Recap: How do we prove that a language is undecidable?

EECS 376 Discussion 8

Sec 27: Th 5:30-6:30 DOW 1017

IA: Eric Khiu

Slide deck available at course drive/Discussion/Slides/Eric Khiu

Announcement

- Combined Discussion Sections (17 and 27)
  - March 14: Eric K will cover Discussion 8 at **DOW 1017**
  - March 21: Chaitanya will cover Discussion 9 at **EECS 1200**
- Bryan will cover my OH on Tue March 19 5:00-7:30 at MLB B116
- Midterm Evaluation:
  - Thanks for the feedback!
  - Want more practice problems
  - Short summary after each topic is helpful

Agenda

- More on undecidability
  - Recording here
- The class P
- The class NP
- P vs NP
- SAT and Cook Levin Theorem (if time)

Recap: Decidable Language

- A language L is decidable iff there exists a TM that
  - accepts all $x \in L$
  - rejects all $x \notin L$
  - halts on all inputs
- One way to prove that a language is decidable is by describing a TM that decides it (i.e., a decider)
- **Note 1:** You are allowed to hardcode constant values in your machine (seen in HW6)
- **Note 2:** If some deciders are known to exists, you can use them in your machine (e.g., D_S and D_T from last week for $S \setminus T$)

Recap: Turing Reductions

- Suppose we want to show that $A \leq_T B$
- **Step 1: Identify the inputs of D_A and D_B**
  - Is the input a number? A string? Multiple strings? A machine?
- **Step 2: Draft Desired Behavior of D_A**
  - Choose between "return same" and "return opposite"
  - Return same: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \in B \Leftrightarrow D_B(x')$ accepts $\Leftrightarrow D_A(x)$ accepts
  - Return opposite: $x \in A \Leftrightarrow \cdots \Leftrightarrow x' \notin B \Leftrightarrow D_B(x')$ rejects $\Leftrightarrow D_A(x)$ accepts
- **Step 3: Generate input(s) for D_B**
  - Return same: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \in B$ and $x \notin A \Rightarrow x' \notin B$?
  - Return opposite: How to generate x', possibly using x, such that $x \in A \Rightarrow x' \notin B$ and $x \notin A \Rightarrow x' \in B$?

Note: Here we condense the two cases using iff

Recap: Undecidability Proof

- Know: L_{BARBER} (or any undecidable language) is undecidable
- Task: Prove L is undecidable
- We use proof by contradiction:
  - Suppose for contradiction that L is decidable
  - Show that $L_{BARBER} \leq_T L$
  - By this theorem: Suppose $A \leq_T B$. If B is decidable, then A is decidable.
  - Since we have $L_{BARBER} \leq_T L$ and L is decidable by assumption
  - Therefore, L_{BARBER} is now decidable. Contradiction.

More Turing Reductions

Course Notes

More Turing Reductions

- Another thing you are allowed to do is to create a machine (without running it) and pass it to the backbox decider
- For example, I want to use the black box decider D_E that decides
$$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

```
D_A = "On input (⟨M⟩,x):
    Construct M' as follows: TODO
    Run D_E on ⟨M'⟩ and return same"
```

- **Discuss:** Why do I have to create M'? Why can't I just use M?
  - The input of D_E must be a machine
  - We don't know $L(M)$, so we can't tell if D_E accepts/ rejects M

More Turing Reductions

- For example, I want to use the black box decider D_E that decides

  $$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$

  to build a decider for L_{ACC}. I can do the following:

```
DA = "On input (⟨M⟩,x):
    Construct M' as follows: TODO
    Run DE on ⟨M'⟩ and return same"
```

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ … $\Rightarrow D_E(\langle M' \rangle)$ accepts $\Rightarrow D_A(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ … $\Rightarrow D_E(\langle M' \rangle)$ rejects $\Rightarrow D_A(\langle M \rangle, x)$ rejects

More Turing Reductions

- For example, I want to use the black box decider D_E that decides
$$L_E = \{\langle M \rangle : L(M) = \emptyset\}$$
to build a decider for L_{ACC}. I can do the following:

```
D_A = "On input (⟨M⟩,x):
    Construct M' as follows: TODO
    Run D_E on ⟨M'⟩ and return same"
```

