CS581 Theory of Computation: Homework #6

Due on March 16 2016 at 12:30pm

Harry H. Porter Winter 2016

Konstantin Macarenco

Chapter 6.2 Decidability of logical theories

1. Well formed formula
   A formula is a well-formed string over this
   1. is an atomic formula
   2. has the form $\phi \wedge \phi$ or $\phi \vee \phi$ or $\neg\phi$ and ϕ are smaller formulas, or
   3. has the form $\exists x_i[\phi_1] or \forall x_i[\phi_1]$ where ϕ_1 is a smaller formula.
2. prenex normal form - all quantifiers apper in the front of the formula.
3. a variable isn't bound withing the scope of a quantifier is called a free variable
4. formula with no free variables is called a sentence of statement.
5. **universe** with assignment of relations to relation symbols is called a **model**
6. if M is a model we let the **theory of M**, written $Th(M)$, be the collection of true sentence in the language of that model
7. **A DECIDABLE THEORY**
8. Tuatology always true in any model
9. Axioms a given set of statements assumed to be true without proof. Rules of inference/deduction
10. **Kurt Godel** showed that no algorithm can decide in general whether statements in number theory are true or false.
11. **Church showed that** $Th(N, +, x)$ is undecidable. (Proof idea : reduce Atm to the problem of deciding number theory).
12. $Th(N, +)$ is decidable
13. The set of provable statements in number theory is Turing Recognizable

    we can enumerate all the provable statements. (List them all out)
14. Some statements are true but not provable!

    Some statements in $Th(N, +, \times)$has no proof.

    Assume all true statements are provable, Look for a proof of ϕ *and* $\neg\phi$one of then will be true
    But $Th(N, +, \times)$is undecidable, hence contradiction!

Chapter 7 TIME COMPLEXITY

1. Measuring the complexity
2. let M be a deteministic Turing machine that halts on all inputs. The **running time** or **time complexity** of M is the function $f : N \to N$, where $f(n)$ is the runnging time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine. Customarily we use n to represent the lenght of the input.
3. **BIG O and SMALL O**
4. Let f and g be functions $f, g : N \to R^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exists= such that for every integer $n \geq n_0$,
   $f(n) \leq cg(n)$
   When $f(n) = O(g(n))$ we say that $g(n)$ is an upper bound for $f(n)$.
5. the big-O interacts wit logarithms in a particular way. $log_b n = log_2 n/log_2 b$
6. Let t(n) be a function, where t(n) $\geq$ n then every t(n) time multitape Turing Machine has an equivalent O(t^2(n)) time single-tape Turing machine.

7. Let N be a nondeterministic TUring machine that is a decider. The **running time** of N is the function $f : N \rightarrow N$, where f(n) is the maximum number of steps that N uses on any branch of its computation of any inputof length n, as shown in the following figure.

   The definition of the running time of a non deterministic TM is not intended to correspond to any real-world computing device. Rather,it is a useful math definition.

8. Let t(n) be function, where $t(n) \geq n$. Then every t(n) time nondeterministic signle-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single tape Turing Machine

9. **the class P** - polynomial.

10. Exponential is bad, polynomial is good.

11. **P** is the class of languages that are decidable in polynomial time on a deterministic single tape Turing machine. (realistically solvable on a computer).

    1. Every context-free language is a member of P.
    2. **Dynamic programming** - break a problem to smaller subproblems, and solve each subproblem only once.
    3. PATH problem is in P

12. **the class NP** - nondeterministically polynomial, i.e. have P time verifier.

    1. HAMILTONIAN PATH problem is in NP, exponential decider, but Polynomial
    2. Composites (natural number is composite if it is the product of two integers greater than 1, i.e. composite is not a prime number). Can be easily verified with given divider.
    3. $\overline{HAMPATH}$ is not in NP

13. **Verifier** for a language A is an algorithm V, where

    $$A = \{w \mid V\ accepts, c\ for\ some\ string\ c\}$$

    We measure the time of a verifier only in terms of the length of w, so a ponynomial time verifier runs in polynomial time in the length of w. A language A is **polynomially verifiable** if it has a polynomial time verifier.

14. Theorem 7.20. A language is in NP iff it is deced by some nondeterministic polynomial time Turing machine.

    1. Forward convert P time verifier to an equivalent NTM.
    2. Back convert NTP to P time verifier.

15. Nondeterministic time complexity class $NTIME(t(n))$

    1. $NTIME(t(n)) = \{L \mid L\ is\ a\ language\ decided\ by\ an\ O(t(n))\ time\ nondeterministic\ TM\}$

16. $NP = \bigcup_k NTIME(n^k)$

    1. NP problems
    2. The QLIQUE problem
    3. The SUBSET problem
    4. Proof: provide polynomial time verifier or NTM (nondeterministic turing machine).

17. **P versus NP**

    1. P can be decided quickly.
    2. NP can be verified quickly.

3. P = NP ? nobody can prove or disprove.
4. The best deterministic method currently known for deciding languages in NP uses exponential time. In other words, we can prove that.
$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$

18. **NP-COMPLETENESS**

    1. If a plynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable.
    2. **Satisfiability problem** a boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.
    3. **POLYNOMIAL TIME REDUCIBILITY**
        1. If $A \leq_p B$ and $B \in P$ then $A \in P$

19. A language B is NP-complete if it satisfies two conditions:
    1. B is NP, and
    2. every A in NP is polynomial time reducible to B.if it satisfies two conditions:
    1. B is NP, and
    2. every A in NP is polynomial time reducible to B.
20. if B is NP complete and $B \in P$, then P = NP.
21. if B is NP complete and $B \leq_p C$, then C is NP complete.
22. **THE COOK-LEVIN THEOREM** the SAT is NP-complete
23. Other NP complete problems:
    1. CLIQUE
    2. VERTEX-COVER
    3. HAMPATH
    4. UHAMPATH (undirected)
    5. SUBSET SUM PROBLEM
24. **SPACE COMPLEXITY**