Reductions

Thomas Schwarz, SJ

Reduction

- So far we have shown:
  - There are problems that are not tractable algorithmically
    - Halting Problem
  - We defined a class $\mathscr{P}$ of problems that are considered computationally tractable, even as the instance size scales up
  - We have defined a class $\mathscr{N}\mathscr{P}$ of problems where we can verify a solution effectively

Reduction

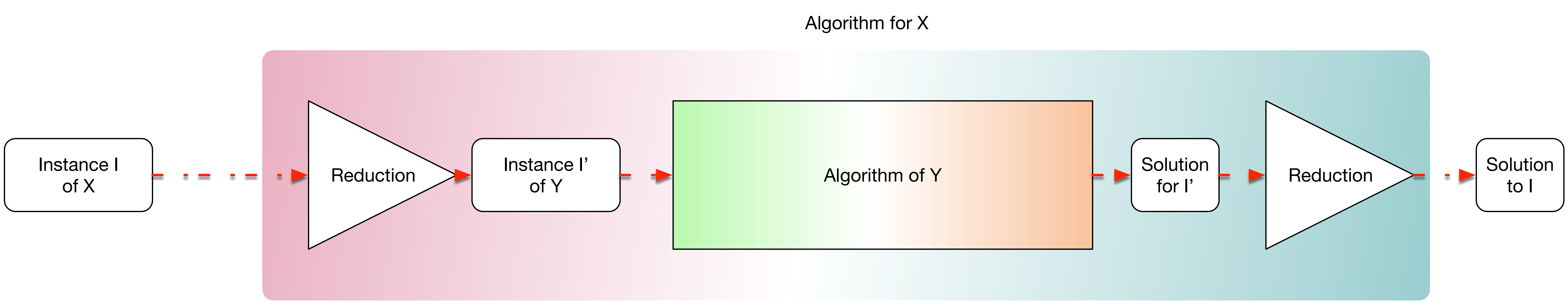
- Fundamental Conjecture in Complexity
  - $\mathscr{P} \neq \mathscr{NP}$
- Reason why people believe in this conjecture
  - There are problems that are $\mathscr{NP}$-complete
- A problem $P \in \mathscr{NP}$ is $\mathscr{NP}$-complete if
  - $(P \in \mathscr{P}) \Rightarrow (\mathscr{P} = \mathscr{NP})$

Reduction

- Can use the solution of one problem to solve another problem
- Example: Matrix multiplication and Matrix Squaring
  - If you can solve matrix multiplication, you can certainly solve matrix squaring
  - However, it also works the other way around:
    - Multiply square matrix A with square matrix B
    - Calculate left side of $\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}^2 = \begin{pmatrix} AB & 0 \\ 0 & BA \end{pmatrix}$
  - From the algorithmic standpoint: squaring a matrix is exactly as complicated as multiplying two matrices

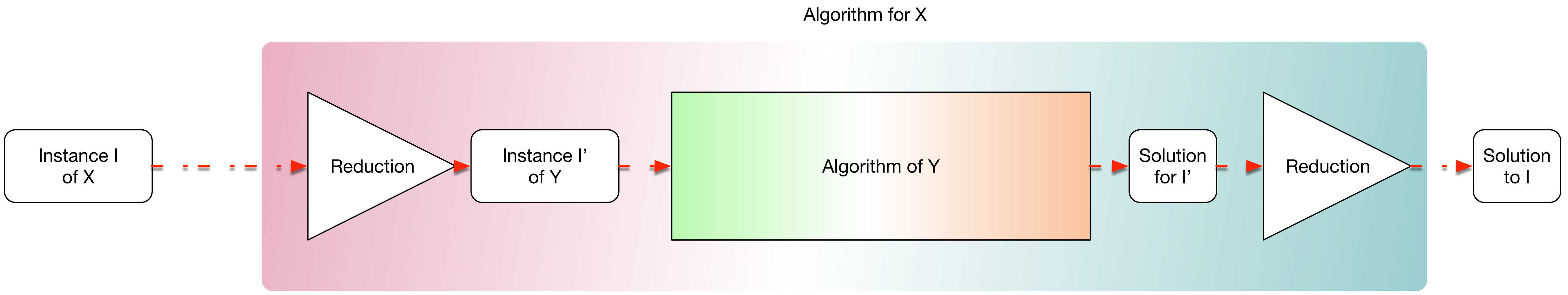
Reduction

- Formally, problem X reduces to problem Y if there is a reduction that converts instances of X to instances of Y and a translation that takes a solution of Y and makes it into a solution for X that solves the original instance.



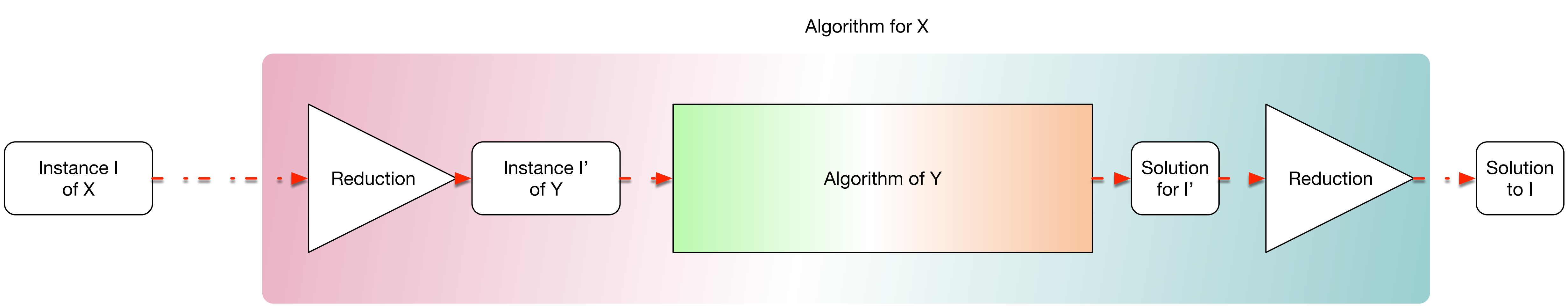
Reduction

- Matrix multiplication reduces to matrix squaring
  - Reduce: Given A, B, construct $\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$
  - calculate the square
  - translate result $\begin{pmatrix} AB & 0 \\ 0 & BA \end{pmatrix} \longrightarrow AB$



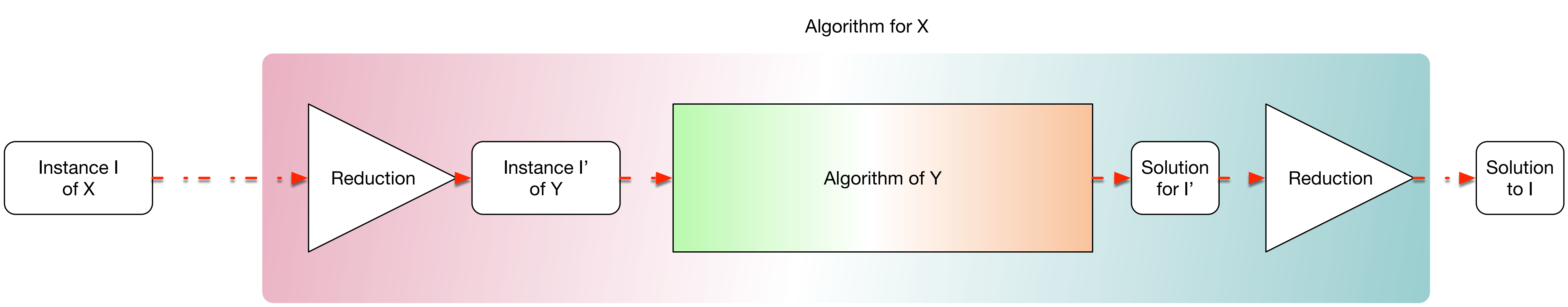
Reduction

- We usually apply reduction to existence problems
  - Answer is True/False
  - No translation is needed



