Circuit satisfiability and Cook-Levin Theorem

Lecture 24

Thursday, December 3, 2020

LaTeXed: October 30, 2020 13:43

24.1
Recap

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** if and only if

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem 24.1 (Cook-Levin).

SAT is **NP-Complete**.

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** if and only if

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem 24.1 (Cook-Levin).

SAT is **NP-Complete**.

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** if and only if

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem 24.1 (Cook-Levin).

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

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** if and only if

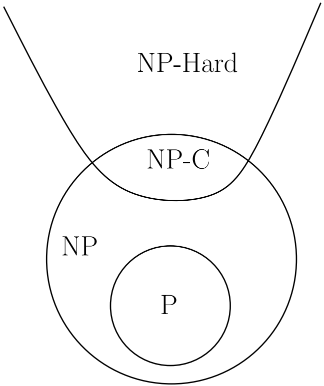
- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem 24.1 (Cook-Levin).

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

Pictorial View



P and NP

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

Question: Suppose $\mathbf{P} \neq \mathbf{NP}$. Is every problem in $\mathbf{NP} \setminus \mathbf{P}$ also **NP-Complete**?

Theorem 24.2 (Ladner).

*If $\mathbf{P} \neq \mathbf{NP}$ then there is a problem/language $X \in \mathbf{NP} \setminus \mathbf{P}$ such that X is not **NP-Complete**.*

P and **NP**

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

Question: Suppose $\mathbf{P} \neq \mathbf{NP}$. Is every problem in $\mathbf{NP} \setminus \mathbf{P}$ also **NP-Complete**?

Theorem 24.2 (Ladner).

If $\mathbf{P} \neq \mathbf{NP}$ *then there is a problem/language* $\boldsymbol{X} \in \mathbf{NP} \setminus \mathbf{P}$ *such that* $\boldsymbol{X}$ *is not* **NP-Complete**.

P and **NP**

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

Question: Suppose $\mathbf{P} \neq \mathbf{NP}$. Is every problem in $\mathbf{NP} \setminus \mathbf{P}$ also **NP-Complete**?

Theorem 24.2 (Ladner).

If $\mathbf{P} \neq \mathbf{NP}$ *then there is a problem/language* $\boldsymbol{X} \in \mathbf{NP} \setminus \mathbf{P}$ *such that* $\boldsymbol{X}$ *is not* **NP-Complete**.

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\cong_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\cong_P$ **3SAT**
6. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover** $\cong_P$ **3SAT** $\cong_P$ **SAT** $\cong_P$ **Hamiltonian Cycle**

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\cong_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\cong_P$ **3SAT**
6. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover** $\cong_P$ **3SAT** $\cong_P$ **SAT** $\cong_P$ **Hamiltonian Cycle**

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\cong_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\cong_P$ **3SAT**
6. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover** $\cong_P$ **3SAT** $\cong_P$ **SAT** $\cong_P$ **Hamiltonian Cycle**

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\approxeq_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\approxeq_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\approxeq_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\approxeq_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\approxeq_P$ **3SAT**
6. **Clique** $\approxeq_P$ **Independent Set** $\approxeq_P$ **Vertex Cover** $\approxeq_P$ **3SAT** $\approxeq_P$ **SAT** $\approxeq_P$ **Hamiltonian Cycle**

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\cong_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\cong_P$ **3SAT**
6. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover** $\cong_P$ **3SAT** $\cong_P$ **SAT** $\cong_P$ **Hamiltonian Cycle**

What do we know so far

1. **Independent Set** $\leq_P$ **Clique**, **Clique** $\leq_P$ **Independent Set**.
   $\implies$ **Clique** $\cong_P$ **Independent Set**.
2. **Vertex Cover** $\leq_P$ **Independent Set**, **Independent Set** $\leq_P$ **Vertex Cover**.
   $\implies$ **Independent Set** $\cong_P$ **Vertex Cover**.
3. **3SAT** $\leq_P$ **SAT**, **SAT** $\leq_P$ **3SAT** $\implies$ **3SAT** $\cong_P$ **SAT**.
4. **3SAT** $\leq_P$ **Independent Set** .
   Exercise (or Cook-Levin theorem): **Independent Set** $\leq_P$ **SAT**
   $\implies$ **3SAT** $\cong_P$ **Independent Set**.
5. **SAT** $\leq_P$ **Hamiltonian Cycle**
   Exercise (or Cook-Levin theorem): **Hamiltonian Cycle** $\leq_P$ **3SAT**
   $\implies$ **Hamiltonian Cycle** $\cong_P$ **3SAT**
6. **Clique** $\cong_P$ **Independent Set** $\cong_P$ **Vertex Cover** $\cong_P$ **3SAT** $\cong_P$ **SAT** $\cong_P$ **Hamiltonian Cycle**

NP Completeness

Clique $\approxeq_P$ **Independent Set** $\approxeq_P$ **Vertex Cover** $\approxeq_P$ **3SAT** $\approxeq_P$ **SAT** $\approxeq_P$ **Hamiltonian Cycle**

All these problems are in **NP**.

SAT is **NPC**.

All these problems are **NP-Complete**.

NP Completeness

Clique $\approxeq_P$ **Independent Set** $\approxeq_P$ **Vertex Cover** $\approxeq_P$ **3SAT** $\approxeq_P$ **SAT** $\approxeq_P$ **Hamiltonian Cycle**

All these problems are in **NP**.

SAT is **NPC**.

All these problems are **NP-Complete**.

NP Completeness

Clique $\approxeq_P$ **Independent Set** $\approxeq_P$ **Vertex Cover** $\approxeq_P$ **3SAT** $\approxeq_P$ **SAT** $\approxeq_P$ **Hamiltonian Cycle**

All these problems are in **NP**.

SAT is **NPC**.

All these problems are **NP-Complete**.

NP Completeness

Clique $\approxeq_P$ **Independent Set** $\approxeq_P$ **Vertex Cover** $\approxeq_P$ **3SAT** $\approxeq_P$ **SAT** $\approxeq_P$ **Hamiltonian Cycle**

All these problems are in **NP**.

SAT is **NPC**.

All these problems are **NP-Complete**.

THE END

...

(for now)

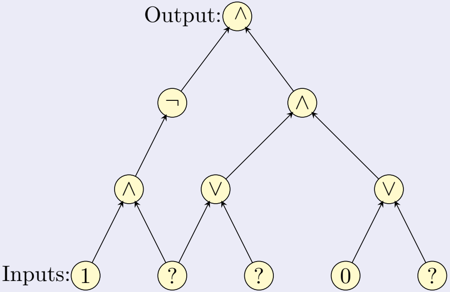
24.2
Circuit SAT

24.2.1
The circuit satisfiability (CSAT) problem

Circuits

Definition 24.1.

A circuit is a directed acyclic graph with



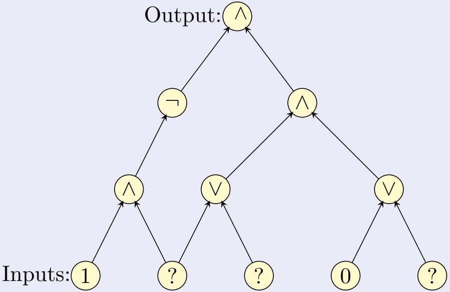
1. Input vertices (without incoming edges) labelled with **0**, **1** or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

Can safely assume every node has at most two incoming edges.

Circuits

Definition 24.1.

A circuit is a directed acyclic graph with



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

Can safely assume every node has at most two incoming edges.

CSAT: Circuit Satisfaction

Definition 24.2 (Circuit Satisfaction (CSAT).).

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

Claim 24.3.

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

Definition 24.2 (Circuit Satisfaction (**CSAT**).).

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

Claim 24.3.

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.

Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

However they are equivalent in terms of polynomial-time solvability.

Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

However they are equivalent in terms of polynomial-time solvability.

Converting a CNF formula into a Circuit

3SAT $\leq_P$ CSAT

Given $3\mathrm{CNF}$ formula φ with n variables and m clauses, create a Circuit C.

