NP-Completeness

Lecture 4

September 3, 2015

4.1: Quick & total recall

Recall...

NP Problems

Definition

Nondeterministic Polynomial Time (denoted by **NP**) is the class of all problems that have efficient certifiers (for YES instances).

Recall...

NP-Complete Problems

Definition

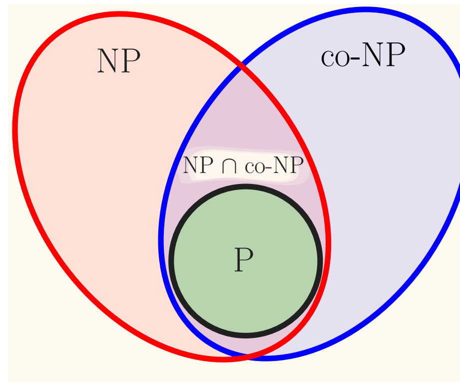
A problem X is said to be **NP-Complete** if

1. $X \in$ **NP**, and
2. (Hardness) For any $Y \in$ **NP**, **Y** $\leq_P$ **X**.

Recall...

NP

Decision problems with a polynomial certifier.
Examples: **SAT**, **Hamiltonian Cycle**, **3-Colorability**.



Definition

co-NP: class of all decision problems X s.t. $\overline{X} \in$ **NP**.
Examples: **UnSAT**, **No-Hamiltonian-Cycle**, **No-3-Colorable**.

Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language L is **NP-Complete** $\iff$
   - L is in **NP**
   - for every L' in **NP**, $L' \leq_P L$
3. L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
4. Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language L is **NP-Complete** $\iff$
   - L is in **NP**
   - for every L' in **NP**, $L' \leq_P L$
3. L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
4. Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language L is **NP-Complete** $\iff$
   - L is in **NP**
   - for every L' in **NP**, $L' \leq_P L$
3. L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
4. Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete**.

Recall...

1. **NP**: languages that have polynomial time certifiers/verifiers.
2. A language L is **NP-Complete** $\iff$
   - L is in **NP**
   - for every L' in **NP**, $L' \leq_P L$
3. L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.
4. Cook-Level theorem...

Theorem (Cook-Levin)

Circuit-SAT *is* **NP-Complete***.*

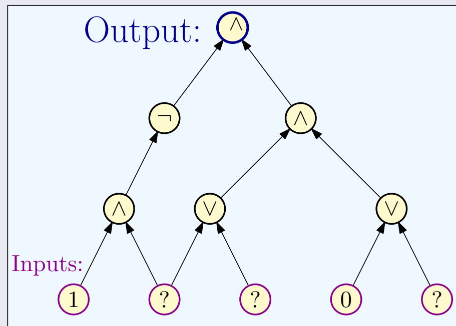
4.2: NP Completeness continued

4.2.1: Preliminaries

Circuits

Definition

A circuit is a directed *acyclic* graph with



1. Input vertices (without incoming edges) labelled with $\mathbf{0}$, $\mathbf{1}$ or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

4.2.2: Cook-Levin Theorem

Cook-Levin Theorem

Definition (Circuit Satisfaction (CSAT).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value 1?

Theorem (Cook-Levin)

CSAT *is* **NP-Complete**.

Need to show

1. **CSAT** is in **NP**.
2. *every* **NP** problem X reduces to **CSAT**.

CSAT: Circuit Satisfaction

Claim

CSAT *is in* **NP**.

1. Certificate: Assignment to input variables.
2. Certifier: Evaluate the value of each gate in a topological sort of DAG and check the output gate value.

CSAT: Circuit Satisfaction

Claim

CSAT *is in* **NP**.

1. Certificate: Assignment to input variables.
2. Certifier: Evaluate the value of each gate in a topological sort of DAG and check the output gate value.

CSAT is NP-hard: Idea

1. Need to show that *every* **NP** problem X reduces to **CSAT**.
2. What does it mean that $X \in$ **NP**?
3. $X \in$ **NP** implies that there are polynomials $p()$ and $q()$ and certifier/verifier program C such that for every string s the following is true:
   1. If s is a YES instance ($s \in X$) then there is a *proof* t of length $p(|s|)$ such that $C(s,t)$ says YES.
   2. If s is a NO instance ($s \notin X$) then for every string t of length at $p(|s|)$, $C(s,t)$ says NO.
   3. $C(s,t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time).

CSAT is NP-hard: Idea

1. Need to show that *every* **NP** problem X reduces to **CSAT**.
2. What does it mean that $X \in$ **NP**?
3. $X \in$ **NP** implies that there are polynomials $p()$ and $q()$ and certifier/verifier program C such that for every string s the following is true:
   1. If s is a YES instance ($s \in X$) then there is a *proof* t of length $p(|s|)$ such that $C(s,t)$ says YES.
   2. If s is a NO instance ($s \notin X$) then for every string t of length at $p(|s|)$, $C(s,t)$ says NO.
   3. $C(s,t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time).

CSAT is NP-hard: Idea

1. Need to show that *every* **NP** problem X reduces to **CSAT**.
2. What does it mean that $X \in$ **NP**?
3. $X \in$ **NP** implies that there are polynomials $p()$ and $q()$ and certifier/verifier program C such that for every string s the following is true:
    1. If s is a YES instance ($s \in X$) then there is a *proof* t of length $p(|s|)$ such that $C(s,t)$ says YES.
    2. If s is a NO instance ($s \notin X$) then for every string t of length at $p(|s|)$, $C(s,t)$ says NO.
    3. $C(s,t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time).