Reduction

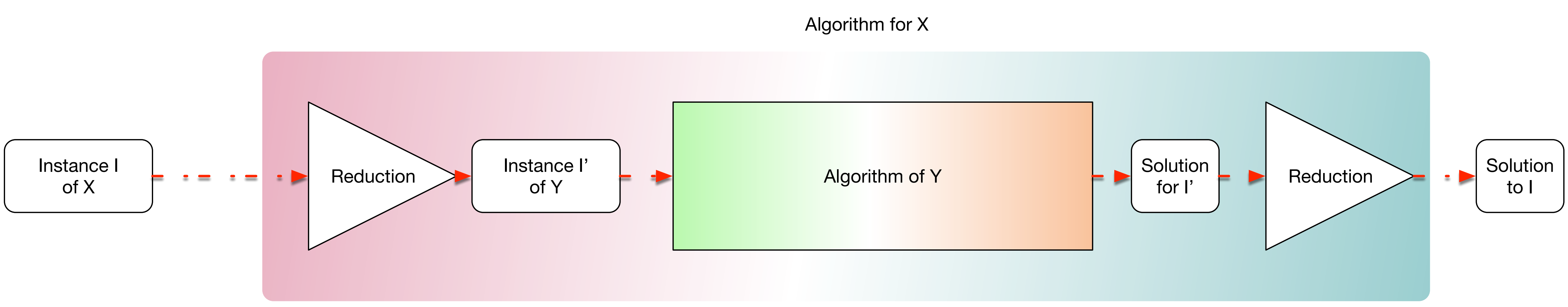
- X reduces ***polynomially*** to Y
  - Reduction works in polynomial time
- Write $X \leq_P Y$
  - Read(Y is at least as hard as X)



Reduction

- Assume $X \leq_p Y$ and $Y \in \mathscr{P}$. Then $X \in \mathscr{P}$.

- Proof: if we have a polynomial reduction, then the diagram below explains how we get a polynomial time algorithm for X



A New Formal Definition of $\mathscr{NP}$ - complete

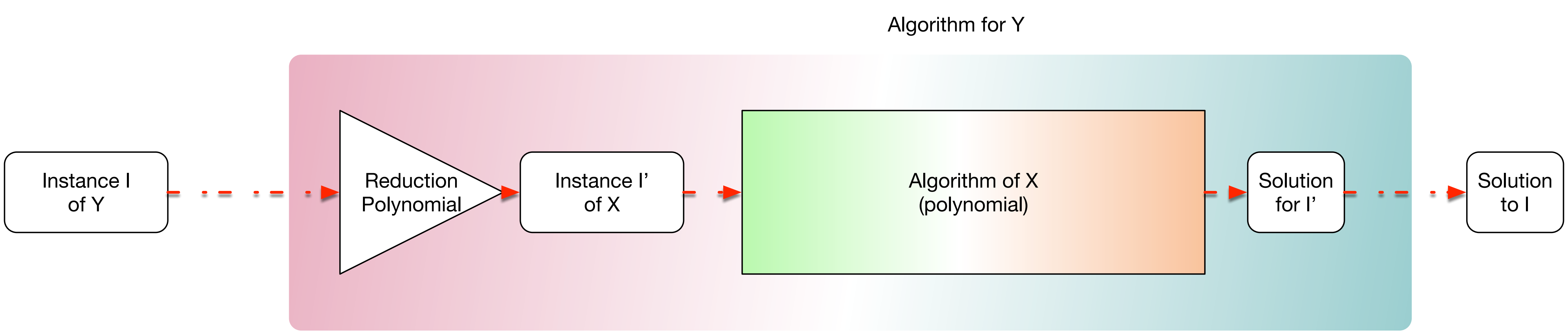
- $X \in \mathscr{NP}$ is $\mathscr{NP}-$complete
  - IF AND ONLY IF
- $\forall Y \in \mathscr{NP} \ : \ Y \leq_P X$

A New Formal Definition of $\mathcal{NP}$ - complete

- Theorem:
  - $X \in \mathcal{NP}$-complete AND $X \in \mathcal{P}$
    - IMPLIES
  - $\mathcal{P} = \mathcal{NP}$

A New Formal Definition of $\mathscr{NP}$ - complete

- Proof:
  - If $Y \in \mathscr{NP}$, then by completeness
  - $Y \leq_P X$
  - Thus:



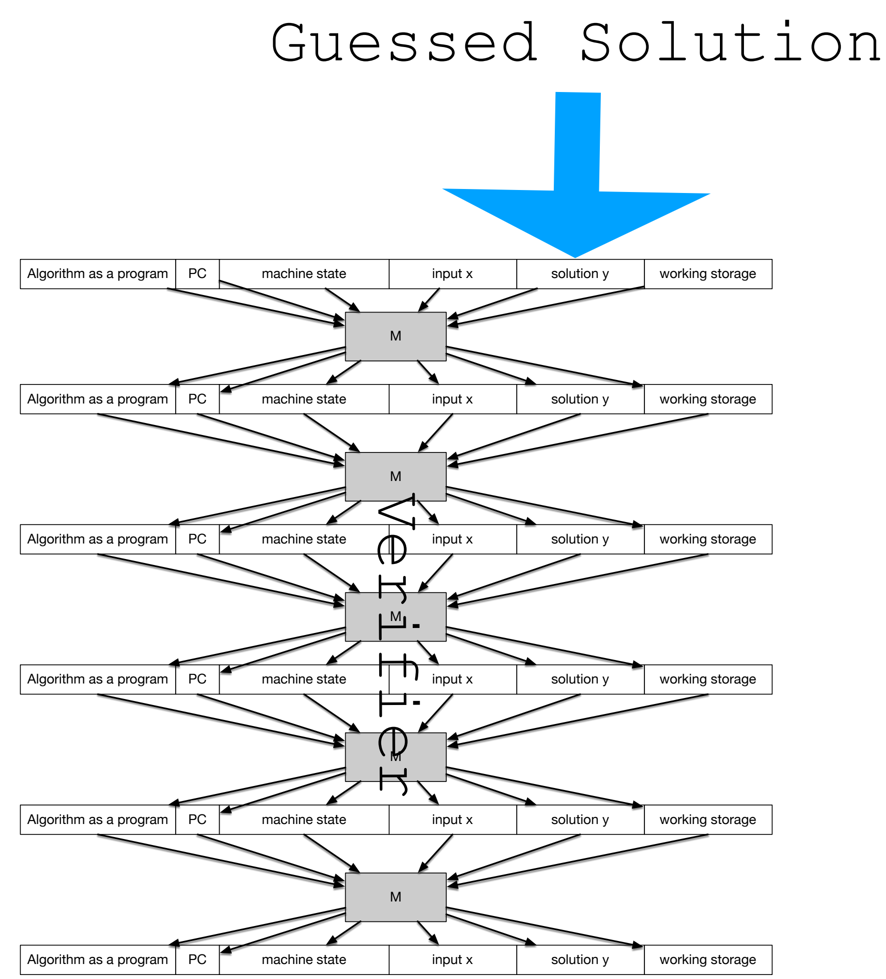
  - There is a polynomial time algorithm for Y
  - Thus: $Y \in \mathscr{P}$

