Models of Computation

Ganesh Gopalakrishnan

Aug 23, 2011

Table of contents

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.

Checklist

- Send email to `peterlee@cs.utah.edu` with these details:
  - Subject: Preferred Email Address
  - Body: Just your preferred email address

  Please do not use personal email addresses in future. (All emails requesting help must be sent to teach-cs3100).
- 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

Learning Objectives

- To understand what "computation" means
- Learn how to compare the "power" of various computers (or computataional devices)
- Build and experiment with Automata and Machines that operate on textual information
- Learn Mathematical Logic
- Study Computability and Complexity
- Be able to appreciate why program testing is hard
- Be able to appreciate how in some cases, it may be rendered easier
- Build up your foundations with respect to all sorts of information processing systems that are all syntax/text based

We now present some questions that this course will help you grapple with (or at least ask with sufficient confidence)

How to extract data from spreadsheets...

- ...and do it without errors
- Panko (2009) summarized the results of seven field audits in which operational spreadsheets were examined, typically by an outsider to the organization. His results show that 94% of spreadsheets have errors and that the average cell error rates (the ratio of cells with errors to all cells with formulas) is 5.2% `http://www.strategy-at-risk.com/2009/03/03/the-risk-of-spreadsheet-errors`

Bioinformatics and sequence matching...

- Bioperl `http://en.wikipedia.org/wiki/BioPerl`
- Biopython `http://news.open-bio.org/news/2011/08/biopython-1-58-released`
- Python Regexp Testing `http://kodos.sourceforge.net/about.html`

How to detect E-Cashier vulnerabilities...

- ...and prevent vulnerabilities through *formal analysis*
- (Google "How to shop for free online") `http://research.microsoft.com/apps/pubs/default.aspx?id=145858`

Why Software is "Eating" the World...

- ...and how to build and test these machines
- Google "Andreessen Eating" (to locate his WSJ article)
  `http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html`
- Importance of debugging software *formally*

The quest for supercomputers

- Even the most powerful of computers
  - Top 500 computers: `http://www.top500.org/`
  - Troubled waters: (Blue Waters: `http://www.theregister.co.uk/2011/08/08/ibm_kills_blue_waters_super/`)
- ...grapple with the same fundamental questions as this Turing machine:
- (A mechanical Turing machine) `http://www.youtube.com/watch?v=E3keLeMwfHY&feature=topics`

Make it more hands-on

- You learn the concepts through declarative programming
- We will use a subset of Python
- I'll teach you the theory part through lecture notes
- Assignments: theory, Python programming, other applied projects
- Final course project: gives you a chance to use some of these ideas

Computation: Foundation of Modern Society

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)

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)

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.)

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

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

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)

Reduction Proofs (using Paper/Pencil)

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

Proofs in Practice

Use automated tools!