CSAT is NP-hard: Idea

1. Need to show that *every* **NP** problem X reduces to **CSAT**.
2. What does it mean that $X \in$ **NP**?
3. $X \in$ **NP** implies that there are polynomials $p()$ and $q()$ and certifier/verifier program C such that for every string s the following is true:
    1. If s is a YES instance ($s \in X$) then there is a *proof* t of length $p(|s|)$ such that $C(s,t)$ says YES.
    2. If s is a NO instance ($s \notin X$) then for every string t of length at $p(|s|)$, $C(s,t)$ says NO.
    3. $C(s,t)$ runs in time $q(|s| + |t|)$ time (hence polynomial time).

Reducing X to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot, \cdot)$.
2. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing $\mathbf{X}$ to **CSAT**

1. X is in **NP** means we have access to $p(), q(), C(\cdot,\cdot)$.
2. What is $C(\cdot,\cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing X to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot, \cdot)$.
2. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing X to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot, \cdot)$.
2. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing X to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot, \cdot)$.
2. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing $\mathbf{X}$ to CSAT

1. X is in **NP** means we have access to $p(), q(), C(\cdot, \cdot)$.
2. What is $C(\cdot, \cdot)$? It is a program or equivalently a Turing Machine!
3. How are $p()$ and $q()$ given?
   As numbers.
4. Example: if 3 is given then $p(n) = n^3$.
5. **NP** problem $\equiv \langle p, q, C \rangle$. where C is a program or a TM.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C \rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C \rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s,t)$ returns YES.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C \rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C \rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s,t)$ returns YES.

Reducing X to **CSAT**

1. **NP** problem: a three tuple $\langle p, q, C \rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s, t)$ returns YES.
   ...$C(s, t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C \rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s, t)$ returns YES.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C\rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C\rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C\rangle$ is fixed so $|\langle p, q, C\rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s,t)$ returns YES.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C \rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s, t)$ returns YES.
   ...$C(s, t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C \rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s, t)$ returns YES.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C \rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s, t)$ returns YES.
   ...$C(s, t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C \rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C \rangle$ is fixed so $|\langle p, q, C \rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s, t)$ returns YES.

Reducing X to CSAT

1. **NP** problem: a three tuple $\langle p, q, C\rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C\rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C\rangle$ is fixed so $|\langle p, q, C\rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s,t)$ returns YES.

Reducing X to **CSAT**

1. **NP** problem: a three tuple $\langle p, q, C\rangle$.
   C: program or TM, $p(\cdot)$, $q(\cdot)$: polynomials.
2. Problem X: Given string s, is $s \in X$?
3. **Equivalent**:
   $\exists$ proof t of length $p(|s|)$ & $C(s,t)$ returns YES.
   ...$C(s,t)$ runs in $q(|s|)$ time.
4. Reduce from X to **CSAT**...
   Need an algorithm **alg** that
   1. takes s (and $\langle p, q, C\rangle$).
      Creates circuit **G** in poly time in $|s|$.
      ($\langle p, q, C\rangle$ is fixed so $|\langle p, q, C\rangle| = O(1)$.)
   2. **G** is satisfiable
      $\iff$ $\exists$ proof t s.t. $C(s,t)$ returns YES.

Reducing X to CSAT

1. **Q:** How do we reduce X to **CSAT**?
2. Need algorithm **alg** that:
   1. Input: s (and $\langle p, q, C \rangle$).
   2. creates circuit **G** in poly-time in $|s|$ ($\langle p, q, C \rangle$ fixed).
   3. **G** satisfiable $\iff$ $\exists$ proof t: $C(s, t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   1. Convert $C(s, t)$ into a circuit **G** with t as unknown inputs (rest is known including s)
   2. Known: $|t| \leq p(|s|)$ so express boolean string t as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3. Asking if there is a proof t that makes $C(s, t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make **G** evaluate to true/YES.

Reducing X to CSAT

1. **Q:** How do we reduce X to **CSAT**?
2. Need algorithm **alg** that:
   1. Input: s (and $\langle p, q, C\rangle$).
   2. creates circuit **G** in poly-time in $|s|$ ($\langle p, q, C\rangle$ fixed).
   3. **G** satisfiable $\iff$ $\exists$ proof t: $C(s,t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   1. Convert $C(s,t)$ into a circuit **G** with t as unknown inputs (rest is known including s)
   2. Known: $|t| \leq p(|s|)$ so express boolean string t as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3. Asking if there is a proof t that makes $C(s,t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make **G** evaluate to true/YES.

Reducing $\mathbf{X}$ to **CSAT**

1. **Q:** How do we reduce X to **CSAT**?
2. Need algorithm **alg** that:
   1. Input: s (and $\langle p, q, C \rangle$).
   2. creates circuit **G** in poly-time in $|s|$ ($\langle p, q, C \rangle$ fixed).
   3. **G** satisfiable $\iff$ $\exists$ proof t: $C(s,t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   1. Convert $C(s,t)$ into a circuit **G** with t as unknown inputs (rest is known including s)
   2. Known: $|t| \leq p(|s|)$ so express boolean string t as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3. Asking if there is a proof t that makes $C(s,t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make **G** evaluate to true/YES.