A New Formal Definition of $\mathscr{NP}$ - complete

- Circuit Satisfiability is $\mathscr{NP}$ - complete
    - We have argued that we can express any problem in $\mathscr{NP}$ using a circuit

- This reduces the problem to Circuit Satisfiability

- Therefore:
    - Circuit Satisfiability remains $\mathscr{NP}-$complete



Family of $\mathscr{NP}$-complete problems

- Boolean Formula Satisfiability:
  - Given a boolean formula, determine whether there is an assignment to the variables such that the formula becomes true

$$(x \Rightarrow \bar{y}) \vee (x \wedge y \vee z) \vee (x \wedge \bar{y} \wedge \bar{z} \wedge (x \oplus y))$$

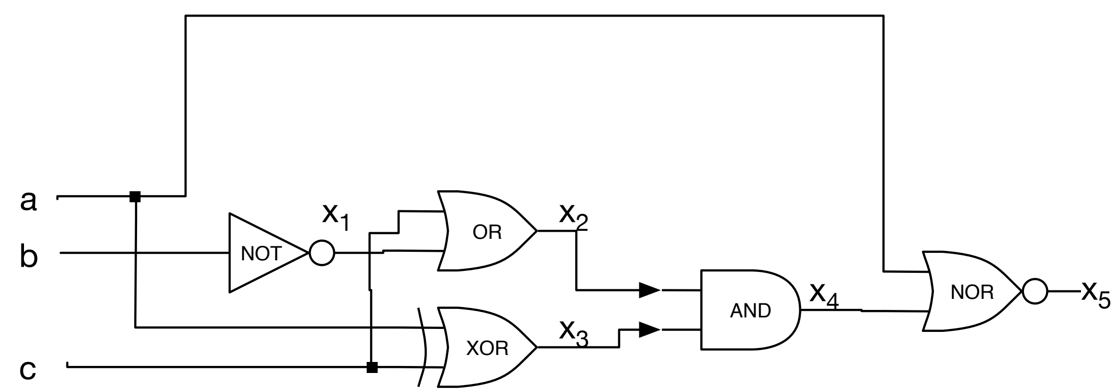
  - Given an assignment, can check the truth value of the formula in polynomial time
  - Thus, Boolean Formula Satisfiability is in $\mathscr{NP}$

Family of $\mathscr{NP}$-complete problems

- We can reduce boolean formula satisfiability to boolean circuit satisfiability
  - Need to show that boolean circuit satisfiability is at least as hard as boolean formula satisfiability
  - Given an instance of boolean circuit satisfiability, show that it can be reduced to boolean formula satisfiability

Family of $\mathscr{NP}$-complete problems

- Need to "translate" a boolean circuit into a formula



- Each circuit element becomes part of a Boolean formula
- $x_1 = \bar{b}, x_2 = c \vee x_1, x_3 = a \oplus c, x_4 = x2 \wedge x3$
  $x_5 = \neg(x_4 \vee a)$
- Make this into a single formula
- $x_1 = (\neg b) \wedge (x_2 = c \vee x_1) \wedge \ldots$

Family of $\mathscr{NP}$-complete problems

- Final formula is satisfiable exactly if the circuit is satisfiable
- Translation is in polynomial time

Family of $\mathscr{NP}$-complete problems

- 3-SAT : Satisfiability of a boolean formula in conjunctive normal form with clauses with three literals

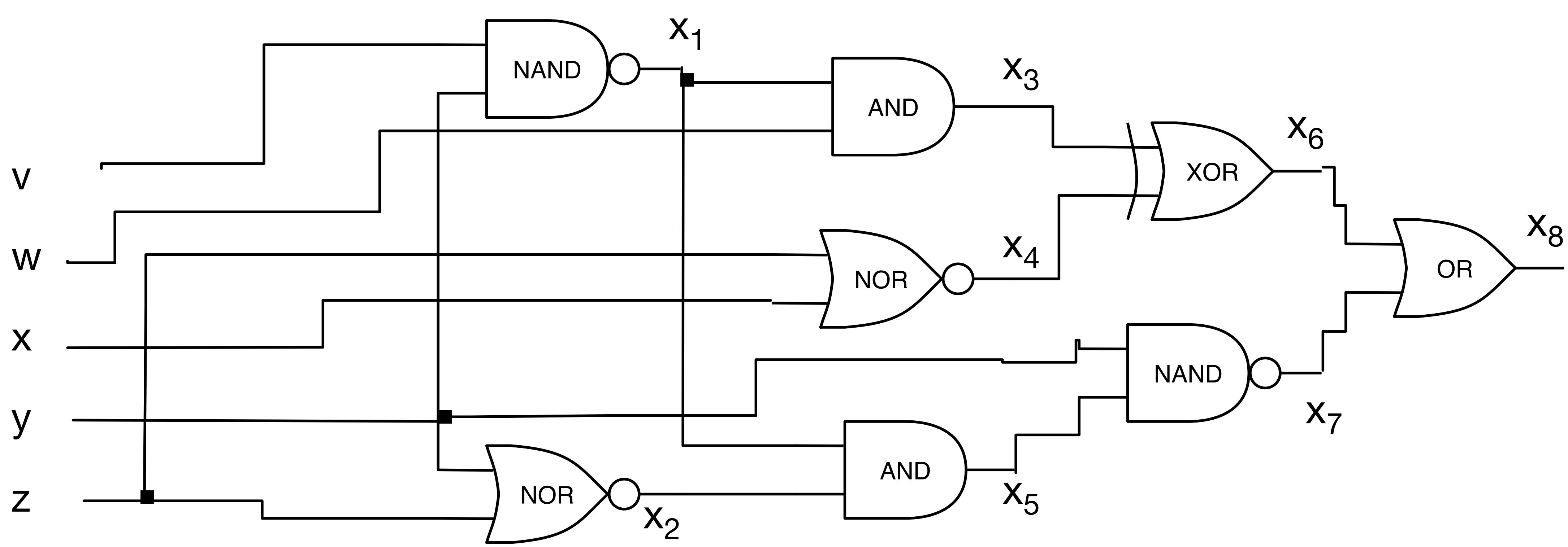
$$(a \vee \bar{b} \vee c) \wedge (a \vee a \vee c) \wedge (b \vee \bar{d} \vee d) \wedge (\bar{a} \vee \bar{d} \vee \bar{f})\ldots$$

Family of $\mathscr{NP}$-complete problems

- Reduction to Boolean Formula Satisfiability
  - Need to transform the boolean circuit satisfiability problem to 3-SAT
  - Reduction: $3-$SAT $\leq_P$ Boolean Circuit Sat
  - As before, describe the circuit in a boolean formula
  - However, now each gate can be expressed with only three-clause conditions:

Family of $\mathscr{NP}$-complete problems

- Translate the circuit into one with two-entry boolean gates



$$\big(x_1 = \neg(v \wedge y)\big) \wedge \big((x_2 = \neg(z \vee y))\big) \wedge \big(x_3 = x_1 \wedge w\big) \wedge \big(x_4 = \neg(x \vee z\big) \wedge$$
$$\big(x_5 = x_1 \wedge x_2\big) \wedge \big(x_6 = x_3 \oplus x_4\big) \wedge \big(x_7 = \neg(y \wedge x_5)\big) \wedge \big(x_8 = x_6 \vee x_7\big) \wedge x_8$$

