# COMP4141 Theory of Computation

## Lecture 12 NP-Completeness


Ron van der Meyden

CSE, UNSW



Revision: 2013/04/21

(Credits: Rob van Glabbeek, Peter Hofner, D Dill, K Engelhardt, M Sipser, W Thomas, T Wilke)

# $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Recall:
**P**—problems that can be solved in polynomial time on a TM.
**NP**—problems that can be solved in polynomial time on an NTM.

**Question**

*Is* $\mathbf{P} = \mathbf{NP}$*?*

*Equivalently: Can we simulate a polynomial time non-deterministic TM (NTM) in polynomial time on a (deterministic) TM?*

At this point, no one knows for sure, but "no" might be a good bet.

# NP-complete problems

This is about decision problems (problems with yes/no answers).

Equivalently, solving the membership problem $x \in L$.

Obviously $\mathbf{P} \subseteq \mathbf{NP}$.

Nobody knows for sure whether $\mathbf{NP} \subseteq \mathbf{P}$

Intuitively, **NP**-complete problems are the "hardest" problems in **NP**.

# **P** Reducibility

Recall how we use mapping-reducibility to transfer (un)decidabilty from one problem to the next.

**Definition**

$f : \Sigma^* \longrightarrow \Sigma^*$ is a *polynomial time computable* (or **P** *computable*) function if some polynomial time TM $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w \in \Sigma^*$.

**Definition**

$A \subseteq \Sigma_1^*$ is *polynomial time mapping reducible* (or **P** *reducible*) to $B \subseteq \Sigma_2^*$, written $A \leq_{\mathbf{P}} B$, if a **P** computable function $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ exists that is also a reduction (from $A$ to $B$).

# P Reducibility cont.

**Theorem**

*If $A \leq_{\mathbf{P}} B$ and $B \in \mathbf{P}$ then $A \in \mathbf{P}$.*

**Proof.**

Suppose $f$ is the **P** reduction from $A$ to $B$ that runs in time $O(n^{k_1})$ and $M_B$ is a **P** decider for $B$ that runs in time $O(n^{k_2})$.

To decide $w \in A$, first compute $f(w)$, then run $M_B$ on $f(w)$.

Note that the input to $M_B$ has size at most $|w|^{k_1}$.

The total running time is $O(|w|^{k_1} + (|w|^{k_1})^{k_2}) = O(|w|^{k_1 k_2})$.

□

# NP-Hardness

**Definition**

A language $B$ is **NP**-*hard* if *every* $A \in$ **NP** is **P** reducible to $B$.

Intuitively, this says $B$ is at least as hard as any problem in **NP**.

**Theorem**

*If $B$ is* **NP***-hard and $B \leq_{\mathbf{P}} C$ then $C$ is* **NP***-hard.*

# NP-Completeness

**Definition**

A language $B$ is **NP**-*complete* if

1. $B \in \mathbf{NP}$
2. $B$ is **NP**-hard (i.e. *every* $A \in \mathbf{NP}$ is **P** reducible to $B$).

**Theorem**

*If $B$ is **NP**-complete and $B \in \mathbf{P}$ then $\mathbf{P} = \mathbf{NP}$.*

**Theorem**

*If $B$ is **NP**-complete and $B \leq_{\mathbf{P}} C$ for $C \in \mathbf{NP}$, then $C$ is **NP**-complete.*

**Proof.**

Polynomial time reductions compose. □

# NP-Completeness

If there are any problems in **NP** $\setminus$ **P**, the **NP**-complete problems are all there.

Every **NP**-complete problem can be translated in deterministic polynomial time to every other **NP**-complete problem.

So, if there is a **P** solution to one **NP**-complete problem, there is a **P** solution to every **NP** problem.

# NP-Hardness by Reduction

Typical method: Reduce a known **NP**-hard problem $P_1$ to the new problem $P_2$ .

# Basic Proof Strategy

**NP**-completeness is a good news/bad news situation.

- Good news: The problem is in **NP**!
- Bad news: The problem is **NP**-hard!

So, a typical **NP**-completeness proof consists of two parts:

1. Prove that the problem is in **NP** (i.e., it has **P** verifier).
2. Prove that the problem is at least as hard as other problems in **NP**.

A TM can simulate an ordinary computer in polynomial time, so it is sufficient to describe a polynomial-time checking algorithm that will run on any reasonable model of computation.

# NP-Hardness

A problem is **NP**-*hard* if having a polynomial-time solution to it would give us a polynomial solution to every problem in **NP**.

**Proving that the problem is NP-hard**: The usual strategy is to find a polynomial-time reduction of a known **NP**-hard problem (say $P_1$) to the problem in question (say $P_2$).

The goal is to show that $P_2$ is at least as hard (in terms of polynomial vs. super-polynomial time) as $P_1$.

Repeated warning: Make sure you are reducing the known problem to the unknown problem!

In practice, there are now thousands of known **NP**-complete problems. A good technique is to look for one similar to the one you are trying to prove **NP**-hard.

Computers and Intractability - A guide to theory of NP-completeness, M.R. Garey and D.S. Johnson, Freeman 1979 lists a whole bunch.

# Boolean Formulae

Let $Prop = \{x, y, \ldots\}$ be a countable set of *Boolean variables* (or *propositions*).