Reducing X to CSAT

1. **Q:** How do we reduce X to **CSAT**?
2. Need algorithm **alg** that:
   1. Input: s (and $\langle p, q, C\rangle$).
   2. creates circuit **G** in poly-time in $|s|$ ($\langle p, q, C\rangle$ fixed).
   3. **G** satisfiable $\iff$ $\exists$ proof t: $C(s, t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   1. Convert $C(s, t)$ into a circuit **G** with t as unknown inputs (rest is known including s)
   2. Known: $|t| \leq p(|s|)$ so express boolean string t as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3. Asking if there is a proof t that makes $C(s, t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make **G** evaluate to true/YES.

Reducing $\mathbf{X}$ to CSAT

1. **Q:** How do we reduce X to **CSAT**?
2. Need algorithm **alg** that:
   1. Input: s (and $\langle p, q, C \rangle$).
   2. creates circuit **G** in poly-time in $|s|$ ($\langle p, q, C \rangle$ fixed).
   3. **G** satisfiable $\iff$ $\exists$ proof t: $C(s, t)$ returns YES.
3. Simple but Big Idea: Programs are the same as Circuits!
   1. Convert $C(s, t)$ into a circuit **G** with t as unknown inputs (rest is known including s)
   2. Known: $|t| \leq p(|s|)$ so express boolean string t as $p(|s|)$ variables $t_1, t_2, \ldots, t_k$ where $k = p(|s|)$.
   3. Asking if there is a proof t that makes $C(s, t)$ say YES is same as whether there is an assignment of values to "unknown" variables $t_1, t_2, \ldots, t_k$ that will make **G** evaluate to true/YES.

Example: Independent Set

1. Formal definition:

Independent Set

> **Instance**: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, k
> **Question:** Does $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ have an **Independent Set** of size $\geq k$

2. Certificate: Set $S \subseteq V$.
3. Certifier: Check $|S| \geq k$ and no pair of vertices in S is connected by an edge.
4. **Q:** Formally, why is **Independent Set** in **NP**?

Example: Independent Set

1. Formal definition:

   Independent Set

   Instance: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, k
   Question: Does $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ have an **Independent Set** of size $\geq k$

2. Certificate: Set $S \subseteq V$.
3. Certifier: Check $|S| \geq k$ and no pair of vertices in S is connected by an edge.
4. **Q:** Formally, why is **Independent Set** in **NP**?

Example: Independent Set

1. Formal definition:

   Independent Set

   > **Instance**: $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, k
   > **Question:** Does $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ have an **Independent Set** of size $\geq k$

2. Certificate: Set $S \subseteq V$.
3. Certifier: Check $|S| \geq k$ and no pair of vertices in S is connected by an edge.
4. **Q:** Formally, why is **Independent Set** in **NP**?

Example: Independent Set

1. Formal definition:

   Independent Set

   Instance: $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, k
   Question: Does $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ have an **Independent Set** of size $\geq k$

2. Certificate: Set $S \subseteq V$.
3. Certifier: Check $|S| \geq k$ and no pair of vertices in S is connected by an edge.
4. **Q:** Formally, why is **Independent Set** in **NP**?

Example: Independent Set

Formally why is Independent Set in NP?

1. Input is a "binary" vector:

$$\langle n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k\rangle$$

   encodes $\langle G, k\rangle$.

   1. n is number of vertices in **G**
   2. $y_{i,j}$ is a bit which is 1 if edge (i, j) is in **G** and 0 otherwise (adjacency matrix representation)
   3. k: size of independent set.

2. **Certificate**: $t = t_1 t_2 \ldots t_n$.
   Interpretation: $t_i = 1$ if vertex i is in independent set.
   ... 0 otherwise.

Example: Independent Set

Formally why is Independent Set in NP?

1. Input is a "binary" vector:

$$\langle n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k \rangle$$

encodes $\langle G, k \rangle$.

   1. n is number of vertices in **G**
   2. $y_{i,j}$ is a bit which is 1 if edge (i, j) is in **G** and 0 otherwise (adjacency matrix representation)
   3. k: size of independent set.

2. **Certificate**: $t = t_1 t_2 \ldots t_n$.
Interpretation: $t_i = 1$ if vertex i is in independent set.
... 0 otherwise.

Example: Independent Set

Formally why is Independent Set in NP?

1. Input is a "binary" vector:

$$\langle n, y_{1,1}, y_{1,2}, \ldots, y_{1,n}, y_{2,1}, \ldots, y_{2,n}, \ldots, y_{n,1}, \ldots, y_{n,n}, k \rangle$$

   encodes $\langle G, k \rangle$.

   1. n is number of vertices in **G**
   2. $y_{i,j}$ is a bit which is 1 if edge (i,j) is in **G** and 0 otherwise (adjacency matrix representation)
   3. k: size of independent set.

2. **Certificate**: $t = t_1 t_2 \ldots t_n$.
   Interpretation: $t_i = 1$ if vertex i is in independent set.
   ... 0 otherwise.

Certifier for Independent Set

Certifier $C(s, t)$ for **Independent Set**:

```
if (t_1 + t_2 + ... + t_n < k) then
    return NO
else
    for each (i, j) do
        if (t_i ∧ t_j ∧ y_{i,j}) then
            return NO

return YES
```

Example: Independent Set

Certifier circuit for Independent Set of size at least 2 for graph with 3 vertices

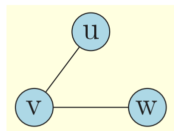


Figure: Graph **G** with $k = 2$

Both ends of an edge?

