NP-Hardness reductions

- Definition: P is the class of problems that can be solved in polynomial time, that is n^c for a constant c

- Roughly, if a problem is in P then it's easy, and if it's not in P then it's hard.

- We'd like to show that many natural problems are not in P. We do not know how to do that. However, we can link the hardness of the problems.

- Next: Define several problems:

  SAT, CLIQUE, SUBSET-SUM, ...

- Prove polynomial-time reductions:

  CLIQUE $\in$ P ⇨ SAT $\in$ P

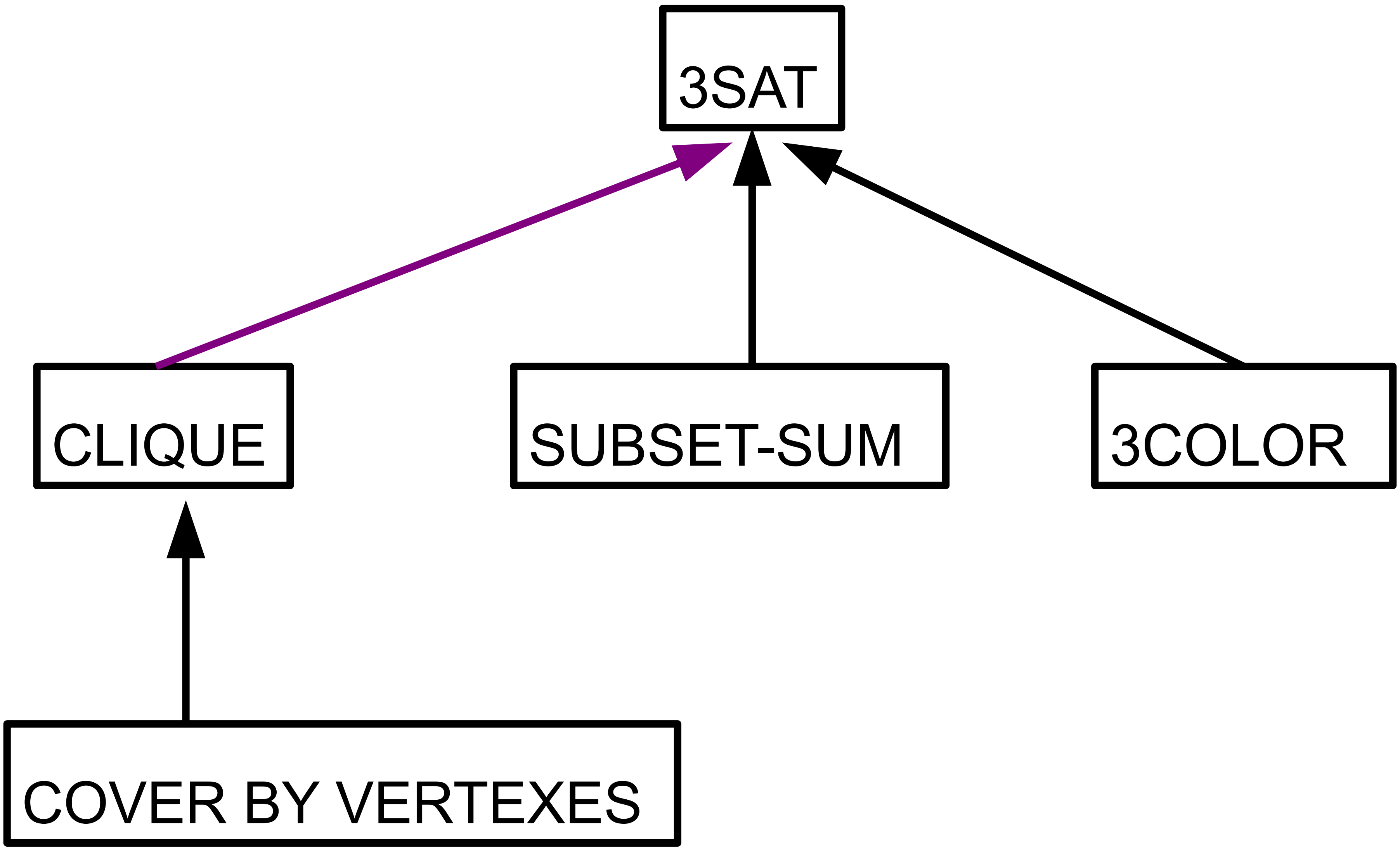
  SUBSET-SUM $\in$ P ⇨ SAT $\in$ P

- Definition: "A reduces to B in polynomial time" means:

  B $\in$ P ⇨ A $\in$ P

- If you encounter problem X,
  instead of trying to show that X is hard,
  try to find a problem Y that people think is hard,
  and reduce Y to X,
  and move on.

- Map of the reductions
- A → B means A $\in$ P implies B $\in$ P



- Definition of boolean formulas

(boolean) variable take either true or false (1 or 0)

literal = variable or its negation $x, \neg x$

clause = OR of literals $(x \vee \neg y \vee z)$

CNF = AND of clauses $(x \vee \neg y \vee z) \wedge (z) \wedge (\neg x \vee y)$

3CNF = CNF where each clause has 3 literals

$$(x \vee \neg y \vee z) \wedge (z \vee y \vee w) \wedge (\neg x \vee y \vee \neg u)$$

A 3CNF is satisfiable if $\exists$ assignment of 1 or 0 to variables that make the formula true

Satisfying assignment for above 3CNF?

- Definition of boolean formulas

(boolean) variable take either true or false (1 or 0)

literal = variable or its negation $x, \neg x$

clause = OR of literals $(x \lor \neg y \lor z)$

CNF = AND of clauses $(x \lor \neg y \lor z) \land (z) \land (\neg x \lor y)$

3CNF = CNF where each clause has 3 literals

$$(x \lor \neg y \lor z) \land (z \lor y \lor w) \land (\neg x \lor y \lor \neg u)$$

A 3CNF is satisfiable if $\exists$ assignment of 1 or 0 to variables that make the formula true

$x = 1, y = 1$ satisfies above

Equivalently, assignment makes each clause true

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ?? 3SAT:

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ∈ 3SAT:

  Assignment x = 1, y = 0, z = 0 gives

  (1 V 0 V 0) Λ (0 V 1 V 0) = 1 Λ 1 = 1

  (x V x V x) Λ (¬x V ¬x V ¬x) ?? 3SAT

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ∈ 3SAT:

  Assignment x = 1, y = 0, z = 0 gives

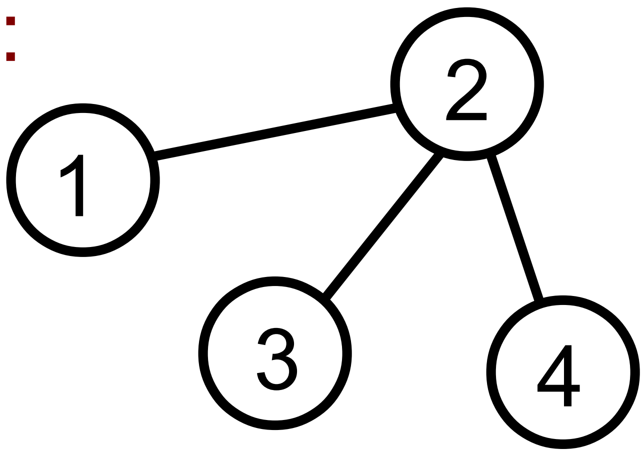
  (1 V 0 V 0) Λ (0 V 1 V 0) = 1 Λ 1 = 1

  (x V x V x) Λ (¬x V ¬x V ¬x) ∉ 3SAT

  x = 0 gives 0 Λ 1 = 0, x = 1 gives 1 Λ 0 = 0

- Conjecture: 3SAT ∉ P
- Best known algorithm takes time exponential in | φ |

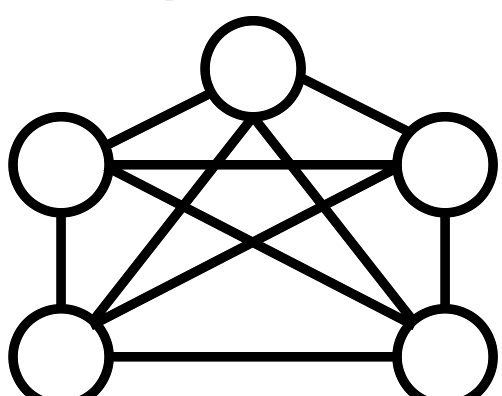
- Definition: a graph G = (V, E) consists of
  - a set of nodes V (also called "vertices")
  - a set of edges E that connect pairs of nodes
- Example:



V = {1, 2, 3, 4}

E = {(1,2), (2,3), (2,4)}

- Definition: a t-clique is a set of t nodes all connected
- Example:

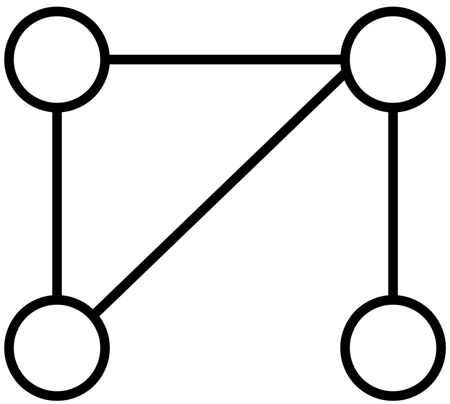


is a 5-clique

- Definition:

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- Example:

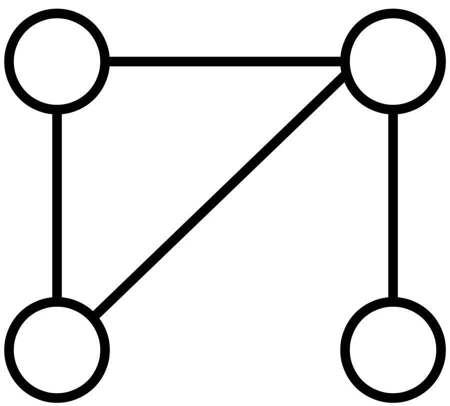
G = 

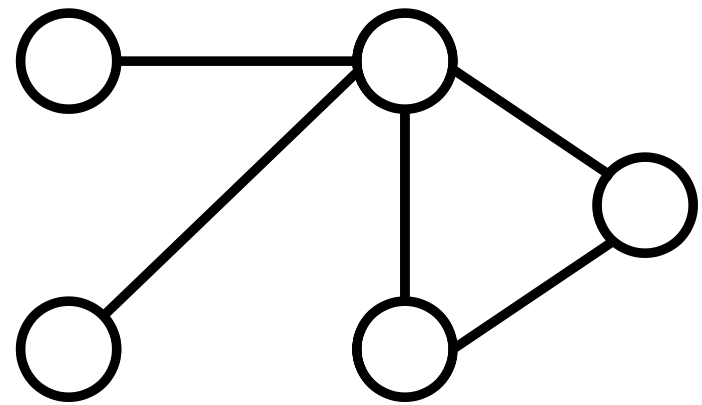
(G, 3) ? CLIQUE

- Definition:

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- Example:

G = 

H = 

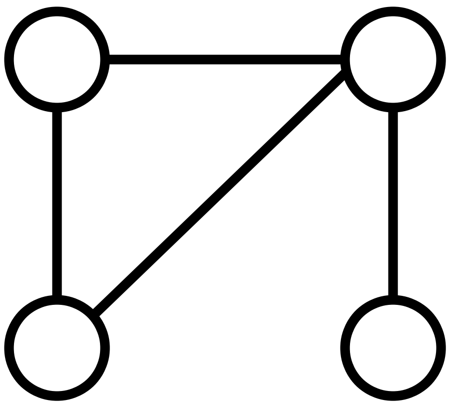
(G, 3) $\in$ CLIQUE

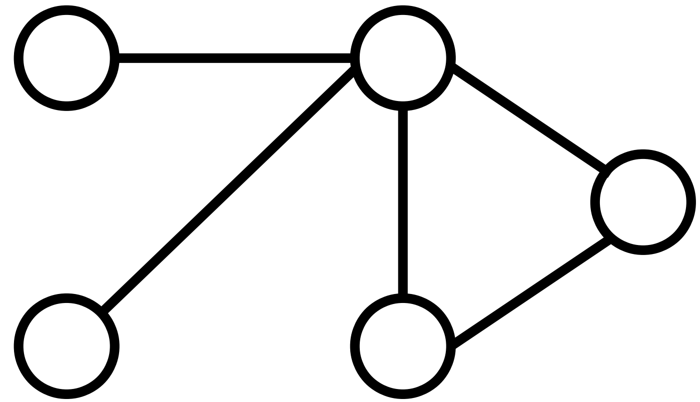
(H, 4) ? CLIQUE

- **Definition:**

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- **Example:**

G = 

H = 

(G, 3) $\in$ CLIQUE

(H, 4) $\notin$ CLIQUE

- **Conjecture:** CLIQUE $\notin$ P

- 3SAT and CLIQUE both believed $\notin$ P

- They seem different problems. And yet:

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- If you think 3SAT $\notin$ P, you also think CLIQUE $\notin$ P

- Above theorem gives what reduction?

- 3SAT and CLIQUE both believed $\notin$ P
- They seem different problems. And yet:
- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P
- If you think 3SAT $\notin$ P, you also think CLIQUE $\notin$ P
- Above theorem gives
  polynomial-time reduction of 3SAT to CLIQUE

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give algorithm R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

Enough to prove the theorem?

