COMP3630 / COMP6363

week 10: **Other Complexity Classes**

This Lecture Covers Chapter 11 of HMU: Other Complexity Classes

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- The Tautology Problem.
- **co-NP** and its relation to **NP**.
- A different (possibly non-equivalent) notion of **NP**-Hardness.
- Optimization Problems.

Additional Reading: Chapter 11 of HMU.

The Tautology Problem

Definition 11.1.1

Recall:

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

Definition 11.1.2

A boolean formula is a tautology if it evaluates to true for all truth value assignments. The Tautology Problem is the set of all boolean formulae that are tautologies:

$$TAUT = \{ \langle\phi\rangle \mid \phi \text{ is a tautological Boolean formula } \}$$

Is *TAUT* in NP?

Is there a check and verify approach?

The Tautology Problem

Definition 11.1.1

Recall:

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

Definition 11.1.2

A boolean formula is a tautology if it evaluates to true for all truth value assignments.
The Tautology Problem is the set of all boolean formulae that are tautologies:

$$TAUT = \{ \langle \phi \rangle \mid \phi \text{ is a tautological Boolean formula } \}$$

Is *TAUT* in **NP**?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is false.
  - Assume π makes our formula true. Now what? (Is that even relevant?)

The Tautology Problem

Definition 11.1.1

Recall:

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

Definition 11.1.2

A boolean formula is a tautology if it evaluates to true for all truth value assignments. The Tautology Problem is the set of all boolean formulae that are tautologies:

$$TAUT = \{ \langle\phi\rangle \mid \phi \text{ is a tautological Boolean formula } \}$$

Is *TAUT* in **NP**?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is false.
  - Assume π makes our formula true. Now what? (Is that even relevant?)
- So, we don't know! TAUT could be in **NP**, but we don't know.
  (I.e., there could be certificate that's not an assignment.)

On the Hardness of *TAUT*

Theorem 11.1.3

If TAUT is in **P**, *then every* **NP** *problem is in* **P**.

Proof.

We show that we could solve any *SAT* problem in **P** if *TAUT* is in **P**. (*SAT* is **NP**-hard!)

On the Hardness of *TAUT*

Theorem 11.1.3

If TAUT is in **P**, *then every* **NP** *problem is in* **P**.

Proof.

We show that we could solve any *SAT* problem in **P** if *TAUT* is in **P**. (*SAT* is **NP**-hard!)

- A formula ϕ is satisfiable if $\neg\phi$ is not a tautology. (You can easily prove this.)
  E.g., $\phi = (x \vee \neg y) \wedge y$, $\neg\phi = (\neg x \wedge y) \vee \neg y$.
  For $\pi(x) = \top$ and $\pi(y) = \top$ we get $\pi \models \phi$ and $\pi \not\models \neg\phi$.
- Solve *SAT* in polytime:
  - If ϕ is the input, run *TAUT* on $\neg\phi$.
  - flip the result.

□

On the Hardness of *TAUT*

Theorem 11.1.3

If TAUT is in **P***, then every* **NP** *problem is in* **P***.*

Proof.

We show that we could solve any *SAT* problem in **P** if *TAUT* is in **P**. (*SAT* is **NP**-hard!)

- A formula ϕ is satisfiable if $\neg\phi$ is not a tautology. (You can easily prove this.)
  E.g., $\phi = (x \vee \neg y) \wedge y$, $\neg\phi = (\neg x \wedge y) \vee \neg y$.
  For $\pi(x) = \top$ and $\pi(y) = \top$ we get $\pi \models \phi$ and $\pi \not\models \neg\phi$.
- Solve *SAT* in polytime:
  - If ϕ is the input, run *TAUT* on $\neg\phi$.
  - flip the result.

□

Question

- Have we shown that *TAUT* is **NP**-hard?

On the Hardness of *TAUT*

Theorem 11.1.3

If TAUT is in **P**, *then every* **NP** *problem is in* **P**.

Proof.

We show that we could solve any *SAT* problem in **P** if *TAUT* is in **P**. (*SAT* is **NP**-hard!)

- A formula ϕ is satisfiable if $\neg\phi$ is not a tautology. (You can easily prove this.)
  E.g., $\phi = (x \vee \neg y) \wedge y$, $\neg\phi = (\neg x \wedge y) \vee \neg y$.
  For $\pi(x) = \top$ and $\pi(y) = \top$ we get $\pi \models \phi$ and $\pi \not\models \neg\phi$.
- Solve *SAT* in polytime:
  - If ϕ is the input, run *TAUT* on $\neg\phi$.
  - flip the result.