At least two vertices?

$\wedge$ $\neg$ $\vee$

u, v u, w v, w u v w

1 0 1

Programs, Turing Machines and Circuits

1. **alg**: "program" that takes $f(|s|)$ steps on input string s.
2. Questions: What computer is used?
   What does *step* mean?
3. "Real" computers difficult to reason with mathematically:
   1. instruction set is too rich
   2. pointers and control flow jumps in one step
   3. assumption that pointer to code fits in one word
4. Turing Machines:
   1. simpler model of computation to reason with
   2. can simulate real computers with *polynomial* slow down
   3. all moves are *local* (head moves only one cell)

Programs, Turing Machines and Circuits

1. **alg**: "program" that takes $f(|s|)$ steps on input string s.
2. Questions: What computer is used?
   What does *step* mean?
3. "Real" computers difficult to reason with mathematically:
   1. instruction set is too rich
   2. pointers and control flow jumps in one step
   3. assumption that pointer to code fits in one word
4. Turing Machines:
   1. simpler model of computation to reason with
   2. can simulate real computers with *polynomial* slow down
   3. all moves are *local* (head moves only one cell)

Programs, Turing Machines and Circuits

1. **alg**: "program" that takes $f(|s|)$ steps on input string s.
2. Questions: What computer is used?
   What does *step* mean?
3. "Real" computers difficult to reason with mathematically:
   1. instruction set is too rich
   2. pointers and control flow jumps in one step
   3. assumption that pointer to code fits in one word
4. Turing Machines:
   1. simpler model of computation to reason with
   2. can simulate real computers with *polynomial* slow down
   3. all moves are *local* (head moves only one cell)

Programs, Turing Machines and Circuits

1. **alg**: "program" that takes $f(|s|)$ steps on input string s.
2. Questions: What computer is used?
   What does *step* mean?
3. "Real" computers difficult to reason with mathematically:
   1. instruction set is too rich
   2. pointers and control flow jumps in one step
   3. assumption that pointer to code fits in one word
4. Turing Machines:
   1. simpler model of computation to reason with
   2. can simulate real computers with *polynomial* slow down
   3. all moves are *local* (head moves only one cell)

Programs, Turing Machines and Circuits

1. **alg**: "program" that takes $f(|s|)$ steps on input string s.
2. Questions: What computer is used?
   What does *step* mean?
3. "Real" computers difficult to reason with mathematically:
   1. instruction set is too rich
   2. pointers and control flow jumps in one step
   3. assumption that pointer to code fits in one word
4. Turing Machines:
   1. simpler model of computation to reason with
   2. can simulate real computers with *polynomial* slow down
   3. all moves are *local* (head moves only one cell)

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot,\cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
    1. $\exists$ proof t of length $\leq p(|s|)$
    2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
    1. computes $p(|s|)$ and $q(|s|)$.
    2. As such, M:
        1. Uses at most $q(|s|)$ memory/tape cells.
        2. M can run for at most $q(|s|)$ time.
    3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot,\cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Certifiers that at TMs

1. Assume $C(\cdot, \cdot)$ is a (deterministic) Turing Machine M
2. Problem: Given M, input s, p, q decide if:
   1. $\exists$ proof t of length $\leq p(|s|)$
   2. M executed on the input s, t halts in $q(|s|)$ time and returns YES.
3. **ConvCSAT** reduces above problem to **CSAT**:
   1. computes $p(|s|)$ and $q(|s|)$.
   2. As such, M:
      1. Uses at most $q(|s|)$ memory/tape cells.
      2. M can run for at most $q(|s|)$ time.
   3. Simulates evolution of the states of M and memory over time, using a big circuit.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of $M =$ input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O\big(q(|s|)\big)$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$. Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^{\ell} = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of M = input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O(q(|s|))$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$. Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of $M =$ input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O(q(|s|))$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$. Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^\ell = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of M = input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O\bigl(q(|s|)\bigr)$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$. Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^\ell = x_1 x_2 \dots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of M = input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_ℓ: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O\big(q(|s|)\big)$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$.
   Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

Simulation of Computation via Circuit

1. M state at time ℓ: A string $x^{\ell} = x_1 x_2 \ldots x_k$ where each $x_i \in \{0, 1, B\} \times Q \cup \{q_{-1}\}$.
2. Time 0: State of M = input string s, a guess t of $p(|s|)$ "unknowns", and rest $q(|s|)$ blank symbols.
3. Time $q(|s|)$? Does M stops in q_{accept} with blank tape.
4. Build circuit C_{ℓ}: Evaluates to YES $\iff$ transition of M from time ℓ to time $\ell + 1$ valid. (Circuit of size $O\big(q(|s|)\big)$.
5. $\mathcal{C}$: $C_0 \wedge C_1 \wedge \cdots \wedge C_{q(|s|)}$.
   Polynomial size!
6. Output of $\mathcal{C}$ true $\iff$ sequence of states of M is legal and leads to an accept state.

NP-Hardness of Circuit Satisfaction

Key Ideas in reduction:

1. Use TMs as the code for certifier for simplicity
2. Since $p()$ and $q()$ are known to $\mathcal{A}$, it can set up all required memory and time steps in advance
3. Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time

Note: Above reduction can be done to **SAT** as well. Reduction to **SAT** was the original proof of Steve Cook.

