CS 311: Computational Structures

James Hook

September 30, 2014

1 Overview

1.1 Course Goals (from course description)

Introduces the foundations of computing. Regular languages and finite automata. Context-free languages and pushdown automata. Turing machines and equivalent models of computation. Computability. Introduction to complexity.

The main goal of the course is that students obtain those skills in the theoretical foundations of computing that are used in the study and practice of computer science. A second goal is that students become familiar with Prolog as an experimental tool for testing properties of computational structures.

Upon the successful completion of this course students will be able to:

1. Find regular grammars and context-free grammars for simple languages whose strings are described by given properties.
2. Apply algorithms to: transform regular expressions to NFAs, NFAs to DFAs, and DFAs to minimum-state DFAs; construct regular expressions from NFAs or DFAs; and transform between regular grammars and NFAs.
3. Apply algorithms to transform: between PDAs that accept by final state and those that accept by empty stack; and between context-free grammars and PDAs that accept by empty stack.
4. Describe LL(k) grammars; perform factorization if possible to reduce the size of k; and write recursive descent procedures and parse tables for simple LL(1) grammars.
5. Transform grammars by removing all left recursion and by removing all possible productions that have the empty string on the right side.
6. Apply pumping lemmas to prove that some simple languages are not regular or not context-free.
7. State the Church-Turing Thesis and solve simple problems with each of the following models of computation: Turing machines (single-tape and multi-tape); while-loop programs; partial recursive functions; Markov algorithms; Post algorithms; and Post systems.

8. Describe the concepts of unsolvable and partially solvable; state the halting problem and prove that it is unsolvable and partially solvable; and use diagonalization to prove that the set of total computable functions cannot be enumerated.

9. Describe the hierarchy of languages and give examples of languages at each level that do not belong in a lower level.

10. Describe the complexity classes P, NP, and PSPACE.

Note: Not all topics will receive equal priority.

1.2 Outline

In this course offering I expect to present topics in approximately this order. [The Syllabus trys to put this on a schedule; it will be updated regularly.]

1. Computation, Algorithms, and Languages.
   (a) Lessons from human computers: focus, symbols, and a common algorithm
   (b) Binary addition as a process
2. Regular languages
   (a) Describing Finite Automata: From cartoons to structures
   (b) Non-deterministic finite automata
   (c) Equivalence of DFAs and NFAs
   (d) Language level properties (Closure properties)
   (e) The regular languages and regular expressions
   (f) Applications of regular languages
   (g) Properties revisited: using language properties to show that some languages are not regular
3. Context-free languages
   (a) Recognizing palindromes
   (b) Push-down automata
   (c) Describing languages with context-free grammars
   (d) Ambiguity
   (e) PDAs can parse CFGs
   (f) CFGs can encode PDAs
4. Computability
   (a) When people were computers

(b) Turing's machine

(c) Simple Turing machine programming

(d) Variations on Turing machines

(e) Language problems and decidablity

(f) Reduction arguments

(g) Relating Turing machines to other models of computation. Recursive functions and the lambda-calculus.

5. Complexity

   (a) Time and Space Complexity

   (b) Polynomial Time

   (c) The classes P and NP

   (d) Satisfiability is NP Complete

   (e) Reduction arguments and NP Completeness

1.3 Motivation: Counting People

Consider two algorithms to count people:

1. Sequentially count off from 1.

2. Everyone stands, counting themselves as 1.

   Find someone else standing. Exchange counts, agree on the sum, elect one to sit and one to stand.

   The last person standing reports the count.

Discusion 1.1 *Questions:*

1. *Which takes longer? Why?*

2. *If we view the arithmetic operation (adding or incrementing) as "expensive", which takes more expensive operations?*

3. *In each algorithm, how many times is each person in the room active?*

4. *If we think of the people as "computational units", both algorithms are parallel. What are the key concepts we need to talk about the differences and similarities between these algorithms?*

5. *When counting off, every person is assigned a serial number. Can the second algorithm be modified to assign a (unique) serial number to each person?*

1.4 Motivation: Grade-school multiplication

Consider a partial computation of the product of two multi-digit numbers. Can the class agree on the next step of the calculation?

This basic observation—that people performing a calculation can pause and resume the calculation—was key to Turing's formation of the model we call the Turing machine in 1938.

Key observations:

1. Worksheet: We execute the algorithms by writing on a 2-dimensional sheet of paper. Abstractly, we can work on problems of unbounded size provided we have a large enough sheet of paper.

2. Focus: When we carry out these calculations we do not need to consider the values in their totality, we manipulate their representations one symbol at a time. In particular, as we execute the algorithm we have just a few points of focus at a time.

3. Finite symbols: The algorithms generally look at a single digit at a time. We can assume that there are only finitely many symbols that may occur at a point of focus.

4. Algorithm: The algorithm can be written down and taught. It has a finite number of steps. Given knowledge of the algorithm, a worker can take a suspend a calculation and return to it at a later time with no loss.

These ideas become the basis for the Turing Machine.

In the Turing machine, the worksheet will be flattened to a one-dimensional "tape". We will have a single point of focus, which is a point on the tape where we can read and write symbols. The symbols will come from a finite alphabet. The algorithm will be built into the "finite control" of the machine.

The first two machine models we study introduce ideas that are common to the Turing machine. The first model, the finite automata, does not have any scratch storage (tape or worksheet). It is the simplest model we will study.

1.5 Addition

Look at binary addition.

Build a finite state transducer that outputs the sum of two suitably encoded binary numbers. (Similar to Sipser problems 1.32, 1.33. and 1.34)

Morph the calculation into a recognition problem.

Generalize to illustration of Deterministic Finite Automata (DFA) by "cartoons."

Generalize to formal definition of DFA and acceptance.