□

Questions

- Have we shown that *TAUT* is **NP**-hard?
- **No!** This was not a polytime reduction from *SAT* to *TAUT*. Why?

On the Hardness of *TAUT*

Theorem 11.1.3

If TAUT is in **P**, *then every* **NP** *problem is in* **P**.

Proof.

We show that we could solve any *SAT* problem in **P** if *TAUT* is in **P**. (*SAT* is **NP**-hard!)

- A formula ϕ is satisfiable if $\neg\phi$ is not a tautology. (You can easily prove this.)
  E.g., $\phi = (x \lor \neg y) \land y$, $\neg\phi = (\neg x \land y) \lor \neg y$.
  For $\pi(x) = \top$ and $\pi(y) = \top$ we get $\pi \models \phi$ and $\pi \not\models \neg\phi$.
- Solve *SAT* in polytime:
  - If ϕ is the input, run *TAUT* on $\neg\phi$.
  - flip the result.

□

Questions

- Have we shown that *TAUT* is **NP**-hard?
- **No!** This was not a polytime reduction from *SAT* to *TAUT*. Why?
- Because we flipped the result! We don't implement $w \in SAT$ iff $f(w) \in TAUT$.

The Tautology Problem – and its relation to its complement

Definition 11.1.4

$$TAUT^c = \{ \langle \phi \rangle \mid \phi \text{ is } \underline{\text{not}} \text{ a tautological Boolean formula} \}$$

Is $TAUT^c$ in NP?

Is there a check and verify approach?

The Tautology Problem – and its relation to its complement

Definition 11.1.4

$$TAUT^c = \{ \langle \phi \rangle \mid \phi \text{ is } \underline{\text{not}} \text{ a tautological Boolean formula } \}$$

Is $TAUT^c$ in **NP**?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is yes. That's enough! This is our certificate.
- So, yes, $TAUT^c$ is in **NP**

The Tautology Problem – and its relation to its complement

Definition 11.1.4

$$TAUT^c = \{ \langle\phi\rangle \mid \phi \text{ is } \underline{\text{not}} \text{ a tautological Boolean formula } \}$$

Is $TAUT^c$ in NP?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is yes. That's enough! This is our certificate.
- So, yes, $TAUT^c$ is in **NP**

Key Messages

So what was the problem on why can't (easily, if at all) show that $TAUT$ is in **NP**?

- The non-det. TM can't (easily) guess and verify for the yes answer.

The Tautology Problem – and its relation to its complement

Definition 11.1.4

$$TAUT^c = \{ \langle\phi\rangle \mid \phi \text{ is } \underline{\text{not}} \text{ a tautological Boolean formula} \}$$

Is $TAUT^c$ in **NP**?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is yes. That's enough! This is our certificate.
- So, yes, $TAUT^c$ is in **NP**

Key Messages

So what was the problem on why can't (easily, if at all) show that *TAUT* is in **NP**?

- The non-det. TM can't (easily) guess and verify for the yes answer.
- A non-det. TM can guess and verify for the no answer. (The complement!)

The Tautology Problem – and its relation to its complement

Definition 11.1.4

$$TAUT^c = \{ \langle\phi\rangle \mid \phi \text{ is } \underline{\text{not}} \text{ a tautological Boolean formula } \}$$

Is $TAUT^c$ in **NP**?

Is there a check and verify approach?

- Guess an assignment π.
  - Assume π makes our formula false. Then we know the answer is yes. That's enough! This is our certificate.
- So, yes, $TAUT^c$ is in **NP**

Key Messages

So what was the problem on why can't (easily, if at all) show that $TAUT$ is in **NP**?

- The non-det. TM can't (easily) guess and verify for the yes answer.
- A non-det. TM can guess and verify for the no answer. (The complement!)
- A non-det. TM *could* decide the problem if it could guess all assignments simultaneously. (And return yes if all of them make the formula true.)

The class **co-NP**

Definition 11.2.1

A problem is in **co-NP** if and only if its complement is in **NP**.

Key Message

- Thus, **co-NP** contains problems where we can guess a certificate and verify it in polytime for <u>no</u> instances.
- These are problems where disproving the property is "easier" than proving it. (Easier in the sense that one witness suffices.)

