Pre-lecture brain teaser

A boolean expression is in disjunctive normal form if it consists of the union of clauses where each clause is composed of the intersection of literals. For example:

\[
(\overline{x_1} \land x_3 \land x_4) \lor (x_2 \land \overline{x_3} \land x_4)
\]

Imagine we have a problem: DNF-SAT, where given a DNF formula, we want to know if there is a satisfying assignment. We know two things:

- Finding a satisfying assignment for a DNF formula takes polynomial time.
- We can rewrite any CNF formula as a DNF formula.

Hence I do the smart thing and say since CNF-SAT \( \leq_P \) DNF-SAT, then CNF-SAT \( \in \) NP.

Am I correct?
CS/ECE-374: Lecture 26 - NP-Complete reductions

Lecturer: Nickvash Kani
Chat moderator: Samir Khan
April 27, 2021

University of Illinois at Urbana-Champaign
A boolean expression is in disjunctive normal form if it consists of the union of clauses where each clause is composed of the intersection of literals. For example:

\[ (\overline{x}_1 \land x_3 \land x_4) \lor (x_2 \land \overline{x}_3 \land x_4) \]

Imagine we have a problem: DNF-SAT, where given a DNF formula, we want to know if there is a satisfying assignment. We know two things:

- Finding a satisfying assignment for a DNF formula takes polynomial time.
- We can rewrite any CNF formula as a DNF formula.

Hence I do the smart thing and say since \( \text{CNF-SAT} \leq_P \text{DNF-SAT} \), then \( \text{CNF-SAT} \in \text{NP} \).

Am I correct?
NP-Completeness of two problems:

- Hamiltonian Cycle
- 3-Color

Important: understanding the problems and that they are hard.

Proofs and reductions will be sketchy and mainly to give a flavor

\[ \text{NP-Complete} \rightarrow \text{NP-hard} \quad (\text{harder than all NP problems}) \]

\[ \text{ENP} \rightarrow \text{Write poly time certifier} \]

\[ C(s,+) \]

\[ P \text{ instance} \text{ solution} \]
Reduction from 3SAT to Hamiltonian Cycle
**Input**  Given a directed graph $G = (V, E)$ with $n$ vertices

**Goal**  Does $G$ have a Hamiltonian cycle?

- 2. A Hamiltonian cycle is a cycle in the graph that visits every vertex in $G$ exactly once
**Input**  Given a directed graph \( G = (V, E) \) with \( n \) vertices

**Goal**  Does \( G \) have a Hamiltonian cycle?

- 2. A Hamiltonian cycle is a cycle in the graph that visits every vertex in \( G \) exactly once

---

Want to prove \( \text{HC} \in \text{NP-hard} \)
Is the following graph Hamiltonian?

a  Yes.

b  No.
Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in \( NP \): exercise
- **Hardness**: We will show
  \( 3\text{-SAT} \leq_p \text{Directed Hamiltonian Cycle} \)

\( \text{NP-hard} \)
Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in \( NP \): exercise

- **Hardness**: We will show
  \( 3\text{-SAT} \leq_p \text{Directed Hamiltonian Cycle} \)

\[ I_x = \phi \text{ (3CNF)} \]
\[ I_y = G(n, E) \]

\[ A_y = \text{ORAC}_{AC} \]
\[ A_x = \text{Deduce_{sat}} \]

If \( \phi \) is satisfiable, have \( HC \) if \( \phi \) is not satisfiable, no \( HC \) if \( \phi \) is not satisfiable
Given 3-SAT formula \( \varphi \) create a graph \( G_\varphi \) such that

- \( G_\varphi \) has a Hamiltonian cycle if and only if \( \varphi \) is satisfiable
- \( G_\varphi \) should be constructible from \( \varphi \) by a polynomial time algorithm \( \mathcal{A} \)

**Notation:** \( \varphi \) 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. $\langle x_1 \lor x_2 \lor x_3 \rangle$
- 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.
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

\[ f(x_1) = 1 \]  

(3)
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

$$f(x_1) = 1$$  \hspace{1cm} (3)$$

We create a cyclic graph that always has a hamiltonian:
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

\[ f(x_1) = 1 \] \hspace{1cm} (3)

