BBM402-Lecture 12: NP-completeness

Lecturer: Lale Özkahya

Resources for the presentation:
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/Syllabus/
https://courses.engr.illinois.edu/cs498374/lectures.html

Independent Sets and Cliques

Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

Independent Sets and Cliques

Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

1. **independent set**: no two vertices of $\mathbf{V'}$ connected by an edge.

Independent Sets and Cliques

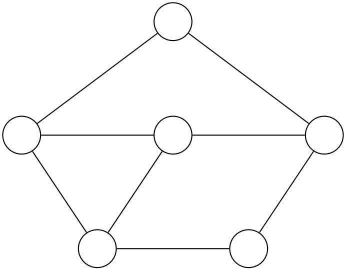
Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

1. **independent set**: no two vertices of $\mathbf{V'}$ connected by an edge.
2. **clique**: *every* pair of vertices in $\mathbf{V'}$ is connected by an edge of $\mathbf{G}$.

Independent Sets and Cliques

Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

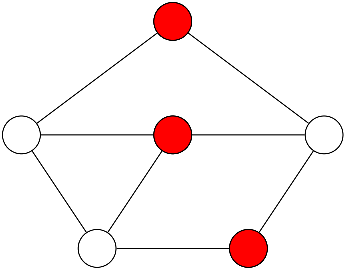
1. **independent set**: no two vertices of $\mathbf{V'}$ connected by an edge.
2. **clique**: *every* pair of vertices in $\mathbf{V'}$ is connected by an edge of $\mathbf{G}$.



Independent Sets and Cliques

Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

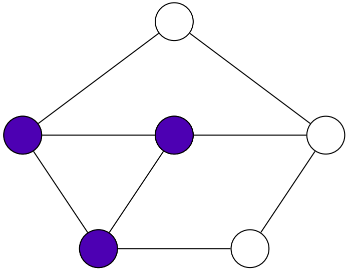
1. **independent set**: no two vertices of $\mathbf{V'}$ connected by an edge.
2. **clique**: *every* pair of vertices in $\mathbf{V'}$ is connected by an edge of $\mathbf{G}$.



Independent Sets and Cliques

Given a graph $\mathbf{G}$, a set of vertices $\mathbf{V'}$ is:

1. **independent set**: no two vertices of $\mathbf{V'}$ connected by an edge.
2. **clique**: *every* pair of vertices in $\mathbf{V'}$ is connected by an edge of $\mathbf{G}$.



The **Independent Set** and **Clique** Problems

Problem: Independent Set

Instance: A graph G and an integer $\mathbf{k}$.
Question: Does G has an independent set of size $\geq \mathbf{k}$?

The Independent Set and Clique Problems

Problem: **Independent Set**

> **Instance:** A graph G and an integer $\mathbf{k}$.
> **Question:** Does G has an independent set of size $\geq \mathbf{k}$?

Problem: **Clique**

> **Instance:** A graph G and an integer $\mathbf{k}$.
> **Question:** Does G has a clique of size $\geq \mathbf{k}$?

Recall

For decision problems $\mathbf{X}, \mathbf{Y}$, a reduction from $\mathbf{X}$ to $\mathbf{Y}$ is:

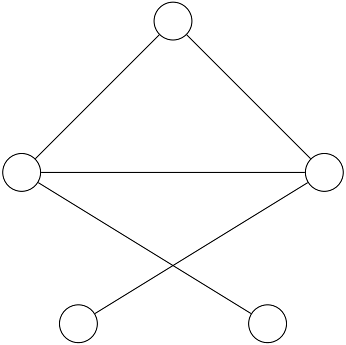
1. An algorithm ...
2. that takes $\mathbf{I_X}$, an instance of $\mathbf{X}$ as input ...
3. and returns $\mathbf{I_Y}$, an instance of $\mathbf{Y}$ as output ...
4. such that the solution (YES/NO) to $\mathbf{I_Y}$ is the same as the solution to $\mathbf{I_X}$.

Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $\mathbf{G}$ and an integer $\mathbf{k}$.

