CSCI/MATH 361
Lecture 1

Welcome.

Roster.

Review of Syllabus. Please note:

- me - where to find me, how to find me, when to find me
- course description
- text, especially the differences between the first and second editions
- exams and homeworks

  two sets of problems with each lecture (occasionally there will be no homework)

  in-class exam *after Reading Period*

  take-home exams

- grades

  percentages from exams and homeworks

  late policy

  attendance

  collaboration on homework – ok!, but don't allow it to become a crutch

- Your responsibilities
- TA – Chris Douglas. Office hours: S, T, Th 9-10:30, CS lounge or 312
- other resources: http://www.cs.williams.edu/~andrea/cs361

  lecture notes (pdf)

  homework solutions (pdf)

- prerequisites – 256 or a 300-level math course (and my permission)

This course is about **computability**. It's about answering questions such as, "Given a specific computing machine, what can we do with it? What can we not do?"

We study these and other questions theoretically: We'll define a mathematical model of a computing machine, and then we'll evaluate it. First, we'll practice "using" it to perform a variety of tasks. Then we'll study it more deeply, understanding theorems related to the model and proving others.

We will

1. Consider specific examples of what the model can do (i.e., what it can compute)
2. Develop intuition for the general types of computations the model can handle
3. Prove properties of the model – generally related to its capabilities/limitations

We'll also see that, while this study is theoretical, it forms the foundation of a great deal of the practice of Computer Science – from grep and awk and perl to compiler design and computational linguistics.

We'll study several models of computing machines, starting with weak (restricted) models and moving to more and more complex models:

1. Finite automata
2. Pushdown automata
3. Turing Machines

Can think of each of these as a formal (mathematical) description of a computing machine. The computing machines they represent differ from each other in:

the quantity and structure of memory;
the amount of information they can take into account in performing a single computational step.

Questions we will answer include:

(1) If you restrict a computing machine so that it has no memory (other than a simple state descriptor), can you still do anything useful?
(2) If you have a computing machine that uses a tape as memory, and now if you add several more tapes, does that give you any additional power?
(3) Are there things that you simply cannot do with the most powerful computational models known?

Some answers are:
(1) Yes, you can still do lexical analysis (tokenizing, for example).
(2) No, at least not in terms of the types of tasks you can accomplish (you might be able to do certain things more elegantly)
(3) Yes.

Here's a sample:

Q: Is it possible to write a C program that will tell whether another C program ever stops?

More specifically

Is there a program U, which takes as input a program P and any input I, and which behaves as follows:

U(P,I) prints TRUE, if P halts on input I
U(P,I) prints FALSE, if P does not halt on input I
(i.e., P goes into an infinite loop)

Let's assume that there is such a program.

Now, write a new program B. B, when given a program P as input, does the following:

```
if U(P,P) then repeat
          else halt
```

Now let's look at B(B)

```
if U(B,B) then repeat
          else halt
```

if B(B) halts, then U(B,B) was FALSE. i.e., B did not halt on input B.
But this is a contradiction.

if B(B) does not halt, then U(B,B) was TRUE. i.e., B halted on input B.
Again, a contradiction.

Therefore, our original assumption of the existence of U(P,I) was false.

There is no way (in general) to write a program that will say whether another will stop.

Another problem:

Q: Is there a program A such that when given P and I as input it will return TRUE if P halts on I.

Yes! Just simulate P on I.

Uses of Theory of Computation:

1. To compare the power of various models of computing.

2. To determine the complexity and/or possibility of making certain computations.

3. To design special-purpose models of computation that are useful for particular (practical) tasks:

parsing (both computer and natural languages)
lexical analysis
terminal emulation