Today's lecture: Course Overview

I. Basic course information
II. Getting started: a few examples

Textbook

Automata and Computability by Dexter Kozen. As an alternate, you can consider *Introduction to the Theory of Computation* by Michael Sipser. I will give the corresponding chapters for the two books when I assign readings. I think that both texts are excellent – which one is better for you is probably a matter of personal taste. If you have access to both, look at them and decide which you like better.

Reading:

September 7: Introduction
Read: *Kozen* lectures 1 & 2 or*Sipser* chapter 0
OK, I know that today is already Sept. 7. This is background material. Take a look and get an idea of what it covers. That way, if sometime in the next few weeks you find that you need a refresher on one of these topics, you'll know where to find it.

September 9: Finite Automata – Read: *Kozen* lecture 3 or *Sipser* 1.1.

September 12: Regular Sets – Read: *Kozen* lecture 4 (or *Sipser* 1.1. as before).

September 14: Non-Determinism – Read: *Kozen* lecture 5 or *Sipser* 1.2.

September 16: Equivalence of DFAs and NFAs – Read: *Kozen* lecture 6 (or *Sipser* 1.2. as before).

September 19: Regular Expressions – Read: *Kozen* lectures 7 & 8 or *Sipser* 1.3.

September 21: Equivalence of Regular expressions and Finite Automata Read: *Kozen* lecture 9 (or *Sipser* 1.3. as before).

September 23: Nonregular Languages – Read: *Kozen* lecture 12 or *Sipser* 1.4.

September 26: More examples of the pumping Lemma Read: *Kozen* lecture 13 or *Sipser* 1.4.

September 28: Applications of finite automata

September 30: More applications

October 26: Midterm: In class.

I. Basic Course Information

A. What is the Theory of Computation?

1. It's the mathematical formalization of computers:

a. 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?

e. 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. It’s the basis for practical applications of computers

- **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

3. Topics covered in this course:

- **a.** Finite automata and regular languages (~ 4 weeks)
- **b.** Pushdown automata and context-free languages (~ 3 weeks)
- **c.** Turing Machines, unrestricted grammars, decidability (~ 4 weeks)
- **d.** Other topics (~ 2 weeks)
  Possibilities include, an introduction to: complexity theory, cryptography, hardware verification, quantum computation, etc. (but only one of the above, depending on time available).

B. Web page: http:www.ugrad.cs.ubc.ca/∼cs421
Should be on-line very soon.

C. People:

1. Instructor: Mark Greenstreet
Contact information:

office:	CICSR/CS 323
e-mail:	mrg@cs.ubc.ca
office hours:	Wednesdays 9-10am
	Thursdays 11am-12 noon
	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 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: Albert Wong and Yury Kholondyrev

3. Grading

Homework	20%
Daily Question	10%
Midterm	25%
Final Exam	45%

a. What's the "Daily Question"?

I will assign one question for each lecture. The question will be intended to be easy – think of it as a warm-up for the homework problems. If you do the assigned reading, you should be able to do the "Daily Question" in under ten minutes. I'm doing this instead of having quizzes. Doesn't everyone hate quizzes?

The challenge is how to grade these fairly, and not to have to deal with a bunch of doctor's notes and other excuses every lecture. Here's my solution. Let's say that by the end of the term, there have been n daily questions, each graded on a scale of $0 \ldots M$ points, and let T be your total score for these problems. Then, your score for this part will be:

$$\min\left(\sqrt{1.1\frac{T}{n*M}},\ 1\right) * 10\%$$

Note the built-in extra credit. You can miss up to 10% of the Daily Questions without having it affect your grade – i.e. no need to bother with excuses for missing class. If you know that you're going to miss a class, you can send in your solution by e-mail or put it in my mailbox with a data and time-stamp from the office staff before class starts. No credit will be given for late solutions – I intend to post solutions for each question shortly after the class in which it was due.

Pay attention to what the square-root does. You get lots of points for the first few that you do, and very little for the final ones if you've already aced all of the others. In other words, if you are struggling, then this is great source of points. On the other hand, if you're already cruising to an $A+$, then relax, you can skip a few and it won't affect your grade.

b. How many homework assignments will there be?

My plan is to hand out one assignment every other week.

c. 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 that is one week or more late gets no credit.

4. Midterm:

The midterm will be on October 26, in class.

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.

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 from somewhere else and don't disclose it, then I will follow the university procedures for academic misconduct.

II. Getting Started: a few examples

A. Languages

1. Types of languages

a. Natural languages
b. Programming languages
c. Formal languages

2. Formal languages

a. A set of symbols called the "alphabet", Σ.
Example: $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.

b. A string is a sequence of zero or more symbols from the alphabet. A string of length 0 is called the "empty" string and is written ϵ. We write Σ^* to denote the set of all strings composed of zero or more symbols from Σ.

c. A language is a subset of the set of all strings: $L \subseteq \Sigma *$.

d. Examples:

i.. The set of all strings where every 'a' is followed immediately by a 'b'.
ii.. The set of all strings where left and right parentheses are "balanced"; in other words, every prefix of the string has at least as many left parentheses as right, and the total number of left and right parentheses in the string are equal.
iii.. The set of all strings that have more 'a's than 'b's.
iv.. The set of all strings over the English alphabet that are statements of valid mathematical theorems.

B. Computing structures

1. logic gates

a. A circuit with two inputs whose output is true if both inputs are true...
b. A circuit with two inputs whose output is true if either input is true...
c. A circuit with two inputs whose output is true if an even number of the inputs are true...
d. A circuit with three inputs whose output is true if the majority of its inputs are true...
e. Repeat each of the above with n inputs.
f. The capabilities of the logic gate model:
i.. What can we compute?
ii.. What cannot be computed?
iii.. What is missing?

2. circuits with state

a. Circuits with a fixed number of state-holding elements
i.. What can they compute?
ii.. What cannot be computed?
iii.. What is missing?
b. Circuits with unlimited memory
i.. How is it accessed: stack, FIFO, arbitrary
ii.. What can be computed?
iii.. What cannot be computed?

C. Combining the two

1. Different computing structures can recognize different classes of languages. We can study the classes of languages to see what each includes, and what is beyond its capabilities. This then lets us understand what can be accomplished by various mechanisms for computation and what is impossible. We can also take a more nuanced approach and find what can be computed efficiently, and what cannot.

2. For each structure, we can identify problems that can be solved and problems that are unsolvable.

3. If a new structure is proposed, we can classify it relative to the structures that we already have. Often, new proposals can be shown to be equivalent to something that we've already studied, in which case we can use everything we know about the earlier version.

4. The Church-Turing thesis is a conjecture originally stated in the 1930s (1935-37 depending on the version that you choose) that anything that can be computed by any machine can be computed by a Turing machine. Philosophers continue to debate what "any machine" means. Church and Turing certainly had a narrower, mathematical sense of what it meant than we now have. What is remarkable is that there is no demonstrated counter-example to the Church-Turing thesis no matter what model we use. In some very strong sense, a Turing machine seems to be the most powerful possible model of computation that can be physically realized.

A Turing machine is a very minimal and simple computer. More to the point, the mechanism is very simple, which makes programming it very convoluted. It's easy to see what the machine does at each step, but more subtle to see that this is enough to implement any computation.

5. While this is the framework, there remain many intriguing open questions of profound significance. For example, computer cryptography hangs on a thread that there are problems that are very hard to compute unless you know a particular "secret" and very easy to compute with the secret. Other areas of intense work include quantum computation, zero-knowledge proofs, applications to verification, and many others.