Reducing **Independent Set** to **Clique**

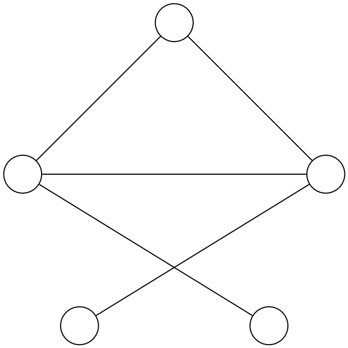
An instance of **Independent Set** is a graph **G** and an integer **k**.



Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $\mathbf{G}$ and an integer $\mathbf{k}$.

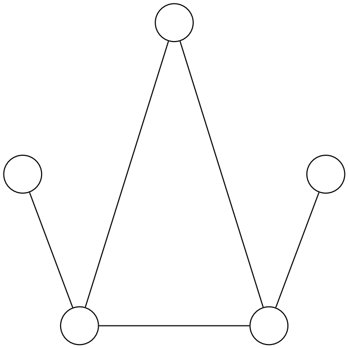
Reduction given $< \mathbf{G}, \mathbf{k} >$ outputs $< \overline{\mathbf{G}}, \mathbf{k} >$ where $\overline{\mathbf{G}}$ is the *complement* of $\mathbf{G}$. $\overline{\mathbf{G}}$ has an edge $(\mathbf{u}, \mathbf{v})$ if and only if $(\mathbf{u}, \mathbf{v})$ is not an edge of $\mathbf{G}$.



Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $\mathbf{G}$ and an integer $\mathbf{k}$.

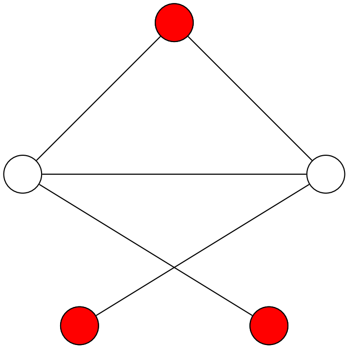
Reduction given $< \mathbf{G}, \mathbf{k} >$ outputs $< \overline{\mathbf{G}}, \mathbf{k} >$ where $\overline{\mathbf{G}}$ is the *complement* of $\mathbf{G}$. $\overline{\mathbf{G}}$ has an edge $(\mathbf{u}, \mathbf{v})$ if and only if $(\mathbf{u}, \mathbf{v})$ is not an edge of $\mathbf{G}$.



Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $\mathbf{G}$ and an integer $\mathbf{k}$.

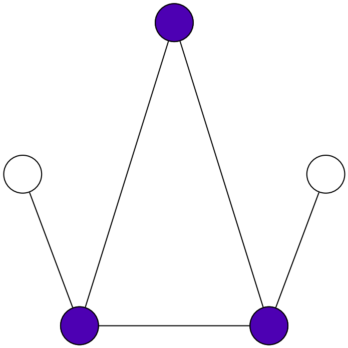
Reduction given $< \mathbf{G}, \mathbf{k} >$ outputs $< \overline{\mathbf{G}}, \mathbf{k} >$ where $\overline{\mathbf{G}}$ is the *complement* of $\mathbf{G}$. $\overline{\mathbf{G}}$ has an edge $\mathbf{(u, v)}$ if and only if $\mathbf{(u, v)}$ is not an edge of $\mathbf{G}$.



Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $\mathbf{G}$ and an integer $\mathbf{k}$.

Reduction given $< \mathbf{G}, \mathbf{k} >$ outputs $< \overline{\mathbf{G}}, \mathbf{k} >$ where $\overline{\mathbf{G}}$ is the *complement* of $\mathbf{G}$. $\overline{\mathbf{G}}$ has an edge $\mathbf{(u, v)}$ if and only if $\mathbf{(u, v)}$ is not an edge of $\mathbf{G}$.



Correctness of reduction

Lemma

$\mathbf{G}$ has an independent set of size $\mathbf{k}$ if and only if $\overline{\mathbf{G}}$ has a clique of size $\mathbf{k}$.