NP-Hardness of Circuit Satisfaction

Key Ideas in reduction:

1. Use TMs as the code for certifier for simplicity
2. Since $p()$ and $q()$ are known to $\mathcal{A}$, it can set up all required memory and time steps in advance
3. Simulate computation of the TM from one time to the next as a circuit that only looks at three adjacent cells at a time

Note: Above reduction can be done to **SAT** as well. Reduction to **SAT** was the original proof of Steve Cook.

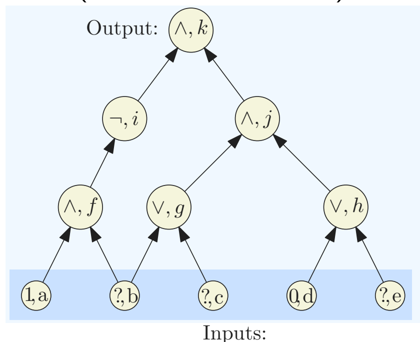
4.3: Showing that **SAT** is **NP-Complete**

4.3.1: Other NP Complete Problems

SAT is **NP-Complete**

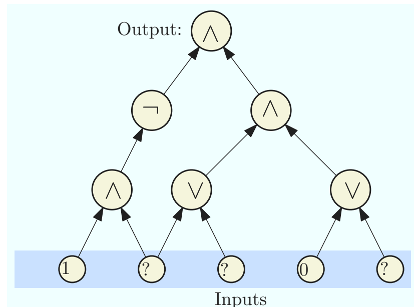
1. We have seen that **SAT** $\in$ **NP**
2. To show **NP-Hardness**, we will reduce Circuit Satisfiability (**CSAT**) to **SAT**

   Instance of **CSAT** (we label each node):

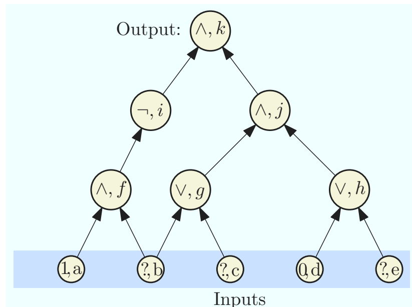


Converting a circuit into a CNF formula

Label the nodes



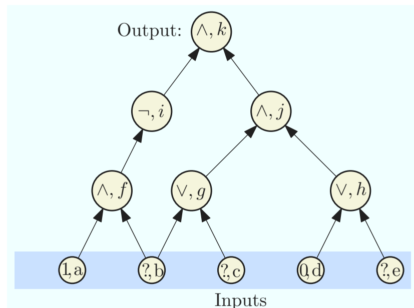
(A) Input circuit



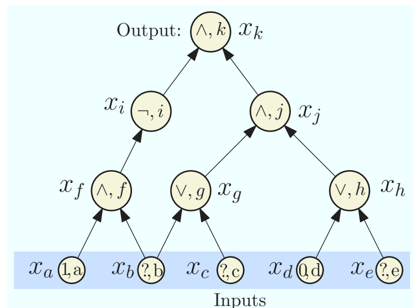
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node



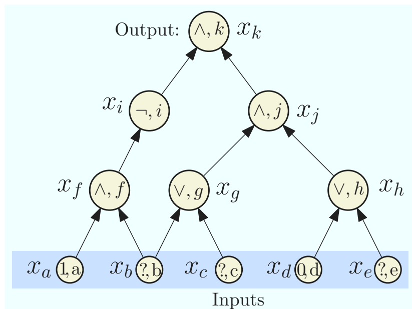
(B) Label the nodes.



(C) Introduce var for each node.

Converting a circuit into a CNF formula

Write a sub-formula for each variable that is true if the var is computed correctly.



(C) Introduce var for each node.

x_k (Demand a sat' assignment!)

$x_k = x_i \wedge x_k$

$x_j = x_g \wedge x_h$

$x_i = \neg x_f$

$x_h = x_d \vee x_e$

$x_g = x_b \vee x_c$

$x_f = x_a \wedge x_b$

$x_d = 0$

$x_a = 1$

(D) Write a sub-formula for each variable that is true if the var is computed correctly.

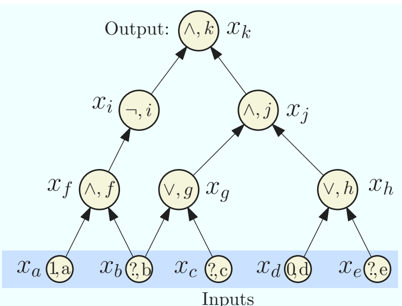
Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula

x_k	x_k
$x_k = x_i \wedge x_j$	$(\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \wedge (x_k \vee \neg x_i \vee \neg x_j)$
$x_j = x_g \wedge x_h$	$(\neg x_j \vee x_g) \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h)$
$x_i = \neg x_f$	$(x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f)$
$x_h = x_d \vee x_e$	$(x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \wedge (\neg x_h \vee x_d \vee x_e)$
$x_g = x_b \vee x_c$	$(x_g \vee \neg x_b) \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c)$
$x_f = x_a \wedge x_b$	$(\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \wedge (x_f \vee \neg x_a \vee \neg x_b)$
$x_d = 0$	$\neg x_d$
$x_a = 1$	x_a

Converting a circuit into a CNF formula

Take the conjunction of all the CNF sub-formulas



$$
\begin{aligned}
& x_k \wedge (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \\
& \wedge (x_k \vee \neg x_i \vee \neg x_j) \wedge (\neg x_j \vee x_g) \\
& \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h) \\
& \wedge (x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f) \\
& \wedge (x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \\
& \wedge (\neg x_h \vee x_d \vee x_e) \wedge (x_g \vee \neg x_b) \\
& \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c) \\
& \wedge (\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \\
& \wedge (x_f \vee \neg x_a \vee \neg x_b) \wedge (\neg x_d) \wedge x_a
\end{aligned}
$$

We got a CNF formula that is satisfiable $\iff$ the original circuit is satisfiable.

