Models of Computation
Revision Notes

James Brown

April 5, 2017

Contents

1 Introduction

These are notes I have written in preparation for the upcoming 2017 Models of Computation exam. This year the module was run by Paul Levy (P.B.Levy@cs.bham.ac.uk). This module is about problems and *computers*. We ask ourselves:

- What problems can be solved on a computer?
- What problems can be solved on a computer with finitely many states?
- What problems can be solved on a computer with only finitely many states, but also a stack of unlimited size?
- What problems can be solved on a computer with only finitely many states, but also a tape of unlimited size that it can read and write to?
- What problems can be solved *fast* on a computer?
- What does "fast" mean anyway?
- What does *computer* mean anyway?

2 Language Membership Problems and Regular Expressions

2.1 Language Membership Problems

Suppose we have a set of characters Σ, which we will call the *alphabet*. A *word* is a finite sequence of characters, and we write Σ^* for the set of all words. We can *concatenate* words. A *language* is a set of words and a subset of Σ^*. Given a word, we want to know is it in the language or not? If we take an example alphabet a, b, c, here are some languages:

- All words which contain exactly 3 b's
- All words whose length is prime
- All words that have more b's than a's
- The words abc, bac and cb
- No words at all
- The empty word

These examples are largely pretty useless, but this problem does have real world applications such as

- Java has rules about what you can call a variable. Is the word read by the compiler a valid variable name?
- A user makes an account and enters a password, is it valid?
- A student has submitted code for an assignment, is it correct?
- Will this code crash when it's run?

In each one of these examples, we are provided with a word and we want to know whether it is an acceptable word. We want to make a computer tell us the answer.

2.2 Regex

Regular Expressions are a useful notation for describing languages. We write Empty for the language consisting of no words (the empty set). We write a for the language consisting of just the single-character a, and ϵ for the language consisting of just the empty word. If we have a language ***L*** and ***L'***, we write ***LL'*** for the set of words that are a concatenation of a word in ***L*** and a word in ***L'***. We can also write ***L—L'*** for the set of words that are in ***L*** or ***L'*** - the union of the two languages. We write ***L**** for the set of words that are a concatenation of some number of words in ***L*** (some number may be 0). Just like arithmetic, regular expressions have precedence rules. $*$ has the highest precedence, then juxtaposition and then $|$.

Regular expressions in theoretical computer science mean an expression built from the above operations and nothing more. Regular expressions are used much more widely in programming with far more operations, but they cannot be used in the module. For example, + is a very common operation but not available to us.

3 Finite State Automata

4 Bisimulation and Minimisation

5 The Halting Problem

6 Properties of Code

7 Turing Machines

8 Church's Thesis

9 Complexity and P

10 NP

11 Lambda-calculus