COMS W3261
Computer Science Theory
Lecture 20
NP and Co-NP

Alexander Roth

2014 – 11 – 17

Outline

1. Review: SAT is NP-Complete
2. CSAT is NP-Complete
3. 3SAT is NP-Complete
4. Co-NP
5. SAT and SMT Solvers

1 Review: SAT is NP-Complete

- SAT is the language $\{ E \,|\, E \text{ is a satisfiable boolean expression } \}$.
- We have shown that SAT is NP-complete by (1) showing SAT is in NP and (2) showing that every language in NP can be reduced to SAT in polynomial time.

2 CSAT is NP-Complete

- A boolean expression is in conjunctive normal form (CNF) if it is the logical AND (conjunction) of clauses where a clause is the logical OR of one or more literals. (Colloquially, the AND of ORs.)
- We define CSAT as the language $\{ E \,|\, E \text{ is a satisfiable CNF boolean expression } \}$.
- We can prove that CSAT is NP-Complete.

  - First, we need to show that CSAT is in NP. But this is easy because a CNF boolean expression is a special case of a boolean expression and SAT is in NP.
  - Second, we need to show we can reduce an instance of E of SAT into an instance F of CSAT in polynomial time such that E is satisfiable iff F is satisfiable. However, this part of the proof is a bit more complicated because we need to do the reduction in polynomial time.
    * Two boolean expressions are equivalent if they have the same value on every truth assignment.
    * However, just converting a boolean expression E to an equivalent CNF boolean expression E' may make E' exponentially longer and thus take exponential time (in the length of E).
    * To avoid this exponential blow-up, we can transform E into another not necessarily equivalent boolean expression F but with the property that E is satisfiable iff F is satisfiable. Moreover, we can do this transformation in time polynomial in the length of E. Section 10.3 of HMU has the details.

3 3SAT is NP-Complete

- A boolean expression is in k-CNF if it is the logical AND of clauses each one of which is the logical OR of exactly k distinct literals. for example, $(w \vee \neg x \vee y) \wedge (x \vee \neg y \vee z)$ is in 3-CNF.
- We define kSAT as the language $\{\, E \,|\, E$ is a satisfiable k-CNF boolean expression $\}$.
- We can show that 1SAT and 2SAT are in P but kSAT is NP-Complete for $k \geq 3$.

4 Co-NP

- P is closed under complement but is not known whether NP is closed under complement.
- CO-NP is the set of languages whose complements are in NP.
- A language is co-NP-Complete if it is in Co-NP and if every language in Co-NP is polynomial-time reducible to it.
  - The language consisting of boolean expressions that are tautologies is Co- NP-Complete.
- It is widely believed that none of the NP-complete languages have their complements in NP.

- Thus, it is believed no NP-Complete language is in Co-NP.
  - We suspect NP $\neq$ Co-NP because we can prove NP = Co-NP iff there is some NP-complete problem whose complements is in NP.

5 SAT and SMT Solvers

- Even though SAT is NP-Complete (a worst-case result), software tools called SAT solvers have been developed using various algorithms and heuristics that seem to work well for instances of large practical SAT problems arising in areas such as electronic design automation, model checking, software verification, and artificial intelligence.
- Satisfiability Modulo Theories (SMT) are generalizations of SAT where an SMT instance is a formula in first-order logic in which some function and predicate symbols have additional interpretations. A simple example would be an instance of SAT in which some of the binary variables are replaced by linear inequalities. Many SMT solvers have been built. In a typical use of an SMT solver, we would translate various kinds of assertions attached to a program into SMT formulas to see if the assertions are always true. SMT solvers allow a richer set of questions to be posed than is possible with SAT solvers.

Class Notes

Proving an NP-Complete problem is NP-Complete: Suppose we have a problem L'. We must:

1. show that L' is in $\mathcal{NP}$.
2. Reduce L' to an NP-complete problem L.

CNF is the AND of a bunch of clauses. (The AND of OR's).

Consider $E = x_1y_1 + x_2y_2 + x_3y_3$ in disjunctive normal form. Map into conjunctive normal form. Transform into an equivalent boolean expression F

$$\begin{aligned} F =& (x_1 + x_2 + x_3)(y_1 + x_2 + x_3)(x_1 + y_2 + x_3)\wedge \\ & (y_1 + y_2 + x_3)(x_1 + x_2 + y_3)(x_1 + x_2 + y_3)\wedge \\ & (x_1 + y_2 + y_3)(y_1 + y_2 + y_3) \end{aligned}$$

OR of n AND's + each AND has 2 vars. AND of 2^n clauses of n vars.

Step 1.

$$\begin{aligned} E &= \neg((\neg(x+y))(\overline{x}+y)) \\ &\Rightarrow \neg(\neg(x+y)) + \neg(\overline{x}+y) \\ &\Rightarrow x + y + \neg(\overline{x}+y) \\ &\Rightarrow x + y + x\overline{y} \end{aligned}$$

Step 2. Transform the resulting expression E of length n into an expression F such that

1. F is in CNF and has at most n clauses.
2. F is constructible from E in $O(|E|)$ time.
3. A truth assignment T for E makes E satisfiable iff there exists an extension S of T that makes F true.