Computational Complexity

Mohammad Mahmoody

6th Session

30 Jan 2014

Last Time

$L \in$

$L \in P \subseteq NP$

- Many natural combinatorial problems are $\mathbf{NP}$-complete: CircSAT, 3SAT, IND-SET, CLIQUE, V-Cover
  So to find out whether $\mathbf{P} \neq \mathbf{NP}$ we can focus on these problems.

$\boxed{P \overset{?}{\neq} NP}$

- Search Problems and their close relation to the class $\mathbf{NP}$.

- Search vs. Decision: For Circ-SAT they were "equivalent".

Today

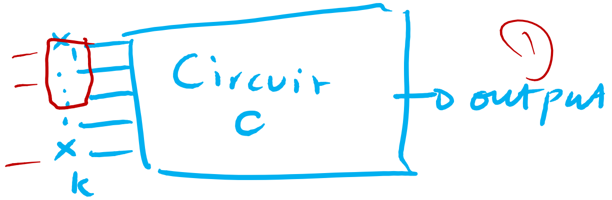
- Search vs. Decision for other problems in $\mathbf{NP}$
- The notion of Turing reductions (and generalizing $\mathbf{NP}$ hardness)
- Complement of languages and relevant classes

Search Vs. Decision: **CircSAT**

Search_Circ-SAT $\leq_?$ Decision-Circ-SAT

- Theorem: If we could solve (Decision) **Circ-SAT** in polynomial time we can also solve the search version: **Search-Circ-SAT** in polynomial time

A
- Suppose B solves Circ-SAT
- We are given circuit C:



- Let $C_0 = C$
- for $i = 1, 2, \ldots, k$: let $C_{i-1}^{x_i = true}$ be C_{i-1} where x_i is always true

  Run B to find out whether $C_{i-1}^{x_i = true}$ is satisfiable

  if so, let $C_i = C_{i-1}^{x_i = true}$ and $x_i = true$

  otherwise let $x_i = false$ and $C_i = C_{i-1}^{x_i = false}$.
- output $X = (X_1, \ldots, X_k)$

What kind of Reduction was that?

- We assumed a "subroutine" B that solves **Circ-SAT**
  Presented an algorithm A that uses B and solves **Search-Circ-SAT**

- It is called a **Cook** or a **Turing** reduction: Given a subroutine B that solves some "problem" X, A uses B and solves some other problem Y. Notation $\cancel{X} \leq_T \cancel{Y}$
  Y X
  poly-time Algorith..

- Notation A^B or just A^X:
  Algorithm A gets accesses a subroutine B or some subroutine that solves X.

- A does not care how X is solved: uses solver as a "black-box" (a.k.a. "oracle").

  Suppose X is Halting problem

- It is meaningful to talk about A^X even if X is not solvable efficiently (or at all).

What is an $\boxed{\textbf{NP}}$-hard problem?

Complete

- Previously we called a language L **NP**-~~hard~~ if:
  1. $L \in NP$
  2. There is a Karp reduction from any $S \in NP$ to L : $S \leq_p L$

NP hardness

- The above definition is just **NP**-hardness under Karp reduction
- We can talk about **NP**-hard under Turing reductions as well:
- Call any problem X **NP**-hard if for all $S \in$ **NP** there is a polynomial-time Turing reduction R such that R^X dcides S on all inputs correctly

NP-hardner

$R^X(\alpha)$ outputs β

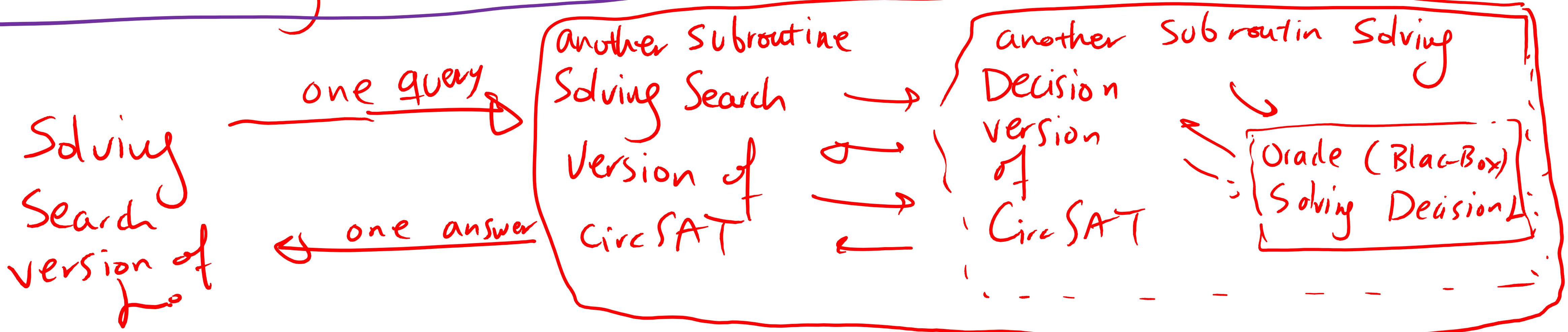
How about other problems?

even under Turing Reductions.

- Theorem: Suppose L is *any* **NP** complete language. Then there is a Turing reduction from the search version of L (where, given x we want to find a witness w that $x \in L$) to (decision version of) L.

We are given x, and have oracle access to B solving Decision L

Goal: find witness w for $x \in L$.



putting steps together:

1. Using oracle for Decision version of L we can implement a subroutin (oracle) for (Decision) CircSAT

2. Using subroutine for Decision-CircuitSAT we can solve Search version of CircSAT

3. Using search version of CircSAT we can Solve Search version of problem L.

1 is because L is NP-hard (which is the case because it is NP-complete)

2 is because of the proof we saw last time

3 is because of Cook-Levin reduction from L to CircuitSAT and that it gives witness for $x \in L$,