Proof.

Need to prove two facts:

$\mathbf{G}$ has independent set of size at least $\mathbf{k}$ implies that $\overline{\mathbf{G}}$ has a clique of size at least $\mathbf{k}$.

$\overline{\mathbf{G}}$ has a clique of size at least $\mathbf{k}$ implies that $\mathbf{G}$ has an independent set of size at least $\mathbf{k}$.

Easy to see both from the fact that $\mathbf{S} \subseteq \mathbf{V}$ is an independent set in $\mathbf{G}$ if and only if $\mathbf{S}$ is a clique in $\overline{\mathbf{G}}$. $\square$

Independent Set and Clique

1. **Independent Set** $\leq$ **Clique**.

Independent Set and Clique

1. **Independent Set** $\leq$ **Clique**.
   What does this mean?
2. If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.

Independent Set and **Clique**

1. **Independent Set** $\leq$ **Clique**.
   What does this mean?
2. If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.
3. **Clique** is *at least as hard as* **Independent Set**.

Independent Set and **Clique**

1. **Independent Set** $\leq$ **Clique**.
   What does this mean?
2. If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.
3. **Clique** is *at least as hard as* **Independent Set**.
4. Also... **Independent Set** is *at least as hard as* **Clique**.

Independent Set and Clique

Assume you can solve the **Clique** problem in $\mathbf{T(n)}$ time. Then you can solve the **Independent Set** problem in

- **(A)** $\mathbf{O(T(n))}$ time.
- **(B)** $\mathbf{O(n \log n + T(n))}$ time.
- **(C)** $\mathbf{O(n^2 T(n^2))}$ time.
- **(D)** $\mathbf{O(n^4 T(n^4))}$ time.
- **(E)** $\mathbf{O(n^2 + T(n^2))}$ time.
- **(F)** Does not matter - all these are polynomial if $\mathbf{T(n)}$ is polynomial, which is good enough for our purposes.

Vertex Cover

Given a graph $\mathbf{G = (V, E)}$, a set of vertices $\mathbf{S}$ is:

Vertex Cover

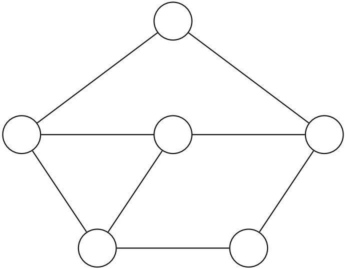
Given a graph $\mathbf{G = (V, E)}$, a set of vertices $\mathbf{S}$ is:

1. A **vertex cover** if every $\mathbf{e} \in \mathbf{E}$ has at least one endpoint in $\mathbf{S}$.

Vertex Cover

Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, a set of vertices $\mathbf{S}$ is:

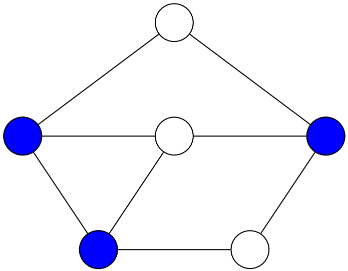
1. A **vertex cover** if every $\mathbf{e} \in \mathbf{E}$ has at least one endpoint in $\mathbf{S}$.



Vertex Cover

Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, a set of vertices $\mathbf{S}$ is:

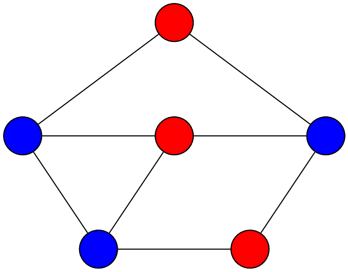
1. A **vertex cover** if every $\mathbf{e} \in \mathbf{E}$ has at least one endpoint in $\mathbf{S}$.



