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 8: Mathematical Background – Read: *Sipser* chapter 0
Homework 0 goes out (due Sept. 18).

September 11: Finite Automata – Read: *Sipser* 1.1.
Lecture will cover through Example 1.15 (i.e. pages 31–40).

September 13: Regular Languages.
The rest of *Sipser* 1.1 (i.e. pages 40–47).

September 15: Non-Determinism – Read: *Sipser* 1.2.
Lecture will cover through Example 1.35 (i.e. pages 47–52).
Homework 1 goes out (due Sept. 25).

September 18: NFAs
Lecture will cover through Example 1.38 (i.e. pages 53-54).
Homework 0 due.

September 20: Equivalence of DFAs and NFAs.
The rest of *Sipser* 1.2 (i.e. pages 54–63).

September 22: Regular Expressions – Read: *Sipser* 1.3.
Lecture will cover through Example 1.58 (i.e. pages 63-59). Homework 2 goes out (due Oct. 2).

September 25: Equivalence of DFAa and Regular Expressions.
The rest of *Sipser* 1.3 (i.e. pages 63–76).
Homework 1 due.

September 27: Nonregular Languages – Read *Sipser* 1.4.
Lecture will cover through Example 1.73 (i.e. pages 77-80).

September 29: Pumping Lemma Examples.
The rest of *Sipser* 1.4 (i.e. pages 80–82). Homework 3 goes out (due Oct. 16).

October 2: Introduction to Context Free Languages – *Sipser* 2.1.
Lecture will cover through "Designing Context-Free Gramamars" (i.e. pages 99-105).
Homework 2 due.

October 2: Ambiguity and Chomsky Normal Form.
The rest of *Sipser* 2.1 (i.e. pages 105–109).

October 6: Midterm Review.

October 11: Midterm 1: Regular Languages.
Including: DFAs, NFAs, Regular Expressions, the Pumping Lemma.

October 13 – 20: The rest of CFLs – *Sipser* 2.2 & 2.3.

October 23 –Nov. 3: Turing Machines and the Halting Problem – *Sipser* 3 & 4.

November 6 – 8: Introduction to Reducibility – *Sipser* 5.1.

November 10: Midterm Review.

November 15: Midterm 2.

November 17 – 24: More Reducibility – *Sipser* 5.2 & 5.3.

November 27 – December 1: To be Decided
Possible topics include: cryptography, proving software correct, quantum computation, molecular computation, catching up because some topics took longer than I planned, . . .

I. What is the Theory of Computation?

A. It's the mathematical formalization of computers:

1. What is a computer?

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

b. When are two computers equivalent to each other?

c. When are two computers different?

d. What are classes of equivalent computers?

2. What are the capabilities of a computer?

a. What can a computer do?

b. What are computers unable to do?

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

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

e. What are classes of equivalent problems?

B. Topics covered in this course:

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

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

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

4. Other topics (~ 3 weeks)
Midterms, review, holidays, and hopefully time for a one of complexity theory, cryptography, hardware verification, quantum computation, etc.

C. Connections to other areas of computer science

1. Programming languages and compilers

a. describing the syntax of programming languages

b. automatically generating lexical analysers and parsers

2. Hardware and software verification

a. Showing that a circuit implements its specification:
Example: showing that the caches of the CPUs in a multi-processor hold consistent copies of data.

b. Verifying properties of network protocols:
Examples: showing freedom from deadlock, proper authentication, etc.

c. Promising results are now arriving for software verification as well:
Verifying that arrays an pointers are used properly, that variables are properly initialized, etc.

3. Computer security

a. Cryptography

b. Virus creation and detection

c. Authentication

II. 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
	Thursdays 11am-12noon
	Fridays 9am-10am
	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.

2. TA's: Mohammad Ali Safari and Jan Ulrich

Who	Office Hours	e-mail
Mohammad Safari	To be announced	safari@cs.ubc.ca
Jan Ulrich	Tuesdays, 11am-12noon	ulrichj@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 hand out one assignment every other week, but skip the two weeks that have midterms, i.e. no homework will be due during the week of the midterm. This means there will be about 10 assignments in total.

b. When is homework due?
At the beginning of class on the due date. Homework that is late loses 10% per day rounded up to the nearest day. Homework will not be accepted after 4:30pm two days after the due date. Typically, homework will be due on Monday; thus the late penalties are as follows:

Turned-in in class	Full credit
In my mailbox or e-mail by 8am, Monday morning	Full credit
Turned in by 4:30pm on Monday:	90% credit
Turned in by 4:30pm on Tuesday:	80% credit
Turned in by 4:30pm on Wednesday:	80% credit
Turned in later than 4:30pm on Wednesday:	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.

4. Midterm:

There will be two midterms. The first will be on October 11, and the second will be on November 15. Both will be in class (i.e. 8-9am). Contact me by September 20 if you cannot 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.

III. Machines and Languages

A. 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.

B. 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 proposations 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.