- Inputs to C are the n boolean variables $x_1, x_2, \ldots, x_n$
- Use NOT gate to generate literal $\neg x_i$ for each variable x_i
- For each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output

Example

3SAT $\leq_P$ CSAT

$$\varphi = \left(x_1 \vee \vee x_3 \vee x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4\right)$$

Example

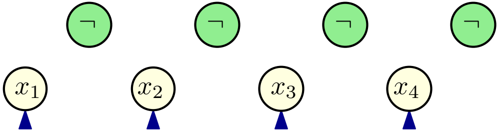
3SAT $\leq_P$ CSAT

$$\varphi = \Big(x_1 \vee \vee x_3 \vee x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \wedge \Big(\neg x_2 \vee \neg x_3 \vee x_4\Big)$$

Example

3SAT $\leq_P$ CSAT

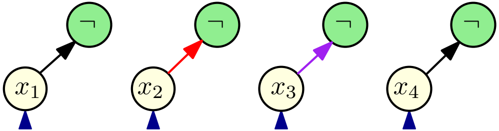
$$\varphi = \Big(x_1 \vee \vee x_3 \vee x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \wedge \Big(\neg x_2 \vee \neg x_3 \vee x_4\Big)$$



Example

3SAT $\leq_P$ CSAT

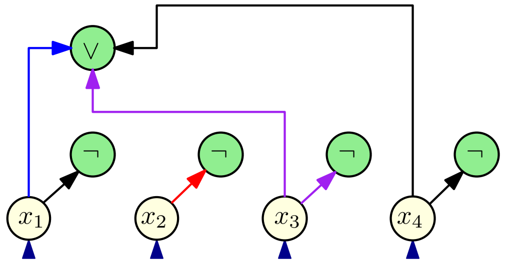
$$\varphi = \left(x_1 \vee \vee x_3 \vee x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4\right)$$



Example

3SAT $\leq_P$ CSAT

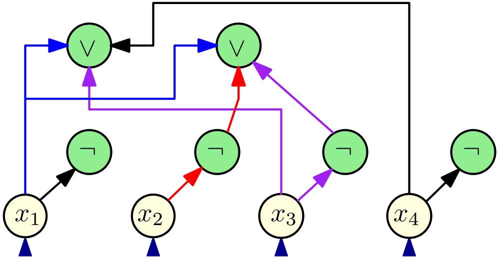
$$\varphi = \Big(x_1 \lor \lor x_3 \lor x_4\Big) \land \Big(x_1 \lor \neg x_2 \lor \neg x_3\Big) \land \Big(\neg x_2 \lor \neg x_3 \lor x_4\Big)$$



Example

3SAT $\leq_P$ CSAT

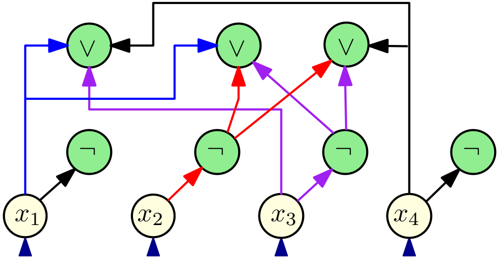
$$\varphi = \left(x_1 \vee \vee x_3 \vee x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4\right)$$



Example

3SAT $\leq_P$ CSAT

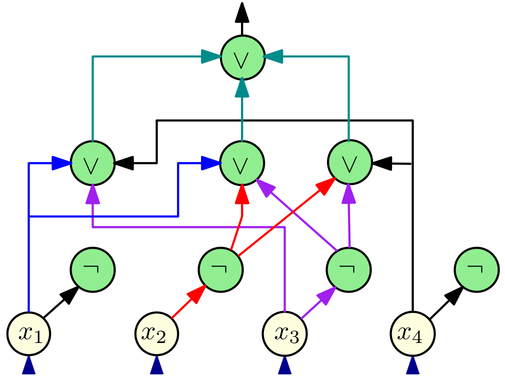
$$\varphi = \Big(x_1 \vee \vee x_3 \vee x_4\Big) \wedge \Big(x_1 \vee \neg x_2 \vee \neg x_3\Big) \wedge \Big(\neg x_2 \vee \neg x_3 \vee x_4\Big)$$



Example

3SAT $\leq_P$ CSAT

$$\varphi = \left(x_1 \vee \vee x_3 \vee x_4\right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3\right) \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4\right)$$



3SAT $\leq_P$ CSAT

Lemma 24.4.

SAT $\leq_P$ *3SAT* $\leq_P$ *CSAT*.

THE END

...

(for now)

24.2.2

Towards reducing CSAT to 3SAT

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y					
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$				
0	0	0	1				
0	0	1	1				
0	1	0	1				
0	1	1	0				
1	0	0	0				
1	0	1	0				
1	1	0	0				
1	1	1	1				

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$				
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}$ *vee*$\overline{y}$			
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	**0**	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}\ vee\overline{y}$	$\overline{z} \vee x \vee y$		
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	**0**	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}$ *vee*$\overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	**0**	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}\ vee\overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	**0**
1	1	1	1	1	1	1	1

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}\ vee\overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Converting $\mathbf{z = x \wedge y}$ to 3SAT

z	x	y	$z = x \wedge y$	$z \vee \overline{x}$ $vee\overline{y}$	$\overline{z} \vee x \vee y$	$\overline{z} \vee x \vee \overline{y}$	$\overline{z} \vee \overline{x} \vee y$
0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

$$\left(z = x \wedge y\right)$$

$$\equiv$$

$$(z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y		
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	0	$\overline{z} \vee x \vee y$
1	1	0	0	$\overline{z} \vee x \vee y$
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

z	x	y	$z = x \wedge y$	clauses
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	0	$\overline{z} \vee x \vee y$
1	1	0	0	$\overline{z} \vee x \vee y$
1	1	1	1	

$$\left(z = x \wedge y\right)$$
$$\equiv$$
$$(z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y) \wedge (\overline{z} \vee x \vee \overline{y}) \wedge (\overline{z} \vee \overline{x} \vee y)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   1.2 $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma 24.5.

$$\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   1.2 $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$
   is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma 24.5.

$$\left(z = x \wedge y\right) \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   1.2 $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$
   is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma 24.5.

$$\left(z = x \wedge y\right) \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   1.2 $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$ is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma 24.5.

$$\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$ to 3SAT

Simplify further if you want to

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(\overline{z} \vee x \vee u) \wedge (\overline{z} \vee x \vee \overline{y}) = (\overline{z} \vee x)$
   1.2 $(\overline{z} \vee x \vee y) \wedge (\overline{z} \vee \overline{x} \vee y) = (\overline{z} \vee y)$
2. Using the above two observation, we have that our formula $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x \vee y\right) \wedge \left(\overline{z} \vee x \vee \overline{y}\right) \wedge \left(\overline{z} \vee \overline{x} \vee y\right)$
   is equivalent to $\psi \equiv \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

Lemma 24.5.

$$\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$$

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

z	x	y		
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

z	x	y	$z = x \vee y$	
0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	$z \vee x \vee \overline{y}$
0	1	0	0	$z \vee \overline{x} \vee y$
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	1	
1	1	0	1	
1	1	1	1	

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

z	x	y	$z = x \vee y$	clauses
0	0	0	1	
0	0	1	0	$z \vee x \vee \overline{y}$
0	1	0	0	$z \vee \overline{x} \vee y$
0	1	1	0	$z \vee \overline{x} \vee \overline{y}$
1	0	0	0	$\overline{z} \vee x \vee y$
1	0	1	1	
1	1	0	1	
1	1	1	1	

$$\Big(z = x \vee y\Big)$$
$$\equiv$$
$$(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   1.2 $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma 24.6.

The formula $z = x \vee y$ is equivalent to the CNF formula

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $\mathbf{z = x \vee y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
    1.1 $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
    1.2 $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma 24.6.