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ … $\Rightarrow L(M') = \emptyset \Rightarrow D_E(\langle M' \rangle)$ accepts $\Rightarrow D_A(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ … $\Rightarrow L(M') \neq \emptyset \Rightarrow D_E(\langle M' \rangle)$ rejects $\Rightarrow D_A(\langle M \rangle, x)$ rejects

Brainstorm Time

- Correctness proof draft
  - $(\langle M \rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow$ ... $\Rightarrow L(M') = \emptyset \Rightarrow D_A(\langle M' \rangle)$ accepts
  - $(\langle M \rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow$ ... $\Rightarrow L(M') \neq \emptyset \Rightarrow D_A(\langle M' \rangle)$ rejects
- **Brainstorm 1:** How to make $L(M') = \emptyset$ happen?
  - Just make M' rejects all inputs!
    - `M’= “On input w: Reject”`

Trigger this if M accepts x

- **Brainstorm 2:** How to make $L(M') \neq \emptyset$ happen?
  - Just make M' accepts *some* inputs, say “duck”
    - `M’= “On input w: If w = ‘duck’ then Accept Else Reject”`
  - Even easier: make M' accepts all inputs, i.e., $L(M') = \Sigma^*$
    - `M’= “On input w: Accept”`

Trigger this if M does not accepts x

Putting Everything together…

- D_A = "On input (⟨M⟩,x):

  Construct M' as follows:

  ```
  M' = "On input w:
      Run M on x
      If M accepts x then Reject
      Else Accept"
  ```

  Run D_E on ⟨M'⟩ and return same"

- Correctness Proof:
  - $(\langle M\rangle, x) \in L_{ACC} \Rightarrow M$ accepts $x \Rightarrow M'$ rejects all inputs $\Rightarrow L(M') = \emptyset \Rightarrow D_E(\langle M'\rangle)$ accepts $\Rightarrow D_A(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \notin L_{ACC} \Rightarrow M$ does not accept $x \Rightarrow M'$ accepts all inputs $\Rightarrow L(M') \neq \emptyset \Rightarrow D_E(\langle M'\rangle)$ rejects $\Rightarrow D_A(\langle M\rangle, x)$ rejects

Unit 3: Complexity

Course Notes

Complexity: Introduction

- Last unit: What *can* and *can't* a computer solve
- This unit: What can and can't a computer solve ***efficiently***
- **Complexity Class:** $DTIME(t(n))$ is the *class* of all languages decidable by a TM with time complexity $O(t(n))$

The Class P

[Course Notes]

The Class P

- **Definition:** P is the class of languages decidable by a polynomial-time Turing machine, where

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

- Given the significant difference between polynomial and exponential time, we informally consider P to be the class of efficiently decidable languages

Discuss: Given the definition of class P, what do we need to show to prove that a language is in class P?

  - Give the algorithm of the efficient decider (TM)
  - Correctness proof (from last unit)
  - Runtime analysis (NEW!)

Example: Spanning Trees

- A spanning tree of a graph $G = (V, E)$ is a subset of edges where every vertex is included in the subgraph

- Consider the following language:

$$L_{SPAN-k}\{\langle G, k\rangle : G \text{ has a spanning tree of weight less than } k\}$$

- We can decide this language efficiently:

```
D = “On input (⟨G,k⟩):
      MST ← KRUSKAL(G)
      If WEIGHT(MST) < k then accept; else reject”
```

Example: Spanning Trees

$L_{SPAN-k}\{\langle G,k\rangle : G \text{ has a spanning tree of weight less than } k\}$

```
D = "On input (⟨G,k⟩):
    MST ← KRUSKAL(G) // O(|E|log|E|)
    If WEIGHT(MST) < k then accept; else reject"
```

- **Correctness analysis:**
  - $\langle G,k\rangle \in L_{SPAN-k} \Rightarrow \exists$ spanning tree T of G s.t. Weight$(T) < k$