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give algorithm R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

Enough to prove the theorem because:

If algorithm C that solves CLIQUE in polynomial time

Then C(R(φ)) solves 3SAT in polynomial time

- Definition of R:

"On input

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

Note a_i b_i c_i are literals, φ has k clauses

- Compute G_φ and t_φ as follows:
- Nodes of G_φ : one for each a_i , b_i , c_i
- Edges of G_φ : Connect all nodes except

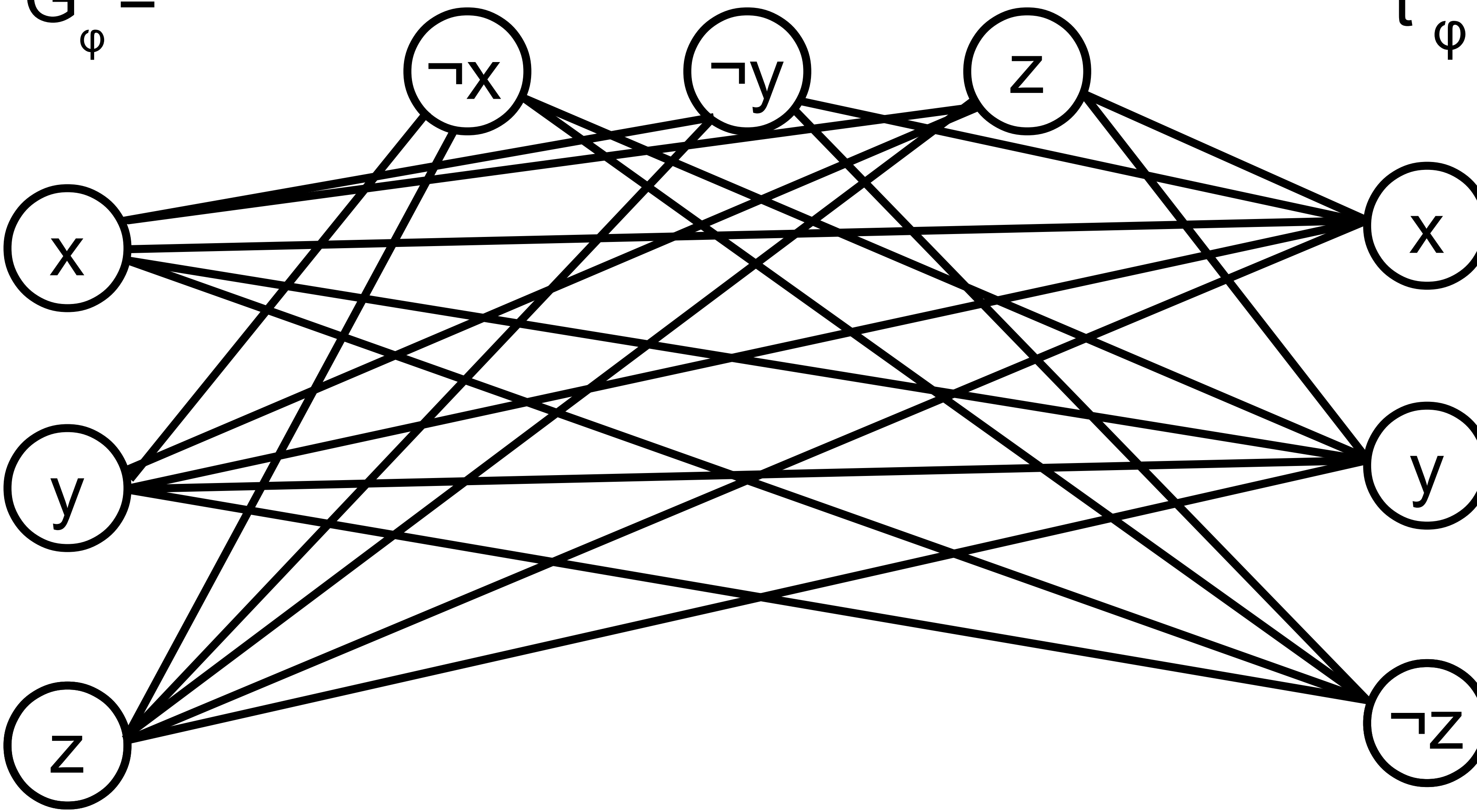
  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x
- t_φ := k"

Example:

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

$G_\varphi =$ $t_\varphi = 3$



- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi, t_\varphi) \in CLIQUE$

- High-level view of proof of $\Rightarrow$

We suppose φ has a satisfying assignment,

and we show a clique of size t_φ in G_φ

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇨

  Suppose φ has satisfying assignment
- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are ??? nodes

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇨

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are $k = t_\varphi$ nodes
- They are a clique because in G_φ we connect all but

(A) Nodes in same clause

???

(B) Contradictory nodes.

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: $\Rightarrow$

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are $k = t_\varphi$ nodes
- They are a clique because in G_φ we connect all but

(A) Nodes in same clause

Our nodes are picked from different clauses

(B) Contradictory nodes. ???

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_{\varphi}, t_{\varphi}) \in CLIQUE$
- Proof: ⇨

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_{φ}
- There are $k = t_{\varphi}$ nodes
- They are a clique because in G_{φ} we connect all but

(A) Nodes in same clause

Our nodes are picked from different clauses

(B) Contradictory nodes. Our nodes correspond to true literals in assignment: if x true then ¬ x can't be

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$

- High-level view of proof of $\Leftarrow$

- We suppose G_φ has a clique of size t_φ,

- then we show a satisfying assignment for φ

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ

- Note you have exactly one node per clause

  because ???

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ
- Note you have exactly one node per clause

  because by (A) there are no edges within clauses
- Define assignment that makes those literals true

  Possible ???

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ $(G_\varphi , t_\varphi) \in$ CLIQUE
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ
- Note you have exactly one node per clause

  because by (A) there are no edges within clauses
- Define assignment that makes those literals true

  Possible by (B): contradictory literals not connected
- Assignment satisfies φ because ???

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow (G_\varphi , t_\varphi) \in$ CLIQUE
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ

- Note you have exactly one node per clause

  because by (A) there are no edges within clauses

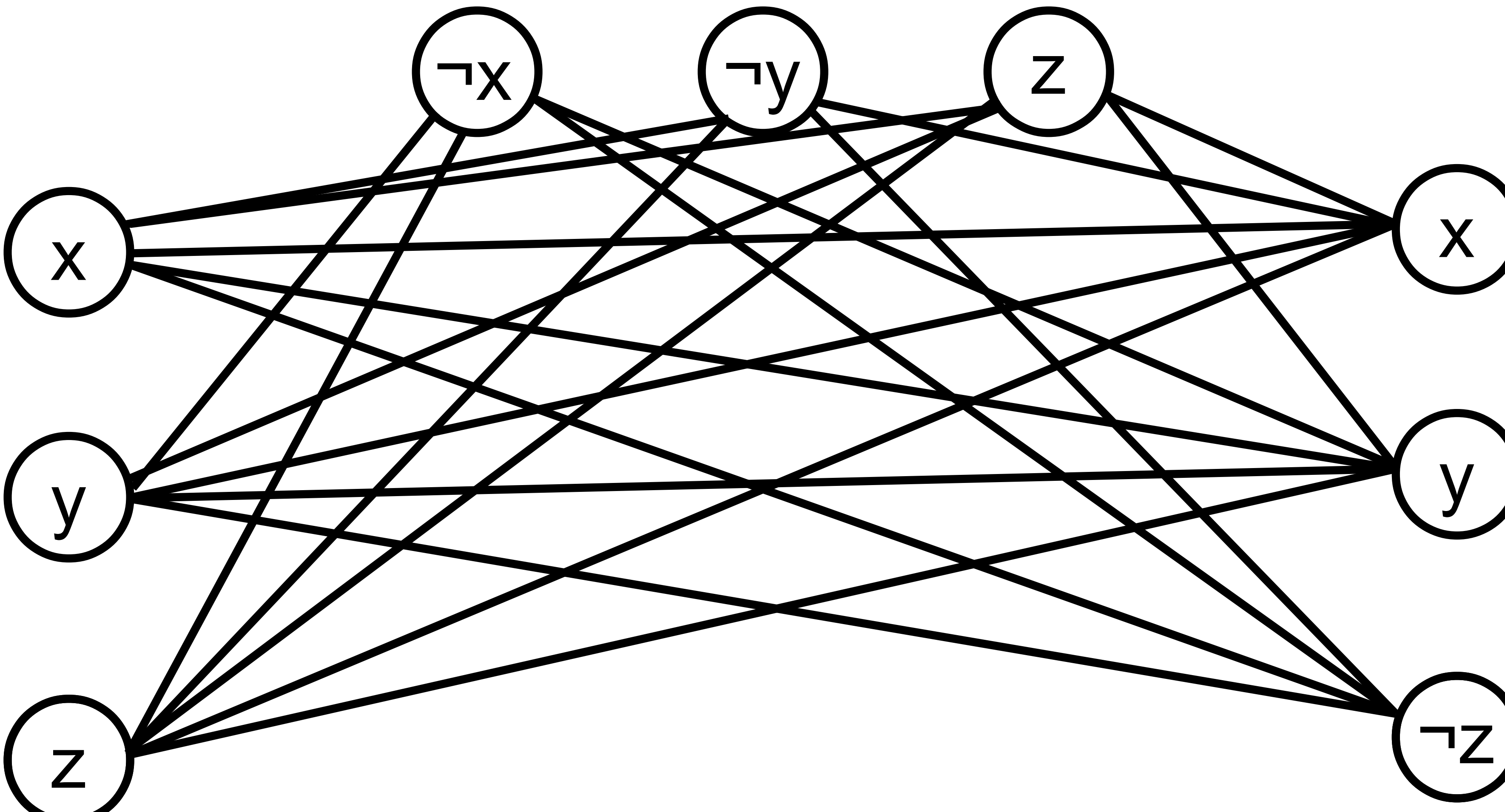
- Define assignment that makes those literals true

  Possible by (B): contradictory literals not connected

- Assignment satisfies φ because every clause is true

Back to example:

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



Back to example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

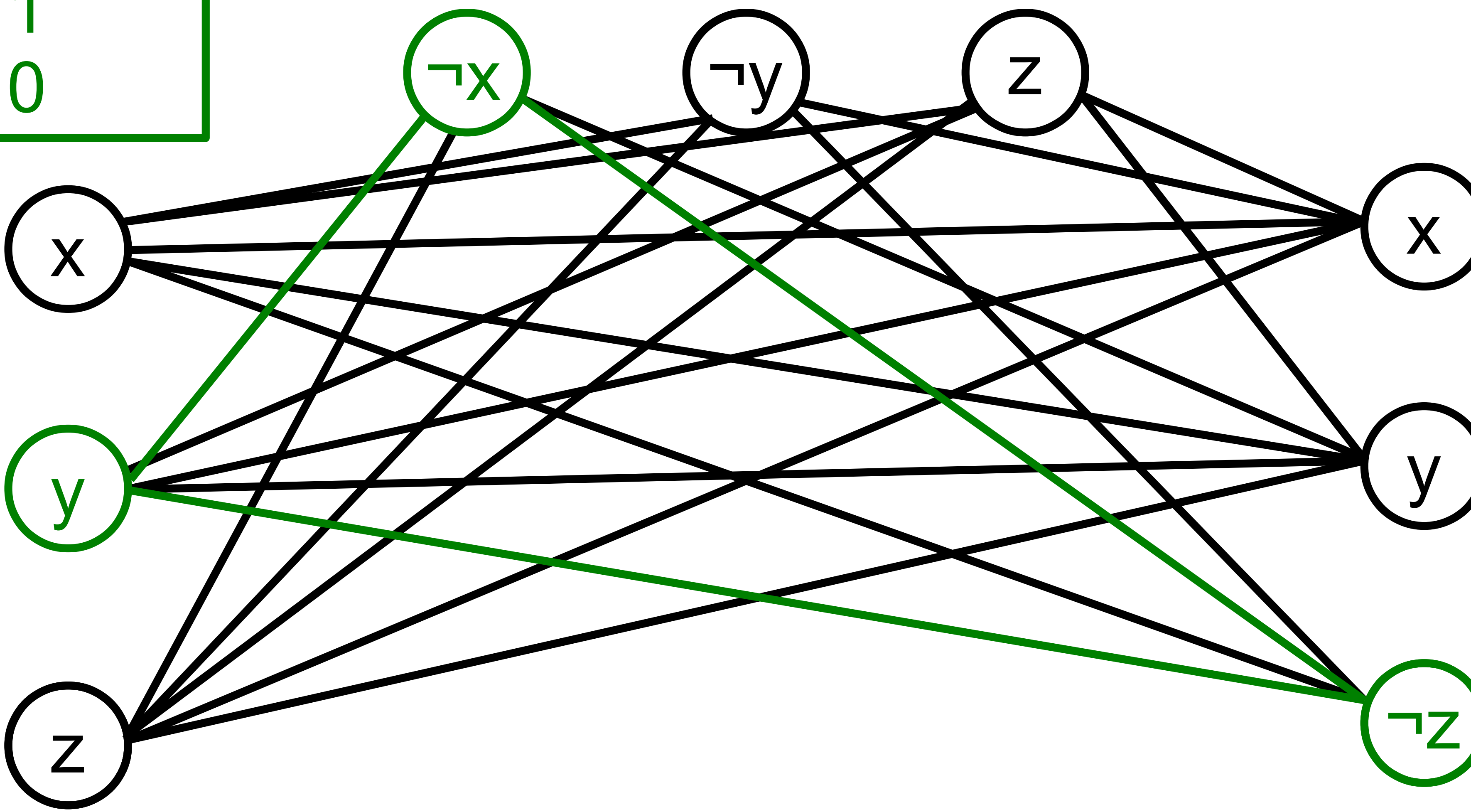
0 1 0   1 0 0   0 1 1

Assignment

$x = 0$

$y = 1$

$z = 0$



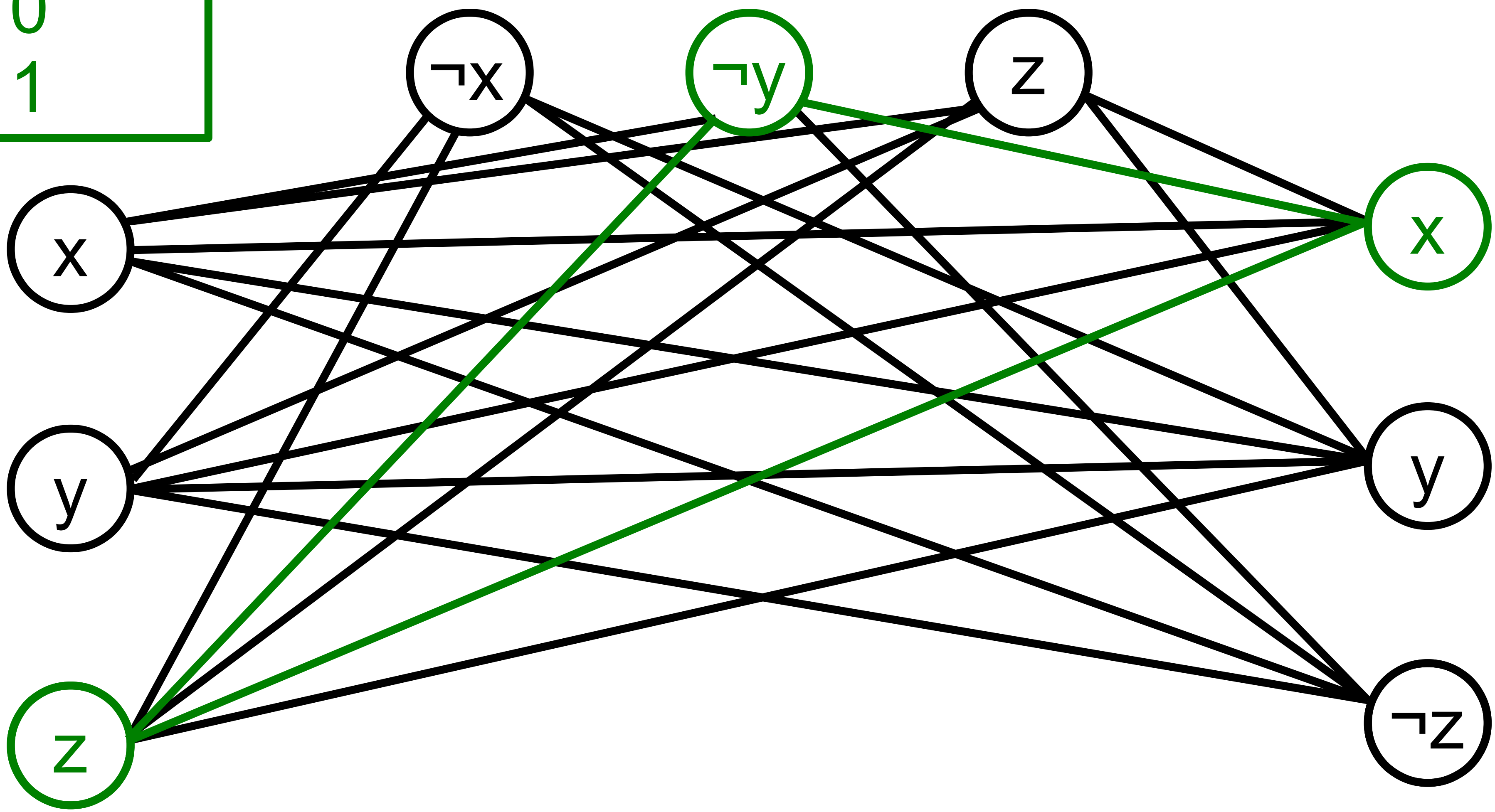
Back to example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

1 0 1 0 1 1 1 0 0

Assignment

x = 1
y = 0
z = 1



- **Theorem**: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- **Proof outline**:

  We give algorithm R that on input φ :

  (1) Computes graph G_φ and integer t_φ such that

  $$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

  (2) R runs in polynomial time

- So far: defined R, proved (1). It remains to see (2)

- (2) is less interesting.

