COMS3261: Computer Science Theory

Fall 2013

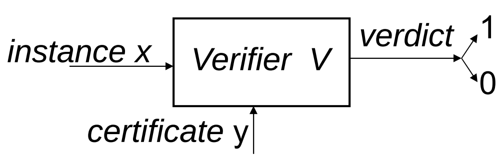
Mihalis Yannakakis

Lecture 25, 12/4/13

NP = short, verifiable certificates

- Verifier V: Polynomial time algorithm which checks that y is a certificate for x

$$L = \{x \in A^* \mid \exists y \in B^*,\ |y| \le |x|^c,\ V(x,y) = 1\}$$

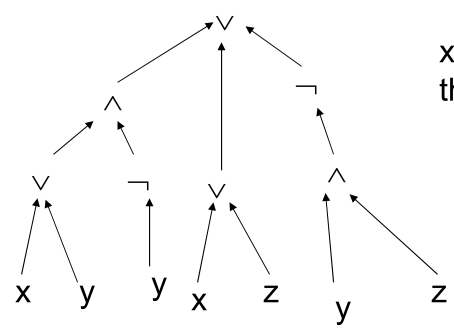


- $x \in L \Rightarrow \exists y \in B^*,\ |y| \le |x|^c$ s.t. $V(x,y) = 1$
- $x \notin L \Rightarrow \forall y \in B^*,\ |y| \le |x|^c.\ V(x,y) = 0$

Example: (Boolean Formula) Satisfiability

- Input: Boolean formula using AND,OR, NOT connectives (operators) with several variables
- Output: Yes, if there is an assignment to the variables of True/False (1/0) that makes the formula true

Example: $((x \vee y) \wedge (\neg y)) \vee (x \vee z) \vee (\neg(y \wedge z))$



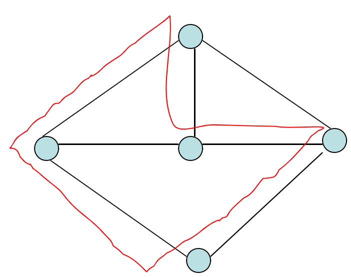
x=y=z=0 satisfies the formula

Examples

- Graph Hamiltonicity: Instance: Graph G

  certificate y = cycle that goes through every node exactly once (Hamiltonian cycle)

  - Eulerian graph (go through every edge once) in P



Traveling Salesman problem (TSP): cities, distances, TSP tour

Optimization problems

Optimization → Decision Problem

- It is common to define complexity classes like P and NP as classes of *decision problems* - problems with Yes/No (0/1) answer.
- Optimization problem Π: Every instance I has a set of solutions, every solution has a value (profit or cost). The problem is to compute the maximum or minimum possible value and a solution that achieves it.
- Corresponding Decision problem for Π:

  Input: instance I of Π, bound b on value

  Output: Yes iff optimum value $\geq$b for maximization problem (resp. optimum value $\leq$b for minimization problem), else No

Examples of Decision versions

- Minimum Coloring Problem:
  Input: Graph G, bound b
  Output: Yes if G can be colored with ($\leq$) b colors, else No
- Traveling Salesman problem
  Input: Distances between n cities, bound b
  Output: Yes, if $\exists$ TSP tour of length $\leq$ b
- Clique Problem
  Input : Graph G, bound b
  Output: Yes, if $\exists$ clique with $\geq$ b (mutually adjacent) nodes
  (mutually compatible entities)

Optimization vs. Decision Problem

- Optimization problem Π is at least as hard as the corresponding Decision problem: solving opt. Π gives immediately answer for decision
- In most cases, it is possible (but harder) to go also the other way using a routine for the Decision problem repeatedly (but only a polynomial number of times).
- Step 1. Find the optimal value
- Step 2. Find an optimal solution, one piece at a time

Functional problems → Decision problems

- Factoring – decision version

Input: Number N, numbers a,b

Output: Yes, if N has a prime factor between a and b, else No

- Can factor N with polynomially many calls to an algorithm for the decision problem. (Use binary search to find a prime factor of N, divide N by the factor, and repeat).