    Def of MST

    $\Rightarrow$ Weight$(MST) \leq$ Weight $(T) < k \Rightarrow D(\langle G,k\rangle)$ accepts
  - $\langle G,k\rangle \notin L_{SPAN-k} \Rightarrow$ Weight$(T) \geq k$ for all spanning trees T of $G \Rightarrow$ Weight $(MST) \geq k$ since MST is a spanning tree $\Rightarrow D(\langle G,k\rangle)$ rejects
- **Runtime analysis:**
  - Kruskal Algorithm runs in $O(|E|\log|E|)$: efficient

Example: GCD

- Prove that the language $GCD = \{(x, y, g): \gcd(x, y) = g\}$ is in class P
- Consider the following decider:

D = "On input (x, y, g):

  $G \leftarrow 0$

  If $x < g$ **or** $y < g$ **or** $g < 0$ **then** reject

  for $d = 1, \ldots, \min(x, y)$ **do**

    if d divides both x and y **then** $G \leftarrow d$

  If $G = g$ **then** accept; **else** reject"

- Correctness analysis:
  - $(x, y, g) \in GCD \Leftrightarrow g = \gcd(x, y) \Leftrightarrow g$ is the last d getting assigned to $G \Leftrightarrow G = g \Leftrightarrow D(x, y, g)$ accepts
- Runtime analysis:
  - $O(\min(x, y))$: ~~efficient~~

Discuss: What is wrong with this proof?

Exercise: $L_{<376376}$

- Show that the following language is in P

  $$L_{<376376} = \{(\langle M \rangle, x) : M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

- Let D be the efficient decider for $L_{<376376}$

- **Correctness Proof Draft**
  - $(\langle M \rangle, x) \in L_{<376376} \Rightarrow$ $\boxed{}$ $\Rightarrow D(\langle M \rangle, x)$ accepts
  - $(\langle M \rangle, x) \notin L_{<376376} \Rightarrow$ $\boxed{}$ $\Rightarrow D(\langle M \rangle, x)$ rejects

- **Exercise:** Design the decider to fill in the two blanks above + runtime analysis
- **Hint:** Consider $|x|^k$ for small x and k for intuition

Exercise: $L_{<376376}$

$$L_{<376376} = \{(\langle M\rangle, x): M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

- Consider the following decider:

```
D = "On input (⟨M⟩, x):
    Counter ← 1
    While M has not halt on x do
        If counter ≥ |x|^376376 then reject
        Execute the (Counter)th step of M on x
        Counter ← Counter + 1
    accept"
```

- Or alternatively,

```
D = "On input (⟨M⟩, x):
    Simulate execution of M(x) for up to |x|^376376 − 1 steps
    If M halts during this execution then accept; else reject"
```

Exercise: $L_{<376376}$

$$L_{<376376} = \{(\langle M\rangle, x) : M \text{ is a TM that halts on } x \text{ in less than } |x|^{376376} \text{ steps}\}$$

```
D = "On input (⟨M⟩,x):
      Simulate execution of M(x) for up to |x|^376376 − 1 steps
      If M halts during this execution then accept; else reject"
```

- **Correctness Analysis**
  - $(\langle M\rangle, x) \in L_{<376376} \Rightarrow M$ will halt on x within $|x|^{376376} - 1$ steps $\Rightarrow D(\langle M\rangle, x)$ accepts
  - $(\langle M\rangle, x) \notin L_{<376376} \Rightarrow M$ won't halt on x within $|x|^{376376} - 1$ steps $\Rightarrow D(\langle M\rangle, x)$ rejects
- **Runtime analysis**
  - Running $|x|^{376376} - 1$ steps of a program is polynomial w.r.t. $|x|$, the only other work done is returning a bool, so this decider is efficient

TL; DPA

- P is the class of languages decidable by a polynomial-time Turing machine
- To prove that a language is in class P
  - Give the algorithm of the efficient decider
  - Correctness analysis
  - Runtime analysis (new)

Take Home Exercise 1

- Let $L_1, L_2 \in P$. Determine whether the following statements are always/ sometimes/ never true (This can be helpful to put on your cheatsheet!)
  - $\overline{L_1} \in P$
  - $L_1 \cap L_2 \in P$
  - $L_1 \cup L_2 \in P$
  - $L_1 \setminus L_2 \in P$

The Class NP

Course Notes

Starter: Verifiers and Certificates

- Consider the following language

$$PAIRSUM = \{(S, k) : \exists a, b \in S \text{ s.t. } a + b = k\}$$

- Determine if the followings are in $PAIRSUM$
  - $S_1 = \{2,4,6\},\ k_1 = 6$
  - $S_2 = \{2,4,6\},\ k_2 = 7$

Share with us: What was your thought process?

