CS 3100, 11/23/10
Ganesh Gopalakrishnan

http://www.cs.utah.edu/fv
Asg8

- 1(f) - convert $\iff$ to $\implies$
- All problems: assume $x, y, k$ are in Nat
- $a, b, c$ are of course Boolean
- I misspoke about mapping reductions
  - They need not be 1-1
- GCD questions: follow defn of GCD
  - Is a divisor
  - Is the largest
  - $X$ and $Y$ divisible by $Z$ means $(X+/Y)$ div by $Z$
- Clique questions: Think of how cliques are built
  - What is a 1-clique? 2-clique? 3-clique? 4-clique? ...
- Do Qn4 without using Rice’s Theorem
  - Similar to Reg(TM) problem
  - “Floor trap-door is opened” based on whether $M$ accepts $w$
• Counting Boolean functions over $N$ inputs
  – Of course, only finitely many
  – But grows quite fast!

• Contrast with counting Nat -> Nat functions
  – Try to enumerate functions
  – We can find a function not in the enumeration
  – Is of higher cardinality
Mapping reductions

• Basic idea:

• Given a set $A$ and a set $B$, we are seeking an “embedding of $A$ in $B$” that
  – Preserves membership
  – $A \leq^m B$ is the notation
  – You can read it also as “$A$ is less hard or the same hardness as $B”
  – We are going to practice it on 2(a) and 2(b) - no computability connotation
    • Simply try to read “IFF”
  – Then do 2(c) which tries to force you to think of language -> language mapping redns
  – $<M,w>$ pairs in $A_{TM}$ are mapped to $<M>$ singletons in the language $A_{bt}$
  – See if all conditions for an MR are satisfied by the constructed mapping reduction
Mapping reductions

• Given an $M$ and $w$
• Build a new TM $M_w$ that has “$w$” embedded in it
  – Say in a “data array”
• Then give $M_w$ to the claimed decider for $A_{bt}$
• What will $M_w$ do when run?
  – Erases input
  – Writes $w$ from data array onto tape
  – Runs $M$’s code on input
• If $D_{A_{bt}}$ can take machines in an “unsuspecting” manner and claim to answer the acceptance of “$e$” of those machines
  – Then it may be fed a “loaded” machine such as $M_w$
Mapping reductions

• Study mapping reduction in the case of NPC (3CNF formula to Graphs) also

• Preserves hardness in both cases
  – If we can solve A_{bt}, we can solve A_{TM} because A_{TM} is <= in hardness
  – If we can solve Clique in poly-time, we can solve 3SAT also in poly-time
MT2

• Language blending
  – S -> 0S | 1S | e | T
  – T -> generates a CFG but its structure is blended away!

Try this:
  S -> T T | U
  U -> 0 U 0 0 | #
  T -> 0 T | T 0 | #
Complexity theory

• Various complexity classes
• Reduction principles remain the same
• Exp-time complete
• P-space complete
  – Pspace and Npspace are the same
  – Space can be reused! Time can’t be!
    • How about energy?
    • Charles Seitz and Tom McKnight (and others) used to talk about “Hot clocking” and “Adiabatic circuits”
    • Charge sloshes back and forth (inductor in clock path; circuit is capacitive)
    • Some energy recovery happens - as opposed to this, in real CMOS ckt, the energy pumped into the capacitors is destroyed and turned into heat
  – So I don’t know whether the “reuse” of energy happens in the same sense
    • Google queries: each can heat a cup of water to near boil
    • But the water in the hydro plant would otherwise have hit the rocks and generated heat that way also
  – Bottomline: if you harvest energy at every spot, perhaps we are OK burning a whole lot (roads and roofs can produce energy)
Complexity theory

• NP-complete
  – Ptime and Nptime are different
• NP-hard
• P-complete
  – Relevant for parallelization
  – BFS can be parallelized more easily
  – DFS - not so
    • Is P-complete
Complexity theory

• Sometimes, complexity classes are not known
• E.g. for some problems, the time-complexity characterization is still an open problem
• In that case, just do what we can! i.e. get space complexity results
• NP-hard : At least as hard as NP
  – All problems in NP have a \(<=m\) to that problem which is NPH
  – Note that Diophantine is NPH
  – At least as hard as NP
  – But really really really hard (undecidable)
  – So to show NPC, must show that it is in NP also
  – ND algorithm has a P-time solution
Complexity theory

- ND algorithm
- Guess and check
- Guess must result in poly-long “certificate”
- Check must be doable in poly-time
- Showing that some problems have poly certificates took effort!
  - Pratt showed that Primality certificates are poly (in 1976)
  - But then we have a cool result: If NPC and CO-NP then NP = Co-NP
  - But since the consequent is unlikely, then for problems that are NP and Co-NP, then it may be that they are not NPC
  - Sure enough, Agrawal, Kayal, and Saxena (the latter two are BS CS students!) showed that primarily has a Det Poly checking algorithm
  - This is NOT the same as prime factorization: the language changes!
Complexity theory

• The same happened to lin programming
• Kachian came up with Poly algorithm
• But it was well known that Lin Prog and its complement are in NP
• (there is more to this... ask Prof. Suresh Venkat)

• Certificate “blowup” is indicative of hardness
• You saw that in PCP and also in Diophantine in a different light (not having succinct certificates is trouble)
Complexity theory

• Strongly NPC
  – Problem hardness does not change by encoding method
  – 3SAT, Tetris, etc are so
  – 3-partitioning is so

• Not strongly NPC (pseudo-polynomial)
  – Can reduce complexity by bloating input
  – 2-partitioning is so
NPC uses

• Don’t run away if NPC
• Don’t run away if undecidable
• All it means is that the FULL language is hard
• Pieces of the language may be easy
• That is what BDDs will sort of teach us
• Will do this + Bool Sat after Turkey-Day
• Gobble Gobble meanwhile!
Wish you...