- R : "On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- We do not directly count the steps of R

  Too low-level, complicated, uninformative.

- We give a more high-level argument

- R : "On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- To compute nodes: examine all literals.

  Number of literals ≤ | φ |

- This is polynomial in the input length | φ |

- R : "On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- To compute edges: examine all pairs of nodes.

  Number of pairs is $\leq$ (number of nodes)$^2 \leq |\varphi|^2$

- Which is polynomial in the input length $|\varphi|$

- R : "On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- Overall, we examine $\leq |\varphi| + |\varphi|^2$

- Which is polynomial in the input length $|\varphi|$
- This concludes the proof.

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- We have concluded the proof of above theorem

- Recall outline:

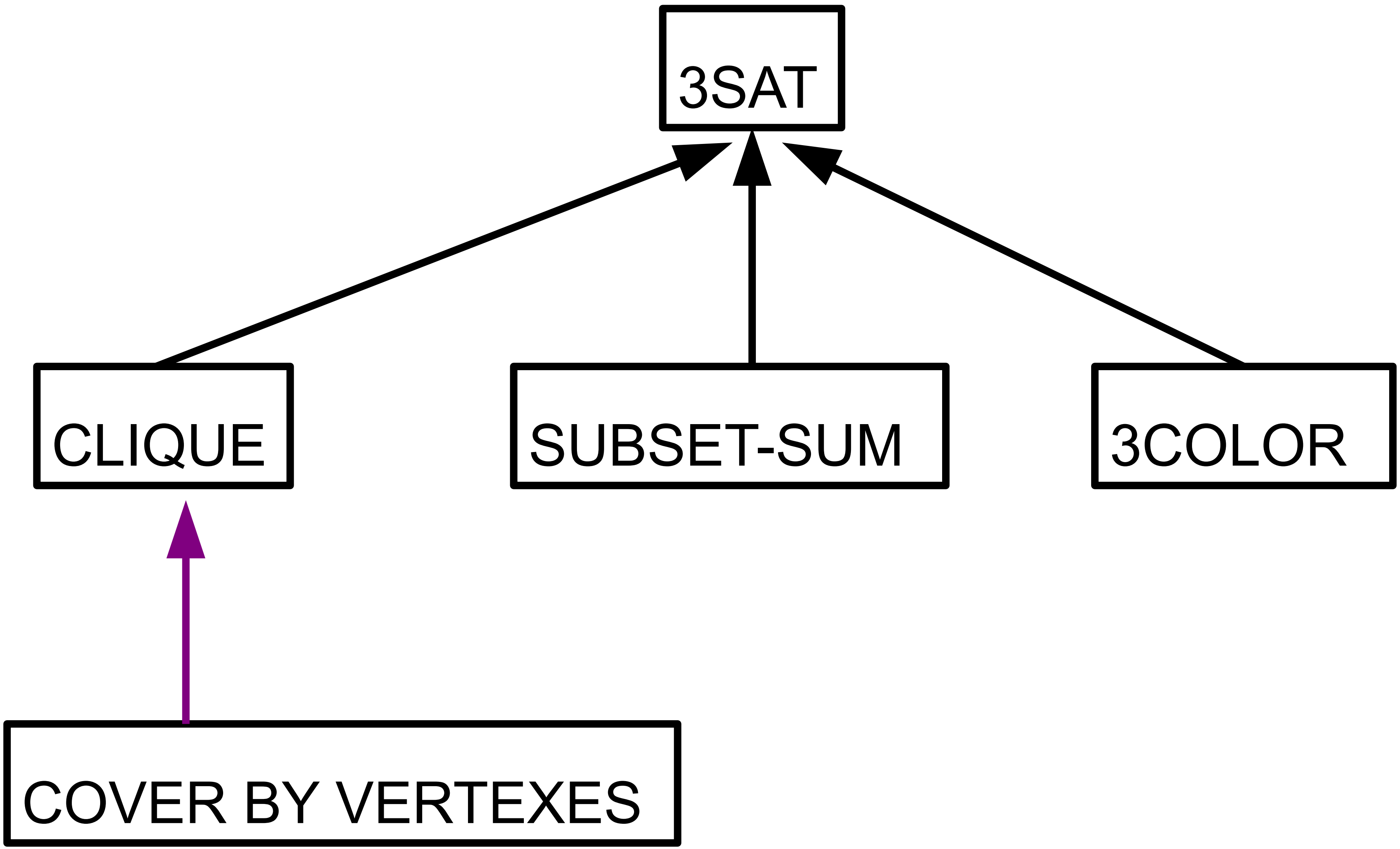
  We give algorithm R that on input φ :

  (1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

  (2) R runs in polynomial time

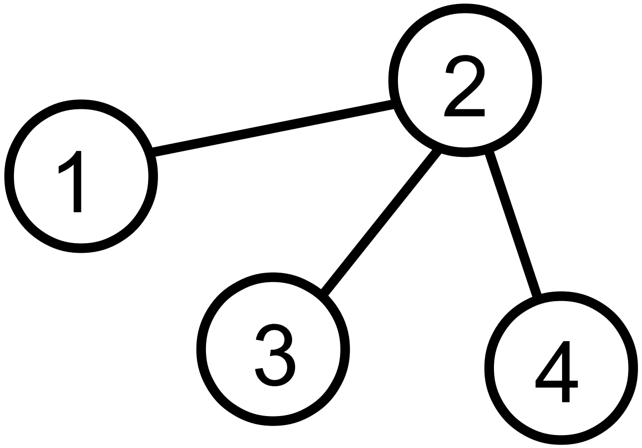
- Map of the reductions
- A → B means A $\in$P implies B $\in$P



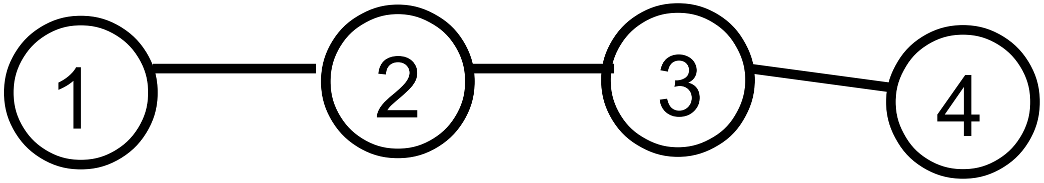
- Definition: In a graph G = (V,E),

a t-cover is a set of t nodes that touch all edges

- Example:



has the 1-cover {2}

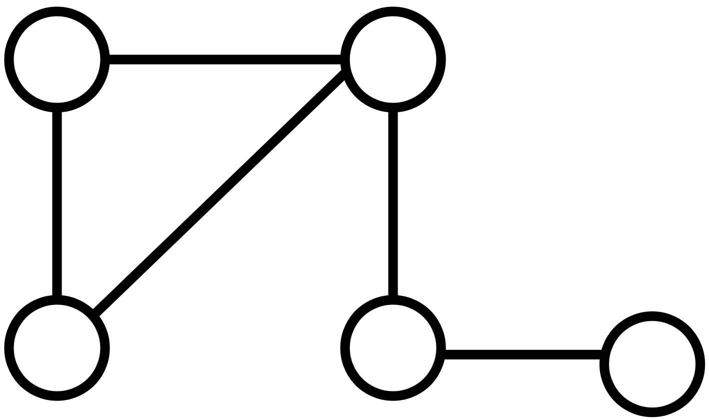


has the 2-cover {2,3}

- Definition: COVER BY VERTEXES

CBV = {(G,t) : G is a graph containing a t-cover}

- Example:

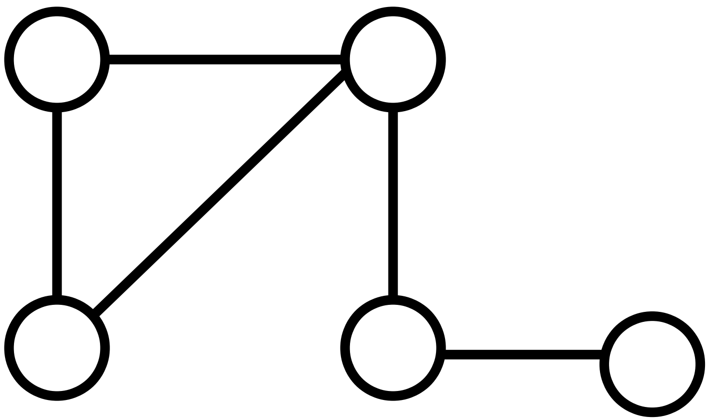
G = 

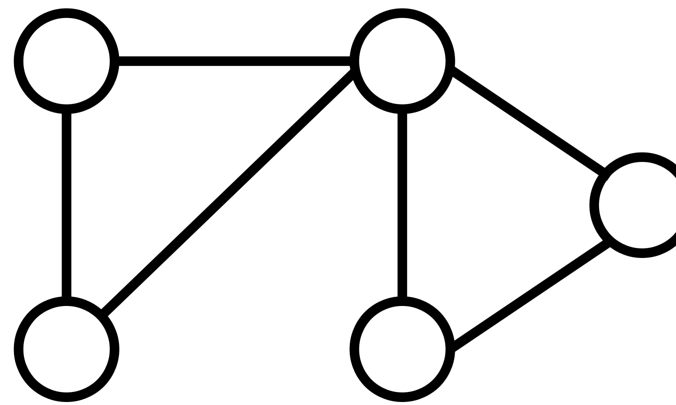
(G, 2) ? CBV

- Definition: COVER BY VERTEXES

CBV = {(G,t) : G is a graph containing a t-cover}

- Example:

G = 

H = 

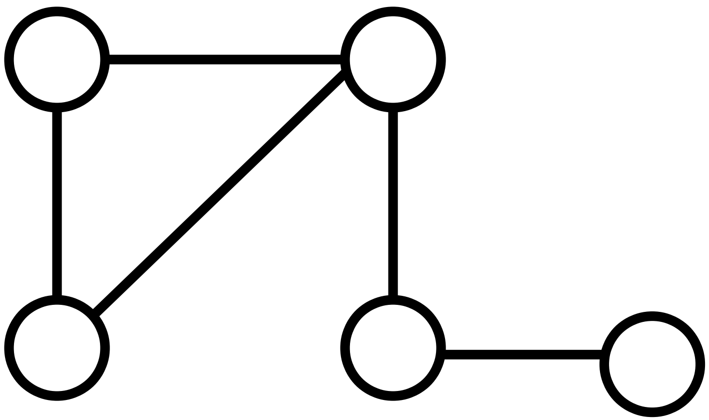
(G, 2) $\notin$ CBV

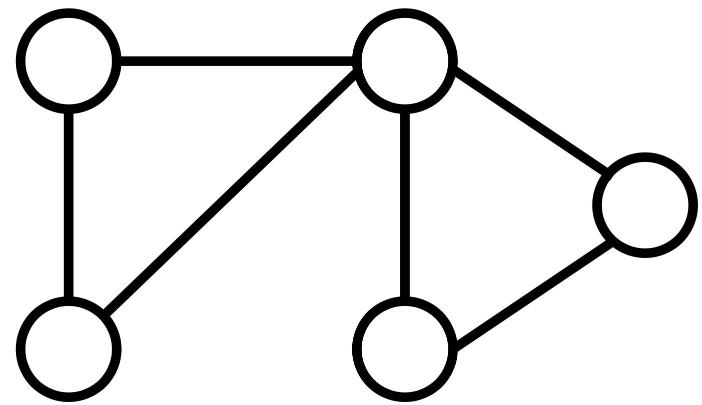
(H, 3) ? CBV

- Definition: COVER BY VERTEXES

CBV = {(G,t) : G is a graph containing a t-cover}

- Example:

G = 

H = 

(G, 2) $\notin$ CBV

(H, 3) $\in$ CBV

- Conjecture: CBV $\notin$ P

- Theorem: CBV $\in$ P $\Rightarrow$ CLIQUE $\in$ P

- Proof outline:

We give algorithm R that on input (G,t) :

(1) Computes graph G' and integer t' such that

G has a t-clique $\Leftrightarrow$ G' has a t'-cover

(2) R runs in polynomial time

- Definition of R:

"On input graph $G = (V,E)$ and integer t

Compute $G' = (V',E')$ and t' as follows:

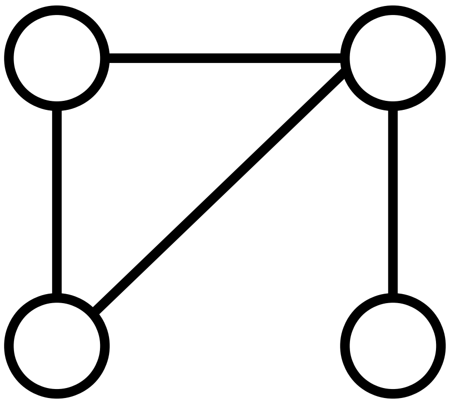
E' is the complement of E

That is, $\{u,v\} \in E'$ if and only if $\{u,v\} \notin E$

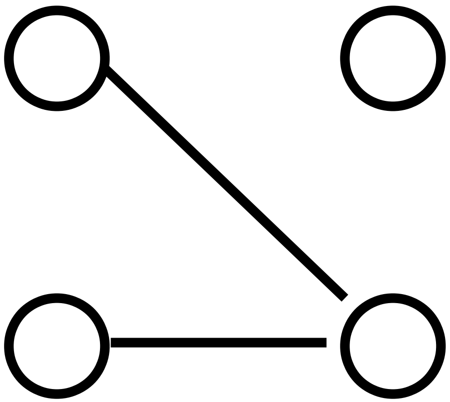
$t' = |V| - t$."