We create a cyclic graph that always has a hamiltonian:

![Cyclic Graph Diagram]

But how do we encode the variable?
Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment. Consider the expression:

\[ f(x_1) = 1 \]  \hspace{1cm} (4)

Maybe we can encode the variable \( x_1 \) in terms of the cycle direction:
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

\[ f(x_1) = 1 \]  \hspace{1cm} (4)

Maybe we can encode the variable \( x_1 \) in terms of the cycle direction:

If \( x_1 = 1 \)

If \( x_1 = 0 \)
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  \hspace{1cm} (5)

Maybe two circles? Now we need to connect them so that we have a single hamiltonian path
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  \hspace{1cm} (5)

Maybe two circles? Now we need to connect them so that we have a single hamiltonian path.

\[ x_1 = 0 \]
\[ x_2 > 1 \]
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \] (6)

Now we need to connect them so that we have a single hamiltonian path
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  \hspace{1cm} (6)

Now we need to connect them so that we have a single hamiltonian path

\[ x_1 x_2 = 0 \text{ if } x_1 x_2 = 1 \text{ and } x_1 x_2 = 1 \]
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  

(7)

Would be nice to have a single start/stop node.
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  \hspace{1cm} (7)

Would be nice to have a single start/stop node.
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  

(8)

Getting a bit messy. Let’s reorganize:
How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  

(8)

Getting a bit messy. Let’s reorganize:
Reduction: Encoding idea II

How do we encode multiple variables?

\[ f(x_1, x_2) = 1 \]  \hspace{1cm} (9)

This is how we encode variable assignments in a variable loop!

\[ x_1, x_2 = [0, 1] \]
How do we handle clauses?

$$f(x_1) = \begin{cases} x_1 \end{cases} \quad \chi_1 = 1$$  \hspace{1cm} (10)

Let's go back to our one variable graph:
How do we handle clauses?

\[ f(x_1) = x_1 \]  \hspace{1cm} (11)

Add node for clause:

\[ x_1 \]
Reduction: Encoding idea III

How do we handle clauses?

\[ f(x_1, x_2) = (x_1 \lor \overline{x_2}) \]  \hspace{1cm} (12)

What do we do if the clause has two literals:
How do we handle clauses?

\[ f(x_1, x_2) = (x_1 \lor \overline{x_2}) \]

(12)

What do we do if the clause has two literals:

\[ [0, 0] \]

[Diagram of a graph with nodes and edges labeled with variables and clauses. The graph shows connections between variables and clauses, indicating the encoding process.]
Reduction: Encoding idea III

How do we handle clauses?

\[ f(x_1, x_2) = (x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_2) \]  

What if the expression has multiple clauses:

Gadget
The Reduction: Review 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 $\varphi$; nodes numbered from left to right (1 to $3m + 3$)
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 $C_j$. 

$$x_1 \lor \neg x_2 \lor x_4$$

$$\neg x_1 \lor \neg x_2 \lor \neg x_3$$
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 $C_j$. 

\[
x_1 \lor \neg x_2 \lor x_4 \\
\neg x_1 \lor \neg x_2 \lor \neg x_3
\]
The Reduction algorithm: Review 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 $C_j$.

$x_1 \lor \neg x_2 \lor x_4$

$\neg x_1 \lor \neg x_2 \lor \neg x_3$

"Buffer" vertices

$\textbf{Diagram}$
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 $C_j$. 

$x_1 \lor \neg x_2 \lor x_4$  

$\neg x_1 \lor \neg x_2 \lor \neg x_3$
The Reduction algorithm: Review II

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

\[
\begin{align*}
x_1 \lor \neg x_2 \lor x_4 & \\
\neg x_1 \lor \neg x_2 \lor \neg x_3 & 
\end{align*}
\]
The Reduction algorithm: Review 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 $C_j$. 

$x_1 \lor \neg x_2 \lor x_4$

$\neg x_1 \lor \neg x_2 \lor \neg x_3$
The Reduction algorithm: Review 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 $C_j$. 