Family of $\mathscr{NP}$-complete problems

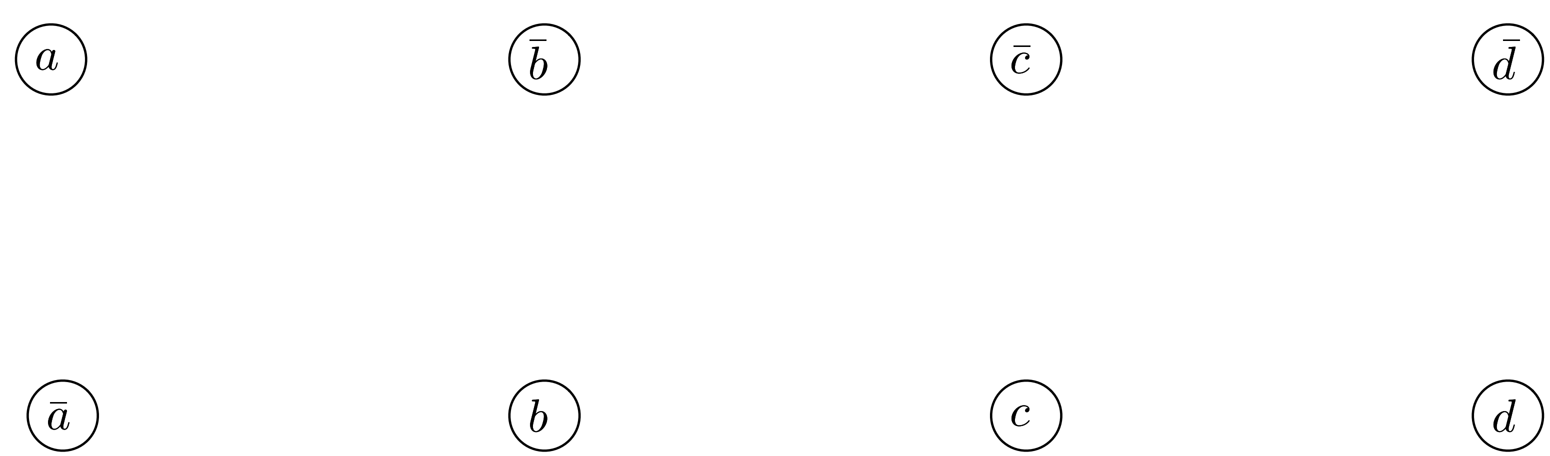
- Clauses directly represent gates
- Transform the clauses into conjunctive normal form with three literals
  - Trick
    - $x = (x \vee p \vee q) \wedge (x \vee \neg p \vee q) \wedge (x \vee p \vee \neg q) \wedge (x \vee \neg p \vee \neg q)$
    - $x \vee y = (x \vee y \vee p) \wedge (x \vee y \vee \neg p)$
- This blows up the representation, but only by a constant amount

Family of $\mathscr{NP}$-complete problems

- As before, the circuit is satisfiable if and only if the formula is satisfiable
- Ergo: We can solve 3-SAT implies we can solve Boolean Circuit Satisfiability

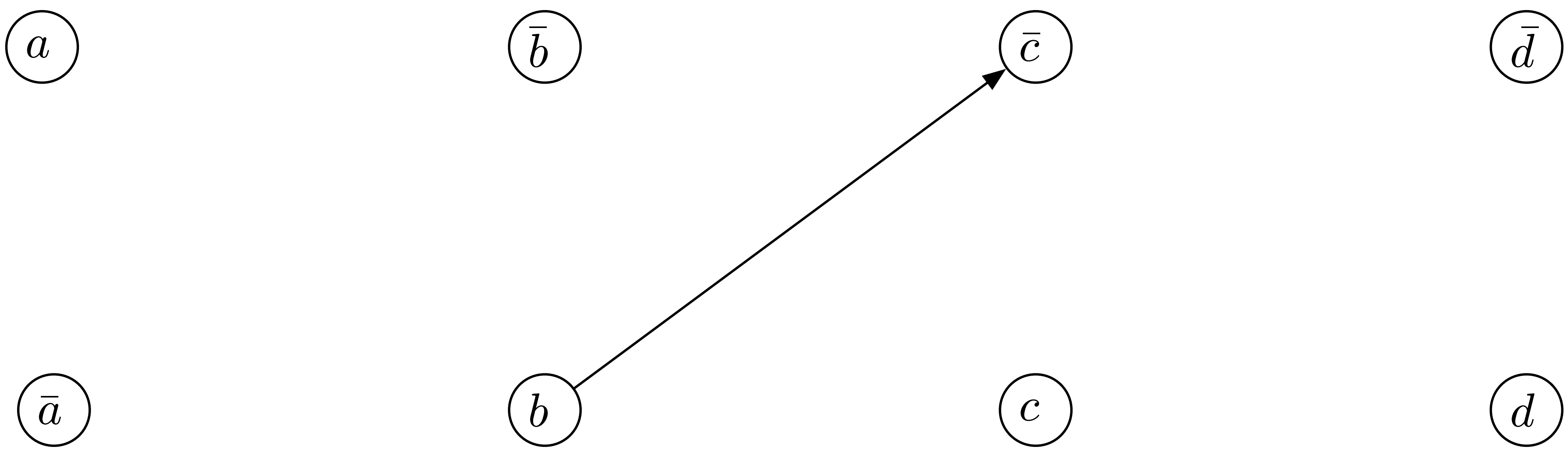
Family of $\mathscr{NP}$-complete problems

- What about 2-SAT
  - Clauses are of the form $(a \vee b)$, which is equivalent to $\neg a \Rightarrow b$
  - Create a graph:
    - Vertices are variables and their negation



Family of $\mathscr{NP}$-complete problems

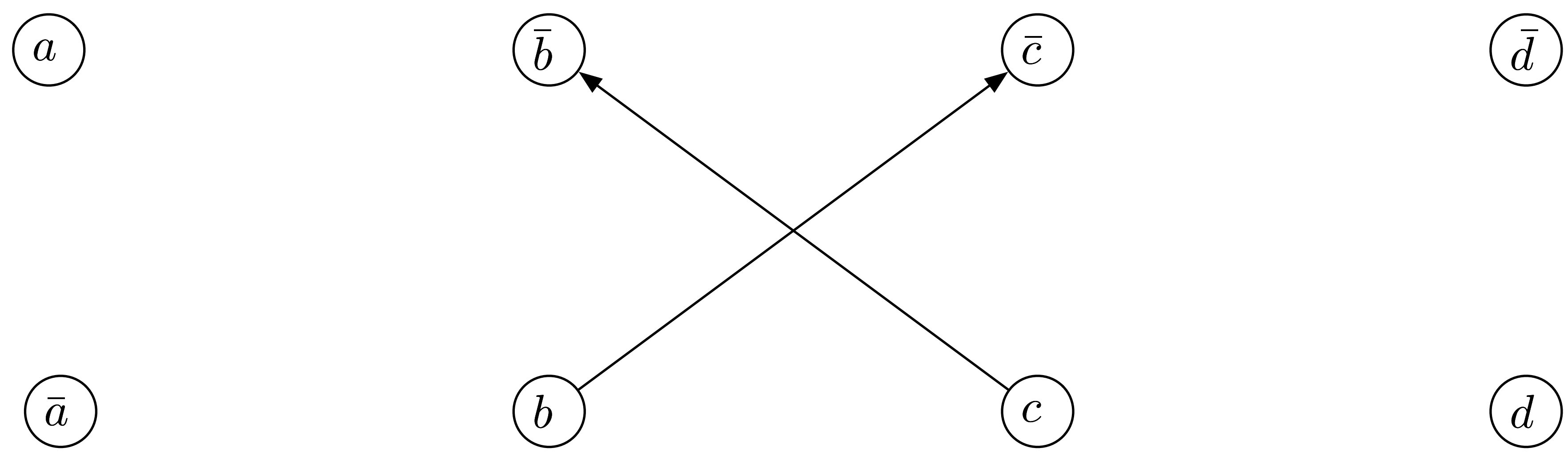
- Draw an edge if there is a clause equivalent to $x \Rightarrow y$ in the formula