- Example

G =



G' =



- Claim: G has a t-clique ⇔ G' has a t'-cover

- Proof:

  (➔) Suppose G = (V,E) has a t-clique C.

  We claim that V - C is a cover of G'.

  Let (u,v) be in E'. Then ?

- Claim: G has a t-clique ⇔ G' has a t'-cover

- Proof:

  (➔) Suppose G = (V,E) has a t-clique C.

  We claim that V - C is a cover of G'.

  Let (u,v) be in E'. Then (u,v) ∉ E. So either u or v does not belong to C. So either u or v belongs to V - C.

  (←) Suppose G' = (V',E') has a t-cover C.

  We claim that V - C is a clique of G.

  Let u and v be two nodes in V - C. Then ?

- Claim: G has a t-clique ⇔ G' has a t'-cover

- Proof:

  (➔) Suppose G = (V,E) has a t-clique C.

  We claim that V - C is a cover of G'.

  Let (u,v) be in E'. Then (u,v) ∉ E. So either u or v does not belong to C. So either u or v belongs to V - C.

  (←) Suppose G' = (V',E') has a t-cover C.

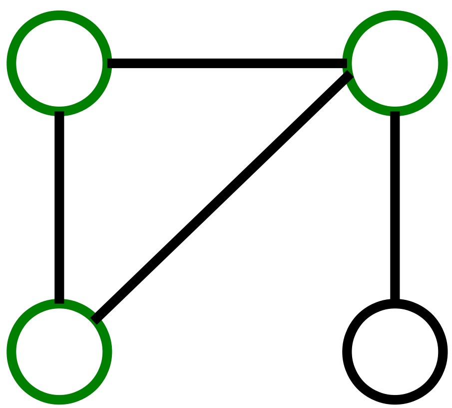
  We claim that V - C is a clique of G.

  Let u and v be two nodes in V - C. Then {u,v} is not in E'. Hence {u,v} is in E.

■

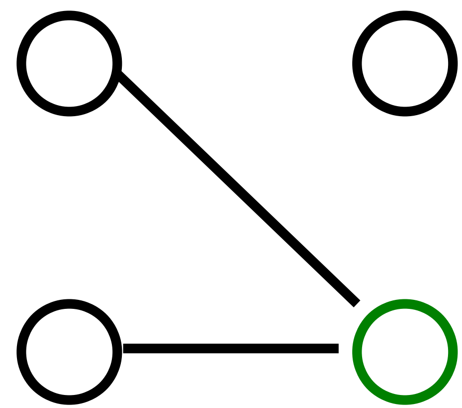
- Example

G =



a t = 3 clique

G' =



a t' = |V| - t = 1 cover

- It remains to argue that R runs in polynomial time

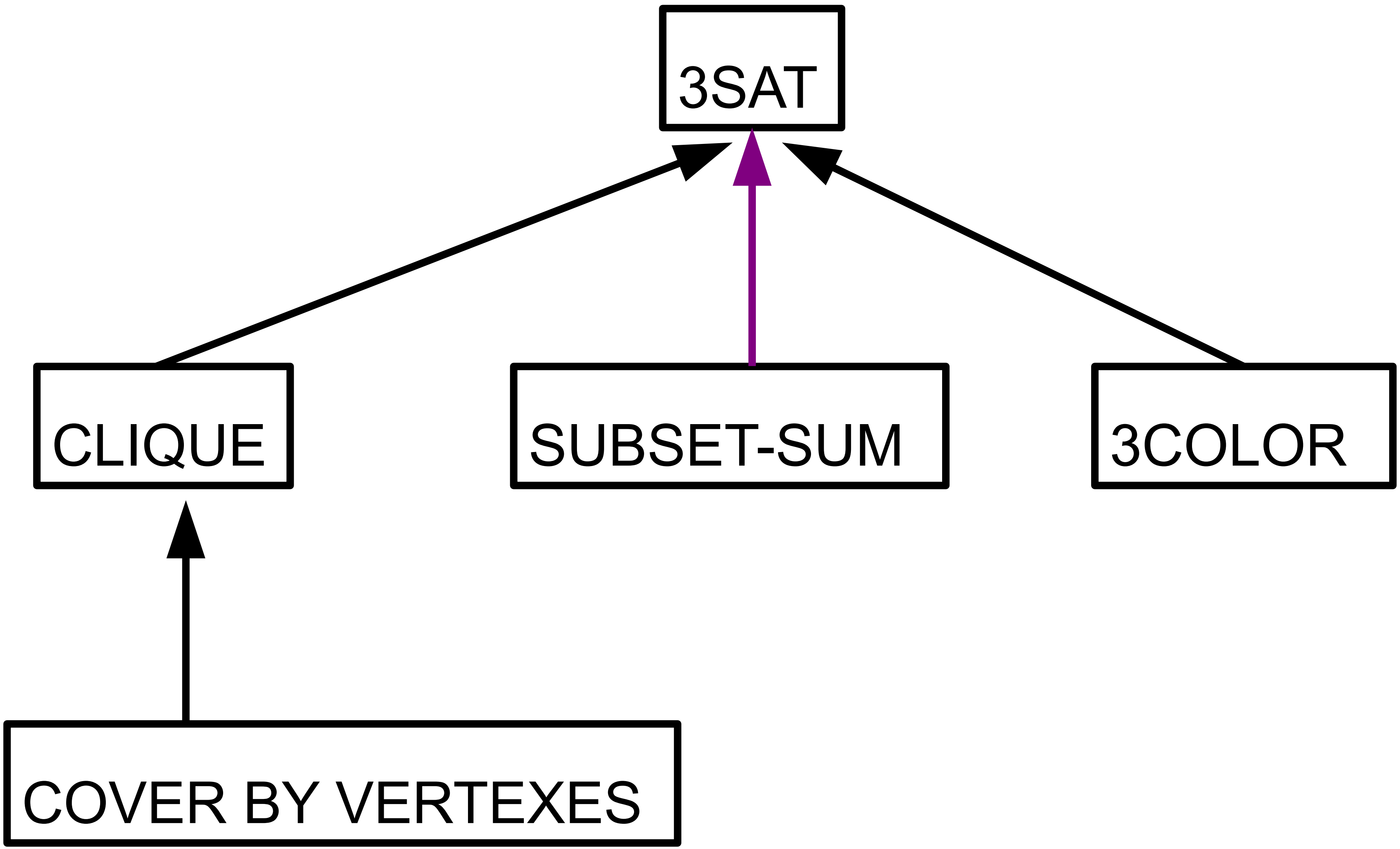
- To compute G' = (V',E') we go through each pair of nodes $\{u,v\}$ and make it an edge if and only if $\{u,v\} \notin E$.

- This takes time $|V|^2$ which is polynomial in the input length

- To compute t' is simple arithmetic.

- End of proof that CBV $\in$ P $\Rightarrow$ CLIQUE $\in$ P

- Map of the reductions
- A → B means A $\in$P implies B $\in$P



- Definition:

SUBSET-SUM = $\{(a_1, a_2, \ldots, a_n, t) : \exists$ i1,i2,...,ik $\leq n$ such that $a_{i1}+a_{i2}+\ldots+a_{ik} = t \}$

- Example:
    - (5, 2, 14, 3, 9, 25) ? SUBSET-SUM

- Definition:

SUBSET-SUM = $\{(a_1, a_2, ..., a_n, t) : \exists\ i1, i2, ..., ik \leq n$ such that $a_{i1} + a_{i2} + ... + a_{ik} = t\ \}$

- Example:
  - (5, 2, 14, 3, 9, 25) $\in$ SUBSET-SUM

    because 2 + 14 + 9 = 25
  - (1, 3, 4, 9, 15) ? SUBSET-SUM

- Definition:

SUBSET-SUM = $\{(a_1, a_2, ..., a_n, t) : \exists\ i1, i2, ..., ik \leq n$ such that $a_{i1} + a_{i2} + ... + a_{ik} = t\}$

- Example:
  - (5, 2, 14, 3, 9, 25) $\in$ SUBSET-SUM

    because 2 + 14 + 9 = 25
  - (1, 3, 4, 9, 15) $\notin$ SUBSET-SUM

    because no subset of {1,3,4,9} sums to 15

- Conjecture: SUBSET-SUM $\notin$ P

- Theorem: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give algorithm R that on input φ :

(1) Computes numbers $a_1, a_2, \ldots, a_n, t$ such that

$$\varphi \in 3SAT \Leftrightarrow (a_1, a_2, \ldots, a_n, t) \in \text{SUBSET-SUM}$$

(2) R runs in polynomial time

- Theorem: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Warm-up for definition of R:
- On input φ with v variables and k clauses:

- R will produce a list of numbers.
- Numbers will have many digits, v + k

  and look like this: 1000010011010011

  First v (most significant) digits correspond to variables
- Other k (least significant) correspond to clauses

- Theorem: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R:
- "On input φ with v variables and k clauses :
- For each variable x include

  a_x^T = 1 in x's digit, and 1 in every digit of a clause where x appears without negation

  a_x^F = 1 in x's digit, and 1 in every digit of a clause where x appears negated
- For each clause C, include twice

  a_C = 1 in C's digit, and 0 in others
- Set t = 1 in first v digits, and 3 in rest k digits"

Example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

3 variables + 3 clauses $\Rightarrow$ 6 digits for each number

	var x	var y	var z	clause 1	clause 2	clause 3
a_x^T =	1	0	0	1	0	1
a_x^F =	1	0	0	0	1	0
a_y^T =	0	1	0	1	0	1
a_y^F =	0	1	0	0	1	0
a_z^T =	0	0	1	1	1	0
a_z^F =	0	0	1	0	0	1
a_{c1} =	0	0	0	1	0	0
a_{c2} =	0	0	0	0	1	0
a_{c3} =	0	0	0	0	0	1
t =	1	1	1	3	3	3

} two copies of each of these (a_{c1}, a_{c2}, a_{c3})

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Rightarrow$

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

  because ???

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Rightarrow$

Suppose φ has satisfying assignment

- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

because a_x^T , a_x^F have 1 in x's digit, 0 in others

and 1, 2, or 3 in last k digits

because ???

- Claim: $\varphi \in$ 3SAT ⇔ R(φ) $\in$ SUBSET-SUM
- Proof: ⇨

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

  because a_x^T , a_x^F have 1 in x's digit, 0 in others

  and 1, 2, or 3 in last k digits

  because each clause has true literal, and

  a_x^T has 1 in clauses where x appears not negated

  a_x^F has 1 in clauses where x appears negated
- By picking ???? ????? ??????? ?? sum reaches t

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Rightarrow$

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits
  because a_x^T, a_x^F have 1 in x's digit, 0 in others

  and 1, 2, or 3 in last k digits

  because each clause has true literal, and
  a_x^T has 1 in clauses where x appears not negated
  a_x^F has 1 in clauses where x appears negated
- By picking appropriate subset of a_C sum reaches t

- Claim: $\varphi \in$ 3SAT ⇔ R(φ) $\in$ SUBSET-SUM
- Proof: ⇐
- Suppose a subset sums to t = 1111111111333333333
- No carry in sum, because ???

- Claim: $\varphi \in$ 3SAT ⇔ R(φ) $\in$ SUBSET-SUM
- Proof: ⇐
- Suppose a subset sums to t = 11111111111333333333
- No carry in sum, because only 3 literals per clause
- So digits behave "independently"
- For each pair a_x^T a_x^F exactly one is included

  otherwise ???

- Claim: $\varphi \in 3SAT \Leftrightarrow R(\varphi) \in$ SUBSET-SUM
- Proof: ⇦
- Suppose a subset sums to t = 111111111113333333
- No carry in sum, because only 3 literals per clause
- So digits behave "independently"
- For each pair a_x^T a_x^F exactly one is included

  otherwise would not get 1 in that digit
- Define x true if a_x^T included, false otherwise
- For any clause C, the a_C contribute ≤ 2 in C's digit
- So each clause must have a true literal

  otherwise ???

- Claim: $\varphi \in 3SAT \Leftrightarrow R(\varphi) \in SUBSET\text{-}SUM$
- Proof: $\Leftarrow$
- Suppose a subset sums to t = 1111111111333333333
- No carry in sum, because only 3 literals per clause
- So digits behave "independently"
- For each pair a_x^T a_x^F exactly one is included

  otherwise would not get 1 in that digit
- Define x true if a_x^T included, false otherwise
- For any clause C, the a_C contribute ≤ 2 in C's digit
- So each clause must have a true literal

  otherwise sum would not get 3 in that digit

Back to example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

		var x	var y	var z	clause 1	clause 2	clause 3
	a_x^T =	1	0	0	1	0	1
	a_x^F =	1	0	0	0	1	0
	a_y^T =	0	1	0	1	0	1
	a_y^F =	0	1	0	0	1	0
	a_z^T =	0	0	1	1	1	0
	a_z^F =	0	0	1	0	0	1
(2x)	a_{c1} =	0	0	0	1	0	0
(2x)	a_{c2} =	0	0	0	0	1	0
(2x)	a_{c3} =	0	0	0	0	0	1
	t =	1	1	1	3	3	3

Back to example:

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

0 1 0 1 0 0 0 1 1

	var x	var y	var z	clause 1	clause 2	clause 3	
$a_x^T =$	1	0	0	1	0	1	
$a_x^F =$	**1**	**0**	**0**	**0**	**1**	**0**	
$a_y^T =$	**0**	**1**	**0**	**1**	**0**	**1**	
$a_y^F =$	0	1	0	0	1	0	
$a_z^T =$	0	0	1	1	1	0	
$a_z^F =$	**0**	**0**	**1**	**0**	**0**	**1**	
(2x) **$a_{c1} =$**	**0**	**0**	**0**	**1**	**0**	**0**	(choose twice)
(2x) **$a_{c2} =$**	**0**	**0**	**0**	**0**	**1**	**0**	(choose twice)
(2x) **$a_{c3} =$**	**0**	**0**	**0**	**0**	**0**	**1**	
$t =$	1	1	1	3	3	3	

Assignment

x = 0

y = 1

z = 0

Back to example:

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

1 1 1 0 0 1 1 1 0

	var x	var y	var z	clause 1	clause 2	clause 3	
a_x^T =	**1**	**0**	**0**	**1**	**0**	**1**	
a_x^F =	1	0	0	0	1	0	
a_y^T =	**0**	**1**	**0**	**1**	**0**	**1**	
a_y^F =	0	1	0	0	1	0	
a_z^T =	**0**	**0**	**1**	**1**	**1**	**0**	
a_z^F =	0	0	1	0	0	1	
(2x) a_{c1} =	0	0	0	1	0	0	
(2x) **a_{c2} =**	**0**	**0**	**0**	**0**	**1**	**0**	(choose twice)
(2x) **a_{c3} =**	**0**	**0**	**0**	**0**	**0**	**1**	
t =	1	1	1	3	3	3	

Assignment

x = 1

y = 1

z = 1

- It remains to argue that ???

