CSE 322 - Introduction to Formal Methods in Computer Science
Introductionary Remarks

Dave Bacon
Department of Computer Science & Engineering, University of Washington

I. INTRODUCTION

Computing devices are unlike most everything else in the universe. They aren't rocks, just sitting there doing nothing. They aren't the vast fusion processors known as stars. They aren't the atoms and molecules floating around in the interstellar medium. No they are something vastly different, and in some sense vastly more powerful (not real power, mind you: your computer can't outshine the Sun!) We all know this–it is an obvious statement–but rarely do we really think hard about what it actually means for something to be a computer. And, given these strange devices called computers, how exactly do we understand them in a deep and meaningful way? In this course we will show you the beginnings of a vast theoretical structure which has been developed to try and come to terms with this strange thing called a computer. Not surprisingly it doesn't look too much like the other theories you might have seen, those of the world of physics like Newtonian mechanics, quantum theory, and special relativity. No the theory of computing devices is a different structure altogether: something vastly more interesting (in many people's opinion) and, get this, actually of practical value!

Let us begin to discuss what we will be doing in this course by describing a set of sample problems centered around everyones favorite activity: flying on an airline. In each of these problems the central object you will be working with will be an airline route map. In Figure 1, I've given an example for such a map for an airline I have personally always dreamed of founding: BaconAir^{TM}.

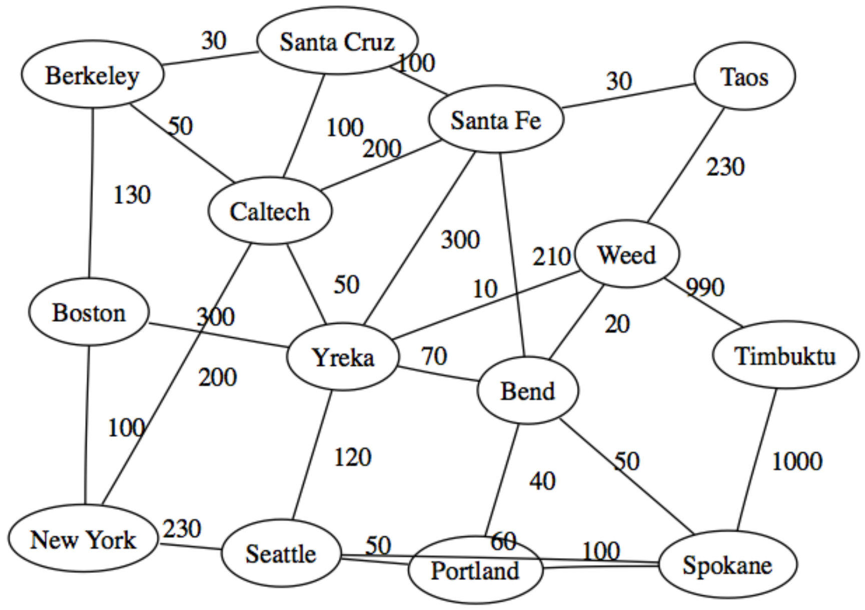

FIG. 1: BaconAir map with fairs on the edges. Take a flight on the porky side of air flight!

Suppose, now, that your really want to work for BaconAir^{TM}(who won't: the benefits are first rate!) So you go to interview for a position, and like all good CS interviews, instead of seeing if your a good nice efficient person, the first thing I do is give you a test problem. In this case the test problem is as follows. Listed below are five problems. Your goal is to assign each of these five problems to five different "computers." These computers are Homer Simpson, Bart Simpson, Lisa Simpson, an Apple II computer, and IBM's supercomputer Blue Gene. As you can see from this list, the computational power of each of these "computers" varies over a wide spectrum. Indeed, I will tell you that Homer < Bart < Lisa < Apple II < Blue Gene. But you could have guessed that yourself! Your goal is to assign the problems below in the best manner (i.e. respecting the computational complexity) to each of these computers. Do this correctly and you get a job (so you can buy a house and go in debt! Yah, for the housing market!) Okay so here are the problems

- **Problem 1**
  Input: An airport name
  Output: Yes, if number of "e"s is greater than the number of "a"s in the airport name. No otherwise.
  For example:
  Input : S**e***a*ttl**e** ⇒ Output: Yes
  Input : S*a*nt*a* F**e** ⇒ Output: No

- **Problem 2**
  Input: Two airports and list of distances between airports.
  Output: Shortest pathway between these two airports.

- **Problem 3**
  Input: An airport name
  Output: Yes if the name has the string "rek" in it. No otherwise.
  For example:
  Input: Seattle ⇒ Output: No
  Input: Y**rek**a ⇒ Output: Yes

- **Problem 4**
  Input: The route map
  Output: Yes if there exists a Hamiltonian circuit on the map. No otherwise. A Hamiltonian circuit is a tour on the graph which visits each node exactly once.

