COMS W3261
Computer Science Theory
Lecture 18
The Classes $\mathcal{P}$ and $\mathcal{NP}$

Alexander Roth

2014 – 11 – 10

Outline

- Rice's Theorem
- Time complexity
- The classes P and NP
- Polynomial-time reductions
- NP-complete problems

1 Rice's Theorem

- We showed that the language $L_e = \{ M \mid L(M) = \emptyset \}$ is not recursively enumerable, and hence is undecidable.
- We showed that the language $L_{ne} = \{ M \mid L(M) \neq \emptyset \}$ is recursively enumerable but not recursive, and hence is undecidable.
- The undecidability of these languages is a special case of a more general theorem about Turing machines languages known as Rice's Theorem.
- Rice's Theorem
  - A *property* of the recursively enumerable languages is a set of recursively enumerable languages.
  - For example, under this definition a language being context free is a property of the recursively enumerable languages: the set of all CFL's is a subset of the recursively enumerable languages.

  - A property is *trivial* if it is either the empty set or it is the set of all recursively enumerable languages.
  - Rice's Theorem: Every nontrivial property of the recursively enumerable languages is undecidable.
  - Section 9.3.3 of HMU has a proof of Rice's Theorem.
  - Rice's Theorem does not pinpoint why a property is undecidable. It could be recursively enumerable and not recursive, or it could be not recursively enumerable.

- From Rice's theorem, we can conclude a number of problems about Turing machine languages that are undecidable:
  1. Is the language accepted by a TM empty?
  2. Is the language accepted by a TM finite?
  3. Is the language accepted by a TM regular?
  4. Is the language accepted by a TM context free?

2 Time Complexity

- We now limit our attention to recursive languages and ask the question "How quickly can a Turing machine determine whether a given input string w is a given recursive language L?"
- We say that a function $T(n)$ is $O(f(n))$ [read "big-O of $f(n)$"]if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n \geq n_0$, we have $T(n) \leq cf(n)$.
- Big-O notation allows us to ignore constant factors and just focus on asymptotic behavior. E.g., $T(n) = 2n^2$ is $O(n^2)$.
- Big-O notation allows us to ignore low-order terms. E.g., $T(n) = 2n^2 + 3n + 4$ is $O(n^2)$.
- A Turing machine M is of time complexity $T(n)$ if on any input of length n, M halts after making at most $T(n)$ moves.
- Note that time complexity is a worst-case measure. If a Turing machine M is of time complexity $T(n)$, then it makes at most $T(n)$ moves on every input of length n.

Class Notes

Big-Omega Notation

We say $T(n)$ is $\Omega(g(n))$ if there exists an integer n_0 and a constant $c > 0$ such that for all $n \geq n_0$, we have $T(n) \geq cg(n)$.

Big-Theta Notation

We say $T(n)$ is $\Theta(g(n))$ if there exists an integer n_0 and a constant $c_1 > 0 + c_2 > 0$ such that for all n $n \geq n_0$, $c_1 g(n) \geq T(n) \geq c_2 g(n)$.

A TM M is of time complexity $T(n)$ if on any input of length n, M halts after making at most $T(n)$ moves.

Big-0 Examples

- $O(1)$ constant time.
- $O(\log n)$ logarithmic time.
- $O((\log n)^c)$ polylogarithmic time.
- $O(n)$ linear time.
- $O(n^2)$ quadratic time.
- $O(n^c)$ polynomial time.
- $O(c^n)$ exponential time.

3 The Classes P and NP

- We define P to be the set of languages L such that $L = L(M)$ for some deterministic Turing machine M of time complexity $T(n)$ where $T(n)$ is a polynomial function of n.
- NP is the set of language L such that $L = L(M)$ for some nondeterministic Turing machine M where on any input of length n, there are no sequences of more than $T(n)$ moves of M where $T(n)$ is a polynomial function of n.
- The question of whether P = NP is one of the most important open problems in computer science and mathematics.
- The Clay Mathematics Institute has identified the P vs NP question as one of its sever Millennium Prize Problems and offers a one million dollar prize for its solution.

Class Notes

Our definition of Turing machine is usually a single tape deterministic turing machine. $\mathcal{NP}$ includes $\mathcal{P}$, the Recursive languages include $\mathcal{NP}$ and the Recursively Enumerable languages include the Recursive languages.

4 Polynomial-Time Reductions

- In our study of problems that can be solved in polynomial time, we restrict ourselves to polynomial-time reductions.
- A polynomial-time reduction is an algorithm that maps any instance I of problem A into an instance J of problem B in a number of steps that is a polynomial function of the length of I such that I is in A iff J is in B.
- Note that in a polynomial-time reduction of the length of J must be a polynomial function of the length of I.

Class Notes

NP-Complete problems are said to be the hardest problems.

5 NP-Complete Problems

- A language L is NP-complete if
  1. L is in NP and
  2. For every language L' in NP, there is a polynomial-time reduction of L' to L.
- A language satisfying condition (2) is said to be NP-hard.
- If A is NP-complete, B is in NP, and there is a polynomial-time reduction of A to B, then B is NP-Complete.
- If any one NP-complete problem is in P, then P=NP.
- An equivalent definition of the NP-Complete languages is that they are the set of languages for which the membership of an input string in the language can be checked for correctness, given a certificate, in deterministic polynomial time in the length of the input.
- Examples of problems in P
  - Determining whether a set of integers has a subset whose sum is negative.
  - Finding the cheapest path between a pair of nodes in a graph.
  - Lots of others.
- Examples of problems in NP not known to be in P or to be NP-complete
  - Determining whether two graphs are isomorphic.
  - Not that many others.
- A problem is said to be *intractable* if it cannot be solved in polynomial time.