- It remains to argue that R runs in polynomial time

- To compute numbers a_x^T a_x^F :

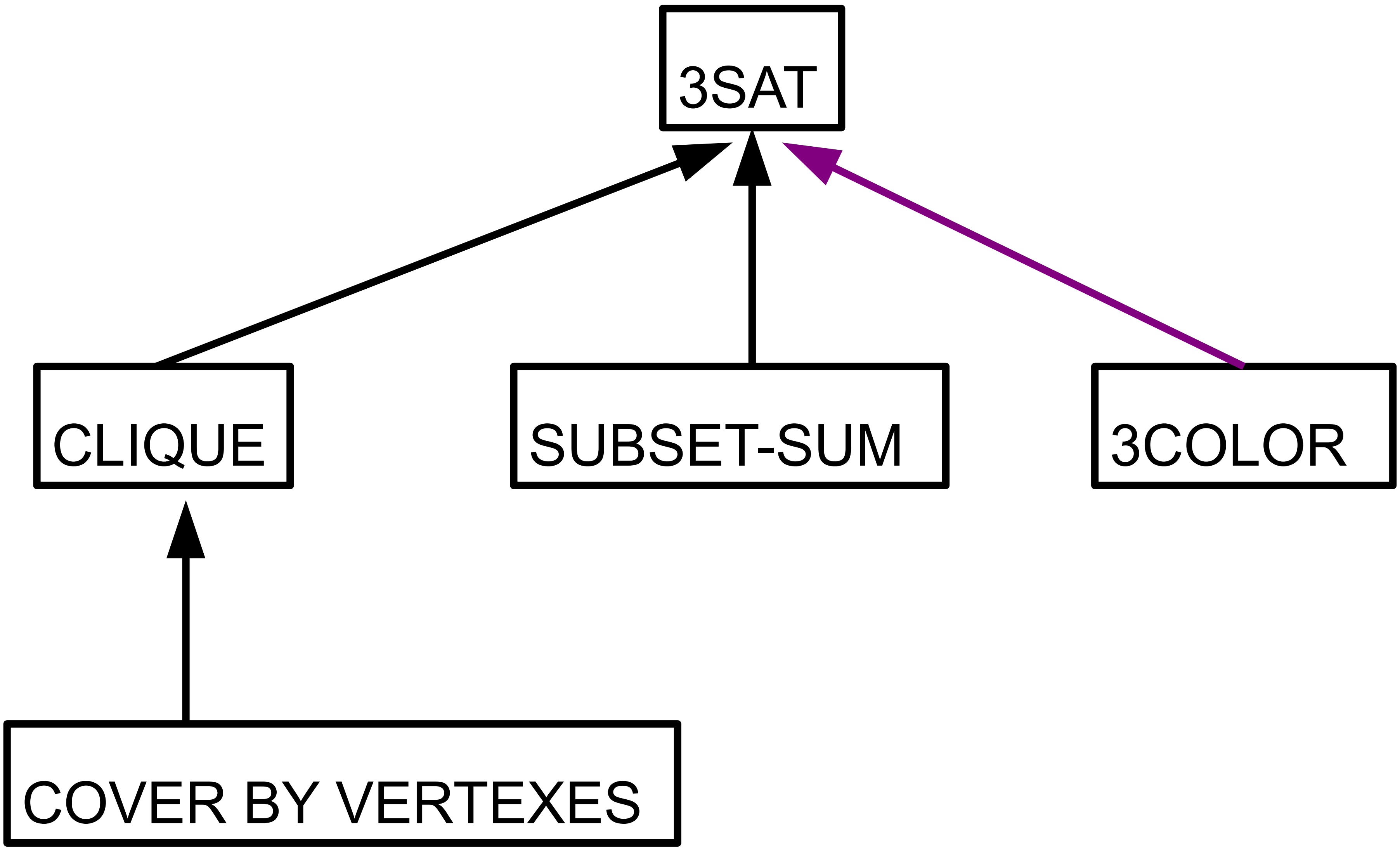
  For each variable x, examine $k \leq |\varphi|$ clauses

  Overall, examine $v\,k \leq |\varphi|^2$ clauses

- To compute numbers a_C examine $k \leq |\varphi|$ clauses

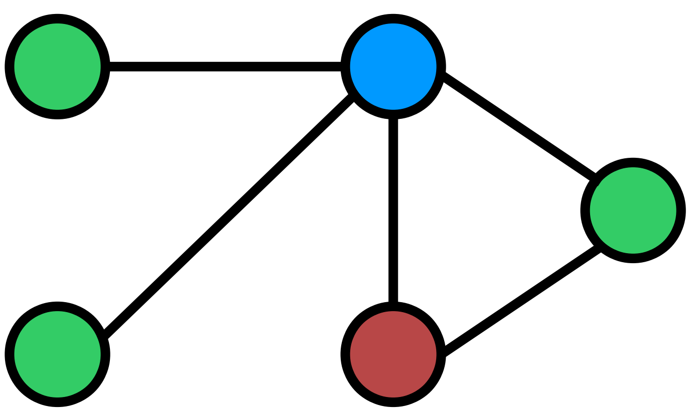
- In total $|\varphi|^2 + |\varphi|$, which is polynomial in input length
- End of proof that SUBSET-SUM $\in$P $\Rightarrow$ 3SAT $\in$P

- Map of the reductions
- A → B means A $\in$P implies B $\in$P

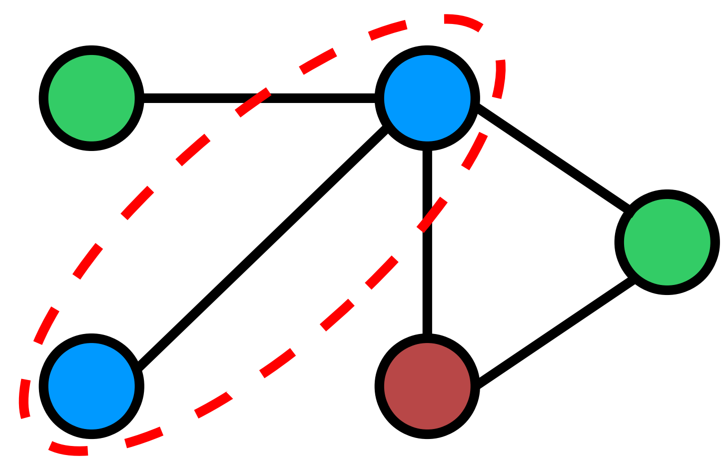


- Definition: A 3-coloring of a graph is a coloring of each node, using at most 3 colors, such that no adjacent nodes have the same color.

- Example:



a 3-coloring

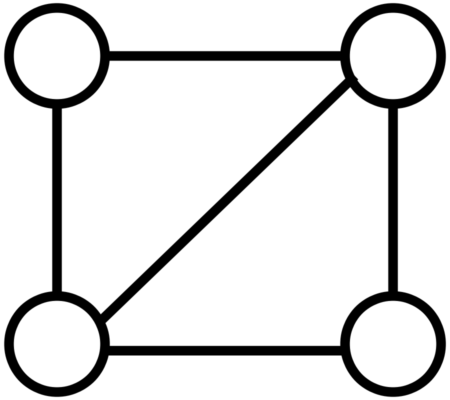


not a 3-coloring

- Definition:

  3COLOR = {G | G is a graph with a 3-coloring}

- Example:

G = 

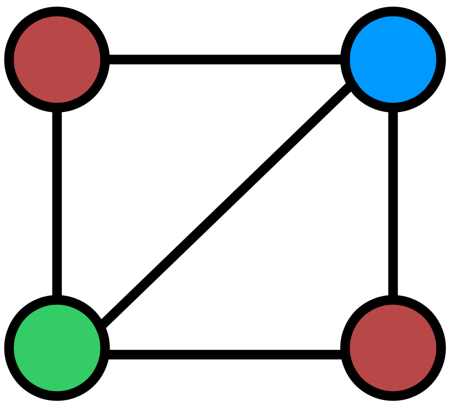
G ?? 3COLOR

- Definition:

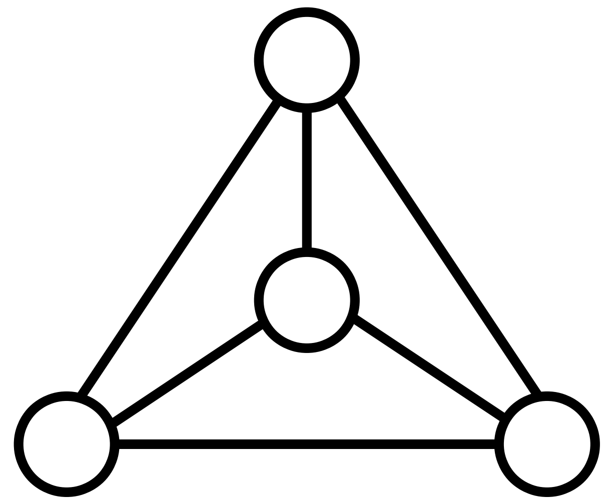
3COLOR = {G | G is a graph with a 3-coloring}

- Example:

G =



H =



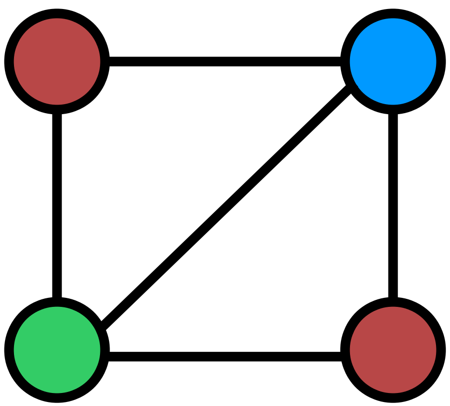
G $\in$ 3COLOR

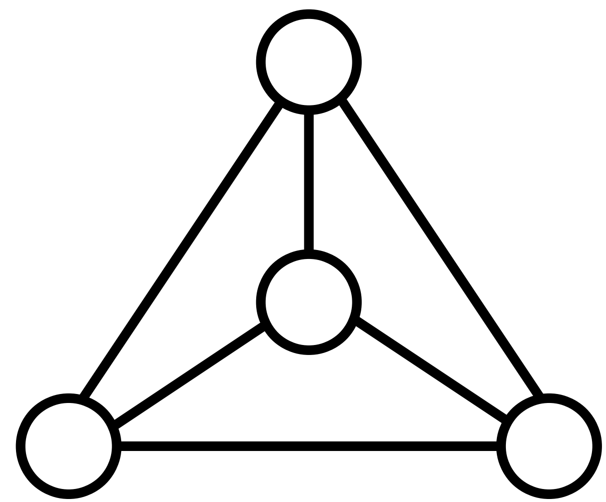
H ? 3COLOR

- Definition:

  3COLOR = {G | G is a graph with a 3-coloring}

- Example:

G = 

H = 

G $\in$ 3COLOR

H $\notin$ 3COLOR
(> 3 nodes, all connected)

- Conjecture: 3COLOR $\notin$ P

- Theorem: $3COLOR \in P \Rightarrow 3SAT \in P$

- Proof outline:

Give algorithm R that on input φ :

(1) Computes a graph G_φ such that

$$\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR.$$

(2) R runs in polynomial time

Enough to prove the theorem ?

- Theorem: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

Give algorithm R that on input φ :

(1) Computes a graph G_φ such that

$$\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR.$$

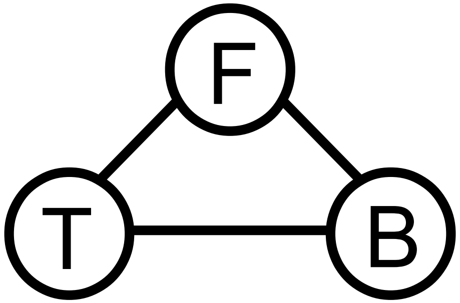
(2) R runs in polynomial time

Enough to prove the theorem because:

If algorithm C that solves 3COLOR in polynomial-time

Then C(R(φ)) solves 3SAT in polynomial-time

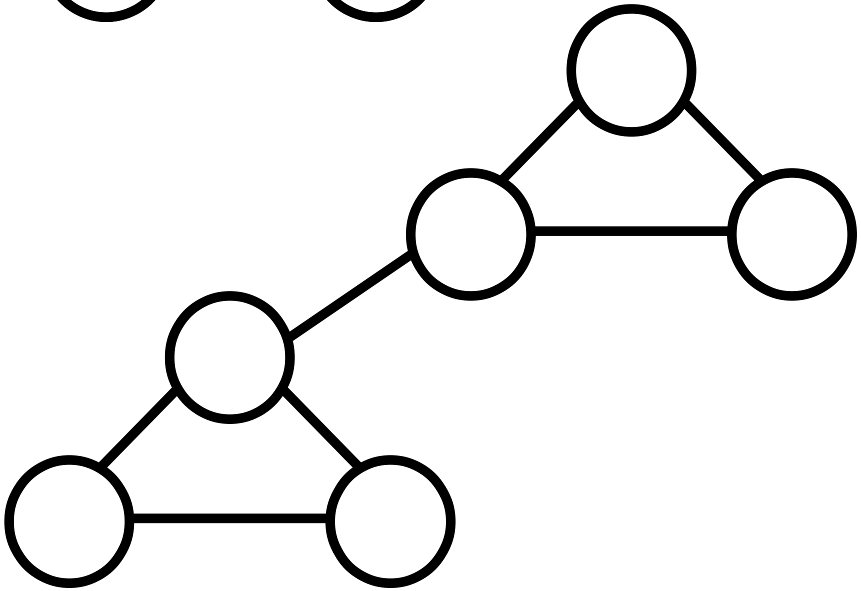
- Theorem: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R:
  - "On input φ, construct G_φ as follows:
  - Add 3 special nodes called the "palette".



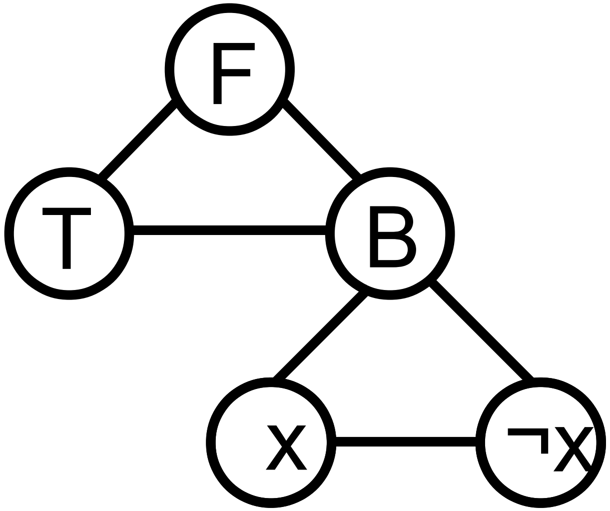
T = "true"
F = "false"
B = "base"

  - For each variable, add 2 literal nodes.

