COMS W3261
Computer Science Theory
Lecture 14
Algorithms and the Church-Turing Thesis

Alexander Roth

2014 – 10 – 20

Outline

1. Midterm review
2. Definition of an algorithm
3. The Church-Turing thesis
4. The diagonalization language L_d is not RE
5. Reducing one problem to another

1 Definition of Algorithm

- Surprisingly, there is no universally agreed-upon definition for the term "algorithm". Informally, we can think of an algorithm as a collection of well-defined instructions for carrying out some task.
- In *The Art of Computer Programming*, Donald Knuth states that an algorithm should have five properties.
    1. Finiteness: An algorithm must always terminate after a finite number of steps.
    2. Definiteness: Each step of an algorithm must be precisely defined.
    3. Input: An algorithm has zero or more inputs.
    4. Output: An algorithm has one or more outputs, quantities which have a specified relation to the inputs.
    5. Effectiveness: All of the operations to be performed in an algorithm can be done exactly and in a finite length of time.

- In this course we will use a Turing machine that halts on all inputs as the definition of an algorithm. The term decider is sometimes used for such a Turing machine.
  - A language L that can be recognized by an algorithm is said to be recursive.
  - If a language L is recursive, we say L is decidable.
  - If a language L is not recursive, we say L is undecidable.
- In general, a Turing machine need not halt all inputs. An input on which a Turing machine never halts is not in the language defined by the Turing machine.
  - A language L that can be recognized by a Turing machine is said to be recursively enumerable.
  - The term Turing-recognizable language is sometimes used for a recursively enumerable language.
  - Note that a language may be undecidable because it is not recursive but is recursively enumerable or because it is not recursively enumerable.

2 The Church-Turing Thesis

- A Turing machine can compute a function from an input to an output by reading the input, making a sequence of moves, and then halting, leaving only the output of the function on the tape.
- A recursive function is one that can be computed by a Turing machine that halts on all inputs.
- A partial-recursive function is one that can be computed by a Turing machine that need not halt on all inputs. The output of the function on an input for which the Turing machine does not halt is said to be undefined.
- The Church-Turing thesis says that any general way to compute will allow us to compute only the partial-recursive functions. The Church-Turing thesis is unprovable because there is no precise definition for “any general way to compute.”
- An informal way of expressing the Church-Turing thesis is that any function that can be effectively computed can be computed by a Turing machine.

3 The Diagonalization Language L_d is not Recursively Enumerable

- We can enumerate all binary strings.
- We can enumerate all Turing machines.
- We define L_d, the diagonalization language, as follows:
  1. Let w_1, w_2, w_3, ... be an enumeration of all binary strings.
  2. Let M_1, M_2, M_3, ... be an enumeration of all Turing machines.
  3. Let $L_d = \{ w_i \mid w_i \text{ is not in } L(M_i) \}$.
- Theorem: L_d is not a recursively enumerable language.
- Proof:
  - Suppose $L_d = L(M_i)$ for some TM M_i.
  - This gives rise to a contradiction. Consider what M_i will do on the input w_i.
    - If M_i accepts w_i, then by definition w_i cannot be in L_d.
    - If M_i does not accept w_i, then by definition w_i is in L_d.
  - Since w_i can neither be in L_d nor not be in L_d, we must conclude there is no Turing machine that can define L_d.

4 Reducing One Problem to Another

- If we have an algorithm to convert instance of a problem P_1 to instances of a problem P_2 that have the same answer, then we say that P_1 reduces to P_2.
- A reduction from P_1 to P_2 must turn every instance of P_1 with a yes answer to an instance of P_2 with a yes answer, and every instance of P_1 with a no answer to an instance of P_2 with a no answer.
- We will frequently use this technique to show that problem P_2 is as hard as problem P_1.
- The direction of the reduction is important.
- For example, if there is a reduction from P_1 to P_2 and if P_1 is not recursive, then P_2 cannot be recursive.
- Similarly, if there is a reduction from P_1 to P_2 and if P_1 is not recursively enumerable, then P_2 cannot be recursively enumerable.