The class **co-NP**

Definition 11.2.1

A problem is in **co-NP** if and only if its complement is in **NP**.

Key Message

- Thus, **co-NP** contains problems where we can guess a certificate and verify it in polytime for no instances.
- These are problems where disproving the property is "easier" than proving it. (Easier in the sense that one witness suffices.)

Theorem 11.2.2

1. $\mathbf{P} \subseteq \textbf{co-NP}$
2. *If* $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{P} = \mathbf{NP} = \textbf{co-NP}$.

The class **co-NP**

Definition 11.2.1

A problem is in **co-NP** if and only if its complement is in **NP**.

Key Message

- Thus, **co-NP** contains problems where we can guess a certificate and verify it in polytime for no instances.
- These are problems where disproving the property is "easier" than proving it. (Easier in the sense that one witness suffices.)

Theorem 11.2.2

1. $\mathbf{P} \subseteq \textbf{co-NP}$
2. *If* $\mathbf{P} = \mathbf{NP}$, *then* $\mathbf{P} = \mathbf{NP} = \textbf{co-NP}$.

Proof.

Because P is closed under complementation. □

co-NP-Hardness and -Completeness

Definition 11.2.3

A problem B is **co-NP**-hard if <u>every</u> $A \in$ **co-NP** is **P**-reducible to B.
A problem B is **co-NP**-complete if it's in **co-NP** and **co-NP**-hard.

co-NP-Hardness and -Completeness

Definition 11.2.3

A problem B is **co-NP**-hard if every $A \in$ **co-NP** is **P**-reducible to B.
A problem B is **co-NP**-complete if it's in **co-NP** and **co-NP**-hard.

Theorem 11.2.4

TAUT is **co-NP***-complete.*

Proof.

See (i.e., try it yourself) in the tutorials. □

Cook-Completeness

Definition 11.3.1

A problem X is Karp-**NP**-hard (resp., complete), if every **NP** problem can be reduced to X in polytime (and $X \in \mathbf{NP}$, resp.). – *That's our standard definition!*

Definition 11.3.2

A problem X is Cook-**NP**-hard (resp., complete), if one can show that if $X \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$ (and $X \in \mathbf{NP}$, resp.).

Example 11.3.3

We have shown that *TAUT* is Cook-**NP**-hard. (But we don't know **NP**-membership.)

Cook-Completeness

Definition 11.3.1

A problem X is Karp-**NP**-hard (resp., complete), if every **NP** problem can be reduced to X in polytime (and $X \in \mathbf{NP}$, resp.). – *That's our standard definition!*

Definition 11.3.2

A problem X is Cook-**NP**-hard (resp., complete), if one can show that if $X \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$ (and $X \in \mathbf{NP}$, resp.).

Example 11.3.3

We have shown that *TAUT* is Cook-**NP**-hard. (But we don't know **NP**-membership.)

Remark

- Cook-completeness is Cook's original definition.
- Cook was interested in why *TAUT* is hard.
- *TAUT* as 'true mathematical theorems'.

Cook vs. Karp

Biggest Difference

- Cook lets us flip the answer after a polytime reduction.
- Karp-completeness implies Cook-completeness.
- If **P** $=$ **NP**, they would both be the same.

Why Karp?

If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any **NP** problem in that time
- with Cook, we cannot conclude anything. (It's too weak.)

Optimization Problems

Observation

So far:

- We have just considered yes/no problems
- E.g., "Does problem X possess 'a solution'?"

In Practice:

- We want to *obtain* a solution! And maybe even the best!
- For example, a satisfying assignment or the size of the smallest node cover.

Observation:

- If we can solve the optimization problem, we can solve the yes/no problem.

Example 11.4.1

- Yes/No problem: Does G have a node cover of size $\leq k$?
- Optimisation problem: What is the size of the smallest node cover for G?

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.
- Since this comparison can be done in **P**, we also solved our decision problem in **P**.

Completeness for Optimisation Problems

Optimisation Problems

Cannot be in **NP**, as they are not yes/no problems

Theorem 11.4.2

If $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of a problem in polytime, if the decision (yes/no) version is* **NP***-complete.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.
- Since this comparison can be done in **P**, we also solved our decision problem in **P**.
- This is a contradiction to $\mathbf{P} \neq \mathbf{NP}$, so the optimization problem is not in **P**.

□