A CFG for Boolean formulae over *Prop* is:

$$\phi \to p \mid \phi \wedge \phi \mid \neg\phi \mid (\phi)$$
$$p \to x \mid y \mid \ldots$$

We use abbreviations such as

$$\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2) \qquad \phi_1 \Rightarrow \phi_2 = \neg\phi_1 \vee \phi_2$$
$$\text{FALSE} = (x \wedge \neg x) \qquad \text{TRUE} = \neg\text{FALSE}$$

Let $Prop(\phi)$ be the propositions that occur in $\phi$.

# Semantics of Boolean Formulae

A Boolean formula is either **TT** (for "true") or **FF** (for "false"), possibly depending on the interpretation of its propositions. Let $\mathbb{B} = \{\mathbf{FF}, \mathbf{TT}\}$.

**Definition**

An *interpretation* (of $\mathit{Prop}(\phi)$) is a function $\pi : \mathit{Prop}(\phi) \longrightarrow \mathbb{B}$.
For Boolean formulae $\phi$ we define $\pi$ *satisfies* $\phi$, written $\pi \models \phi$, inductively by:

**Base:** $\pi \models x$ iff $\pi(x) = \mathbf{TT}$.

**Induction:**

- $\pi \models \neg\phi$ iff $\pi \not\models \phi$.
- $\pi \models \phi_1 \wedge \phi_2$ iff both $\pi \models \phi_1$ and $\pi \models \phi_2$.
- $\pi \models (\phi)$ iff $\pi \models \phi$.

$\phi$ is *satisfiable* if there exists an interpretation $\pi$ such that $\pi \models \phi$.

# SAT—An NP-Complete Problem

$$SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula } \}$$

**Theorem**

*SAT is* **NP***-complete.*

**Proof of SAT $\in$ NP.**

If $\pi \models \phi$ we use $\langle \pi \rangle$ as certicate. $\square$

# Proof of NP-Hardness of SAT

Let $A \in \mathbf{NP}$. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a deciding NTM with $L(M) = A$ and let $p$ be a polynomial such that $M$ takes at most $p(|w|)$ steps on any computation for any $w \in \Sigma^*$.

Construct a $\mathbf{P}$ reduction from $A$ to $SAT$. On input $w$ a Boolean formula $\phi_w$ that describes $M$'s possible computations on $w$. $M$ accepts $w$ iff $\phi_w$ is satisfiable. The satisfying interpretation resolves the nondeterminism in the computation tree to arrive at an accepting branch of the computation tree.

Remains to be done: define $\phi_w$.

# Proof of NP-Hardness of SAT cont.

Recall that $M$ accepts $w$ if an $n \leq p(|w|)$ and a sequence $(C_i)_{0<i\leq n}$ of configurations exist, where

1. $C_1 = q_0 w$,
2. each $C_i$ can yield $C_{i+1}$, and
3. $C_n$ is an accepting configuration.

Let $\Delta = Q \cup \Gamma \cup \{\#\}$. Each $C_i$ can be represented as a $\#$-enclosed string over alphabet $\Delta$ no longer than $n + 3$.

# $\phi_w$

The Boolean formula $\phi_w$ shall represent *all* such sequences $(C_i)_{0<i\leq n}$ beginning with $q_0 w$.

$$\phi_w = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

# $\phi_{\textbf{cell}}$

… describes an $n^2$ grid using propositions

$$Prop = \{\ x_{i,k,s} \mid i, k \in \{1, \ldots, n\} \wedge s \in \Delta\ \}\ .$$

$$\phi_{\text{cell}} = \bigwedge_{0<i,k\leq n} \left( \bigvee_{s\in\Delta} x_{i,k,s} \wedge \bigwedge_{s,t\in\Delta,s\neq t} (\neg x_{i,k,s} \vee \neg x_{i,k,t}) \right)$$

Row $i$ in the grid corresponds to configuration $C_i$. Unused tape cells are blank.

Every grid cell contains exactly one symbol or a state.

# $\phi_{\textbf{start}}$

...specifies that the first row of the grid contains $q_0 w$ where $w = w_1 \dots w_{|w|}$:

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \bigwedge_{2<i\leq|w|+2} x_{1,i,w_{i-2}} \wedge \bigwedge_{|w|+2<i\leq n-1} x_{1,i,\sqcup} \wedge x_{1,n,\#}$$

## $\phi_{\textbf{move}}$

...ensures that $C_i$ yields $C_{i+1}$ by describing *legal* $2 \times 3$ windows of cells.

$$\phi_{\text{move}} = \bigwedge_{0<i,k<n} \bigvee_{\begin{array}{|c|c|c|}\hline a_1 & a_2 & a_3 \\ \hline a_4 & a_5 & a_6 \\ \hline \end{array} \text{ is legal}} \left( \begin{array}{l} x_{i,k-1,a_1} \wedge x_{i,k,a_2} \wedge x_{i,k+1,a_3} \wedge \\ x_{i+1,k-1,a_4} \wedge x_{i+1,k,a_5} \wedge x_{i+1,k+1,a_6} \end{array} \right)$$

what is legal depends on the transition function $\delta$.

# $\phi_{\textbf{accept}}$

...states that the accept state is reached:

$$\phi_{\text{accept}} = \bigvee_{0<i,k\leq n} x_{i,k,q_{\text{accept}}}$$

Finally we check that the size of $\phi_w$ is polynomial in $|w|$ and that $\phi_w$ is constructable in polynomial time.

—THE END—