The formula $z = x \vee y$ is equivalent to the CNF *formula*

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $\mathbf{z = x \lor y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \lor y\right) \equiv (z \lor x \lor \overline{y}) \land (z \lor \overline{x} \lor y) \land (z \lor \overline{x} \lor \overline{y}) \land (\overline{z} \lor x \lor y)$$

1. Using that $(x \lor y) \land (x \lor \overline{y}) = x$, we have that:
   1.1 $(z \lor x \lor \overline{y}) \land (z \lor \overline{x} \lor \overline{y}) = z \lor \overline{y}$.
   1.2 $(z \lor \overline{x} \lor y) \land (z \lor \overline{x} \lor \overline{y}) = z \lor \overline{x}$
2. Using the above two observation, we have the following.

Lemma 24.6.

The formula $z = x \lor y$ is equivalent to the CNF formula

$$\left(z = x \lor y\right) \equiv (z \lor \overline{y}) \land (z \lor \overline{x}) \land (\overline{z} \lor x \lor y)$$

Converting $\mathbf{z} = \mathbf{x} \vee \mathbf{y}$ to 3SAT

Simplify further if you want to

$$\left(z = x \vee y\right) \equiv (z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x \vee y)$$

1. Using that $(x \vee y) \wedge (x \vee \overline{y}) = x$, we have that:
   1.1 $(z \vee x \vee \overline{y}) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{y}$.
   1.2 $(z \vee \overline{x} \vee y) \wedge (z \vee \overline{x} \vee \overline{y}) = z \vee \overline{x}$
2. Using the above two observation, we have the following.

Lemma 24.6.

The formula $z = x \vee y$ *is equivalent to the* CNF *formula*

$$\left(z = x \vee y\right) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$$

Converting $z = \overline{x}$ to CNF

Lemma 24.7.

$$z = \overline{x} \qquad \equiv \qquad (z \vee x) \wedge (\overline{z} \vee \overline{x}).$$

Summary of formulas we derived

Lemma 24.8.

The following identities hold:

1. $z = \overline{x} \quad \equiv \quad (z \vee x) \wedge (\overline{z} \vee \overline{x})$.
2. $\left(z = x \vee y\right) \quad \equiv \quad (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$
3. $\left(z = x \wedge y\right) \quad \equiv \quad \left(z \vee \overline{x} \vee \overline{y}\right) \wedge \left(\overline{z} \vee x\right) \wedge \left(\overline{z} \vee y\right)$

THE END

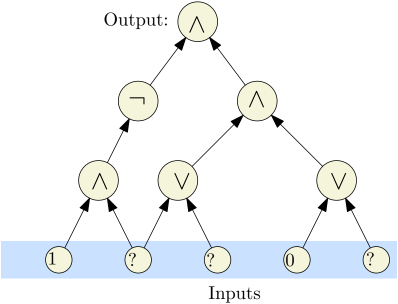
...

(for now)

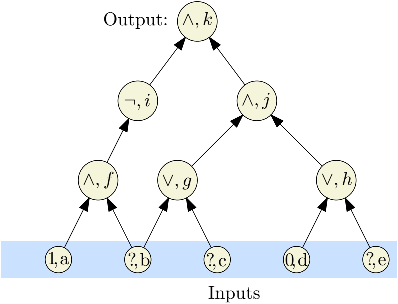
24.2.3
Reduction from CSAT to SAT

Converting a circuit into a CNF formula

Label the nodes



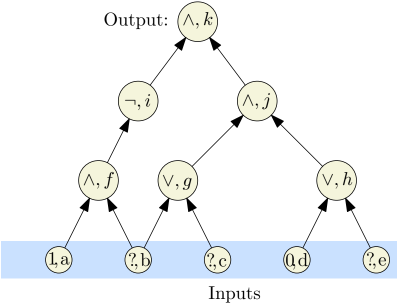
(A) Input circuit



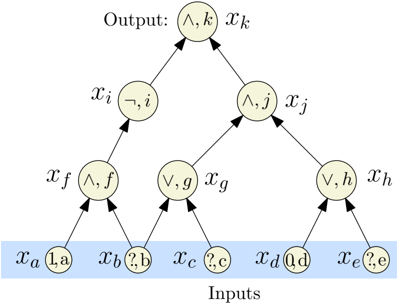
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node



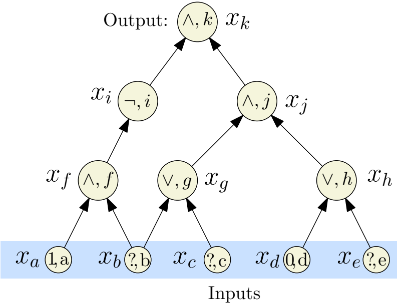
(B) Label the nodes.



(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.

$\boldsymbol{x_k}$ (Demand a sat' assignment!)

$\boldsymbol{x_k = x_i \wedge x_j}$

$\boldsymbol{x_j = x_g \wedge x_h}$

$\boldsymbol{x_i = \neg x_f}$

$\boldsymbol{x_h = x_d \vee x_e}$

$\boldsymbol{x_g = x_b \vee x_c}$

$\boldsymbol{x_f = x_a \wedge x_b}$

$\boldsymbol{x_d = 0}$

$\boldsymbol{x_a = 1}$

(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

From **Lemma 24.8** :

1. $z = \overline{x} \quad \equiv \quad (z \vee x) \wedge (\overline{z} \vee \overline{x})$
2. $(z = x \vee y) \quad \equiv \quad (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$
3. $(z = x \wedge y) \quad \equiv \quad (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x) \wedge (\overline{z} \vee y)$

Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula

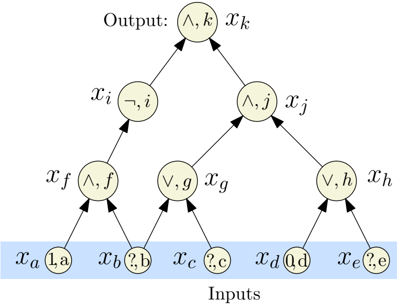
x_k	x_k
$x_k = x_i \wedge x_j$	$(\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \wedge (x_k \vee \neg x_i \vee \neg x_j)$
$x_j = x_g \wedge x_h$	$(\neg x_j \vee x_g) \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h)$
$x_i = \neg x_f$	$(x_i \vee x_f) \wedge (\neg x_i \vee \neg x_f)$
$x_h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

From **Lemma 24.8** :

1. $z = \overline{x} \equiv (z \vee x) \wedge (\overline{z} \vee \overline{x})$
2. $(z = x \vee y) \equiv (z \vee \overline{y}) \wedge (z \vee \overline{x}) \wedge (\overline{z} \vee x \vee y)$
3. $(z = x \wedge y) \equiv (z \vee \overline{x} \vee \overline{y}) \wedge (\overline{z} \vee x) \wedge (\overline{z} \vee y)$

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 if and only if the original circuit is satisfiable.

Correctness of Reduction

Need to show circuit $\boldsymbol{C}$ is satisfiable if and only if $\varphi_{\boldsymbol{C}}$ is satisfiable

$\Rightarrow$ Consider a satisfying assignment $\boldsymbol{a}$ for $\boldsymbol{C}$

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

$\Leftarrow$ Consider a satisfying assignment $\boldsymbol{a}$ for $\varphi_{\boldsymbol{C}}$

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

The result

Lemma 24.9.

CSAT $\le_P$ ***SAT*** $\le_P$ ***3SAT***.

Theorem 24.10.

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

The result

Lemma 24.9.

CSAT $\leq_P$ ***SAT*** $\leq_P$ ***3SAT***.

Theorem 24.10.

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

THE END

...

(for now)

24.3
NP-Completeness of Graph Coloring

24.3.1
The coloring problem

Graph Coloring

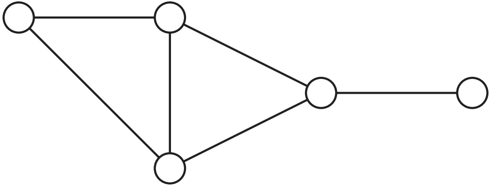
Problem: Graph Coloring

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?

