iii.. The machine transforms the input sequence into an output sequence. For example, the input sequence could be a list of numbers, and the output could be their sum, or could be the same numbers sorted into ascending order. Or, the input sequence could be a list of coordinate pairs, and the output could order them into the shortest "tour" that each point (i.e. a solution to the "Traveling Salesman" problem).

b. A decision process

i.. A single input tape as above.

ii.. The machine answers a "yes or no" question about the tape.

iii.. The inputs for which it answers "yes" are said to be the *language* of the machine.

C. Theory of Computation?

1. It's the mathematical formalization of computers:

a. What is a computer?

i.. What are the bare essentials of something that computes?

ii.. When are two computers equivalent to each other?

iii.. When are two computers different?

iv.. What are classes of equivalent computers?

b. What are the capabilities of a computer?

i.. What can a computer do?

ii.. What are computers unable to do?

iii.. What are "easy" problems for a computer?

iv.. What are "hard" problems for a computer?

v.. What are classes of equivalent problems?

2. Topics covered in this course:

a. Finite automata and regular languages (~ 3 weeks)

b. Pushdown automata and context-free languages (~ 2 weeks)

c. Turing Machines, unrestricted grammars, decidability (~ 4 weeks)

d. Other topics (~ 3 weeks)

Midterms, review, holidays, and hopefully time for a one of complexity theory, cryptography, hardware verification, quantum computation, etc.

3. Connections to other areas of computer science

a. Programming languages and compilers

i.. describing the syntax of programming languages

ii.. automatically generating lexical analysers and parsers

b. Hardware and software verification

i.. Showing that a circuit implements its specification:

Example: showing that the caches of the CPUs in a multi-processor hold consistent copies of data.

ii.. Verifying properties of network protocols:

Examples: showing freedom from deadlock, proper authentication, etc.

iii.. Promising results are now arriving for software verification as well:

Verifying that arrays an pointers are used properly, that variables are properly initialized, etc.

c. Computer security

i.. Cryptography

ii.. Virus creation and detection

iii.. Authentication