Outline

• What is an algorithm?
• Hilbert’s 10th problem
• Church-Turing thesis
• Undecidable problems
• Algorithmic description of a TM

From Sipser Chapter 3.3
What is an Algorithm?

• How would you describe an algorithm?
• An algorithm is a collection of simple instructions for carrying out some task
  – A procedure or recipe
  – Algorithms abound in mathematics and have for thousands of years
Hilbert’s Problems

• In 1900 David Hilbert proposed 23 mathematical problems for next century

• Hilbert’s 10th problem:
  – Devise an algorithm for determining if a polynomial has an integral root (i.e., polynomial will evaluate to 0 with this root)
  – Hilbert said “a process by which it can be determined by a finite number of operations”
  – For example, $6x^3yz^2 + 3xy^2 − x^3 -10$ has integral root $x=5$, $y=3$, and $z=0$.
  – He assumed that a method exists.
    • He was wrong
Church-Turing Thesis

• It could not really be proved that an algorithm did not exist without a clear definition of what an algorithm is

• Definition provided in 1936
  – Alonzo Church’s λ-calculus
  – Alan Turing’s Turing Machines
  – The two definitions were shown to be equivalent
  – Connection between the information notion of an algorithm and the precise one is the Church-Turing thesis
    • The thesis: the intuitive notion of algorithm equals Turing machine

• In 1970 it was shown that no algorithm exists for testing whether a polynomial has integral roots
  – Still there is a correct answer: it either does or does not
We can rephrase Hilbert’s 10th problem as if we were asking if the language

\[ D = \{ p \mid p \text{ is a polynomial with an integral root} \} \]

is **decidable** (not just Turing-recognizable)!

• Can you come up with a procedure to answer this question?
  – Try all possible integers. E.g., start and 0 and loop out: 0, 1, -1, 2, -2, ...
  – For multivariate case, just lots of combinations
  – Is this decidable, Turing recognizable, or neither?
    • May never terminate
    • You will never **know** whether it will not terminate or will accept shortly
  – So, based on this method the problem is Turing-recognizable but not decidable
  – Could there be another method that is decidable?
More on Hilbert’s 10th Problem

• For the univariate case, there is an upper bound on the possible value of the root of the polynomial
  – Hence, we can build an algorithm that halts!
  – The problem is decidable!

• For the multivariate cases, it has been proven that the problem is not decidable

• Extremely significant result
  – There are limits to what it is computable
  – Inherent limitation not due to technology
Ways of Describing Turing Machines

• We can specify the design of a machine (FA, PDA) formally or informally.
  – The same hold true with a Turing Machine
  – The informal description still describes the implementation of the machine—just more informally

• With a TM we can go up one further level of abstraction:
  – We do not describe actual machine (e.g., tape heads, etc.).
  – We describe it algorithmically
Algorithmic description

• The input to a TM is always a string
  – Other objects (e.g., graphs, lists, etc) must be encoded as a string
  – The encoding of object O as a string is denoted as <O>

• We implicitly assume that the TM verifies that the input to conforms to the proper encoding

• If not properly encoded, the string is rejected
Example: Algorithmic Level

Let $A$ be the language of all strings representing connected graphs (i.e., any node can be reached by any other).

$$A = \{<G>| \text{ G is a connected undirected graph}\}$$

- High level description of TM $M$ which decides $A$

$M = \text{"On input } <G> \text{ the encoding of a graph G:
1. Select and mark the first node in G}
2. Repeat the following until no new nodes are marked
   - For each node in G, mark it if it is attached by an edge to a node that is already marked
3. Scan all nodes of G to determine whether they are all marked. If they are, then accept; else reject