$\overline{x_1} \lor \overline{x_2} \lor x_4$

$\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}$
Correctness Proof

**Theorem**
\( \varphi \) has a satisfying assignment iff \( G_\varphi \) has a Hamiltonian cycle.

Based on proving following two lemmas.

**Lemma**
If \( \varphi \) has a satisfying assignment then \( G_\varphi \) has a Hamilton cycle.

**Lemma**
If \( G_\varphi \) has a Hamilton cycle then \( \varphi \) has a satisfying assignment.
Lemma
If $\varphi$ has a satisfying assignment then $G_\varphi$ has a Hamilton cycle.

Proof.

$\Rightarrow$ Let $a$ be the satisfying assignment for $\varphi$. Define Hamiltonian cycle as follows

- If $a(x_i) = 1$ then traverse path $i$ from left to right
- If $a(x_i) = 0$ then traverse path $i$ from right to left
- For each clause, path of at least one variable is in the “right” direction to splice in the node corresponding to clause
Suppose $\Pi$ is a Hamiltonian cycle in $G_\varphi$.

**Definition**
We say $\Pi$ is *canonical* if for each clause vertex $c_j$ the edge of $\Pi$ entering $c_j$ and edge of $\Pi$ leaving $c_j$ are from the same path corresponding to some variable $x_i$. Otherwise $\Pi$ is *non-canonical* or emphcheating.
Suppose $\Pi$ is a Hamiltonian cycle in $G_\varphi$.

**Definition**
We say $\Pi$ is *canonical* if for each clause vertex $c_j$ the edge of $\Pi$ entering $c_j$ and edge of $\Pi$ leaving $c_j$ are from the same path corresponding to some variable $x_i$. Otherwise $\Pi$ is *non-canonical* or emphcheating.

**Lemma**
*Every Hamilton cycle in $G_\varphi$ is canonical.*
Proof of Lemma

**Lemma**

*Every Hamilton cycle in $G_\varphi$ is canonical.*

- If $\Pi$ enters $c_j$ (vertex for clause $C_j$) from vertex $3j$ on path $i$ then it must leave the clause vertex on edge to $3j + 1$ on the same path $i$
  - If not, then only unvisited neighbor of $3j + 1$ on path $i$ is $3j + 2$
  - Thus, we don’t have two unvisited neighbors (one to enter from, and the other to leave) to have a Hamiltonian Cycle
- Similarly, if $\Pi$ enters $c_j$ from vertex $3j + 1$ on path $i$ then it must leave the clause vertex $c_j$ on edge to $3j$ on path $i$
Lemma
Any canonical Hamilton cycle in $G_\varphi$ corresponds to a satisfying truth assignment to $\varphi$.

Consider a canonical Hamilton cycle $\Pi$.

- For every clause vertex $c_j$, vertices visited immediately before and after $c_j$ are connected by an edge on same path corresponding to some variable $x_i$
- We can remove $c_j$ from cycle, and get Hamiltonian cycle in $G - c_j$
- Hamiltonian cycle from $\Pi$ in $G - \{c_1, \ldots, c_m\}$ traverses each path in only one direction, which determines truth assignment
- Easy to verify that this truth assignment satisfies $\varphi$
Hamiltonian cycle in undirected graph
Hamiltonian Cycle in *Undirected* Graphs

Problem

**Input**  Given *undirected* graph $G = (V, E)$

**Goal**  Does $G$ have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?

We've proved directed-HC $\leq_p$ Undirected-HC

![Diagram](image)
NP-Completeness

Theorem
*Hamiltonian cycle* problem for undirected graphs is *NP-Complete*.

Proof.
- The problem is in *NP*; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem
**Goal:** Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

**Reduction**

- 
- 

![](image.png)
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}, v, v_{out}$
- A directed edge $(a, b)$ is replaced by edge $(a_{out}, b_{in})$
Goal: Given directed graph $G$, need to construct undirected graph $G'$ such that $G$ has Hamiltonian Path iff $G'$ has Hamiltonian path

Reduction

- Replace each vertex $v$ by 3 vertices: $v_{in}$, $v$, and $v_{out}$
- A directed edge $(a, b)$ is replaced by edge $(a_{out}, b_{in})$
Reduction: Wrapup

- The reduction is polynomial time (exercise)
- The reduction is correct (exercise)
Hamiltonian Path

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

**Goal**  Does $G$ have a Hamiltonian path?

- A Hamiltonian path is a path in the graph that visits every vertex in $G$ exactly once.
Hamiltonian Path

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

**Goal**  Does $G$ have a Hamiltonian path?

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

**Theorem**

*Directed Hamiltonian Path* and *Undirected Hamiltonian Path* are NP-Complete.

