CS F364
Design & Analysis of Algorithms

ALGORITHMS - COMPLEXITY

Complexity Classes

- NP-Completeness Via Reductions
  - Examples: CNF-SAT

PROBLEM: CNF-SAT

- CNF-SAT:
  - Given a Boolean expression F in CNF (i.e. Conjunctive Normal Form)
    - *find whether there is an input assignment such that the F is satisfied.*
- Question:
  - Can you design an algorithm to solve CNF-SAT ?
    - Can you design a polynomial time algorithm to solve CNF-SAT?
      - Points to ponder:
        - *Verifying validity of a formula in CNF can be done in polynomial time*.
        - **(Dual):** *Verifying satisfiability of a formula in DNF can be done in polynomial time.*
  - But we argued that SAT is NP-complete:
    - Is there an implication on converting a SAT instance to equivalent CNF (or DNF) ?

$\mathbb{NP}$-Completeness Via Reductions: CNF-SAT

- CNF-SAT:
  - Given a Boolean expression F in CNF
    - *find whether there is an input assignment such that the F is satisfied.*
- CNF-SAT is $\mathbb{NP}$-complete
  1. CNF-SAT is in $\mathbb{NP}$.

     Proof:

     Given an assignment of values (as a certificate) verification can be done in linear time.
  2. CNF-SAT is $\mathbb{NP}$-hard

     i.e. SAT $\precsim$ CNF-SAT

     (*see following slides for a reduction*).

NP-Completeness via Reductions: CNF-SAT [2]

- SAT $\precsim$ CNF-SAT
  - Is there an algorithm to convert a Boolean expression F to its equivalent CNF?
    - What is the time complexity of the algorithm?
  - We want a mapping

    g: I(SAT) --> I(CNF-SAT)

  that ONLY needs to preserve satisfiability!

SAT $\precsim$ CNF-SAT

- Mapping: **g(F) = F'** where **F'** is in CNF.
  - Construct the parse tree of the given expression F.
  - Label each edge with a variable (*including the one incoming edge to the root*)
    - Label each leaf with the input variable
  - Construct formula **F'** as **r_0 AND (AND_i v_i)**
    - where each **v_i** corresponds to a vertex of the tree and is of the form:
      - **e_o <--> e_{i1} op e_{i2}** if **op** is *binary*
      - **e_o <--> op e_i** if **op** is *unary*
        - for output edge **e_o** and input edges **e_{i1}** and **e_{i2}**

SAT ≾ CNF-SAT [2]

- Claims (A and B):
  A. The mapping **g** can be computed in polynomial time:
     1. Time taken for constructing a parse tree, given a formula of length **n**
        1. Time taken for parsing: **O(n³)**

           refer to **CYK algorithm (Dynamic Programming)**
        2. Time taken for construction of tree: **O(n)**
     2. Time taken for constructing the formula from the tree:
        1. Time taken for traversing the tree: **O(n)**
        2. Time taken for writing the formula: **O(k*n)** where k is the (constant) length of each clause

  B. **g(F) is satisfiable iff F is satisfiable**.