Verifiers and Certificates

- A certificate c is an additional information used to check whether an input x is in a language L
- A verifier V for a language L is a TM that takes in (x, c) that satisfies:
  - $\forall x \in L$, there exists some certificate c such that $V(x, c)$ accepts
  - $\forall x \notin L$, there is no c such that $V(x, c)$ accepts
- For $PAIRSUM = \{(S, k): \exists a, b \in S \text{ s.t. } a + b = k\}$, $x = (S, k)$, we can have $c = (a, b)$, and we can build a verifier as follows:

V = "On input $(x = (S, k), c = (a, b))$:

  if $a \notin S$ or $b \notin S$ **then** reject

  if $a + b = k$ **then** accept; else **reject**"

Note: You format how you want the certificate to look like!

Formatting Certificate

- Be creative! There is no *one correct way* to format the certificate. For example, we could have

V = "On input $(x = (S, k), c = (e^a, e^b))$:

    if $\ln e^a \notin S$ or $\ln e^b \notin S$ **then** reject

    if $e^a \cdot e^b = e^k$ **then** accept; **else** reject"

- You can even take in a certificate and ignore it!

V = "On input $(x = (S, k), c = \text{"}duck\text{"})$:

    for pairs (a, b): $a, b \in S$ **do**

        if $a + b = k$ **then** accept

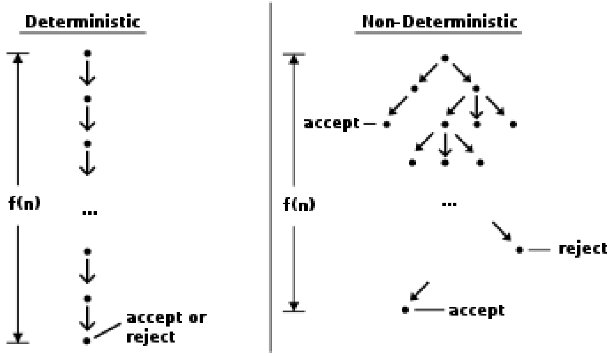
    reject"

The Class NP

- A language is *efficiently verifiable* if there exists an efficient verifier that verifies it
  - **Note 1:** The certificate must be polynomial in $|x|$
  - **Note 2:** Since the runtime of V is polynomial in $|x|$, the machine halts on any input
- **Definition:** NP is the set of efficiently verifiable languages
- To prove that a language is in class NP
  - Give the algorithm of the efficient verifier
  - Correctness analysis
  - Runtime analysis

Why is it called NP? (Out of scope)

- There is an alternative but equivalent definition often used that says the class NP is the set of languages that have a Non-deterministic Polynomial-time (efficient) decider



- Don't worry about this definition in this class, we will only use the definition on the previous slide
- Main takeaway: All NP languages are decidable (Will prove $NP \subseteq EXP$ in HW7 EC)

Example: PairSum

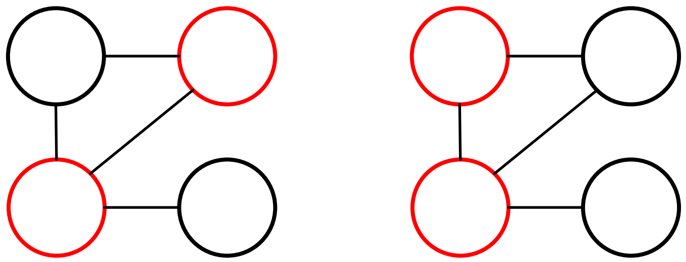
- Prove that $PAIRSUM = \{(S, k) : \exists a, b \in S \text{ s.t. } a + b = k\}$ is in class NP.
- **Efficient Verifier:**

```
V = "On input (x = (S,k), c = (a,b)):
        if a ∉ S or b ∉ S then reject
        if a + b ≠ k then reject
        accept"
```