Vertex Cover

Given a graph $\mathbf{G = (V, E)}$, a set of vertices $\mathbf{S}$ is:

1. A **vertex cover** if every $\mathbf{e} \in \mathbf{E}$ has at least one endpoint in $\mathbf{S}$.



The **Vertex Cover** Problem

Problem (**Vertex Cover**)

Input: *A graph G and integer $\mathbf{k}$.*
Goal: *Is there a vertex cover of size $\leq \mathbf{k}$ in G?*

The **Vertex Cover** Problem

Problem (**Vertex Cover**)

Input: *A graph G and integer $\mathbf{k}$.*
Goal: *Is there a vertex cover of size $\leq \mathbf{k}$ in G?*

Can we relate **Independent Set** and **Vertex Cover**?

Proposition

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *be a graph.* $\mathbf{S}$ *is an independent set if and only if* $\mathbf{V} \setminus \mathbf{S}$ *is a vertex cover.*

Proof.

$(\Rightarrow)$ Let $\mathbf{S}$ be an independent set

1. Consider any edge $\mathbf{uv} \in \mathbf{E}$.
2. Since $\mathbf{S}$ is an independent set, either $\mathbf{u} \notin \mathbf{S}$ or $\mathbf{v} \notin \mathbf{S}$.
3. Thus, either $\mathbf{u} \in \mathbf{V} \setminus \mathbf{S}$ or $\mathbf{v} \in \mathbf{V} \setminus \mathbf{S}$.
4. $\mathbf{V} \setminus \mathbf{S}$ is a vertex cover.

$(\Leftarrow)$ Let $\mathbf{V} \setminus \mathbf{S}$ be some vertex cover:

1. Consider $\mathbf{u}, \mathbf{v} \in \mathbf{S}$
2. $\mathbf{uv}$ is not an edge of G, as otherwise $\mathbf{V} \setminus \mathbf{S}$ does not cover $\mathbf{uv}$.
3. $\implies$ $\mathbf{S}$ is thus an independent set. $\square$

Independent Set $\leq_P$ Vertex Cover

1. **G**: graph with **n** vertices, and an integer **k** be an instance of the **Independent Set** problem.

Independent Set $\leq_P$ Vertex Cover

1. $\mathbf{G}$: graph with $\mathbf{n}$ vertices, and an integer $\mathbf{k}$ be an instance of the **Independent Set** problem.
2. $\mathbf{G}$ has an independent set of size $\geq \mathbf{k}$ iff $\mathbf{G}$ has a vertex cover of size $\leq \mathbf{n} - \mathbf{k}$

Independent Set $\leq_P$ Vertex Cover

1. $\mathbf{G}$: graph with $\mathbf{n}$ vertices, and an integer $\mathbf{k}$ be an instance of the **Independent Set** problem.
2. $\mathbf{G}$ has an independent set of size $\geq \mathbf{k}$ iff $\mathbf{G}$ has a vertex cover of size $\leq \mathbf{n} - \mathbf{k}$
3. $(\mathbf{G}, \mathbf{k})$ is an instance of **Independent Set** , and $(\mathbf{G}, \mathbf{n} - \mathbf{k})$ is an instance of **Vertex Cover** with the same answer.

Independent Set $\leq_P$ Vertex Cover

1. $\mathbf{G}$: graph with $\mathbf{n}$ vertices, and an integer $\mathbf{k}$ be an instance of the **Independent Set** problem.
2. $\mathbf{G}$ has an independent set of size $\geq \mathbf{k}$ iff $\mathbf{G}$ has a vertex cover of size $\leq \mathbf{n} - \mathbf{k}$
3. $(\mathbf{G}, \mathbf{k})$ is an instance of **Independent Set** , and $(\mathbf{G}, \mathbf{n} - \mathbf{k})$ is an instance of **Vertex Cover** with the same answer.
4. Therefore, **Independent Set** $\leq_P$ **Vertex Cover**. Also **Vertex Cover** $\leq_P$ **Independent Set**.