Reduction: CSAT $\leq_P$ SAT

1. For each gate (vertex) v in the circuit, create a variable x_v
2. Case $\neg$: v is labeled $\neg$ and has one incoming edge from u (so $x_v = \neg x_u$). In **SAT** formula generate, add clauses $(x_u \vee x_v)$, $(\neg x_u \vee \neg x_v)$. Observe that

$$x_v = \neg x_u \text{ is true} \iff \begin{matrix} (x_u \vee x_v) \\ (\neg x_u \vee \neg x_v) \end{matrix} \text{ both true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

1. Case $\vee$: So $x_v = x_u \vee x_w$. In **SAT** formula generated, add clauses $(x_v \vee \neg x_u)$, $(x_v \vee \neg x_w)$, and $(\neg x_v \vee x_u \vee x_w)$. Again, observe that

$$\Bigl(x_v = x_u \vee x_w\Bigr) \text{ is true} \quad \iff \quad \begin{matrix} (x_v \vee \neg x_u), \\ (x_v \vee \neg x_w), \\ (\neg x_v \vee x_u \vee x_w) \end{matrix} \quad \text{all true.}$$

1. Case $\wedge$: So $x_v = x_u \wedge x_w$. In **SAT** formula generated, add clauses $(\neg x_v \vee x_u)$, $(\neg x_v \vee x_w)$, and $(x_v \vee \neg x_u \vee \neg x_w)$. Again observe that

$$x_v = x_u \wedge x_w \text{ is true} \iff \begin{array}{l} (\neg x_v \vee x_u), \\ (\neg x_v \vee x_w), \\ (x_v \vee \neg x_u \vee \neg x_w) \end{array} \text{ all true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

1. If v is an input gate with a fixed value then we do the following. If $x_v = 1$ add clause x_v. If $x_v = 0$ add clause $\neg x_v$
2. Add the clause x_v where v is the variable for the output gate

Correctness of Reduction

Need to show circuit C is satisfiable iff φ_C is satisfiable

$\Rightarrow$ Consider a satisfying assignment a for C

1. Find values of all gates in C under a
2. Give value of gate v to variable x_v; call this assignment a'
3. a' satisfies φ_C (exercise)

$\Leftarrow$ Consider a satisfying assignment a for φ_C

1. Let a' be the restriction of a to only the input variables
2. Value of gate v under a' is the same as value of x_v in a
3. Thus, a' satisfies C

Theorem

SAT *is* **NP-Complete**.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why? Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why? Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why? Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

Proving that a problem X is NP-Complete

1. To prove X is **NP-Complete**, show
   1. Show X is in **NP**.
      1. certificate/proof of polynomial size in input
      2. polynomial time certifier $C(s, t)$
   2. Reduction from a known **NP-Complete** problem such as **CSAT** or **SAT** to X
2. **SAT** $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why? Transitivity of reductions:
3. $Y \leq_P$ **SAT** and **SAT** $\leq_P X$ and hence $Y \leq_P X$.

NP-Completeness via Reductions

1. What we currently know:
   1. **CSAT** is **NP-Complete**.
   2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
   3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
   4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
   5. **Vertex Cover** is **NP-Complete**.
   6. **Clique** is **NP-Complete**.
2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
3. A surprisingly frequent phenomenon!

NP-Completeness via Reductions

1. What we currently know:
   1. **CSAT** is **NP-Complete**.
   2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
   3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
   4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
   5. **Vertex Cover** is **NP-Complete**.
   6. **Clique** is **NP-Complete**.
2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
3. A surprisingly frequent phenomenon!

NP-Completeness via Reductions

1. What we currently know:
   1. **CSAT** is **NP-Complete**.
   2. **CSAT** $\leq_P$ **SAT** and **SAT** is in **NP** and hence **SAT** is **NP-Complete**.
   3. **SAT** $\leq_P$ **3SAT** and hence **3SAT** is **NP-Complete**.
   4. **3SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
   5. **Vertex Cover** is **NP-Complete**.
   6. **Clique** is **NP-Complete**.
2. Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.
3. A surprisingly frequent phenomenon!

4.4: More reductions...

Next...

Prove

- **Hamiltonian Cycle** Problem is **NP-Complete**.
- 3-Coloring is **NP-Complete**.
- **Subset Sum**.