- **Runtime analysis:**
  - It takes $O(|S|)$ to check if a and b are in S, and $O(1)$ to check if $a + b = k$. Therefore, verifier is efficient
- **Correctness analysis:**
  - $(S, k) \in PAIRSUM \Rightarrow \exists a, b \in S: a + b = k \Rightarrow V$ will accept when given this (a, b) pair as certificate
  - $(S, K) \notin PAIRSUM \Rightarrow \nexists a, b \in S: a + b = k \Rightarrow V$ will never accept any (a, b) certificate

Exercise: k-CLIQUE

- A clique of an undirected graph $G = (V, E)$ is a subset of vertices where every pair of vertices have an edge in E
- For example:



- A k-clique is a clique with k vertices
- Prove that the following language is in class NP

$$k\text{-}CLIQUE = \{\langle G = (V, E), k\rangle : G \text{ is an undirected graph that has a } k\text{-clique}\}$$

Exercise: k-CLIQUE

$$k\text{-}CLIQUE = \{\langle G = (V, E), k\rangle : G \text{ is an undirected graph that has a } k\text{-clique}\}$$

- Let F be an efficient verifier for $k-CLIQUE$
- **Step 1: Format of the certificate**
  - Let $V' \subseteq V$ such that $|V'| = k$
- **Step 2: Correctness proof draft**
  - $(G, k) \in k-CLIQUE \Rightarrow \exists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow$
    $\Rightarrow F$ will accept when given this V' subset as certificate
  - $(G, k) \in k-CLIQUE \Rightarrow \nexists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow$
    $\Rightarrow F$ will never accept any V' certificate
- **Step 3: Construct efficient verifier**
  - Exercise: Give an algorithm of the efficient verifier
  - Hint: What does "every pair of vertices in V' have an edge in E" mean?

Exercise: k-CLIQUE

$k\text{-}CLIQUE = \{\langle G = (V, E), k\rangle : G \text{ is an undirected graph that has a } k\text{-clique}\}$

- **Efficient Verifier**

  F = "On input $(G = (V, E), V')$:

  If $|V'| \neq k$ **or** $\exists v \in V' : v \notin V$ **then** reject

  $E' \leftarrow \bigcup_{u,v \in V'} (u, v)$

  If $E' = E$ **then** accept; **else** reject"

- **Runtime analysis:**
  - It takes $O(|V|)$ to check if every vertex in V' is in V
  - It takes $O(|E|)$ to check if there is an edges between every pair of vertices in V'

- **Correctness Analysis**
  - $(G, k) \in k{-}CLIQUE \Rightarrow \exists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow \bigcup_{u,v \in V'} (u, v) = E \Rightarrow$ F will accept when given this V' subset as certificate
  - $(G, k) \in k{-}CLIQUE \Rightarrow \nexists V' \subseteq V$ s.t. V' is a clique of size $k \Rightarrow \forall V', |V'| \neq k$ or $V' \nsubseteq V$ $\bigcup_{u,v \in V'} (u, v) \neq E \;\Rightarrow F$ will never accept any V' certificate

TL; DPA

- NP is the set of efficiently verifiable languages
- To prove that a language is in class NP
  - Give the algorithm of the efficient verifier
  - Correctness analysis
  - Runtime analysis (both certificate's size and verifier's runtime must be polynomial)
- Useful phrases for correctness analysis:
  - Verifier V will accept when given this c as certificate
  - Verifier V will never accept any certificate c

Take Home Exercise 2

- Let $L_1, L_2 \in NP$. Determine whether the following statements are always/ sometimes/ never true/ unknown.
  - $\overline{L_1} \in NP$
  - $L_1 \cap L_2 \in NP$
  - $L_1 \cup L_2 \in NP$
  - $L_1 \setminus L_2 \in NP$

P vs NP

[Course Notes]