Proving Correctness of Reductions

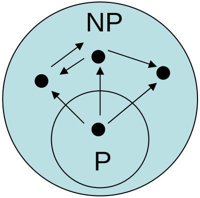
To prove that $\mathbf{X} \leq_P \mathbf{Y}$ you need to give an algorithm $\mathcal{A}$ that:

1. Transforms an instance $\mathbf{I_X}$ of $\mathbf{X}$ into an instance $\mathbf{I_Y}$ of $\mathbf{Y}$.
2. Satisfies the property that answer to $\mathbf{I_X}$ is YES iff $\mathbf{I_Y}$ is YES.
    1. typical easy direction to prove: answer to $\mathbf{I_Y}$ is YES if answer to $\mathbf{I_X}$ is YES
    2. typical difficult direction to prove: answer to $\mathbf{I_X}$ is YES if answer to $\mathbf{I_Y}$ is YES (equivalently answer to $\mathbf{I_X}$ is NO if answer to $\mathbf{I_Y}$ is NO).
3. Runs in **polynomial** time.

NP-Completeness

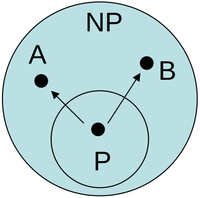
NP-Completeness

- $\leq_p$ allows us to relate problems in NP, saying which allow us to solve which others efficiently.
- Even though we don't know whether all of these problems are in P, we can use $\leq_p$ to impose some structure on the class NP:



- $A \rightarrow B$ here means $A \leq_p B$.

- Sets in NP – P might not be totally ordered by $\leq_p$: we might have A, B with neither $A \leq_p B$ nor $B \leq_p A$:

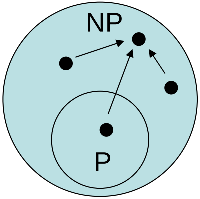


NP-Completeness

- Some languages in NP are hardest, in the sense that every language in NP is $\leq_p$-reducible to them.
- Call these NP-complete.
- Definition: Language B is NP-complete if both of the following hold:

  (a) $B \in NP$, and

  (b) For any language $A \in NP$, $A \leq_p B$.



- Sometimes, we consider languages that aren't, or might not be, in NP, but to which all NP languages are reducible.
- Call these NP-hard.
- Definition: Language B is NP-hard if, for any language $A \in NP$, $A \leq_p B$.

NP-Completeness

- Today, and next time, we'll:
  - Give examples of interesting problems that are NP-complete, and
  - Develop methods for showing NP-completeness.

- Theorem: $\exists$B, B is NP-complete.
  - There is at least one NP-complete problem.
  - We'll show this later.
- Theorem: If A, B, are NP-complete, then $A \leq_p B$.
  - Two NP-complete problems are essentially equivalent (up to $\leq_p$).
- Proof: $A \in NP$, B is NP-hard, so $A \leq_p B$ by definition.

NP-Completeness

- Theorem: If some NP-complete language is in P, then P = NP.
  - That is, if a polynomial-time algorithm exists for any NP-complete problem, then the entire class NP collapses into P.
  - Polynomial algorithms immediately arise for all problems in NP.
- Proof:
  - Suppose B is NP-complete and $B \in P$.
  - Let A be any language in NP; show $A \in P$.
  - We know $A \leq_p B$ since B is NP-complete.
  - Then $A \in P$, since $B \in P$ and “easiness propagates downward”.
  - Since every A in NP is also in P, $NP \subseteq P$.
  - Since $P \subseteq NP$, it follows that P = NP.

NP-Completeness

- Theorem: The following are equivalent.
  1. P = NP.
  2. Every NP-complete language is in P.
  3. Some NP-complete language is in P.
- Proof:

  1 $\Rightarrow$ 2:
  - Assume P = NP, and suppose that B is NP-complete.
  - Then B $\in$ NP, so B $\in$ P, as needed.

  2 $\Rightarrow$ 3:
  - Immediate because there is at least NP-complete language.

  3 $\Rightarrow$ 1:
  - By the previous theorem.

Beliefs about P vs. NP

- Most theoretical computer scientists believe P $\neq$ NP.
- Why?
- Many interesting NP-complete problems have been discovered over the years, and many smart people have tried to find fast algorithms; no one has succeeded.
- The problems have arisen in many different settings, including logic, graph theory, number theory, operations research, games and puzzles.
- Entire book devoted to them [Garey, Johnson].
- All these problems are essentially the same since all NP-complete problems are polynomial-reducible to each other.
- So essentially the same problem has been studied in many different contexts, by different groups of people, with different backgrounds, using different methods.