Graph 3-Coloring

Problem: 3 Coloring

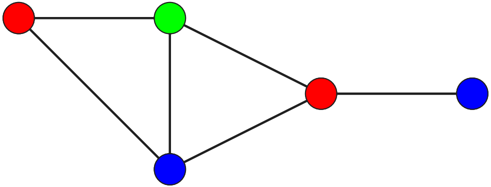
Instance: $G = (V, E)$: Undirected graph.
Question: Can the vertices of the graph be colored using **3** colors so that vertices connected by an edge do not get the same color?



Graph 3-Coloring

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?



Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets $\iff$ G is k-colorable.
3. Graph 2-Coloring can be decided in polynomial time.
4. G is 2-colorable $\iff$ G is bipartite.
5. There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets $\iff$ G is k-colorable.
3. Graph 2-Coloring can be decided in polynomial time.
4. G is 2-colorable $\iff$ G is bipartite.
5. There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets $\iff$ G is k-colorable.
3. Graph $\mathbf{2}$-Coloring can be decided in polynomial time.
4. G is $\mathbf{2}$-colorable $\iff$ G is bipartite.
5. There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. Observation: If G is colored with $\boldsymbol{k}$ colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into $\boldsymbol{k}$ independent sets $\iff$ G is $\boldsymbol{k}$-colorable.
3. Graph $\mathbf{2}$-Coloring can be decided in polynomial time.
4. G is $\mathbf{2}$-colorable $\iff$ G is bipartite.
5. There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets $\iff$ G is k-colorable.
3. Graph **2**-Coloring can be decided in polynomial time.
4. G is **2**-colorable $\iff$ G is bipartite.
5. There is a linear time algorithm to check if G is bipartite using **BFS** (we saw this earlier).

THE END

...

(for now)

24.3.2
Problems related to graph coloring

Register allocation during compilation

1. When a compiler generates the assembly/VM code it needs to allocation registers to values being handled.
2. Need to make sure registers are not in conflict.
3. Build a conflict graph.
4. Color the conflict graph.
5. Every color is a register.
6. If not enough registers, then use memory/stack to store values.
7. CISC v.s. RISC.

Register allocation during compilation

1. When a compiler generates the assembly/VM code it needs to allocation registers to values being handled.
2. Need to make sure registers are not in conflict.
3. Build a conflict graph.
4. Color the conflict graph.
5. Every color is a register.
6. If not enough registers, then use memory/stack to store values.
7. CISC v.s. RISC.

Graph Coloring and Register Allocation

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

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. Create graph G
   - a node v_i for each class i
   - an edge between v_i and v_j if classes i and j conflict
4. Exercise: G is k-colorable $\iff$ k rooms are sufficient.

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. Create graph G
   - a node v_i for each class i
   - an edge between v_i and v_j if classes i and j conflict
4. Exercise: G is k-colorable $\iff$ k rooms are sufficient.

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. Create graph G
   - a node v_i for each class i
   - an edge between v_i and v_j if classes i and j conflict
4. Exercise: G is k-colorable $\iff$ k rooms are sufficient.

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. Create graph G
   - a node v_i for each class i
   - an edge between v_i and v_j if classes i and j <u>conflict</u>
4. Exercise: G is k-colorable $\iff$ k rooms are sufficient.

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

THE END

...

(for now)

24.3.3

Showing NP-Completeness of 3 COLORING

24.3.3.1
The variable assignment gadget

3-Coloring is **NP-Complete**

- **3-Coloring** is in **NP**.
  - Certificate: for each node a color from $\{1, 2, 3\}$.
  - Certifier: Check if for each edge (u, v), the color of u is different from that of v.
- Hardness: We will show **3-SAT** $\leq_P$ **3-Coloring**.

Reduction idea

1. φ: Given **3SAT** formula (i.e., **3**CNF formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Reduction idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Reduction idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Reduction idea

1. φ: Given **3SAT** formula (i.e., **3**CNF formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Reduction idea

1. φ: Given **3SAT** formula (i.e., **3**CNF formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

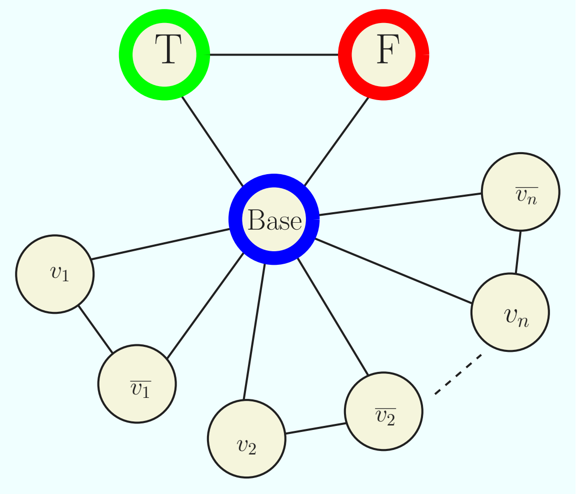
Reduction idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Reduction idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base
   - for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
   - If graph is 3-colored, either v_i or $\bar{v}_i$ 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)

Assignment encoding using **3**-coloring



THE END

...

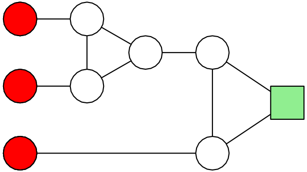
(for now)

24.3.3.2
The clause gadget

3 color this gadget.

Clicker question

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



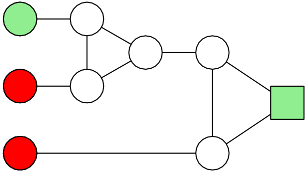
(A) Yes.

(B) No.

3 color this gadget II

Clicker question

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



(A) Yes.

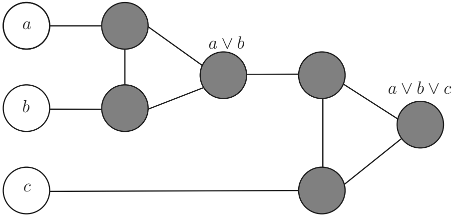
(B) No.

Clause Satisfiability Gadget

1. For each clause $C_j = (a \vee b \vee c)$, create a small gadget graph
   - gadget graph connects to nodes corresponding to a, b, c
   - needs to implement OR
2. OR-gadget-graph:

Clause Satisfiability Gadget

1. 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
2. OR-gadget-graph:



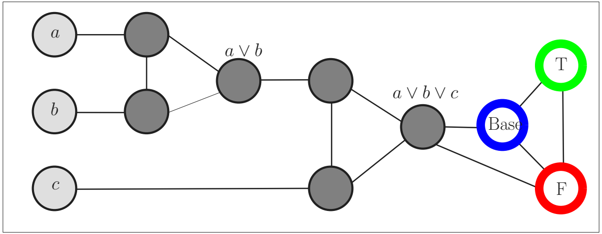
OR-Gadget Graph

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

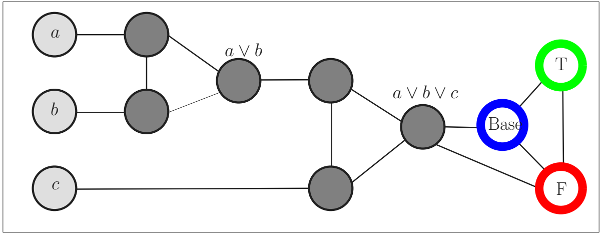
Property: if one of a, b, c is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

Reduction

- create triangle with nodes True, False, Base
- for each variable $\boldsymbol{x_i}$ two nodes $\boldsymbol{v_i}$ and $\boldsymbol{\bar{v}_i}$ connected in a triangle with common Base
- for each clause $\boldsymbol{C_j} = (\boldsymbol{a} \vee \boldsymbol{b} \vee \boldsymbol{c})$, add OR-gadget graph with input nodes $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and connect output node of gadget to both False and Base