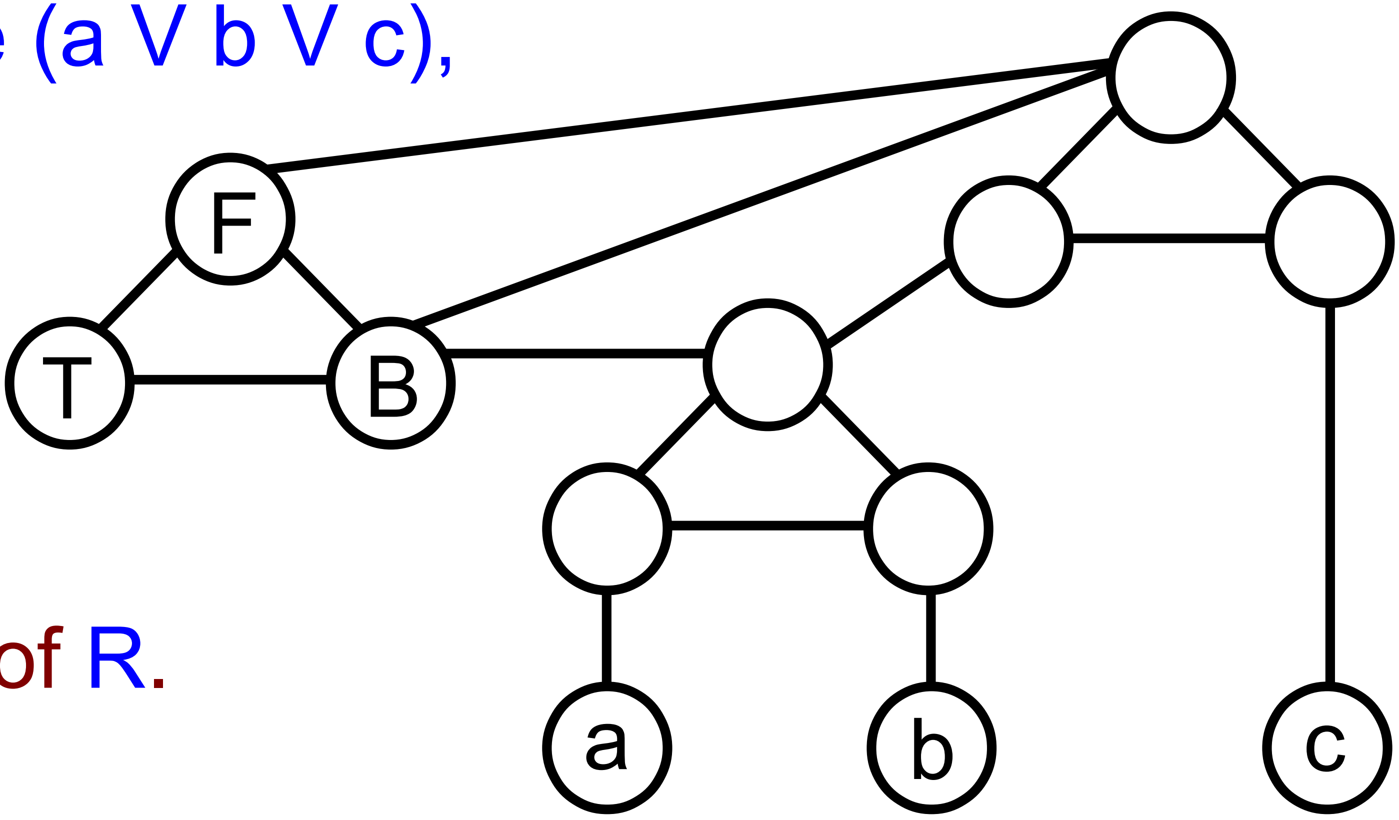
  - For each clause, add 6 clause nodes.



- **Theorem**: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R (continued):
  - For each variable x, connect:



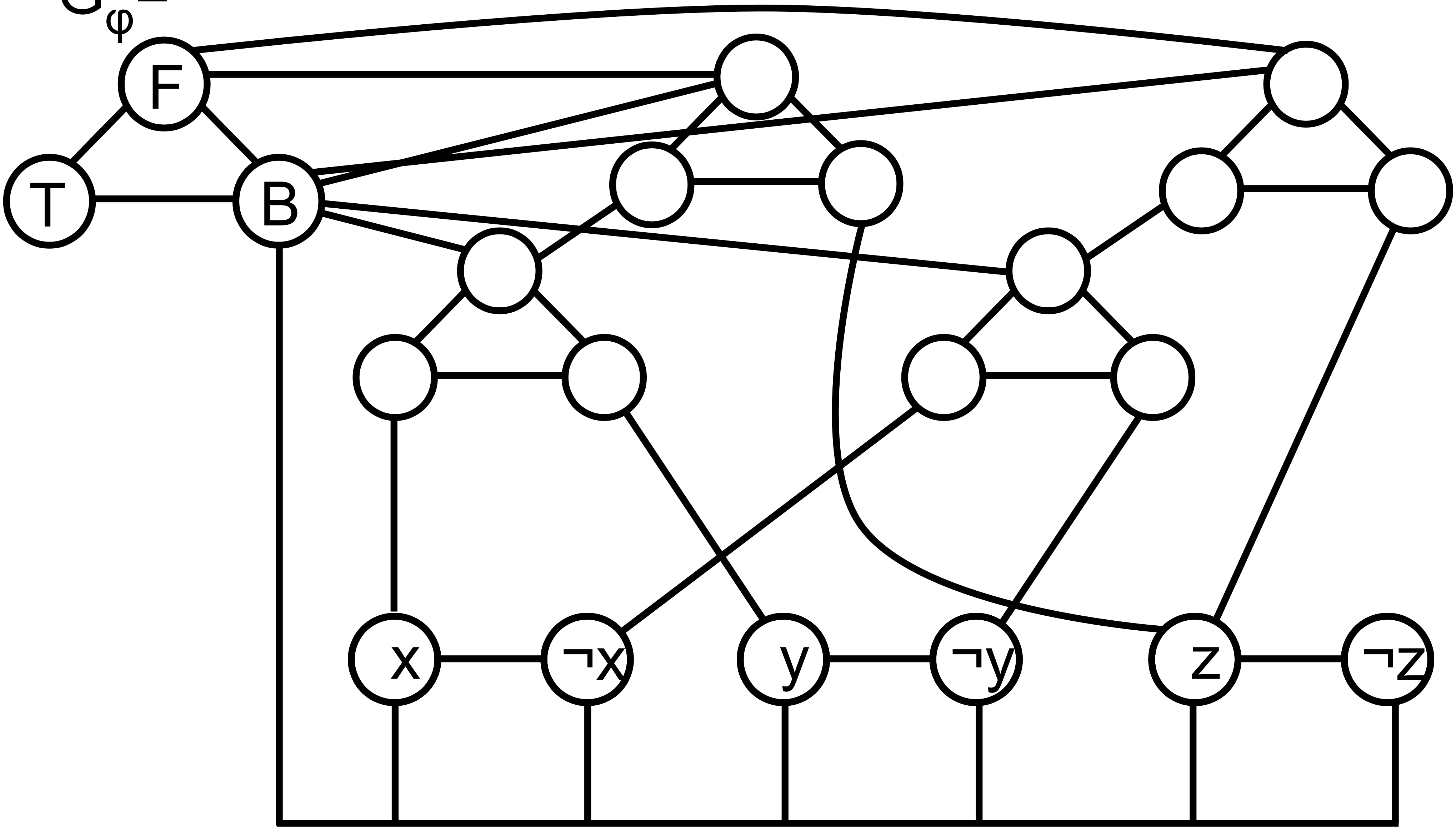
  - For each clause (a V b V c), connect:



- End of definition of R.

Example: $\varphi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$

$G_\varphi =$



- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$

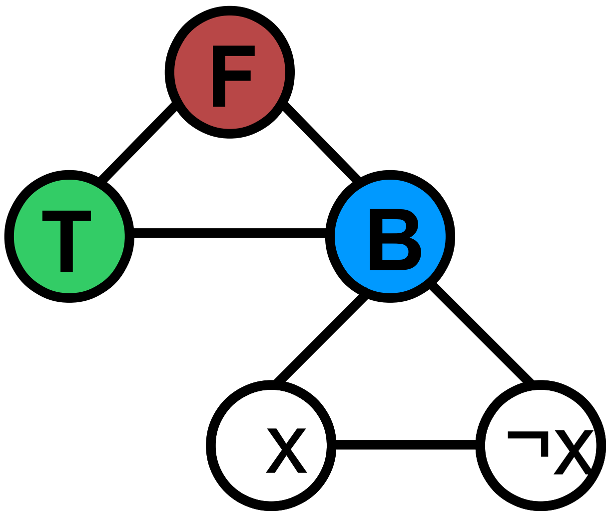
- Before proving the claim, we make some remarks,

  and prove a fact that will be useful

Remark

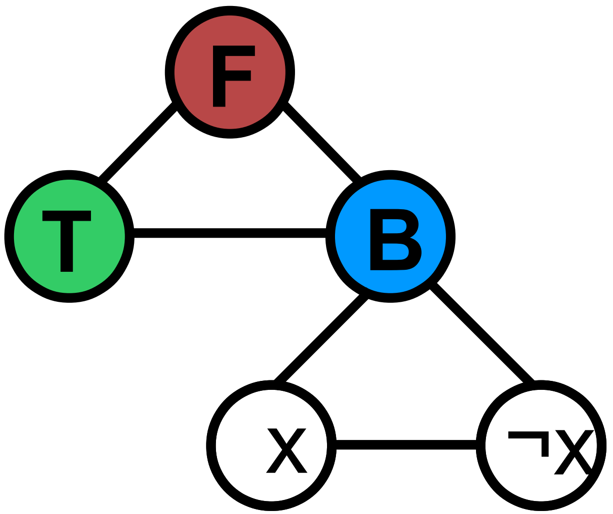
- Idea: T's color represents TRUE
  F's color represents FALSE

  - In a 3-coloring, all variable nodes must be colored T or F because?



Remark

- Idea: T's color represents TRUE
  F's color represents FALSE

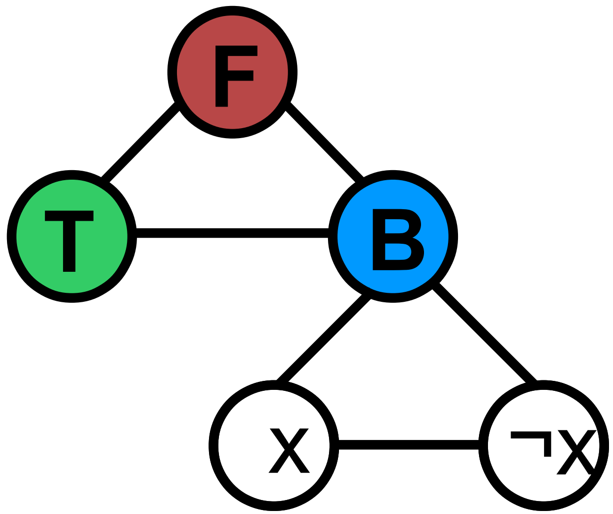


- In a 3-coloring, all variable nodes must be colored T or F because connected to **B**.

  Also, x and ¬x must have different colors because?

Remark

- Idea: T's color represents TRUE
  F's color represents FALSE



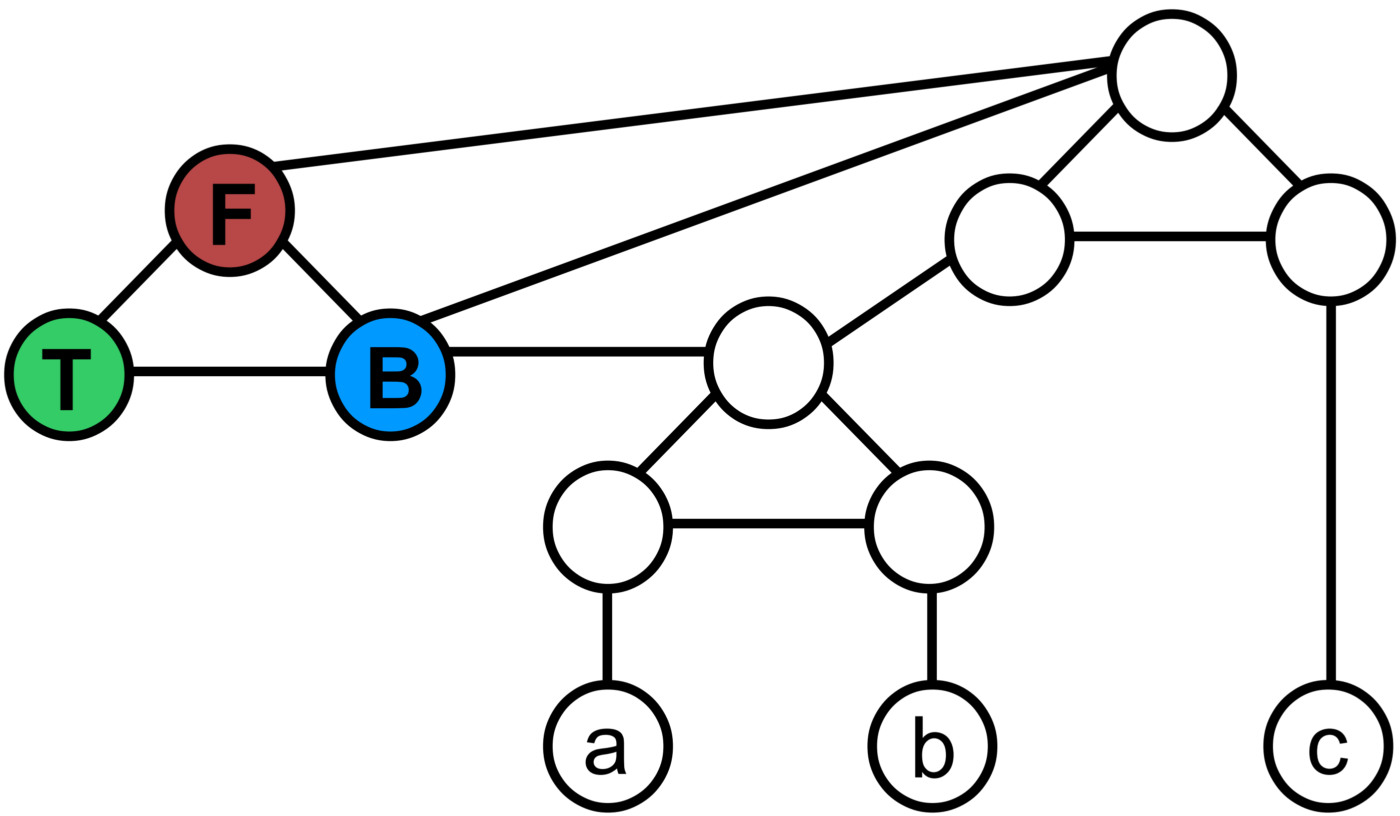
- In a 3-coloring, all variable nodes must be colored T or F because connected to B.