Beliefs about P vs. NP

- Most theoretical computer scientists believe $P \neq NP$.
- Because many smart people have tried to find fast algorithms and no one has succeeded.
- That doesn't mean $P \neq NP$; this is just some kind of empirical evidence.
- The essence of why NP-complete problems seem hard:
  - They have NP structure:

    $$x \in L \text{ iff } (\exists\ c, |c| \leq p(|x|))\ [\ V(\ x, c\ ) \text{ accepts }],$$

    where V is poly-time.
  - Guess and verify.
  - Seems to involve exploring a tree of possible choices, exponential blowup.
- However, no one has yet succeeded in proving that they actually are hard!
  - We don't have sharp enough methods.
  - So in the meantime, we just show problems are NP-complete.

Satisfiability is NP-Complete

Satisfiability is NP-Complete

- SAT = { < ϕ > | ϕ is a satisfiable Boolean formula }
- Definition: (Boolean formula):
  - Variables: $x, x_1, x_2, \ldots, y, \ldots, z, \ldots$
    - Can take on values 1 (true) or 0 (false).
  - Literal: A variable or its negated version: $x, \neg x, \neg x_1, \ldots$
  - Operations: $\wedge \;\; \vee \;\; \neg$
  - Boolean formula: Constructed from literals using operations, e.g.:

    $\phi = x \wedge ((y \wedge z) \vee (\neg y \wedge \neg z)) \wedge \neg(x \wedge z)$
- Definition: (Satisfiability):
  - A Boolean formula is satisfiable iff there is an assignment of 0s and 1s to the variables that makes the entire formula evaluate to 1 (true).

Satisfiability is NP-Complete

- SAT = { < ϕ > | ϕ is a satisfiable Boolean formula }
- Boolean formula: Constructed from literals using operations, e.g.:

  $$\phi = x \wedge ((y \wedge z) \vee (\neg y \wedge \neg z)) \wedge \neg(x \wedge z)$$

- A Boolean formula is satisfiable iff there is an assignment of 0s and 1s to the variables that makes the entire formula evaluate to 1 (true).
- Example: ϕ above
  - Satisfiable, using the assignment x = 1, y = 0, z = 0.
  - So $\phi \in$ SAT.
- Example: $x \wedge ((y \wedge z) \vee (\neg y \wedge z)) \wedge \neg(x \wedge z)$
  - Not in SAT.
  - x must be set to 1, so z must = 0.

Satisfiability is NP-Complete

- SAT = { < ϕ > | ϕ is a satisfiable Boolean formula }
- Theorem: SAT is NP-complete.
- Lemma 1: SAT $\in$ NP.
- Lemma 2: SAT is NP-hard.
- Proof of Lemma 1:
  - Recall: L $\in$ NP if and only if ($\exists$ V, poly-time verifier) ($\exists$ p, poly)

    x $\in$ L iff ($\exists$ c, $|c| \leq p(|x|)$) [V(x, c) accepts]
  - So, to show SAT $\in$ NP, it's enough to show ($\exists$ V) ($\exists$ p)

    ϕ $\in$ SAT iff ($\exists$ c, $|c| \leq p(|x|)$) [V(ϕ, c) accepts]
  - We know: $\phi \in$ SAT iff there is an assignment to the variables such that ϕ with this assignment evaluates to 1.
  - So, let certificate c be the assignment.
  - Let verifier V take a formula ϕ and an assignment c and accept exactly if ϕ with c evaluates to true.
  - Evaluate ϕ bottom-up, takes poly time.

Satisfiability is NP-Complete

- Lemma 2: SAT is NP-hard.
- Proof of Lemma 2:
  - Need to show that, for any $A \in NP$, $A \leq_p SAT$.
  - Fix $A \in NP$.
  - Construct a poly-time f such that

    $w \in A$ if and only if $f(w) \in SAT$.