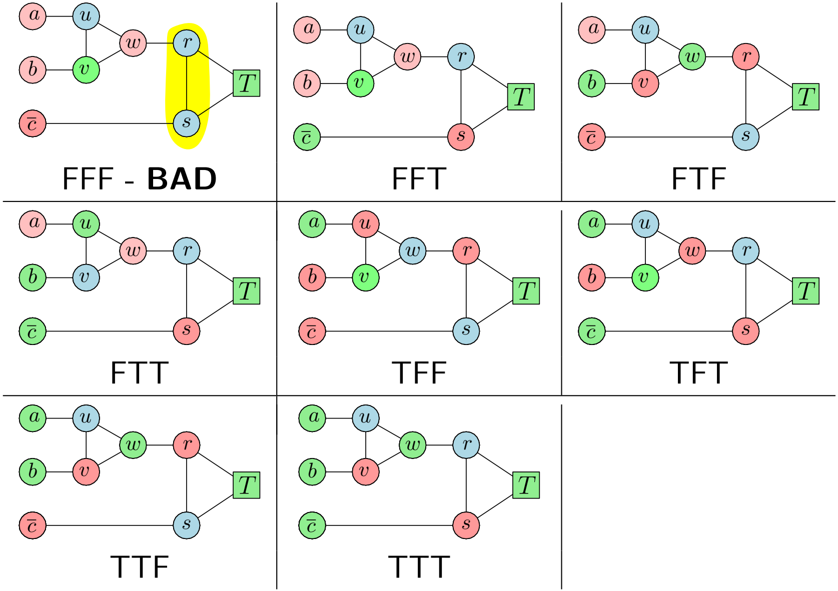
Reduction



Claim 24.1.

*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.*

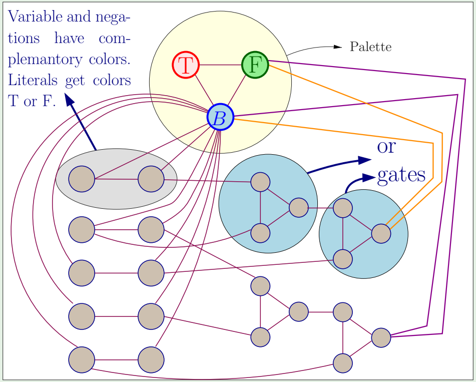
3 coloring of the clause gadget



Reduction Outline

Example 24.2.

$\varphi = (u \lor \neg v \lor w) \land (v \lor x \lor \neg y)$



Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ 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_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment
- consider any clause $C_j = (a \vee b \vee c)$. 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!

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ 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_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment
- consider any clause $C_j = (a \vee b \vee c)$. 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!

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ 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_φ is 3-colorable implies φ 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!

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ 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_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment
- consider any clause $C_j = (a \vee b \vee c)$. 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!

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

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

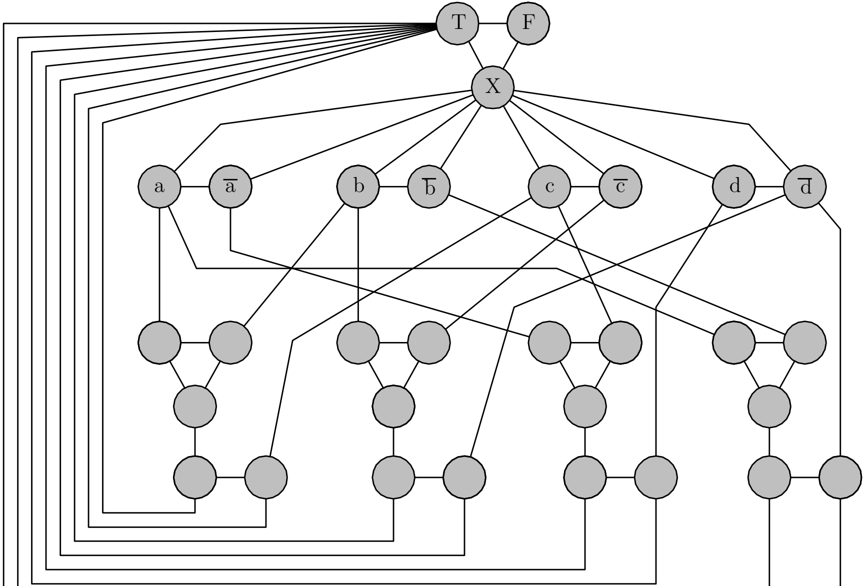
G_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment
- consider any clause $C_j = (a \vee b \vee c)$. 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

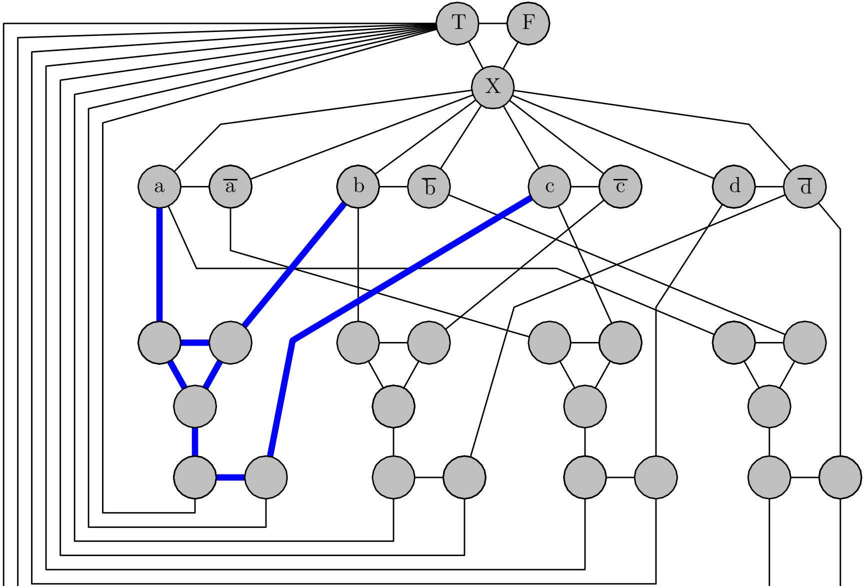
$$(a \vee b \vee c) \wedge (b \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b} \vee \overline{d})$$



Graph generated in reduction...

... from 3SAT to 3COLOR

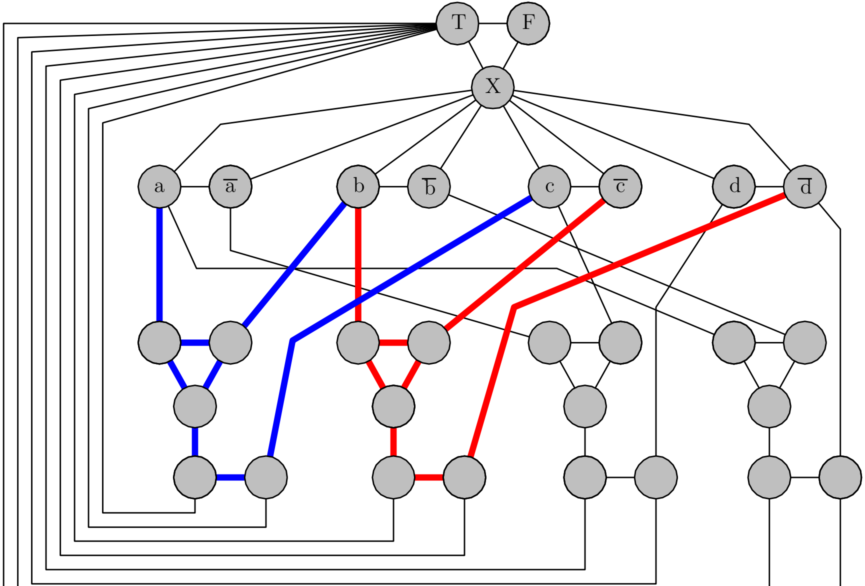
$$(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})$$



Graph generated in reduction...