$b \Rightarrow \neg c$ is equivalent to $(\neg b \vee \neg c)$

Family of $\mathscr{NP}$-complete problems

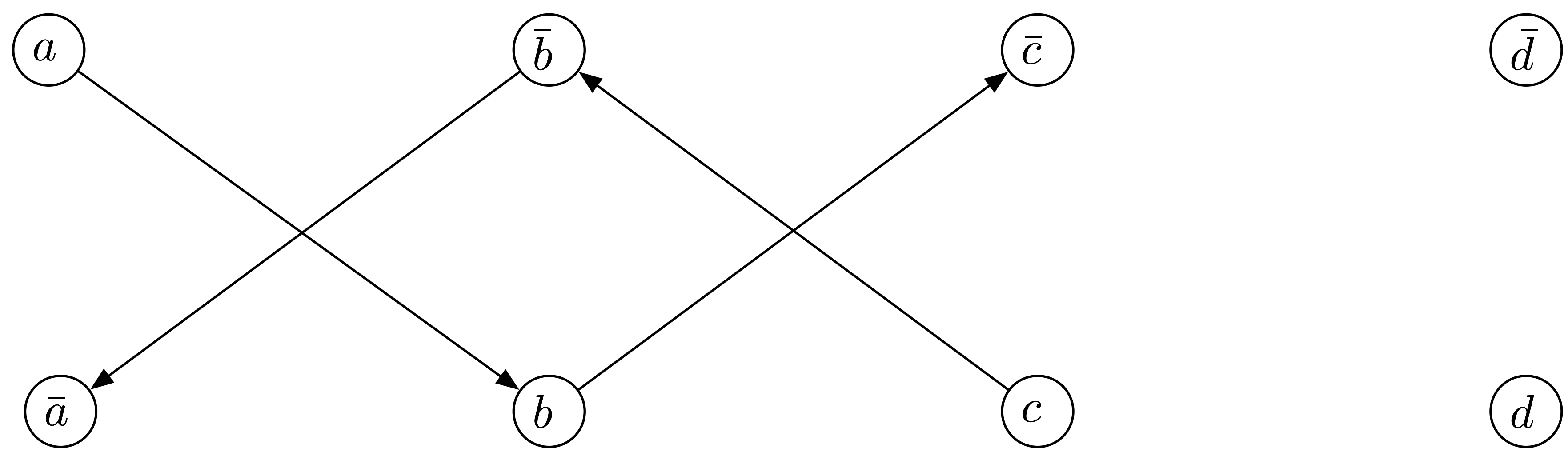
- Draw an edge if there is a clause equivalent to $x \Rightarrow y$ in the formula



$b \Rightarrow \neg c$ is equivalent to $(\neg b \vee \neg c)$
which is equivalent also to $c \Rightarrow \neg b$

Family of $\mathscr{NP}$-complete problems

- Draw an edge if there is a clause equivalent to $x \Rightarrow y$ in the formula



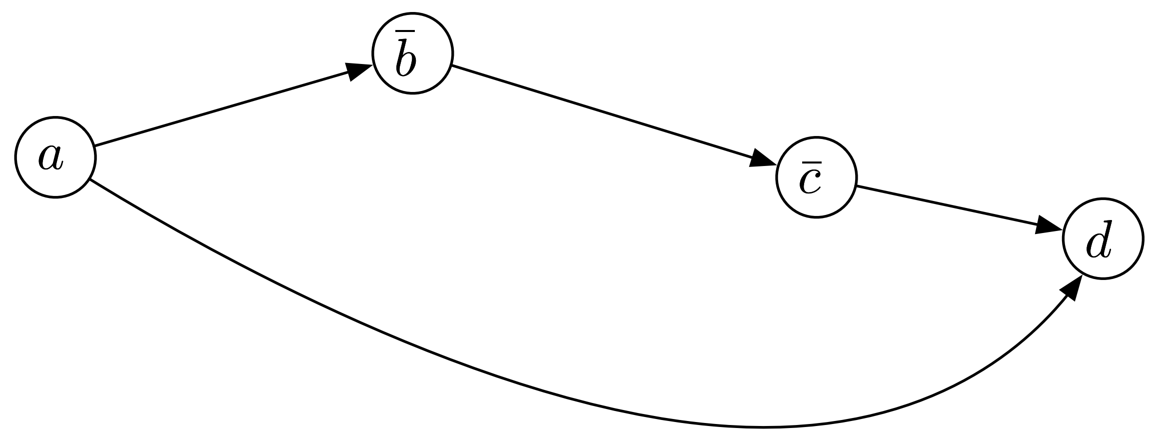
$a \Rightarrow b$ means $(\neg a \vee b)$
which also means $\neg b \Rightarrow \neg a$

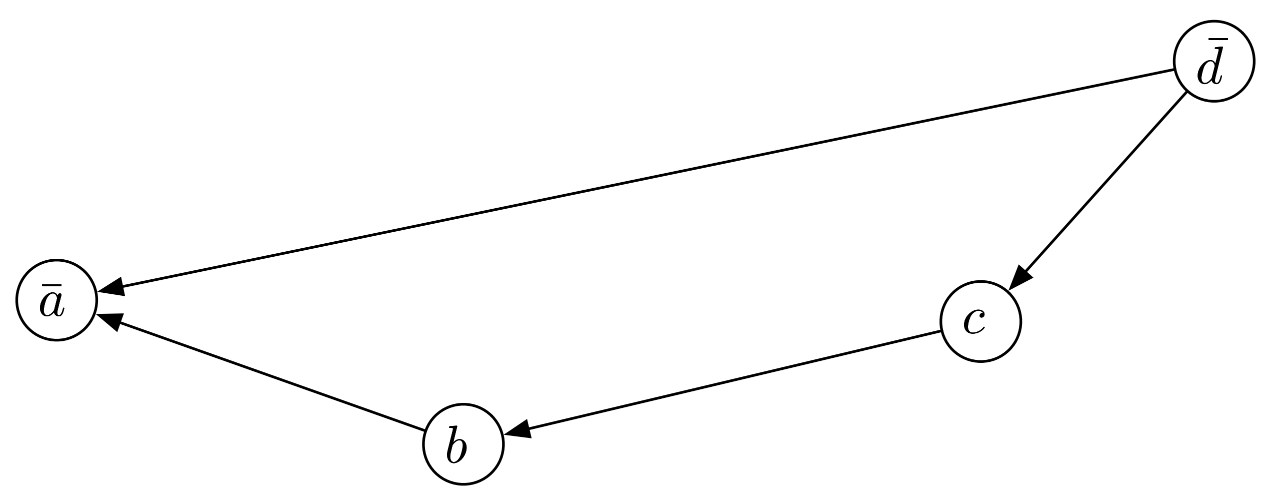
Family of $\mathcal{NP}$-complete problems

- Assume that x and $\neg x$ are in a cycle. Then we can conclude that x implies $\neg x$ that implies x.
- Thus, there is no possibility to find a truth assignment that works
- If there is never such a cycle, then we just start with one variable and assign it true, then we follow the implications et cet.

