CS21
Decidability and Tractability
Lecture 1
January 3, 2022

Outline
• administrative stuff
• motivation and overview of the course
• problems and languages
• Finite Automata

Administrative Stuff
• Text: Introduction to the Theory of Computation – 3rd Edition by Mike Sipser
• Lectures self-contained
• Weekly homework
  – collaboration in small groups encouraged
  – separate write-ups (clarity counts)
• Midterm and final
  – indistinguishable from homework except cumulative, no collaboration allowed

Motivation/Overview
• This course: introduction to Theory of Computation
  – what does it mean?
  – why do we care?
  – what will this course cover?
Motivation/Overview

• At the heart of programs lie \textbf{algorithms}.

• To study algorithms we must be able to speak \textit{mathematically} about:
  – computational problems
  – computers
  – algorithms

Motivation/Overview

• In a perfect world
  – for each \textit{problem} we would have an \textit{algorithm}
  – the algorithm would be the fastest possible
    (requires \textit{proof} that no others are faster)

  What would CS look like in this world?

Motivation/Overview

• Our world (fortunately) is not so perfect:
  – not all problems have algorithms \textit{(we will prove this)}
  – for \textit{many} problems we know embarrassingly little about what the fastest algorithm is
    • multiplying two integers
    • factoring an integer into primes
    • determining shortest tour of given \( n \) cities
  – for certain problems, fast algorithms would change the world \textit{(we will see this)}

Motivation/Overview

Part One: \textit{computational problems, models of computation, characterizations of the problems they solve, and limits on their power}

• Finite Automata and Regular Languages
• Pushdown Automata and Context Free Grammars

Motivation/Overview

Part Two: \textit{computational problems, models of computation, characterizations of the problems they solve, and limits on their power}

Part Three: \textit{complexity classes P and NP, NP-completeness, limits of efficient computation}

Main Points of Course

\textit{(un)-decidability} \hfill \textit{(in)-tractability}

Some problems have no algorithms!

Many problems that we’d like to solve have no \textit{efficient} algorithms!

(no one knows how to \textit{prove} this yet…)
What is a problem?

- Some examples:
  - given \( n \) integers, produce a sorted list
  - given a graph and nodes \( s \) and \( t \), find the (first) shortest path from \( s \) to \( t \)
  - given an integer, find its prime factors
- problem associates each input to an output
- input and output are strings over a finite alphabet \( \Sigma \)

What is a problem?

- A problem is a function:
  \[ f: \Sigma^* \rightarrow \Sigma^* \]
- Simple. Can we make it simpler?
- Yes. Decision problems:
  \[ f: \Sigma^* \rightarrow \{ \text{accept, reject} \} \]
- Does this still capture our notion of problem, or is it too restrictive?

What is a problem?

- Example: factoring:
  - given an integer \( m \), find its prime factors
  \[ f_{\text{factor}}: \{0,1\}^* \rightarrow \{0,1\}^* \]
- Decision version:
  - given 2 integers \( m,k \), accept iff \( m \) has a prime factor \( p < k \)
- Can use one to solve the other and vice versa. True in general (homework).

What is a problem?

- For most of this course, a problem is a decision problem:
  \[ f: \Sigma^* \rightarrow \{ \text{accept, reject} \} \]
- Equivalent notion: language
  \[ L \subseteq \Sigma^* \]
  the set of strings that map to "accept"
- Example: \( L = \) set of pairs \((m,k)\) for which \( m \) has a prime factor \( p < k \)