... from 3SAT to 3COLOR

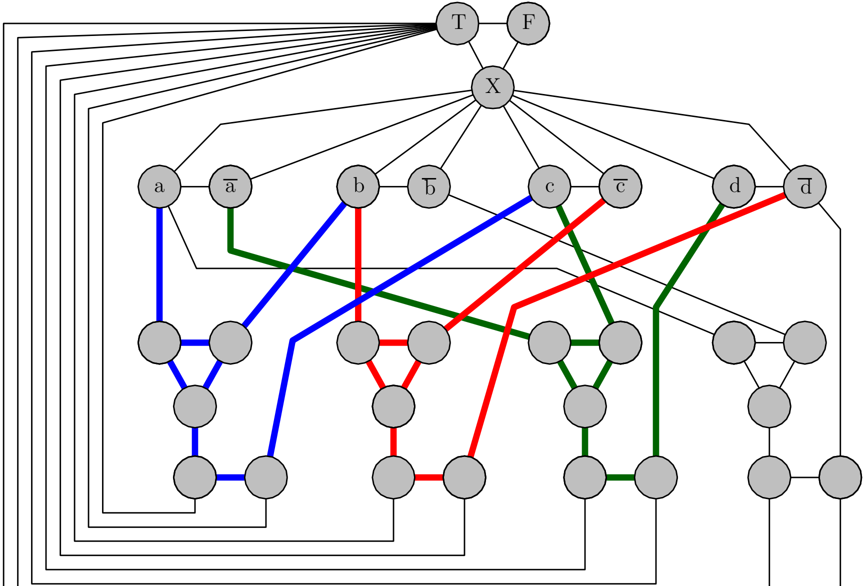
$$(a \vee b \vee c) \wedge (b \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b} \vee \overline{d})$$



Graph generated in reduction...

... from 3SAT to 3COLOR

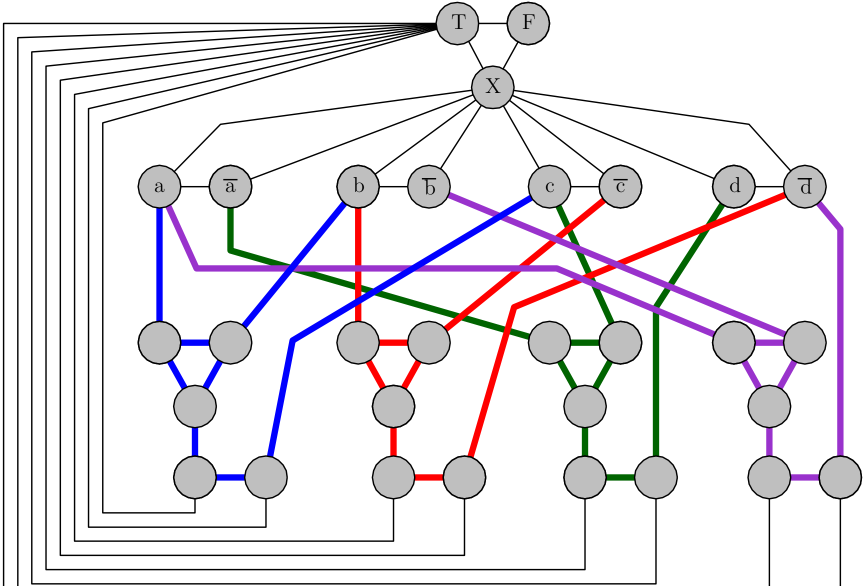
$$(a \lor b \lor c) \land (b \lor \bar{c} \lor \bar{d}) \land (\bar{a} \lor c \lor d) \land (a \lor \bar{b} \lor \bar{d})$$



Graph generated in reduction...

... from 3SAT to 3COLOR

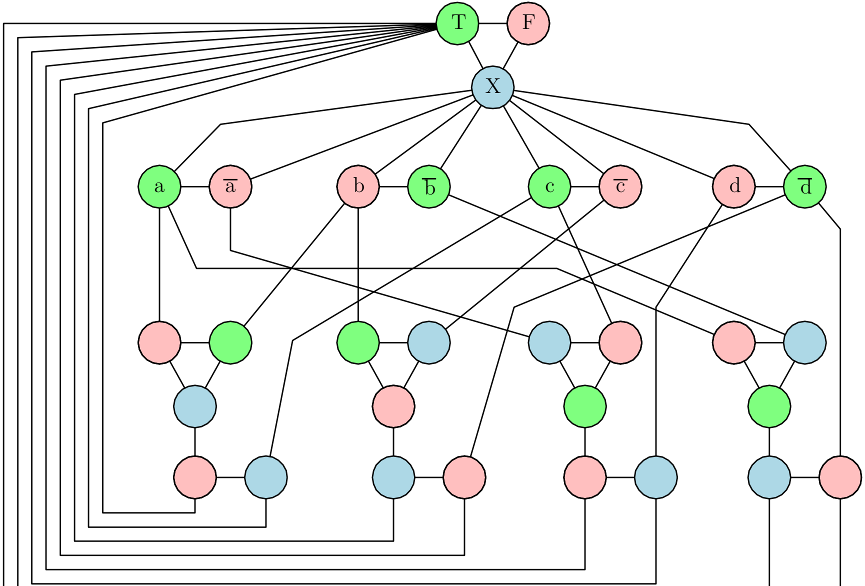
$$(a \vee b \vee c) \wedge (b \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee d) \wedge (a \vee \overline{b} \vee \overline{d})$$



Graph generated in reduction...

... from 3SAT to 3COLOR

$$(a \lor b \lor c) \land (b \lor \overline{c} \lor \overline{d}) \land (\overline{a} \lor c \lor d) \land (a \lor \overline{b} \lor \overline{d})$$



THE END

…

(for now)

24.4
Proof of Cook-Levin Theorem

24.4.1
Statement and sketch of idea for the proof

Cook-Levin Theorem

Theorem 24.1 (Cook-Levin).

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

We have already seen that **SAT** is in **NP**.

Need to prove that every language $L \in$ **NP**, $L \leq_P$ **SAT**

Difficulty: Infinite number of languages in **NP**. Must simultaneously show a generic reduction strategy.

Cook-Levin Theorem

Theorem 24.1 (Cook-Levin).

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

We have already seen that **SAT** is in **NP**.

Need to prove that every language $\boldsymbol{L} \in$ **NP**, $\boldsymbol{L} \leq_{\boldsymbol{P}}$ **SAT**

Difficulty: Infinite number of languages in **NP**. Must simultaneously show a generic reduction strategy.

The plot against SAT

High-level plan to proving the Cook-Levin theorem

What does it mean that $L \in \mathbf{NP}$?

$L \in NP$ implies that there is a non-deterministic TM M and polynomial $p()$ such that

$$L = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ in at most } p(|x|) \text{ steps}\}$$

Input: M, x, p.
Question: Does M stops on input x after $p(|x|)$ steps?

Describe a reduction R that computes from M, x, p a **SAT** formula φ.

- R takes as input a string x and outputs a SAT formula φ
- R runs in time polynomial in $|x|, |M|$
- $x \in L$ if and only if φ is satisfiable

The plot against SAT

High-level plan to proving the Cook-Levin theorem

What does it mean that $L \in \mathbf{NP}$?

$L \in NP$ implies that there is a non-deterministic TM M and polynomial $p()$ such that

$$L = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ in at most } p(|x|) \text{ steps}\}$$

Input: M, x, p.
Question: Does M stops on input x after $p(|x|)$ steps?

Describe a reduction R that computes from M, x, p a **SAT** formula φ.

- R takes as input a string x and outputs a SAT formula φ
- R runs in time polynomial in $|x|, |M|$
- $x \in L$ if and only if φ is satisfiable

The plot against SAT

High-level plan to proving the Cook-Levin theorem

What does it mean that $L \in$ **NP**?