Easy to modify the reduction from *3-SAT* to *Haltonian Cycle* or do a reduction from *Halitonian Cycle*
Hamiltonian Path

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

**Goal**  Does $G$ have a Hamiltonian path?

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

**Theorem**  
*Directed Hamiltonian Path and Undirected Hamiltonian Path are NP-Complete.*

Easy to modify the reduction from 3-SAT to Halitonian Cycle or do a reduction from Halitonian Cycle

Implies that Longest Simple Path in a graph is NP-Complete.
NP-Completeness of Graph Coloring
Problem: **Graph Coloring**

**Instance:** $G = (V, E)$: Undirected graph, integer $k$.

**Question:** Can the vertices of the graph be colored using $k$ colors so that vertices connected by an edge do not get the same color?
Problem: 3 Coloring

**Instance:** \( G = (V, E) \): Undirected graph.

**Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?
Problem: 3 Coloring

**Instance:** $G = (V, E)$: Undirected graph.

**Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?
Observation: If $G$ is colored with $k$ colors then each color class (nodes of same color) form an independent set in $G$. Thus, $G$ can be partitioned into $k$ independent sets iff $G$ is $k$-colorable.

Graph 2-Coloring can be decided in polynomial time.

$G$ is 2-colorable iff $G$ is bipartite! There is a linear time algorithm to check if $G$ is bipartite using Breadth-first-Search.
Problems related to graph coloring
Register Allocation
Assign variables to (at most) $k$ registers such that variables needed at the same time are not assigned to the same register.

Interference Graph
Vertices are variables, and there is an edge between two vertices, if the two variables are “live” at the same time.

Observations

- [Chaitin] Register allocation problem is equivalent to coloring the interference graph with $k$ colors
- Moreover, $3$-COLOR $\leq_P k$ – Register Allocation, for any $k \geq 3$
Class Room Scheduling

Given \( n \) classes and their meeting times, are \( k \) rooms sufficient?

Reduce to Graph \( k \)-Coloring problem

Create graph \( G \)

\[ \begin{align*}
\cdot & \text{ a node } v_i \text{ for each class } i \\
\cdot & \text{ an edge between } v_i \text{ and } v_j \text{ if classes } i \text{ and } j \text{ conflict}
\end{align*} \]

Exercise: \( G \) is \( k \)-colorable iff \( k \) rooms are sufficient
Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint bands of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interfere
Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range \([a, b]\) into disjoint bands of frequencies \([a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]\)
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

**Problem:** given \(k\) bands and some region with \(n\) towers, is there a way to assign the bands to avoid interference?

Can reduce to \(k\)-coloring by creating interference/conflict graph on towers.
Showing hardness of 3 COLORING
3-Coloring is NP-Complete

- **3-Coloring** is in **NP**.
  - Non-deterministically guess a 3-coloring for each node
  - Check if for each edge \((u, v)\), the color of \(u\) is different from that of \(v\).

- **Hardness**: We will show \(3\text{-SAT} \leq_p 3\text{-Coloring}\).
Reduction Idea

Start with $3\text{SAT}$ formula (i.e., $3\text{CNF}$ formula) $\varphi$ with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$. Create graph $G_\varphi$ such that $G_\varphi$ is $3$-colorable iff $\varphi$ is satisfiable

- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in $G_\varphi$.
- create triangle with node True, False, Base
- for each variable $x_i$ two nodes $v_i$ and $\overline{v}_i$ connected in a triangle with common Base
- If graph is $3$-colored, either $v_i$ or $\overline{v}_i$ gets the same color as True. Interpret this as a truth assignment to $v_i$
- Need to add constraints to ensure clauses are satisfied (next phase)
We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true
Reduction Idea 1 - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Let’s start off with the simplest SAT we can think of:

\[ f(x_1, x_2) = (x_1 \lor x_2) \]  \hspace{1cm} (14)
We want to create a gadget that:

• Is 3 colorable if at least one of the literals is true
• Not 3-colorable if none of the literals are true

Let’s start off with the simplest SAT we can think of:

\[ f(x_1, x_2) = (x_1 \lor x_2) \] (14)

Assume green=true and red=false,
We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Let’s try some stuff:
We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Seems to work:
Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true
Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

How do we do the same thing for 3 variables?:

\[ f(x_1, x_2, x_3) = (x_1 \lor x_2 \lor x_3) \] (15)
Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

How do we do the same thing for 3 variables?:

\[ f(x_1, x_2, x_3) = (x_1 \lor x_2 \lor x_3) \]  

(15)

Assume green=true and red=false,
3 color this gadget II

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).

a  Yes.
b  No.
3 color this gadget.

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).

a  Yes.

b  No.
3-coloring of the clause gadget

FFF - BAD

FFT

FTF

FTT

TFF

TFT

TTF

TTT
Next we need a gadget that assigns literals. Our previously constructed gadget assumes:

- All literals are either red or green.
- Need to limit graph so only $x_1$ or $\overline{x_1}$ is green. Other must be red.
Reduction Idea II - Literal Assignment II

![Graph diagram showing T, F, and variables v1, v2, vn, v̅1, v̅2, v̅n](image)
Review Clause Satisfiability Gadget

For each clause $C_j = (a \lor b \lor c)$, create a small gadget graph

- gadget graph connects to nodes corresponding to $a, b, c$
- needs to implement OR

OR-gadget-graph:
OR-Gadget Graph

Property: if $a, b, c$ are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

Property: if one of $a, b, c$ is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.
Reduction

- create triangle with nodes True, False, Base
- for each variable $x_i$ two nodes $v_i$ and $\overline{v}_i$ connected in a triangle with common Base
- for each clause $C_j = (a \lor b \lor c)$, add OR-gadget graph with input nodes $a, b, c$ and connect output node of gadget to both False and Base
Lemma
No legal 3-coloring of above graph (with coloring of nodes $T, F, B$ fixed) in which $a, b, c$ are colored False. If any of $a, b, c$ are colored True then there is a legal 3-coloring of above graph.
Example

\( \varphi = (u \lor \neg v \lor w) \land (v \lor x \lor \neg y) \)
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_\varphi \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v}_i \) False
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_\varphi \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v_i} \) False
- for each clause \( C_j = (a \lor b \lor c) \) at least one of \( a, b, c \) is colored True. OR-gadget for \( C_j \) can be 3-colored such that output is True.
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_{\varphi} \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v}_i \) False
- for each clause \( C_j = (a \lor b \lor c) \) at least one of \( a, b, c \) is colored True. OR-gadget for \( C_j \) can be 3-colored such that output is True.
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_{\varphi} \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v}_i \) False
- for each clause \( C_j = (a \lor b \lor c) \) at least one of \( a, b, c \) is colored True. OR-gadget for \( C_j \) can be 3-colored such that output is True.

\( G_{\varphi} \) is 3-colorable implies \( \varphi \) is satisfiable

- if \( v_i \) is colored True then set \( x_i \) to be True, this is a legal truth assignment
Correctness of Reduction

\( \varphi \) is satisfiable implies \( G_\varphi \) is 3-colorable

- if \( x_i \) is assigned True, color \( v_i \) True and \( \overline{v}_i \) False
- for each clause \( C_j = (a \lor b \lor c) \) at least one of \( a, b, c \) is colored True. OR-gadget for \( C_j \) can be 3-colored such that output is True.

\( G_\varphi \) is 3-colorable implies \( \varphi \) is satisfiable

- if \( v_i \) is colored True then set \( x_i \) to be True, this is a legal truth assignment
- consider any clause \( C_j = (a \lor b \lor c) \). It cannot be that all \( a, b, c \) are False. If so, output of OR-gadget for \( C_j \) has to be colored False but output is connected to Base and False!
Graph generated in reduction from 3SAT to 3COLOR