Part I

NP-Completeness of Hamiltonian Cycle

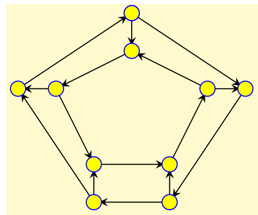
4.5: Reduction from **3SAT** to **Hamiltonian Cycle**

Directed Hamiltonian Cycle

Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once

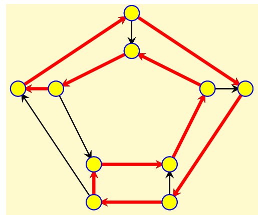


Directed Hamiltonian Cycle

Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once



Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in NP
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: Will prove...
  3SAT $\leq_P$ Directed Hamiltonian Cycle.

Directed Hamiltonian Cycle is **NP-Complete**

- Directed Hamiltonian Cycle is in $\boldsymbol{NP}$
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: Will prove...
  3SAT $\leq_P$ **Directed Hamiltonian Cycle**.

Reduction

1. **3SAT** formula φ create a graph G_φ such that
   - G_φ has a Hamiltonian cycle $\iff$ φ is satisfiable
   - G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$
2. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction

1. **3SAT** formula φ create a graph G_φ such that
   - G_φ has a Hamiltonian cycle $\iff$ φ is satisfiable
   - G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$
2. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction

1. **3SAT** formula φ create a graph G_φ such that
   - G_φ has a Hamiltonian cycle $\iff$ φ is satisfiable
   - G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$
2. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction

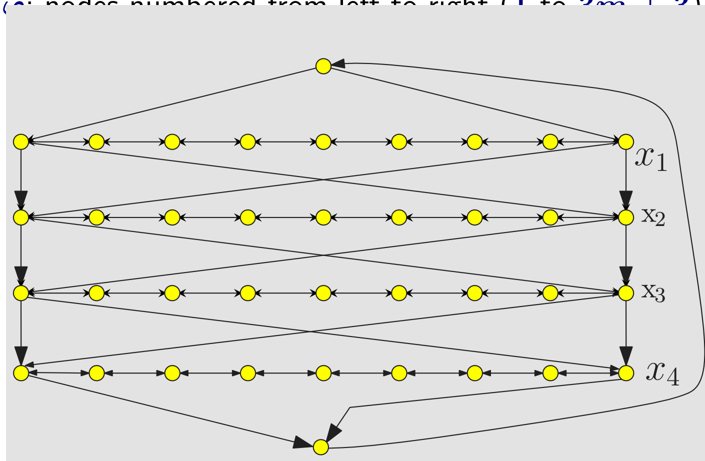
1. **3SAT** formula φ create a graph G_φ such that
   - G_φ has a Hamiltonian cycle $\iff$ φ is satisfiable
   - G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$
2. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction: First Ideas

- Viewing SAT: Assign values to n variables, and each clauses has 3 ways in which it can be satisfied.
- Construct graph with 2^n Hamiltonian cycles, where each cycle corresponds to some boolean assignment.
- Then add more graph structure to encode constraints on assignments imposed by the clauses.

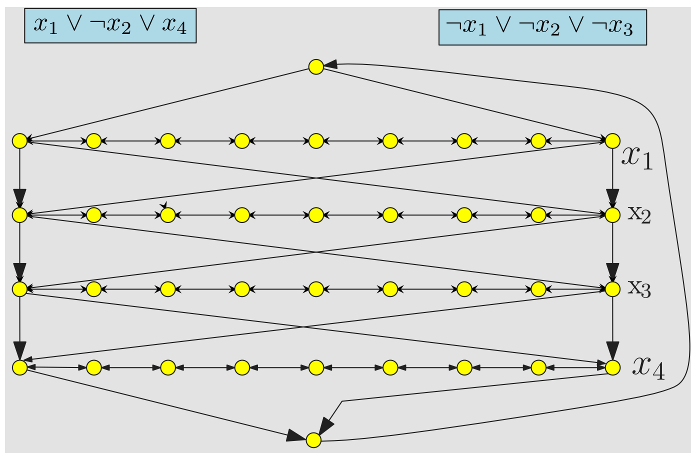
The Reduction: Phase I

- Traverse path i from left to right $\iff$ x_i is set to true.
- Each path has $3(m+1)$ nodes where m is number of clauses in φ; nodes numbered from left to right (1 to $3m+3$)



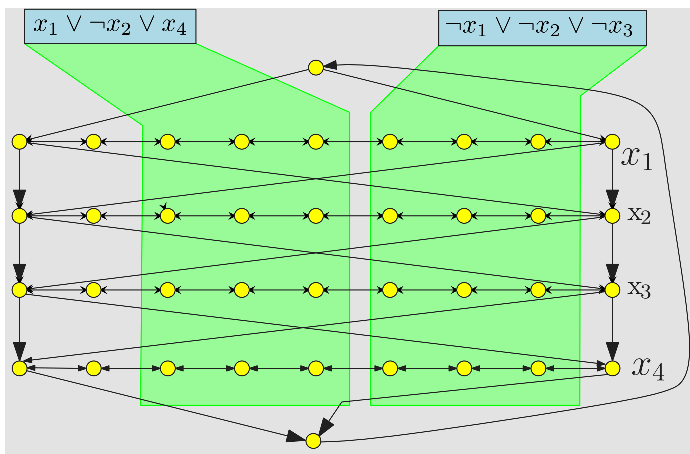
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