    A formula, write it as ϕ_w.
  - By definition, since $A \in NP$, there is a nondeterministic TM M that decides A in polynomial time.
  - Fix polynomial p such that M on input w always halts, on all branches, in time $\leq p(|w|)$; assume $p(|w|) \geq |w|$.
  - $w \in A$ if and only if there is an accepting computation history (CH) of M on w.

Satisfiability is NP-Complete

- Lemma 2: SAT is NP-hard.
- Proof, cont'd:
  - Need $w \in A$ if and only if $f(w)$ ($= \phi_w$) $\in$ SAT.
  - $w \in A$ if and only if there is an accepting CH of M on w.
  - So we must construct formula ϕ_w to be satisfiable iff there is an accepting CH of M on w.
  - Recall definitions of computation history and accepting computation history from Post Correspondence Problem: $\#\ C_0\ \#\ C_1\ \#\ C_2 \ldots$
    - Configurations include tape contents, state, head position.
  - We construct ϕ_w to describe an accepting CH.
  - Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ as usual.
  - Instead of lining up configs in a row as before, arrange in $(p(|w|) + 1)$ row $\times$ $(p(|w|) + 3)$ column matrix:

Proof that SAT is NP-hard

- ϕ_w will be satisfiable iff there is an accepting CH of M on w.
- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$.
- Arrange configs in $(p(|w|) + 1) \times (p(|w|) + 3)$ matrix:

$$
\begin{array}{cccccccccccc}
\# & q_0 & w_1 & w_2 & w_3 & \ldots & w_n & \text{--} & \text{--} & \ldots & \text{--} & \# \\
\# & \ldots & & & & & & & & & & \# \\
\# & \ldots & & & & & & & & & & \# \\
\vdots & & & & & & & & & & & \vdots \\
\# & \ldots & & & & & & & & & & \#
\end{array}
$$

- Successive configs, ending with accepting config.
- Assume WLOG that each computation takes exactly p(|w|) steps, so we use p(|w|) + 1 rows.
- p(|w|) + 3 columns: p(|w|) for the interesting portion of the tape, one for head and state, two for endmarkers.

Proof that SAT is NP-hard

- ϕ_w is satisfiable iff there is an accepting CH of M on w.
- Entries in the matrix are represented by Boolean variables:
  - Define $C = Q \cup \Gamma \cup \{ \# \}$, alphabet of possible matrix entries.
  - Variable $x_{i,j,c}$ represents "the entry in position (i, j) is c".
- Define ϕ_w as a formula over these $x_{i,j,c}$ variables, satisfiable if and only if there is an accepting computation history for w (in matrix form).
- Moreover, an assignment of values to the $x_{i,j,c}$ variables that satisfies ϕ_w will correspond to an encoding of an accepting computation.
- Specifically, $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, where:
  - ϕ_{cell} : There is exactly one value in each matrix location.
  - ϕ_{start} : The first row represents the starting configuration.
  - ϕ_{accept} : The last row is an accepting configuration.
  - ϕ_{move} : Successive rows represent allowable moves of M.

ϕ_{cell}

- For each position (i,j), write the conjunction of two formulas:

  $\bigvee_{c \in C} x_{i,j,c}$: Some value appears in position (i,j).

  $\bigwedge_{c,\, d \in C,\, c \neq d} (\neg x_{i,j,c} \vee \neg x_{i,j,d})$: Position (i,j) doesn't contain two values.

- ϕ_{cell}: Conjoin formulas for all positions (i,j).

- Easy to construct the entire formula ϕ_{cell} given w input.
- Construct it in polynomial time.
- Sanity check: Length of formula is polynomial in |w|:
  - $O((p(|w|)^2)$ subformulas, one for each (i,j).
  - Length of each subformula depends on C, $O(|C|^2)$.