$L \in NP$ implies that there is a non-deterministic TM M and polynomial $p()$ such that

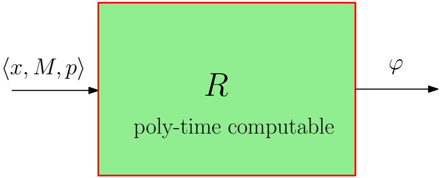
$$L = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ in at most } p(|x|) \text{ steps}\}$$

Input: M, x, p.
Question: Does M stops on input x after $p(|x|)$ steps?

Describe a reduction R that computes from M, x, p a **SAT** formula φ.

- R takes as input a string x and outputs a SAT formula φ
- R runs in time polynomial in $|x|, |M|$
- $x \in L$ if and only if φ is satisfiable

The plot against SAT continued



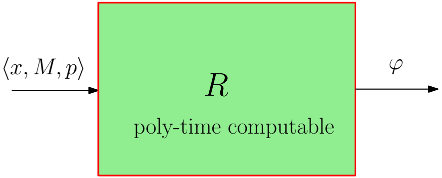
φ is satisfiable if and only if $x \in L$
φ is satisfiable if and only if nondeterministic M accepts x in $p(|x|)$ steps

BIG IDEA

- φ will express "M on input x accepts in $p(|x|)$ steps"
- φ will encode a computation history of M on x

φ: CNF formula s.t if we have a satisfying assignment to it $\implies$ accepting computation of M on x down to the last details (where the head is, what transition is chosen, what the tape contents are, at each step, etc).

The plot against SAT continued



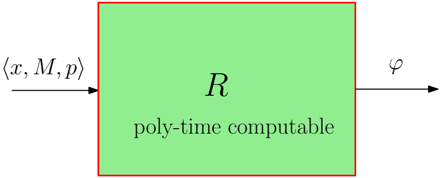
φ is satisfiable if and only if $x \in L$
φ is satisfiable if and only if nondeterministic M accepts x in $p(|x|)$ steps

BIG IDEA

- φ will express "M on input x accepts in $p(|x|)$ steps"
- φ will encode a computation history of M on x

φ: CNF formula s.t if we have a satisfying assignment to it $\implies$ accepting computation of M on x down to the last details (where the head is, what transition is chosen, what the tape contents are, at each step, etc).

The plot against SAT continued



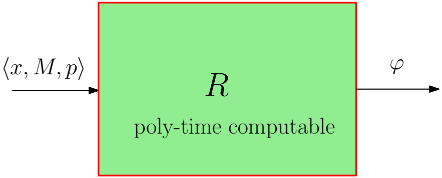
φ is satisfiable if and only if $x \in L$
φ is satisfiable if and only if nondeterministic M accepts x in $p(|x|)$ steps

BIG IDEA

- φ will express "M on input x accepts in $p(|x|)$ steps"
- φ will encode a computation history of M on x

φ: CNF formula s.t if we have a satisfying assignment to it $\implies$ accepting computation of M on x down to the last details (where the head is, what transition is chosen, what the tape contents are, at each step, etc).

The plot against SAT continued



φ is satisfiable if and only if $x \in L$

φ is satisfiable if and only if nondeterministic M accepts x in $p(|x|)$ steps

BIG IDEA

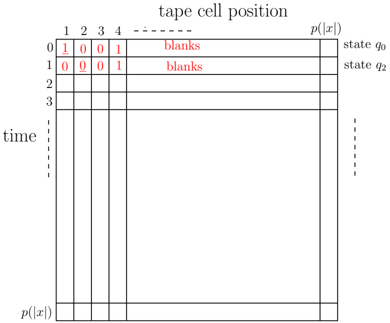
- φ will express "M on input x accepts in $p(|x|)$ steps"
- φ will encode a computation history of M on x

φ: CNF formula s.t if we have a satisfying assignment to it $\implies$ accepting computation of M on x down to the last details (where the head is, what transition is chosen, what the tape contents are, at each step, etc).

The Matrix Executions

Tableau of Computation

M runs in time ***p***(|***x***|) on ***x***. Entire computation of ***M*** on ***x*** can be represented by a "tableau"



Row ***i*** gives contents of all cells at time ***i***
At time **0** tape has input ***x*** followed by blanks
Each row long enough to hold all cells ***M*** might ever have scanned.

Variables of φ

Four types of variables to describe computation of M on x

- $T(b, h, i)$: tape cell at position h holds symbol b at time i.
  For $h = 1, \ldots, p(|x|)$, $b \in \Gamma$, $i = 0, \ldots, p(|x|)$.
- $H(h, i)$: read/write head is at position h at time i.
  Fir $h = 1, \ldots, p(|x|)$, and $i = 0, \ldots, p(|x|)$
- $S(q, i)$ state of M is q at time i.
  For all $q \in Q$ and $i = 0, \ldots, p(|x|)$.
- $I(j, i)$ instruction number j is executed at time i
  M is non-deterministic, need to specify transitions in some way. Number transitions as $1, 2, \ldots, \ell$ where jth transition is $< q_j, b_j, q_j', b_j', d_j >$ indication $(q_j', b_j', d_j) \in \delta(q_j, b_j)$, direction $d_j \in \{-1, 0, 1\}$.

Number of variables is $O(p(|x|)^2|M|^2)$

Notation

Some abbreviations for ease of notation

$\bigwedge_{k=1}^{m} x_k$ means $x_1 \wedge x_2 \wedge \ldots \wedge x_m$

$\bigvee_{k=1}^{m} x_k$ means $x_1 \vee x_2 \vee \ldots \vee x_m$

$\bigoplus(x_1, x_2, \ldots, x_k)$ is a formula that means **exactly one** of $x_1, x_2, \ldots, x_m$ is true. Can be converted to CNF form

---

CNF formula showing making sure that at most one variable is assigned value $\mathbf{1}$:

$$\bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j})$$

Making sure that one of the variables is true: $\bigvee_{i=1}^{k} x_i$.

$$\bigoplus(x_1, x_2, \ldots, x_k) = \bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j}) \bigwedge (x_1 \vee x_2 \vee \cdots \vee x_k).$$

Notation

Some abbreviations for ease of notation

$\bigwedge_{k=1}^{m} x_k$ means $x_1 \wedge x_2 \wedge \ldots \wedge x_m$

$\bigvee_{k=1}^{m} x_k$ means $x_1 \vee x_2 \vee \ldots \vee x_m$

$\bigoplus(x_1, x_2, \ldots, x_k)$ is a formula that means **exactly one** of $x_1, x_2, \ldots, x_m$ is true. Can be converted to CNF form

---

CNF formula showing making sure that at most one variable is assigned value $\mathbf{1}$:

$$\bigwedge_{1 \le i < j \le k} \left(\overline{x_i} \vee \overline{x_j}\right)$$

Making sure that one of the variables is true: $\bigvee_{i=1}^{k} x_i$.

$$\bigoplus(x_1, x_2, \ldots, x_k) = \bigwedge_{1 \le i < j \le k} \left(\overline{x_i} \vee \overline{x_j}\right) \bigwedge \left(x_1 \vee x_2 \vee \cdots \vee x_k\right).$$

Notation

Some abbreviations for ease of notation

$\bigwedge_{k=1}^{m} x_k$ means $x_1 \wedge x_2 \wedge \ldots \wedge x_m$

$\bigvee_{k=1}^{m} x_k$ means $x_1 \vee x_2 \vee \ldots \vee x_m$

$\bigoplus(x_1, x_2, \ldots, x_k)$ is a formula that means **exactly one** of $x_1, x_2, \ldots, x_m$ is true. Can be converted to CNF form

---

CNF formula showing making sure that at most one variable is assigned value $\mathbf{1}$:

$$\bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j})$$

Making sure that one of the variables is true: $\bigvee_{i=1}^{k} x_i$.

$$\bigoplus(x_1, x_2, \ldots, x_k) = \bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j}) \bigwedge (x_1 \vee x_2 \vee \cdots \vee x_k).$$

Notation

Some abbreviations for ease of notation

$\bigwedge_{k=1}^{m} x_k$ means $x_1 \wedge x_2 \wedge \ldots \wedge x_m$

$\bigvee_{k=1}^{m} x_k$ means $x_1 \vee x_2 \vee \ldots \vee x_m$

$\bigoplus(x_1, x_2, \ldots, x_k)$ is a formula that means **exactly one** of $x_1, x_2, \ldots, x_m$ is true. Can be converted to CNF form

---

CNF formula showing making sure that at most one variable is assigned value $\mathbf{1}$:

$$\bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j})$$

Making sure that one of the variables is true: $\bigvee_{i=1}^{k} x_i$.

$$\bigoplus(x_1, x_2, \ldots, x_k) = \bigwedge_{1 \le i < j \le k} (\overline{x_i} \vee \overline{x_j}) \bigwedge (x_1 \vee x_2 \vee \cdots \vee x_k).$$

Clauses of φ

φ is the conjunction of **8** clause groups:

$$\varphi = \bigwedge_{i=1}^{12} \varphi_i$$

where each φ_i is a CNF formula. Described in subsequent slides.