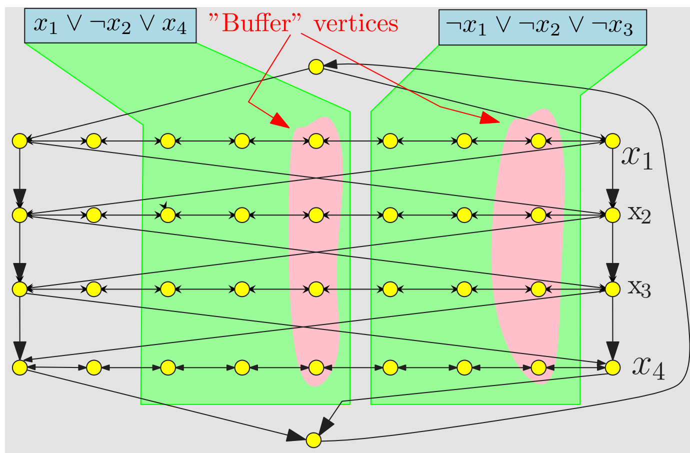
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



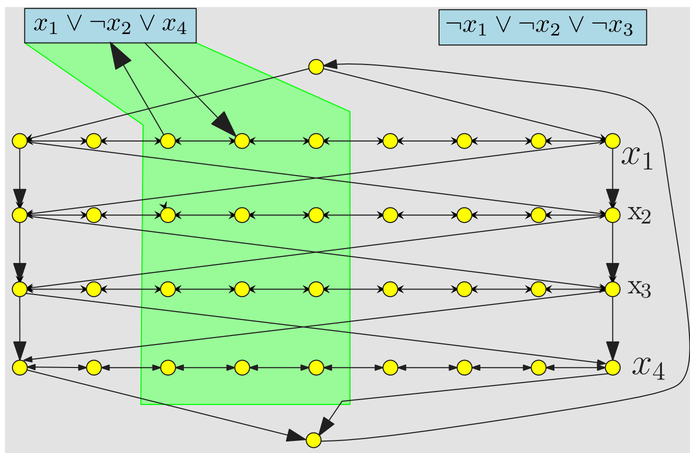
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



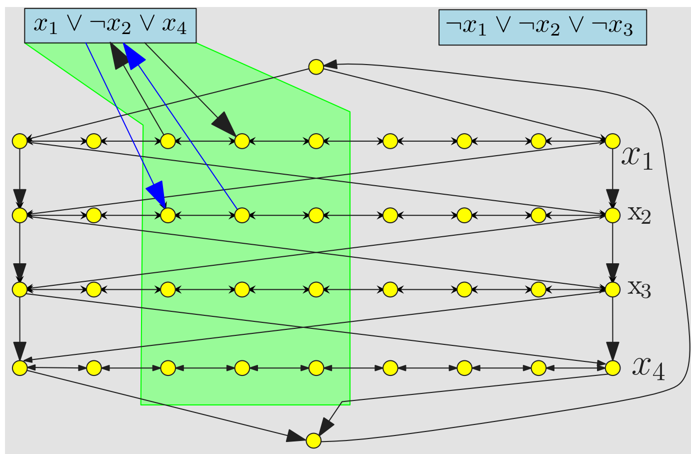
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j + 1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j + 1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



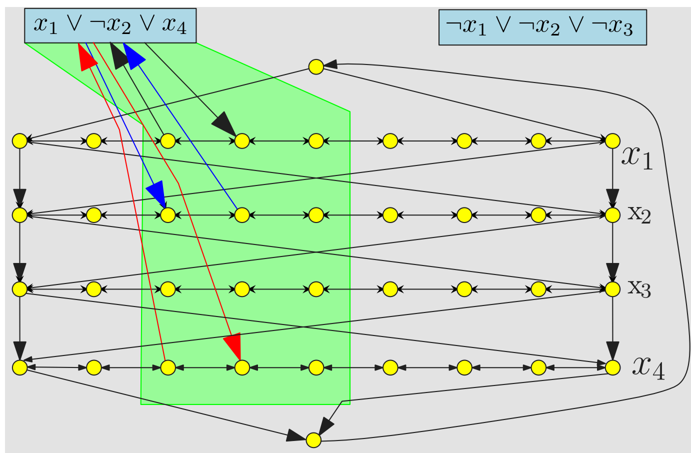
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



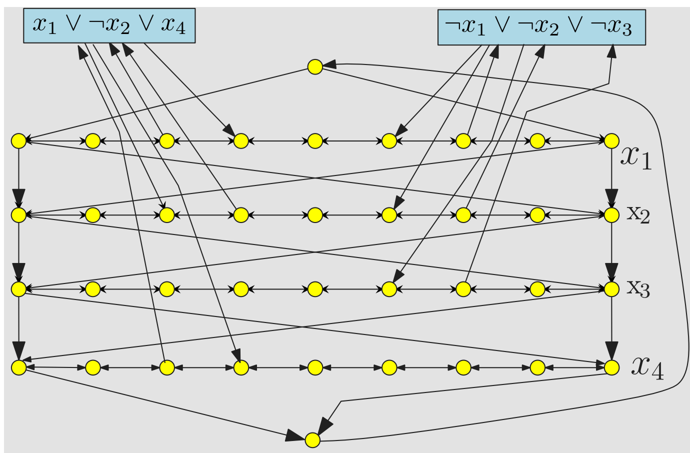
The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



The Reduction: Phase II

- Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



In the next lecture...

Correctness proof of the above reduction, and more **NPC** problems.

Notes

Notes

Notes

Notes