  Also, x and ¬x must have different colors because they are connected.

  So we can "translate" a 3-coloring of G_φ into a true/false assignment to variables of φ

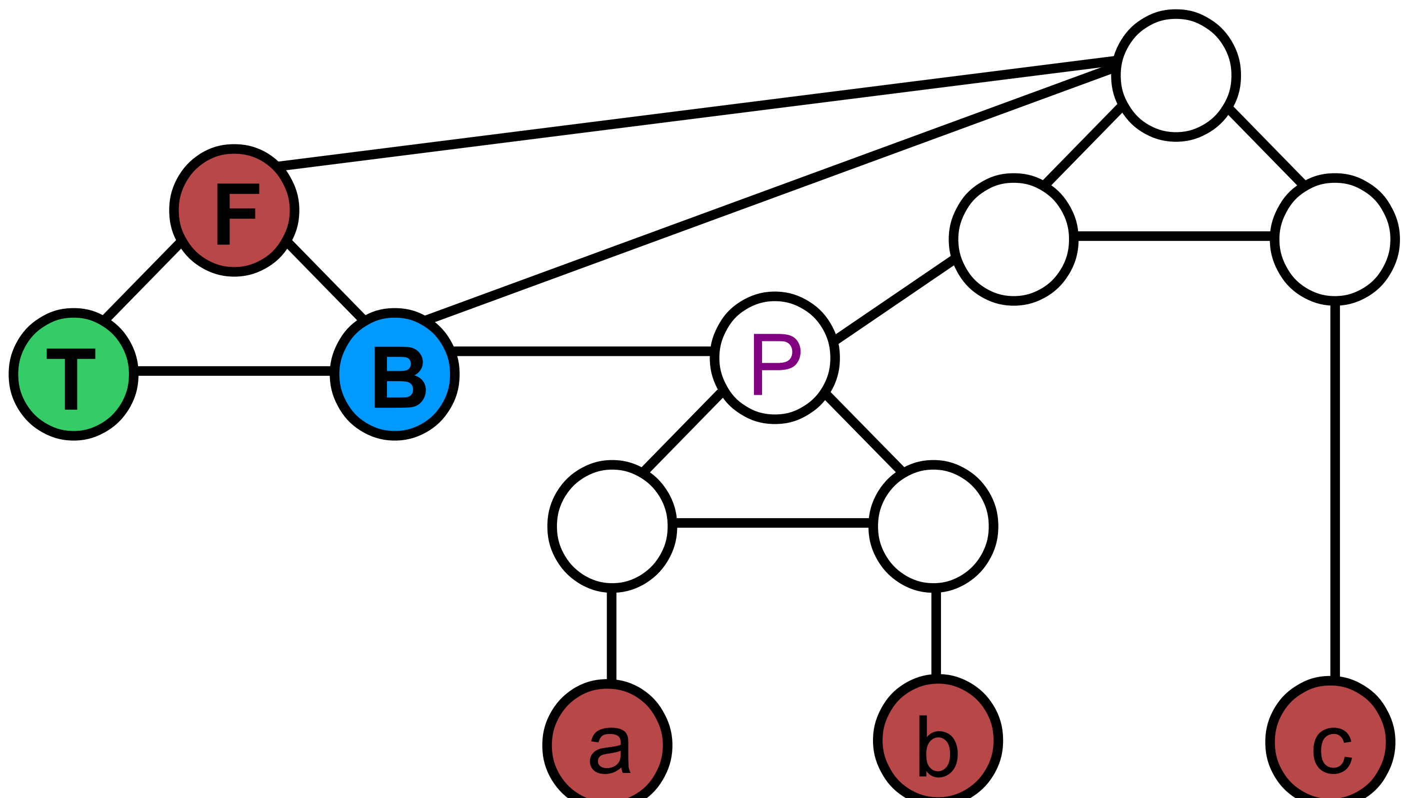
Fact: Graph below 3-colorable ⇔ a, b, or c colored T

The idea is simply that each triangle computes "Or"



Fact: Graph below 3-colorable ⇔ a, b, or c colored T

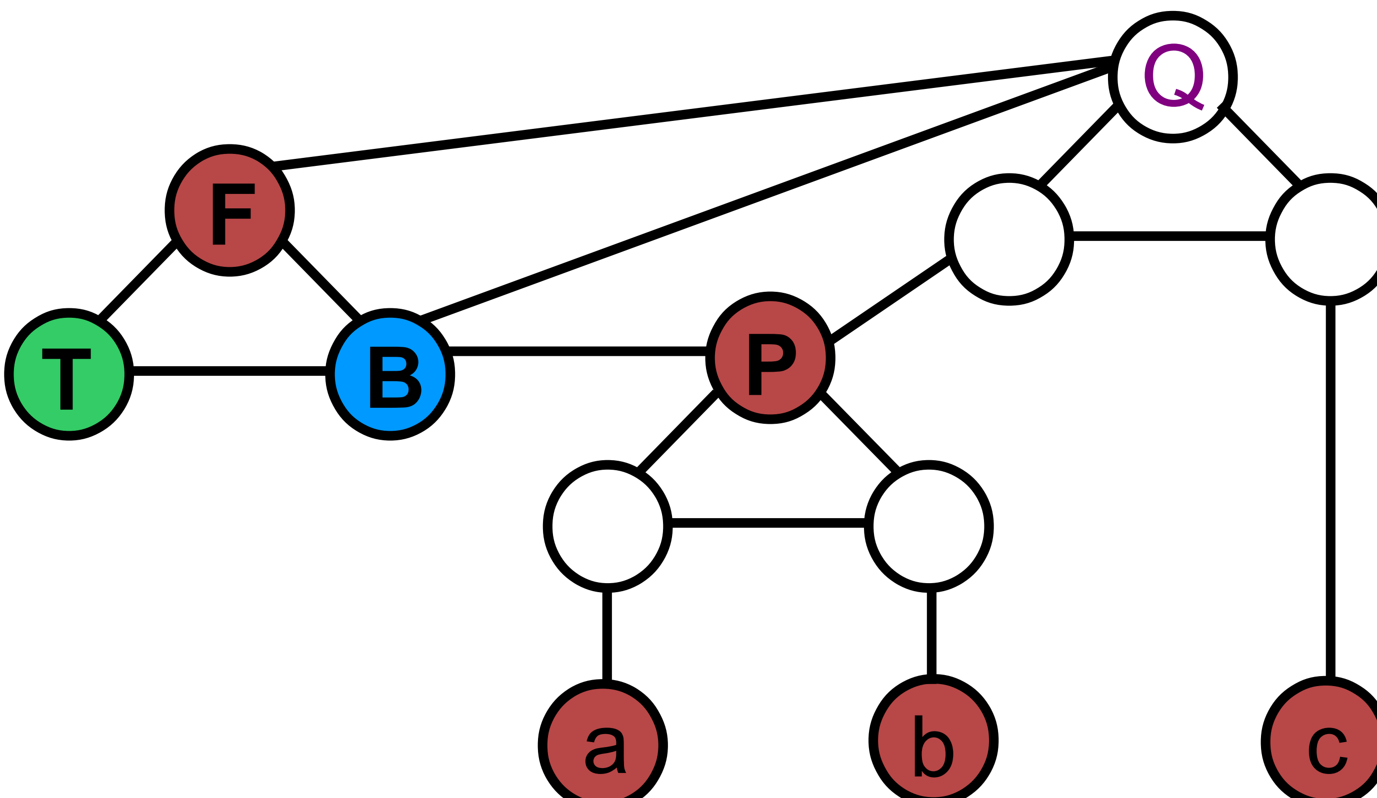
Proof of ⇒: Suppose by contradiction that a, b, and c are all colored F then P colored how?



Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇒: Suppose by contradiction that

a, b, and c are all colored F then P colored F.
Then Q colored how?

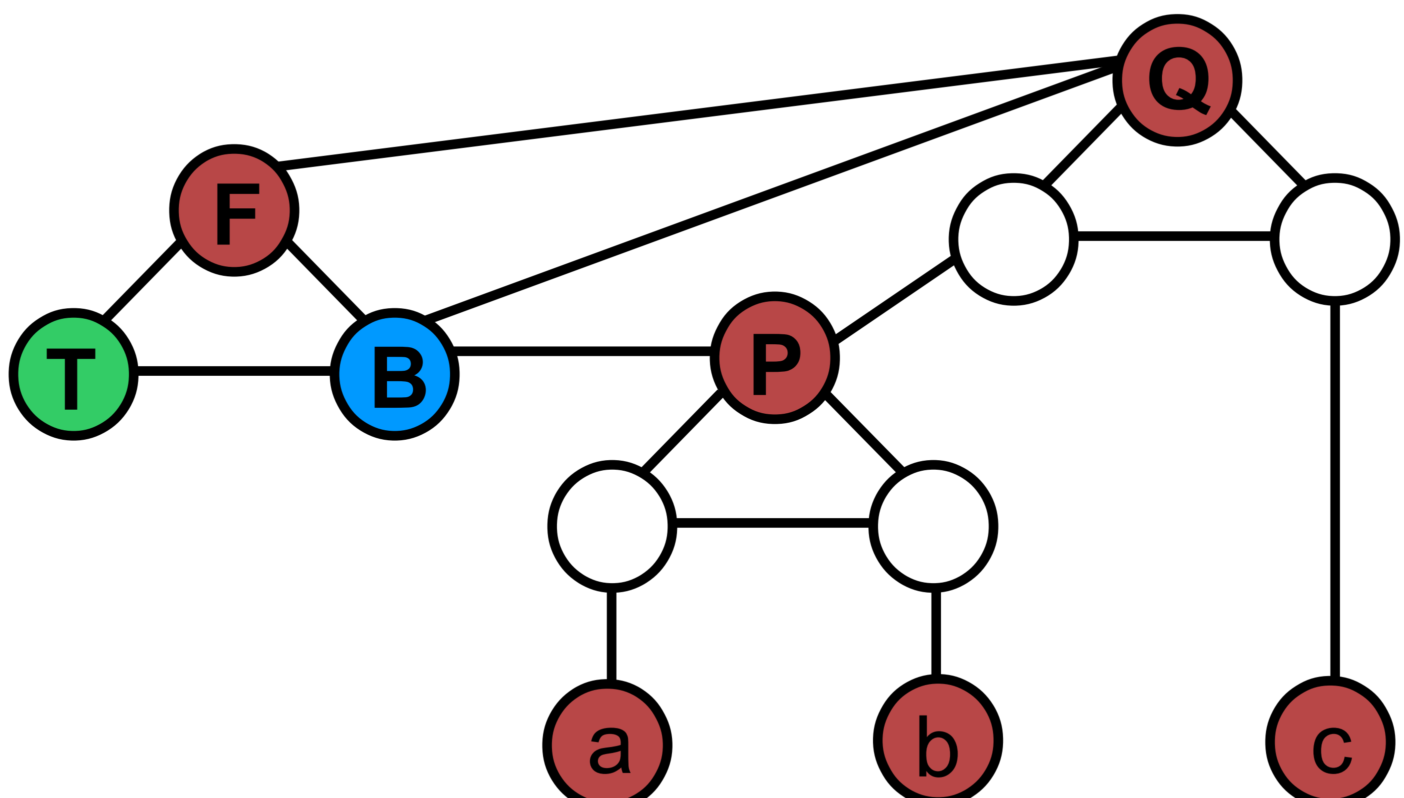


Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇒: Suppose by contradiction that

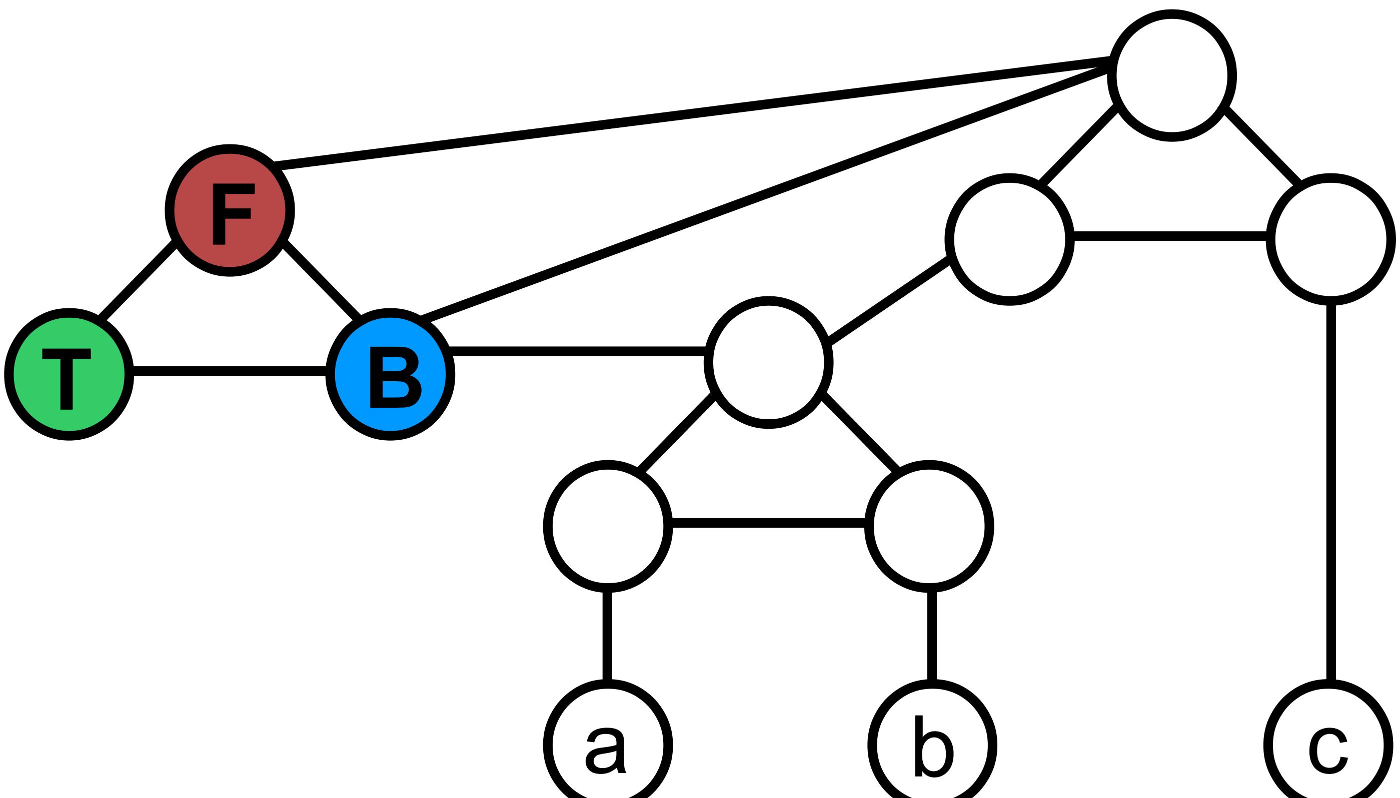
a, b, and c are all colored F then P colored F.
Then Q colored F. But this is not a valid 3-coloring