Property: φ is satisfied $\iff$ there is an execution of M on x that accepts the language in $p(|x|)$ time.

THE END

...

(for now)

24.4.2
The consistency of execution

The variables of φ

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j, i)$: Instruction j was issued at time i.
$H(h, i)$: The head is at location h at time i.
$T(c, h, i)$: The tape at location h at time i stored the character c.

φ_1: The input is encoded correctly

φ_1 asserts (is true iff) the variables are set T/F indicating that M starts in state q_0 at time 0 with tape contents containing x followed by blanks. Let $x = x_1 x_2 \ldots x_n$

$$
\begin{aligned}
\varphi_1 = \quad & S(q_0, 0) && \texttt{// state at time } 0 \texttt{ is } q_0 \\
& \bigwedge_{h=1}^{n} T(x_h, h, 0) && \texttt{// at time } 0 \texttt{ cells } 1 \texttt{ to } n \texttt{ have value } x_1 \texttt{ to } x_n \\
& \wedge \bigwedge_{h=n+1}^{p(n)} T(\sqcup, h, 0) && \texttt{// all remaining cells are blank} \\
& \wedge H(1, 0) && \texttt{// The head is at time } 0 \texttt{ at start of tape}
\end{aligned}
$$

φ_2: M is in exactly one state at any point in time

φ_2 asserts M in exactly one state at any time i:

$$\varphi_2 = \bigwedge_{i=0}^{p(|x|)} \Big(\oplus\big(S(q_0, i), S(q_1, i), \ldots, S(q_{|Q|}, i)\big) \Big)$$

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j, i)$: Instruction j was issued at time i.
$H(h, i)$: The head is at location h at time i.
$T(c, h, i)$: The tape at location h at time i stored the character c.

φ_3: Each tape cell holds a unique symbol at any time

φ_3 asserts that each tape cell holds a unique symbol at any given time.

$$\varphi_3 = \bigwedge_{i=0}^{p(|x|)} \bigwedge_{h=1}^{p(|x|)} \oplus(T(b_1, h, i), T(b_2, h, i), \ldots, T(b_{|\Gamma|}, h, i))$$

For each time i and for each cell position h exactly one symbol $b \in \Gamma$ at cell position h at time i

---

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j, i)$: Instruction j was issued at time i.
$H(h, i)$: The head is at location h at time i.
$T(c, h, i)$: The tape at location h at time i stored the character c.

φ_4: tape head of M is in exactly one position at any time i

φ_4 asserts that the read/write head of M is in exactly one position at any time i

$$\varphi_4 = \bigwedge_{i=0}^{p(|x|)} \left(\bigoplus \left(H(1, i), H(2, i), \ldots, H(p(|x|), i)\right)\right)$$

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j, i)$: Instruction j was issued at time i.
$H(h, i)$: The head is at location h at time i.
$T(c, h, i)$: The tape at location h at time i stored the character c.

φ_5: M accepts the input

φ_5 asserts that M accepts

- Let q_a be unique accept state of M
- without loss of generality assume M runs all $p(|x|)$ steps

$$\varphi_5 = S(q_a, p(|x|))$$

State at time $p(|x|)$ is q_a the accept state.

If we don't want to make assumption of running for all steps

$$\varphi_5 = \bigvee_{i=1}^{p(|x|)} S(q_a, i)$$

which means M enters accepts state at some time.

φ_6: M executes a unique instruction at each time

φ_6 asserts that M executes a unique instruction at each time

$$\varphi_6 = \bigwedge_{i=0}^{p(|x|)} \oplus(I(1,i), I(2,i), \ldots, I(m,i))$$

where m is max instruction number.

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

φ_7: Tape changes only because of the head writing something

φ_7 ensures that variables don't allow tape to change from one moment to next if the read/write head was not there.

"If head is **not** at position h at time i then at time $i+1$ the symbol at cell h must be unchanged"

$$\varphi_7 = \bigwedge_i \bigwedge_h \bigwedge_{b \neq c} \left(\overline{H(h,i)} \Rightarrow \overline{T(b,h,i) \bigwedge T(c,h,i+1)} \right)$$

since $A \Rightarrow B$ is same as $\neg A \vee B$, rewrite above in CNF form

$$\varphi_7 = \bigwedge_i \bigwedge_h \bigwedge_{b \neq c} (H(h,i) \vee \neg T(b,h,i) \vee \neg T(c,h,i+1))$$

φ_8: Transitions are done from correct states

jth instruction of M: $< q_j, b_j, q'_j, b'_j, d_j >$

$$\varphi_8 = \bigwedge_i \bigwedge_j (I(j,i) \Rightarrow S(q_j, i))$$

If instruction j is executed at time i then state at time i must be q_j.

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

φ_9: Transitions are done into correct state

jth instruction of M: $< q_j, b_j, q'_j, b'_j, d_j >$

$$\varphi_9 = \bigwedge_i \bigwedge_j (I(j,i) \Rightarrow S(q'_j, i+1))$$

If instruction j was performed at time i, then state at time $i+1$ must be q'_j.

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

φ_{10}: The character written on tape that triggered an instruction, is the correct one

$$\varphi_{10} = \bigwedge_i \bigwedge_h \bigwedge_j [(I(j,i) \bigwedge H(h,i)) \Rightarrow T(b_j, h, i)]$$

If instruction j was executed at time i and head was at position h, then cell h has the symbol needed to issue instruction j is written under the head location on the tape.

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

φ_{11}: The correct symbol was written to the tape at time i

$$\varphi_{11} = \bigwedge_i \bigwedge_j \bigwedge_h [(I(j,i) \wedge H(h,i)) \Rightarrow T(b'_j, h, i+1)]$$

If instruction j was executed time i with head at h, then at next time step symbol b'_j was written in position h

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

φ_{12}: Head was moved in the right direction at time i

$$\varphi_{12} = \bigwedge_i \bigwedge_j \bigwedge_h [(I(j,i) \wedge H(h,i)) \Rightarrow H(h+d_j, i+1)]$$

The head is moved properly according to instr j.

Variables:

$\left\langle q_j, b_j, q'_j, b'_j, d_j \right\rangle$: jth instruction of M
$I(j,i)$: Instruction j was issued at time i.
$H(h,i)$: The head is at location h at time i.
$T(c,h,i)$: The tape at location h at time i stored the character c.

THE END

...

(for now)

24.4.3
Proof of correctness

Proof of Correctness

(Sketch)

- Given M, x, poly-time algorithm to construct φ
- if φ is satisfiable then the truth assignment completely specifies an accepting computation of M on x
- if M accepts x then the accepting computation leads to an "obvious" truth assignment to φ. Simply assign the variables according to the state of M and cells at each time i.

Thus M accepts x if and only if φ is satisfiable

THE END

...

(for now)

24.5
NP-Complete problems to know and remember

List of NP-Complete Problems to Remember

Problems

1. **SAT**
2. **3SAT**
3. **CircuitSAT**
4. **Independent Set**
5. **Clique**
6. **Vertex Cover**
7. **Hamilton Cycle** and **Hamilton Path** in both directed and undirected graphs
8. **3Color** and **Color**