CS 3100 – Models of Computation – Fall 2011

August 22, 2011

Notes 1, Handed out: August 21, 2011

1 General Announcements

- This course covers concepts from Automata Theory, Computability, Logic, and Reasoning about Programs
- We will use Python and Functional Programming heavily for much of the course
- There will be other tools introduced in the course
- Details about exams and grading are online
- There will be a course project worth half of your final-exam points
- The final exam will also examine your project to some extent; but generally each exam covers portions since the previous exam or the beginning of the course, whichever is more recent.

2 Checklist

- Send email to teach-cs3100 giving your preferred email addresses
- Login to cade machines and make sure that you can run simple Python3 programs
- TAs: Set up the score lookup facility
- TAs: Set up handin for Assignment #1, due 8/30/11

3 Lecture

3.1 Why study Computation?

1. We rely on computers for everything
2. We need tools to reason about computers (hardware and software) to make sure that they are working correctly
3. We need to understand the limits of computers and computation (and how to define these ideas)

3.2 Learning Objectives

1. Learn to design and experiment with various automata (prerequisite for your future classes such as Compilers)
2. Become proficient in basic mathematical logic
3. Learn how to write proofs and believe in proofs you write (and to learn first-hand how brittle proofs can be - much like programs can be - if you leave 'holes' in them)

3.3 Formal Notions of Computation

- Turing machines (will study)
- Lambda calculus (will study when you write functional programs)
- Other formal systems (Rewrite systems - aka Thue systems, Counter machines, etc.)

3.4 Minimalist Approach

Theoreticians seek the smallest number of concepts using which to state ideas

- State (the "Control" memory)
- State Transition
- Alphabet
- Additional memory needed if you want more computing power
  - One stack - gives you Push-down automata
  - Two stacks - gives you Turing machines, or everything

3.5 Notions of Hardness

- How hard are very hard problems (programs that may even loop, or mathematical functions for which there can't exist programs to implement them..)
  - Non Recursively Enumerable (Non RE)
  - RE (programs exist, but may loop)
  - Recursive (halting programs guaranteed - may take too much time/space though)
- How hard are problems within the Recursive family?
  - NP-complete
  - Polynomial

3.6 Correctness Checking and Computer-Aided Verification

- To express correctness, we need to employ mathematical logic
- Logics are intimately related to automata
- Logics are workhorses in the industry now - e.g. Decidable (Recursive) logic engines now routinely poke holes in browsers, and have even shown how to shop free online (a result that has caused patches in very reputed E-commerce sites)
- We will obtain a taste for logic through
  - reading the above inspiring papers
  - using some of these tools
  - If time permits, writing a simple tool yourself (BDD-based)

3.7 Proof methods

Reduction arguments: Gang up problems into a set so that when one day you solve one problem, you have a solution to all problems in that set