Family of $\mathscr{NP}$-complete problems

- Example:
  - Assign b=True
  - Then a=False
  - Then c=True
  - Then d=False



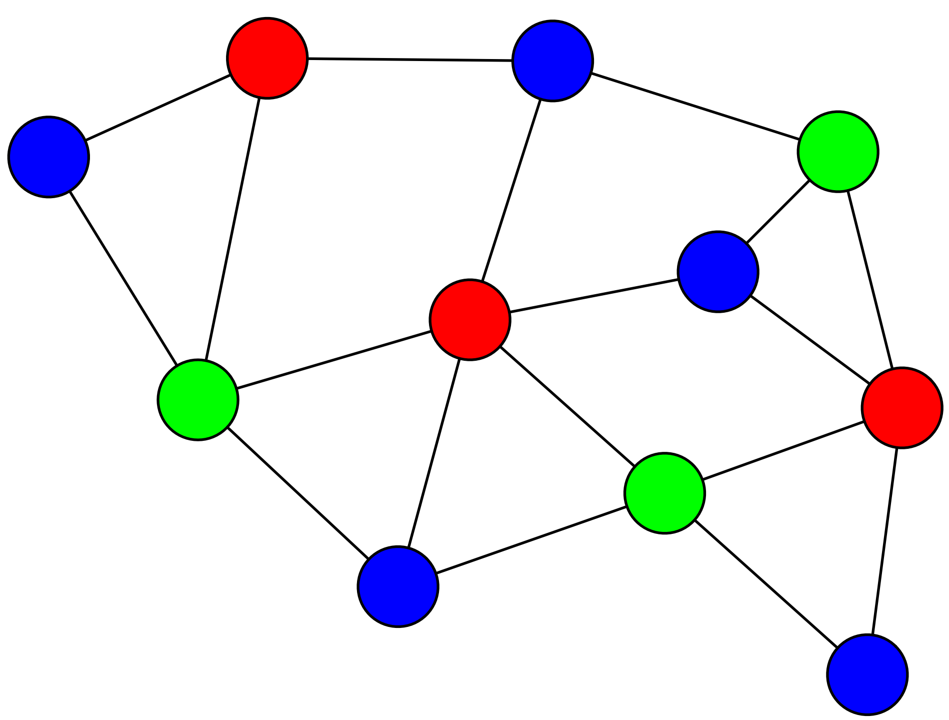


Family of $\mathscr{NP}$-complete problems

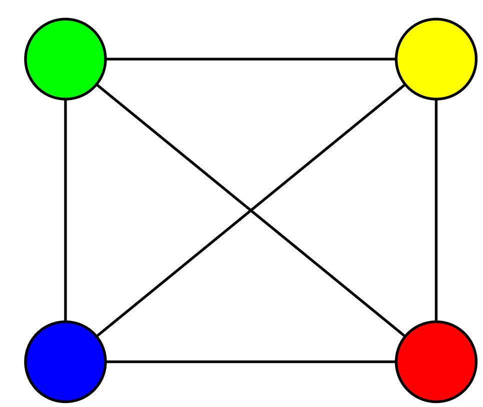
- To calculate this, we calculate the strongly connected components
- If there is a component that contains at most one of a variable and its negation, then assign those literals True
  - By working through all the components, we obtain a satisfiable assignment
- If there is a component that contains both a variable and its negative, then the formula is not satisfiable
- ERGO: we can solve 2-SAT with DFS in polynomial time

Graph 3-Colorability

- Graph 3-colorability
  - Given a graph, can we color its vertices with three colors such that no edge is between vertices with the same color



3 colorable



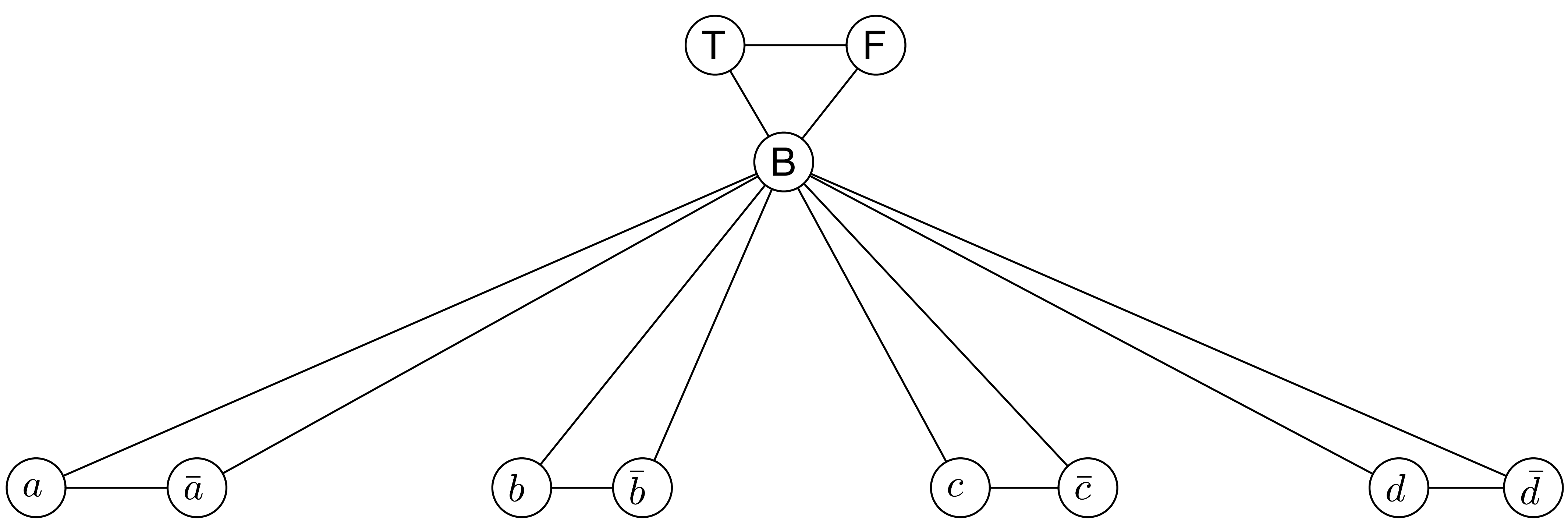
not 3 colorable

Graph 3-Colorability

- Claim: $3-$SAT $\leq_P 3-$color
- Given a $3-$SAT instance, we construct (in polynomial time) an instance of $3-$color such that the graph is colorable if and only if the boolean formula is satisfiable