ϕ_{start}

- The right symbols appear in the first row:

$$\#\quad q_0\quad w_1\quad w_2\quad w_3\quad \ldots\quad w_n\quad \text{--}\quad \text{--}\quad \ldots\quad \text{--}\quad \#$$

$$\phi_{start}:\; x_{1,1,\#} \wedge x_{1,2,q0} \wedge x_{1,3,w1} \wedge x_{1,4,w2} \wedge \ldots$$
$$\wedge\; x_{1,n+2,wn} \wedge x_{1,n+3,\text{--}} \wedge \ldots$$
$$\wedge\; x_{1,p(n)+2,\text{--}} \wedge x_{1,p(n)+3,\#}$$

ϕ_{accept}

- For each j, $2 \leq j \leq p(|w|) + 2$, write the formula:

$$x_{p(|w|)+1,j,qacc}$$

- q_{acc} appears in position j of the last row.
- ϕ_{accept}: Take disjunction (or) of all formulas for all j.
- That is, q_{acc} appears in some position of the last row.

ϕ_{move}

- As for PCP, correct moves depend on correct changes to local portions of configurations.
- It's enough to consider 2×3 rectangles:
- If every 2×3 rectangle is "good", i.e., consistent with the transitions, then the entire matrix represents an accepting CH.
- For each position (i,j), $1 \leq i \leq p(|w|)$, $1 \leq j \leq p(|w|)+1$, write a formula saying that the rectangle with upper left at (i,j) is "good".
- Then conjoin all of these, $O(p(|w|)^2)$ clauses.
- Good tiles for (i,j), for a, b, c in Γ:

a	b	c
a	b	c

#	a	b
#	a	b

a	b	#
a	b	#

ϕ_{move}

- Other good tiles are defined in terms of the nondeterministic transition function δ.
- E.g., if $\delta(q_1, a)$ includes tuple (q_2, b, L), then the following are good:
  - Represents the move directly; for any c:
  - Head moves left out of the rectangle; for any c, d:
  - Head is just to the left of the rectangle; for any c, d:
  - Head at right; for any c, d, e:
  - And more, for #, etc.
- Analogously if $\delta(q_1, a)$ includes (q_2, b, R).
- Since M is nondeterministic, $\delta(q_1, a)$ may contain several moves, so include all the tiles.

c	q_1	a
q_2	c	b

q_1	a	c
d	b	c

a	c	d
b	c	d

d	c	q_1
d	q_2	c

e	d	c
e	d	q_2

ϕ_{move}

- The good tiles give partial constraints on the computation.
- When taken together, they give enough constraints so that only a correct CH can satisfy them all.
- The part (conjunct) of ϕ_{move} for (i,j) should say that the rectangle with upper left at (i,j) is good:
- It is simply the disjunction (or), over all allowable tiles, of the subformula:

a1	a2	a3
b1	b2	b3

$$x_{i,j,a1} \wedge x_{i,j+1,a2} \wedge x_{i,j+2,a3} \wedge x_{i+1,j,b1} \wedge x_{i+1,j+1,b2} \wedge x_{i+1,j+2,b3}$$

- Thus, ϕ_{move} is the conjunction over all (i,j), of the disjunction over all good tiles, of the formula just above.

ϕ_{move}

- ϕ_{move} is the conjunction over all (i,j), of the disjunction over all good tiles, of the given six-term conjunctive formula.
- Q: How big is the formula ϕ_{move}?
- $O(p(|w|)^2)$ clauses, one for each (i,j) pair.
- Each clause is only constant length, O(1).
  - Because machine M yields only a constant number of good tiles.
  - And there are only 6 terms for each tile.
- Thus, length of ϕ_{move} is polynomial in |w|.
- $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, length also poly in |w|.

ϕ_{move}

- $\phi_w = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$, length poly in |w|.
- More importantly, can produce ϕ_w from w in time that is polynomial in |w|.
- w $\in$ A if and only if M has an accepting CH for w if and only if ϕ_w is satisfiable.
- Thus, A $\leq_p$ SAT.
- Since A was any language in NP, this proves that SAT is NP-hard.
- Since SAT is in NP and is NP-hard, SAT is NP-complete.