Done



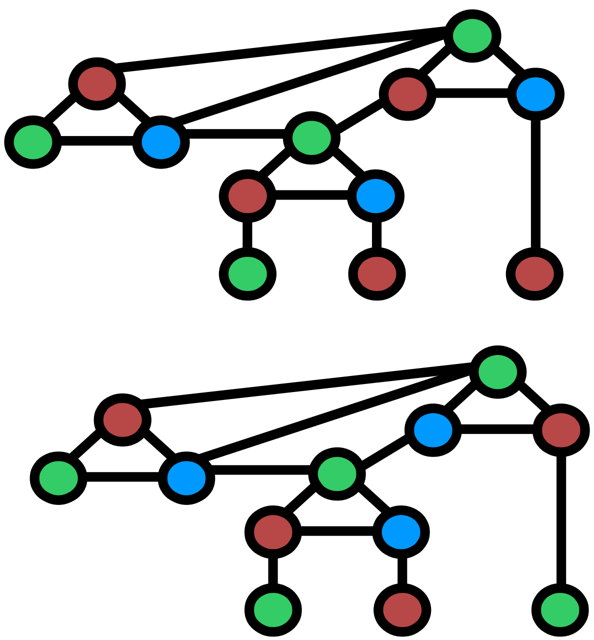
Fact: Graph below 3-colorable ⇔ a, b, or c colored T

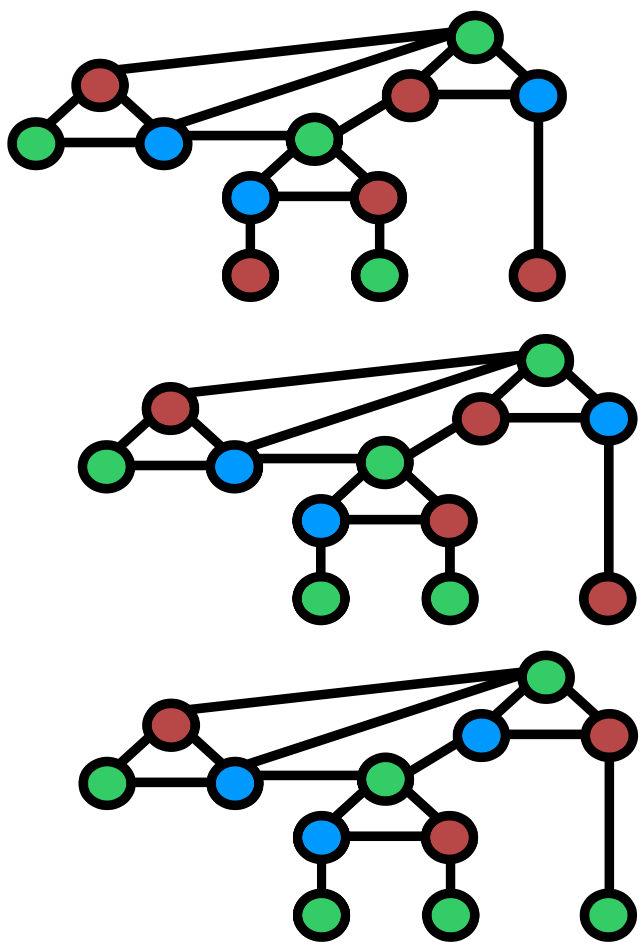
Proof of ⇐: We show a 3-coloring for each way in which a, b, and c may be colored

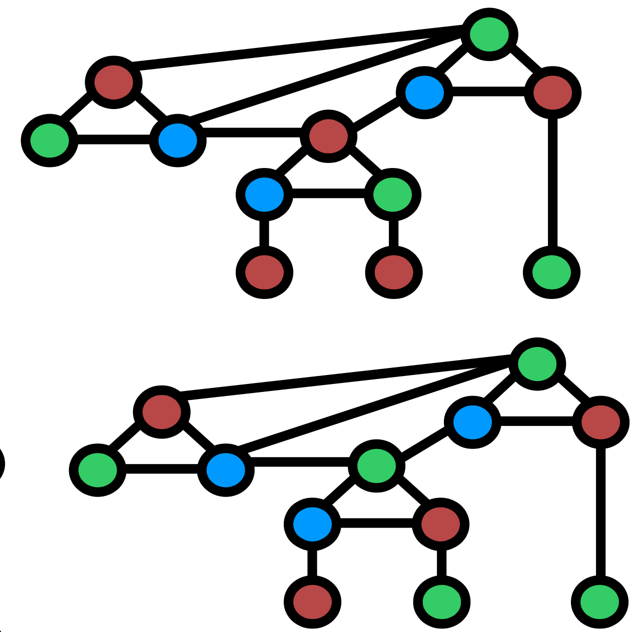


Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇐: We show a 3-coloring for each way in which a, b, and c may be colored

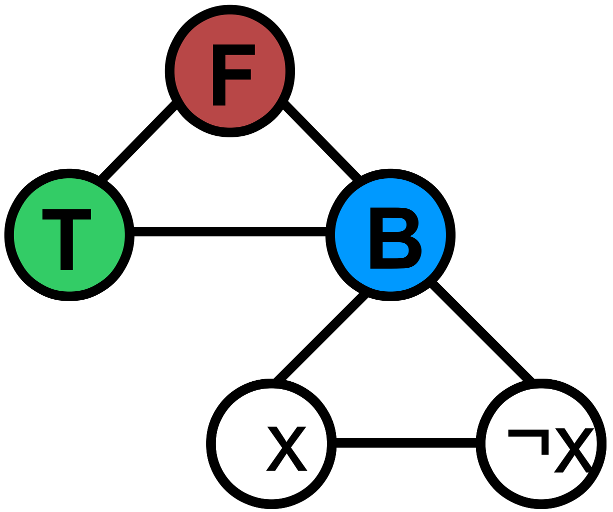




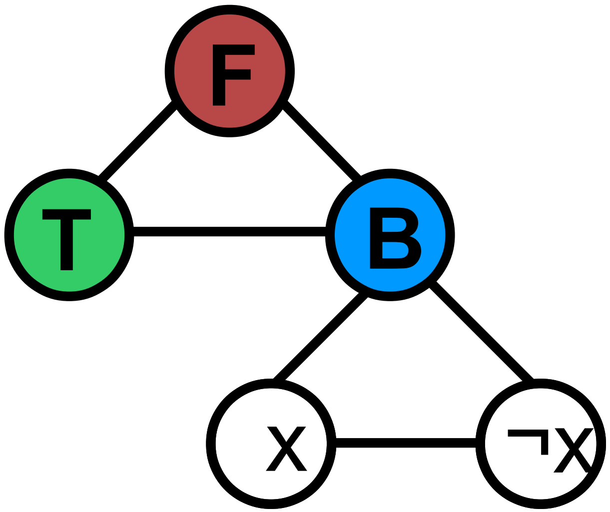


Done

- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: $\Rightarrow$
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly
  Ok because ?



- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: ⇨
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly
  Ok because they don't touch
  T or F in palette, and x and ¬ x
  are given different colors



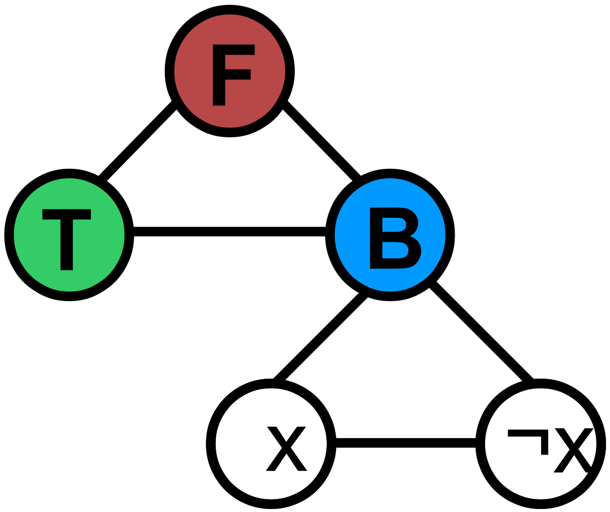
- Color clause nodes using previous Fact.
  Ok because?

- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: ⇨
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly

  Ok because they don't touch

  T or F in palette, and x and ¬ x

  are given different colors



- Color clause nodes using previous Fact.

  Ok because each clause has some true literal

- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: $\Leftarrow$
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because?

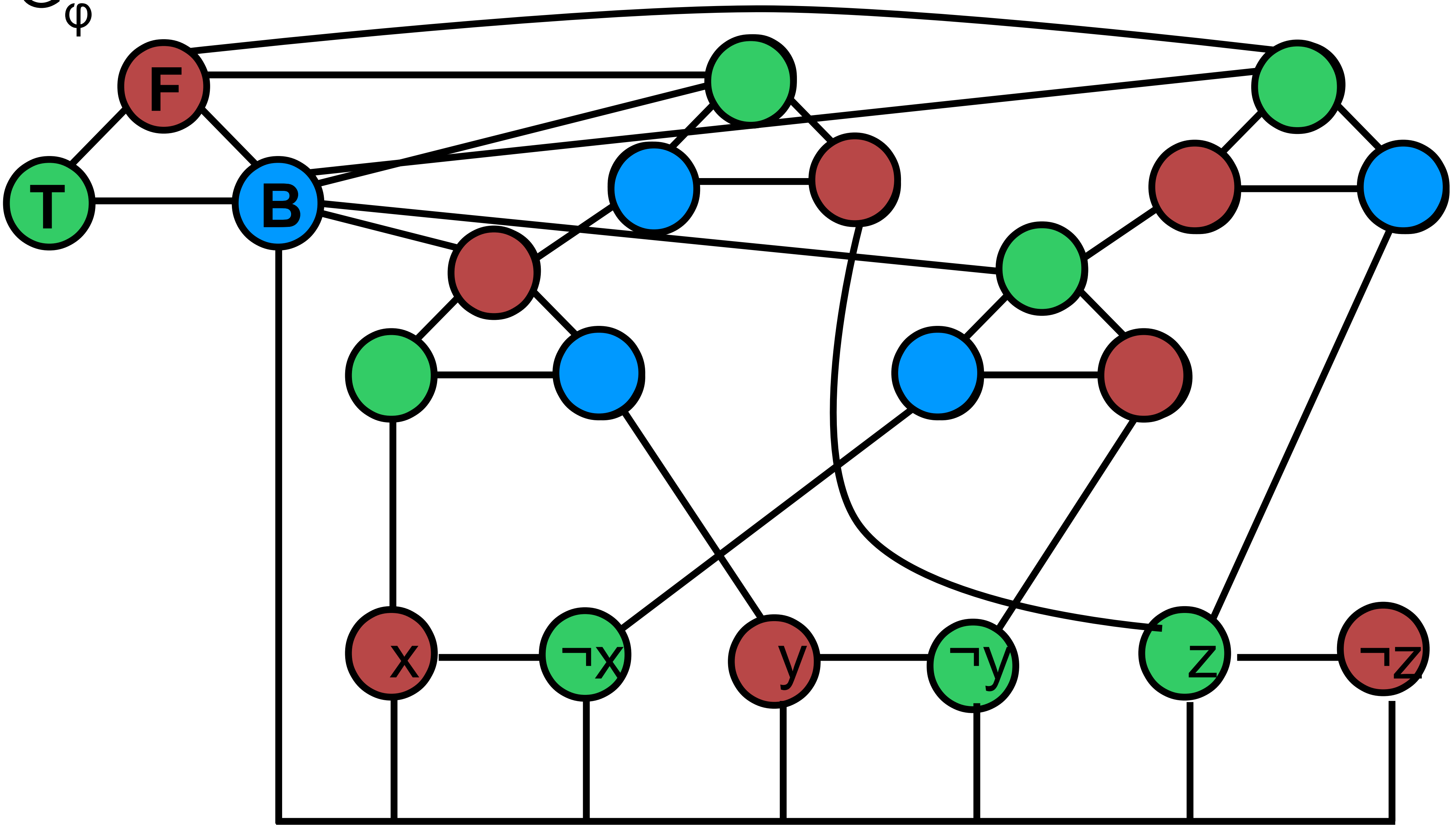
- Claim: $\varphi \in$ 3SAT ⇔ $G_\varphi \in$ 3COLOR
- Proof: ⇐
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because by Remark, x and ¬x are colored **T** or **F** and don't conflict.
- This gives some true literal per clause because?

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ $G_\varphi \in$ 3COLOR
- Proof: $\Leftarrow$
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because by Remark, x and ¬x are colored **T** or **F** and don't conflict.
- This gives some true literal per clause because clause is colored correctly, and by previous Fact
- All clauses are satisfied, so φ is satisfied.

Example: $\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z)$

Satisfying assignment: x = **0**, y = **0**, z = **1**

G_φ=



- It remains to argue that ???

- It remains to argue that R runs in polynomial time

- To add variable nodes and edges, cycle over $v \leq | \varphi |$ variables

- To add clause nodes and edges, cycle over $c \leq | \varphi |$ clauses

- Overall, $\leq | \varphi | + | \varphi |$, which is polynomial in input length $| \varphi |$

- End of proof that 3COLOR $\in$ P ⇨ 3SAT $\in$ P

COLORING NUGGET

- Definition: PLANAR-k-COLOR =

{G : G is a planar graph that can be colored with k colors.}

- PLANAR-2-COLOR is

COLORING NUGGET

- Definition: PLANAR-k-COLOR =

{G : G is a planar graph that can be colored with k colors.}

- PLANAR-2-COLOR is easy

- PLANAR-3-COLOR is

COLORING NUGGET

- Definition: PLANAR-k-COLOR =

{G : G is a planar graph that can be colored with k colors.}

- PLANAR-2-COLOR is easy

- PLANAR-3-COLOR is hard (variant of proof we saw)

- PLANAR-4-COLOR is

COLORING NUGGET

- Definition: PLANAR-k-COLOR =

{G : G is a planar graph that can be colored with k colors.}

- PLANAR-2-COLOR is easy

- PLANAR-3-COLOR is hard (variant of proof we saw)

- PLANAR-4-COLOR is easy (answer is always “YES”)

- We saw polynomial-time reductions

  from 3SAT to CLIQUE

  SUBSET-SUM

  3COLOR

  from CLIQUE to COVER BY VERTEXES

- There are many other polynomial-time reductions

- They form a fascinating web

- Coming up with reductions is “art”