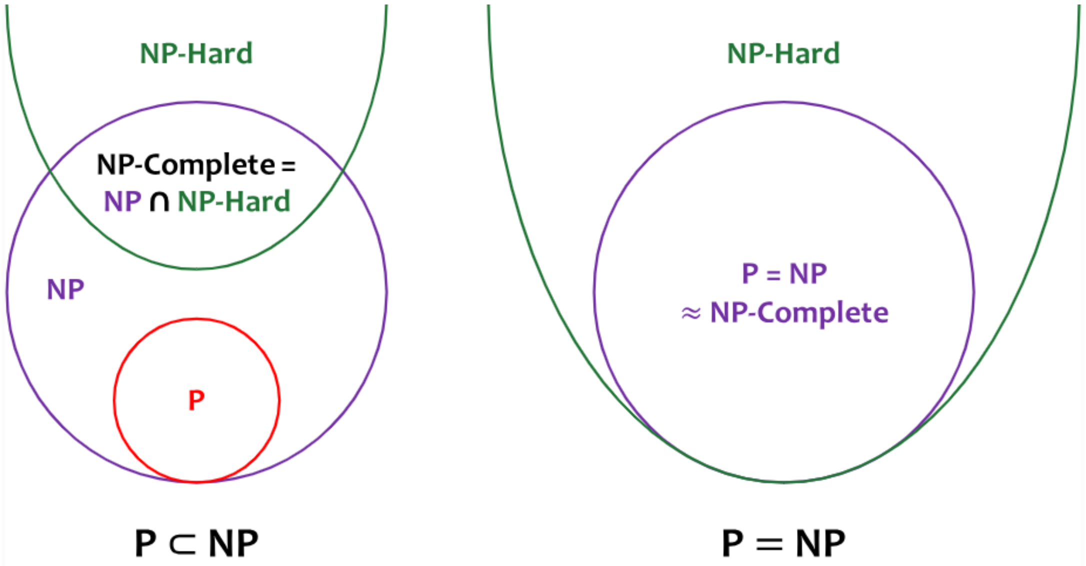
P vs NP Starter

Discussion: Which of the following statements is/are true?
A. $\forall L \in P,\ L \in NP$
B. $\forall L \in NP, L \in P$
C. $\exists L \in P\colon L \notin NP$
D. $\exists L \in NP\colon L \notin P$

- A is true, you will prove this in HW 7 $(P \subseteq NP)$
- C is false since A is true
- B and D are unknown!
  - If B is true, then all languages in NP are in P, so $P = NP$
  - If D is true, then there exists a language in NP that is not in P, so $P \neq NP$

Relationship Between P and NP

- There is no proof as to whether $P = NP$



- If $P = NP$, then all languages in NP are NP-complete, except the two trivial languages Σ^* and $\emptyset$.

Our First NP-Complete Language

- We don’t know if P = NP, but we’ve been able to establish relationships between the problems within NP
- Imagine some “hard” language in NP such that if we could decide this language efficiently, then we’d be able to decide **all** languages in NP efficiently
  - Knowledge about only this one language would give us a result for the entire class NP
  - This type of language is known as **NP-Complete**
- **The language SAT is NP-Complete by the Cook Levin Theorem**
  - Now if we could decide SAT efficiently, we could decide any language in NP efficiently, which would mean that NP $\subseteq$ P and thus P=NP!

The Language SAT

- *Literal*: Boolean variable with a value of TRUE or FALSE (such as x or $\neg x$)
- *Boolean Formula* ϕ: formula made up of literals and logical operators (such as $x \land \neg y$ or $x \lor y \lor z$)
- *Satisfying assignment:* a set of truth values assigned to each literal in a boolean formula ϕ such that ϕ evaluates to true

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

- Example: $x \lor \neg y \in SAT$; $x \land \neg x \in SAT$
- The first step in the Cook Levin proof is to show that SAT is in NP

High Level Cook Levin Theorem Proof

- The meat of the Cook Levin theorem is a reduction from an arbitrary language L in NP to SAT
- This reduction takes the form of a function f that meets the following requirements:
  - $x \in L$ implies that $f(x) \in \mathrm{SAT}$
  - $x \notin L$ implies that $f(x) \notin \mathrm{SAT}$
  - f is computable in polynomial time
- As L is in NP, we know there is an efficient verifier for L
  - f can inspect V's "source code" to build a corresponding (poly-sized) formula ϕ
  - ϕ is designed so that it is satisfied by some assignment of its variables *if and only if* $V(x, c)$ would accept for some c