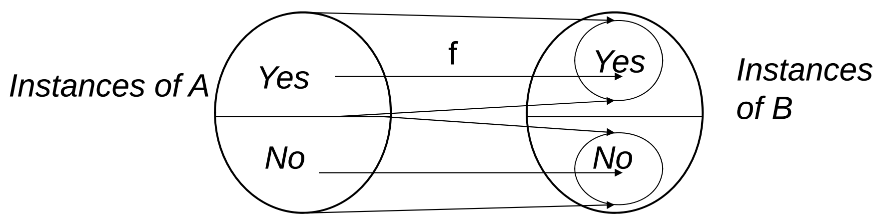
- Any "functional" problem Π (problem with output) can be reduced to a series of calls to a decision problem:

Input: Instance x of Π, index i

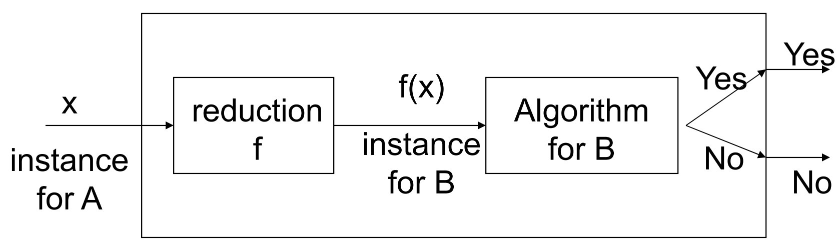
Output: Yes, if the i-th bit of the output of Π on input x is 1, No otherwise

Reductions between Decision Problems

- Polynomial time Reduction from language L to L' (notation $L \leq_P L'$): polynomial time computable function
  f: $\{0,1\}^* \rightarrow \{0,1\}^*$ s.t. $\forall x \in \{0,1\}^*$, $x \in L$ iff $f(x) \in L'$
- Polynomial time Reduction from decision problem A to B (notation $A \leq_P B$): reduction from L_A to L_B (where L_A = set of Yes instances of A and L_B = set of Yes instances of B)
  f maps Yes instances of A to Yes instances of B and No instances of A to No instances of B



Reductions between Decision Problems



- If $B \in P$ then also $A \in P$
- If $A \notin P$ then also $B \notin P$
- If $B \in NP$ then also $A \in NP$
- If $A \notin NP$ then also $B \notin NP$

Reductions compose: $A \leq_P B$ and $B \leq_P C \Rightarrow A \leq_P C$

Complementation: $A \leq_P B \Rightarrow \bar{A} \leq_P \bar{B}$

NP-hardness and NP-completeness

- A language L (decision problem) is NP-hard if every language L' in NP reduces to it: $\forall L' \in NP$, $L' \leq_P L$
- A language L (decision problem) is NP-complete if
  1. L is in NP, and
  2. L is NP-hard

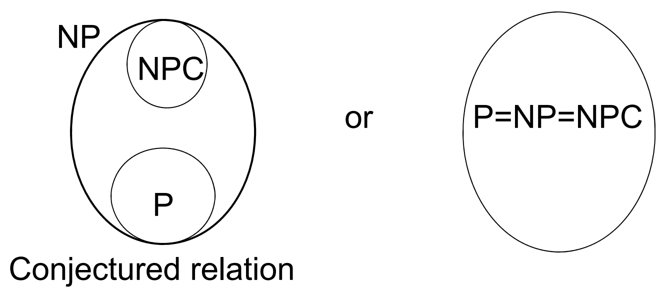
- If A is NP-hard and $A \leq_P B$ then B is also NP-hard

- All NP-complete problems reduce to each other

NP-complete problems and P vs NP

- If any NP-hard problem L is in P then P=NP
  Proof: For every $A \in NP$, $A \leq_P L$ and $L \in P$ imply $A \in P$

- Either all NP-complete problems are in P (and P=NP) or no NP-complete problem is in P (and $P \neq NP$



NP-complete Problems

- (Boolean Formula) Satisfiability
- Graph 3-colorability
- Graph Hamiltonicity
- Travelling Salesman problem
- Clique
- ….
- And thousands and thousands of other problems from all disciplines are NP-complete or NP-hard.
- Widespread phenomenon!

- We only have exponential time algorithms for solving NP-complete problems.
- It is widely believed that it is impossible to solve them in polynomial time

NP-hard problems for automata and regular expressions

- Is a given regular expression ≡ (0+1)* ?
- Is the language of a given NFA = (0+1)* ?
- Is L(A)= L(B) for two given NFA A, B ?
- Is L(R)= L(S) for two given regular expressions R,S ?

- All are NP-hard (at least as hard as all problems in NP) but they are not known and are not believed to be in NP
  - Shortest counterexample string to the equality may be exponentially long in the size of the regular expressions or the NFA.