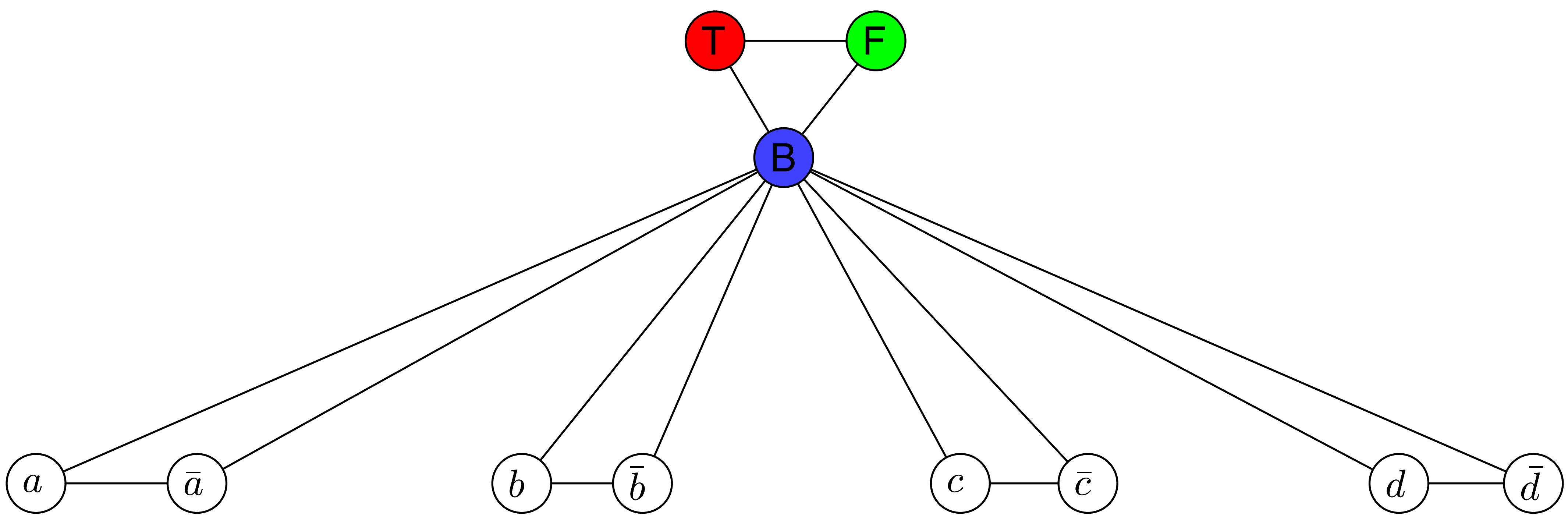
Graph 3-Colorability

- We create a node for each variable and its negation
- We also create a triangle with labels B, False, and True
- Connect as shown below



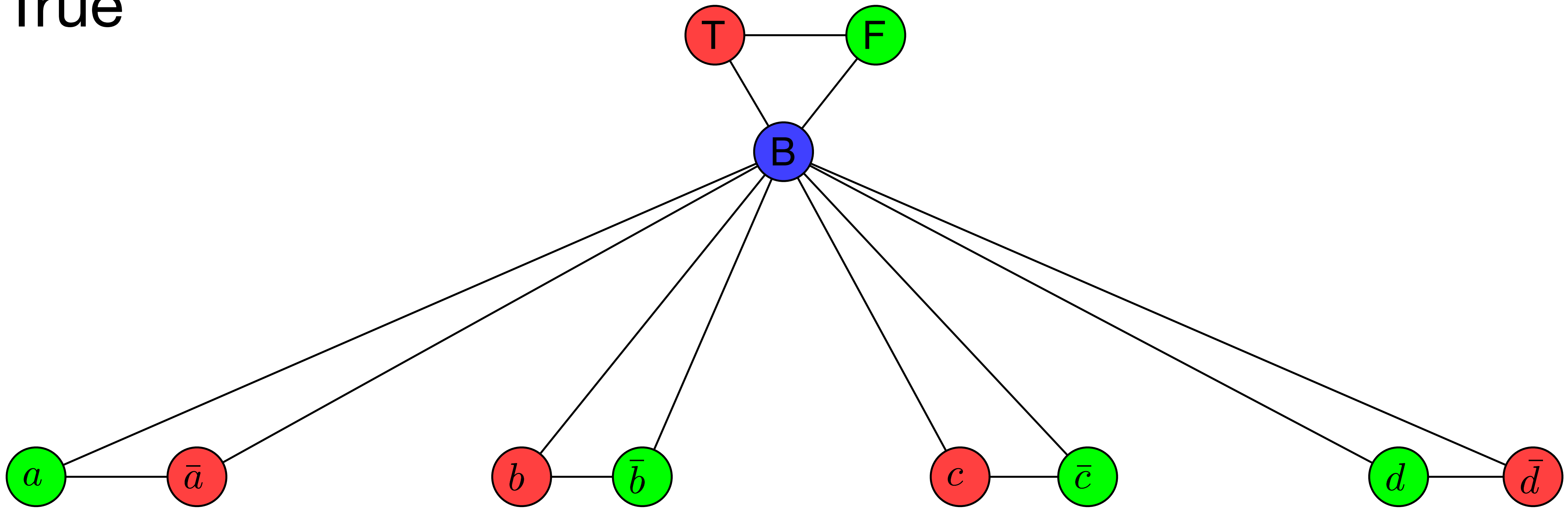
Graph 3-Colorability

- This graph is clearly colorable
  - Start with the upper triangle



Graph 3-Colorability

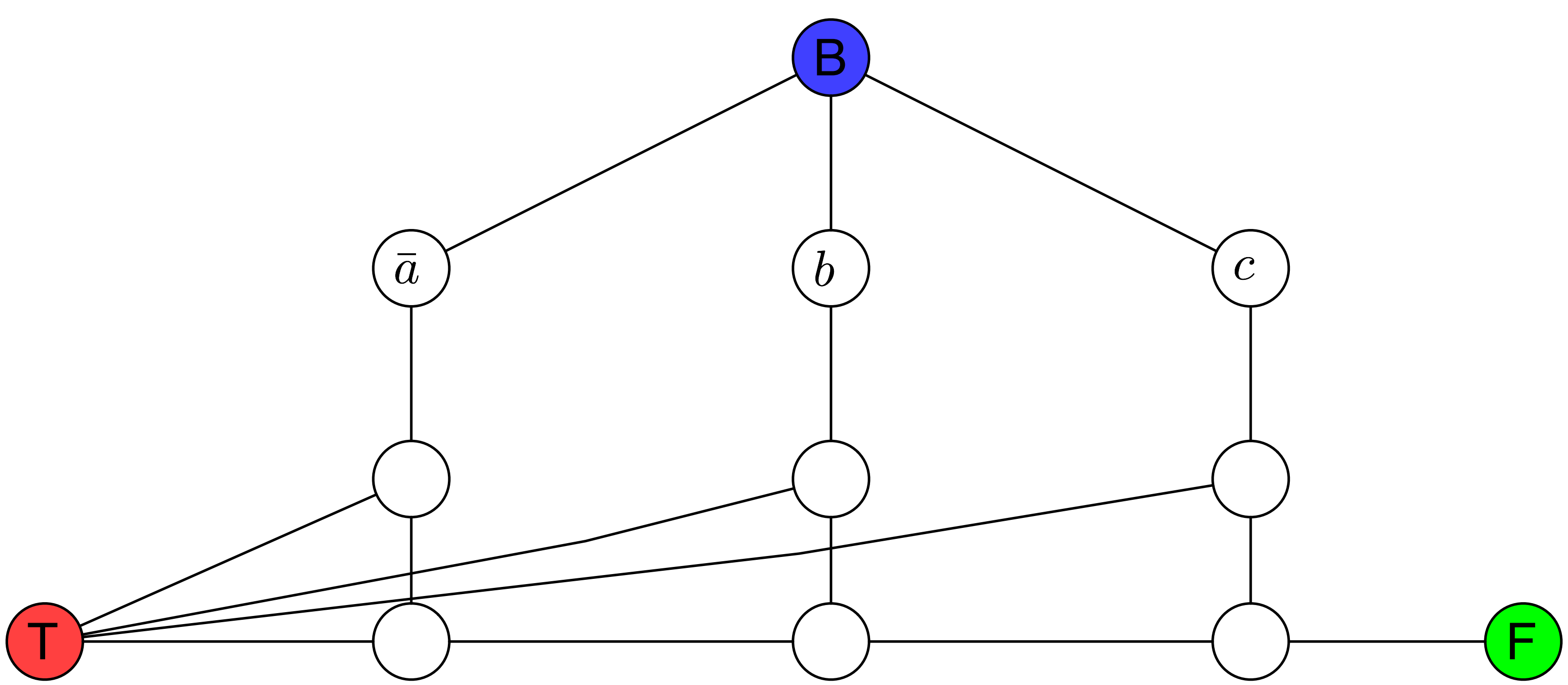
- Then for each variable, we get to decide whether we want to color the variable or its negation with the same color as True