- **Problem 5**
  Given: Access to every pricing scheme of every airline.
  Input: Two airports.
  Output: The cheapest route between these two airports.

So, given these problems, what order should they be assigned to the Simpson family and computers? Don't just read this next sentence hoping that it will give you the answer, think about it, damnit!

The answer, it turns out is that

$$\textbf{Problem 3} < \textbf{Problem 1} < \textbf{Problem 2} \leq \textbf{Problem 4} < \textbf{Problem 5}$$

Thus Homer gets problem 3, Bart gets problem 1, Lisa gets problem 2, the Apple II gets problem 4, and Blue Gene gets problem 5. I will be one of the main "big picture" results of this class that the problems listed above, actually do have a natural ordering in terms of computing power. Further the fact that there is a $\leq$ in this ordering, will turn out to be one of the most interesting open problems, so interesting that there are people who will pay you one million dollars to turn that less than or equals into just a less than!

But what do I mean by this ordering? What does it mean for different problems like those above to have an "ordering" in terms of computational power. Well part of this class will be understanding what such an ordering could possibly mean.

More formally, in this class we will be studying problems like those above within a particular "computational model." This means that we will be formally defining these models of computation and discussing what they can and cannot do. Thus, for example we will define a computational model, finite automata, which can solve problems like problem 3, but *cannot* solve any of the other problems on the list above. And by *cannot* I mean that we will be able to *prove* this! Indeed the five problems I have listed above are meant to be sample problems for different computational models and what they can and cannot do. Thus I can make a list of correspondences, which won't mean much to you at this point, but hopefully will at the end of this course for these problem:

- **Problem 3** can be solved by the weakest of our computational models, a **Finite Automata**.

- **Problem 1** can be solved by the next strongest of our computational models, a **Pushdown Automata**.

- **Problem 2** can be solved by the model which is stronger than the finite automata or pushdown automata, and which corresponds roughly to our desktop computers, a **Turing Machine**.

- **Problem 4** can also be solved by a Turing machine. However, it is widely believed that this problem is an example of an **intractable** problem for the Turing machine. Intractable roughly means that the time required to solve problems of this type grows faster than any polynomial function of the problem size. We won't really get into this subject (NP-completeness) in any detail, but if you take the next theory course you'll get this in spades.

- **Problem 5** (as I've phrased it) is a problem which is **undecidable** for a Turing machine. In other words this is a problem which even your desktop machine cannot (really, truly) solve! This result, that there are problems which a Turing machine cannot really solve in a formal sense, is one of the deepest and most profound results in all of computational science. This means, of course, that it is often not the most practically important problem. But it should shake your core beliefs about mathematics and computation for reasons we'll get to at the end of this course.

So roughly this course will proceed through the ordering in the list above through each of these computational models where we will try to understand what the computational model is and what computational power the model possesses.

Okay so now you might say, well if finite automata cannot solve these other problems, then why should we study them at all! Rightly you might point out that since Turing machines can solve the problems posed for finite automata and pushdown automata, why not just study Turing machines? There are couple of reasons for this which I'll now enumerate

- Studying models of "less" computational power gives us insight into studying computational models of more power. Indeed, these models are of such considerably less complexity that we can more easily prove and understand the power of these models than for the much more complex Turing machine model. As the saying grows, you must learn to walk grasshopper, before you can fly.

- These lower models are often useful for studying machines which aren't like your desktop machine. Indeed simple finite automata are all over the place in embedded systems. Since it is often less expensive to not have a full microprocessor around, studying what finite automata can and cannot do is a good way to make you or your employer save big dollars.

- Finite automata and pushdown automata turn out to be extremely useful for certain computational tasks, especially in compilers and programming languages. For example, finite automata are used in the first step of, say, the C compiler, in the lexical parser, where the raw text of the program is turned into a list of tokens. As another example, you may be surprised to learn that finite automata are often used in video games for the "bots." Next time a bot beats you in a video game you can now really cry because you might have just lost to the lowest of all computational models! Doh. Pushdown automata, on the other hand, are near and dear to things like parsers for XML. Thus while this course will be a "theory CS" course, you should not lose site of the fact that the fundamental insights and algorithms for this course are also of great importance in the grand scheme of being a computer scientist.

- Finally, you should take this course because it will give you an idea about what theory in computer science really means. While most of you are already extremely computer savvy, I'll bet a lot fewer of you are magicians of the theory of computer science. And, what you'll discover in the course of this class, I hope, is that you will be proving some fundamental and beautiful mathematical statements. The pure theoretical computer scientists, indeed, is more of a mathematician than an engineer, proving and understanding computation because this gives them insight into the concrete, steady world of computing devices. Indeed, you may find that, like many a theorist before you, that happiness is proving a theorem. And even if you don't fall in love with the theory of computer science, at least this class will show you what it is, and hopefully you can emphasize with those who catch the theory bug.

So come, join with me, in this wonderful class, "CS Theory 101." And may you learn to dream of finite automata, context free grammars and undecidable languages!