- This "encodes" a truth assignment to the variables

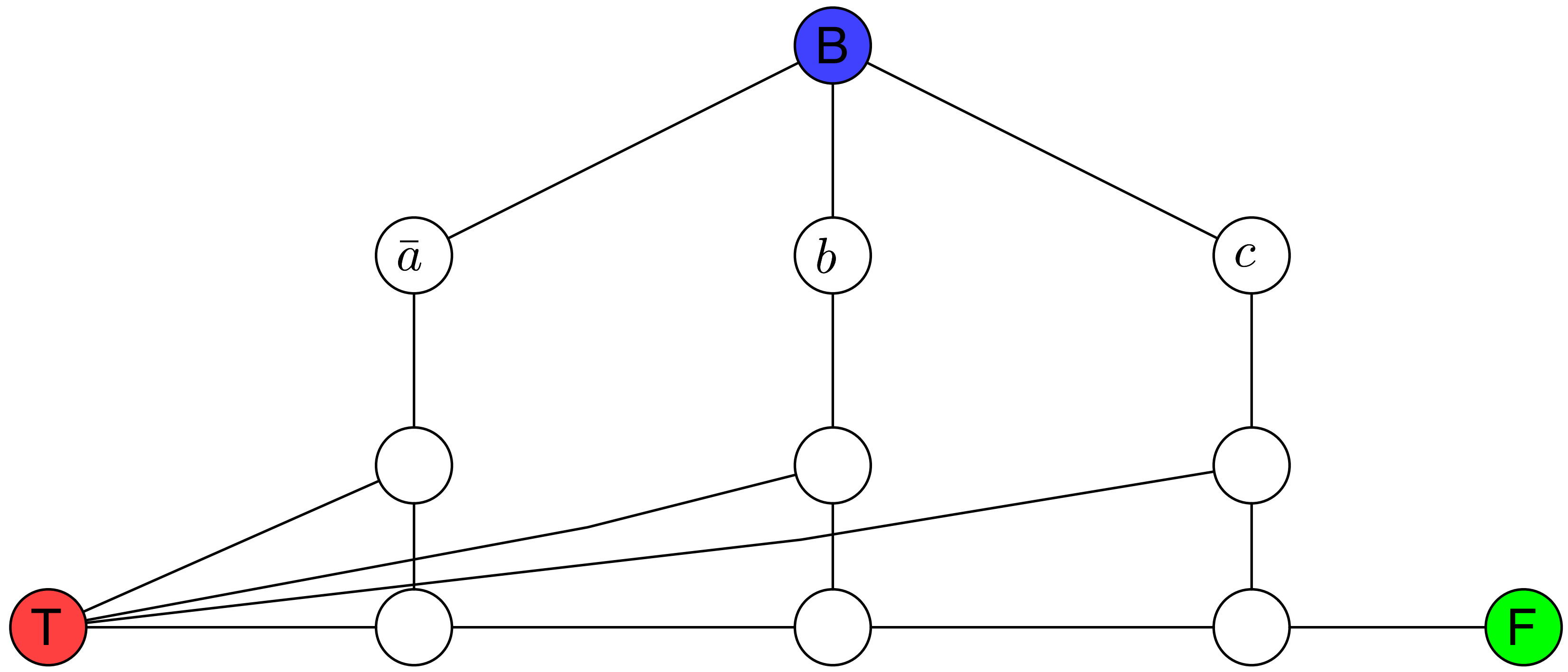
Graph 3-Colorability

- Add for each clause in the formula a 6-node 13-edge gadget
  - Add the six nodes, then add edges as shown below
- Example: $\neg a \vee b \vee c$



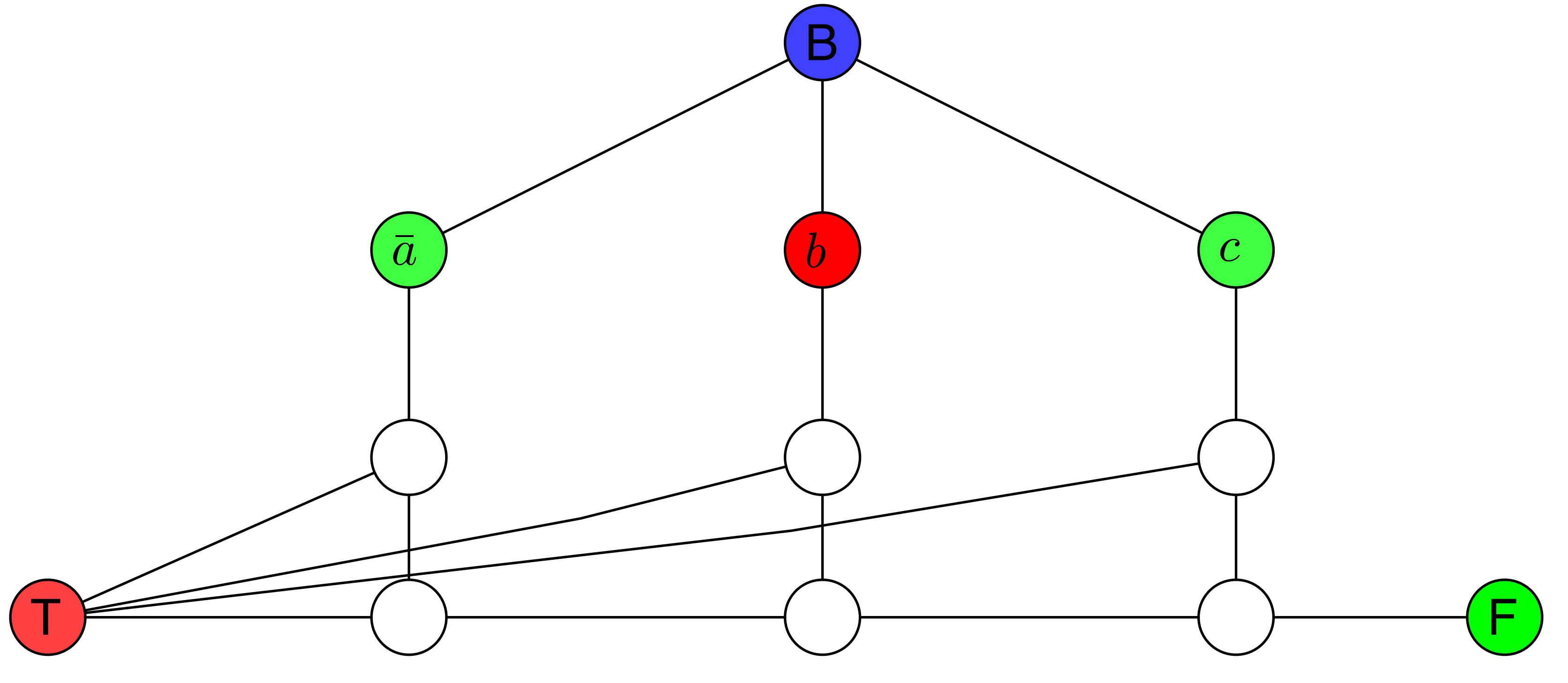
Graph 3-Colorability

- If the graph is three colorable, then:
  - The gadget ensures that at least one element in the clause is True
- Example: $\neg a \vee b \vee c$



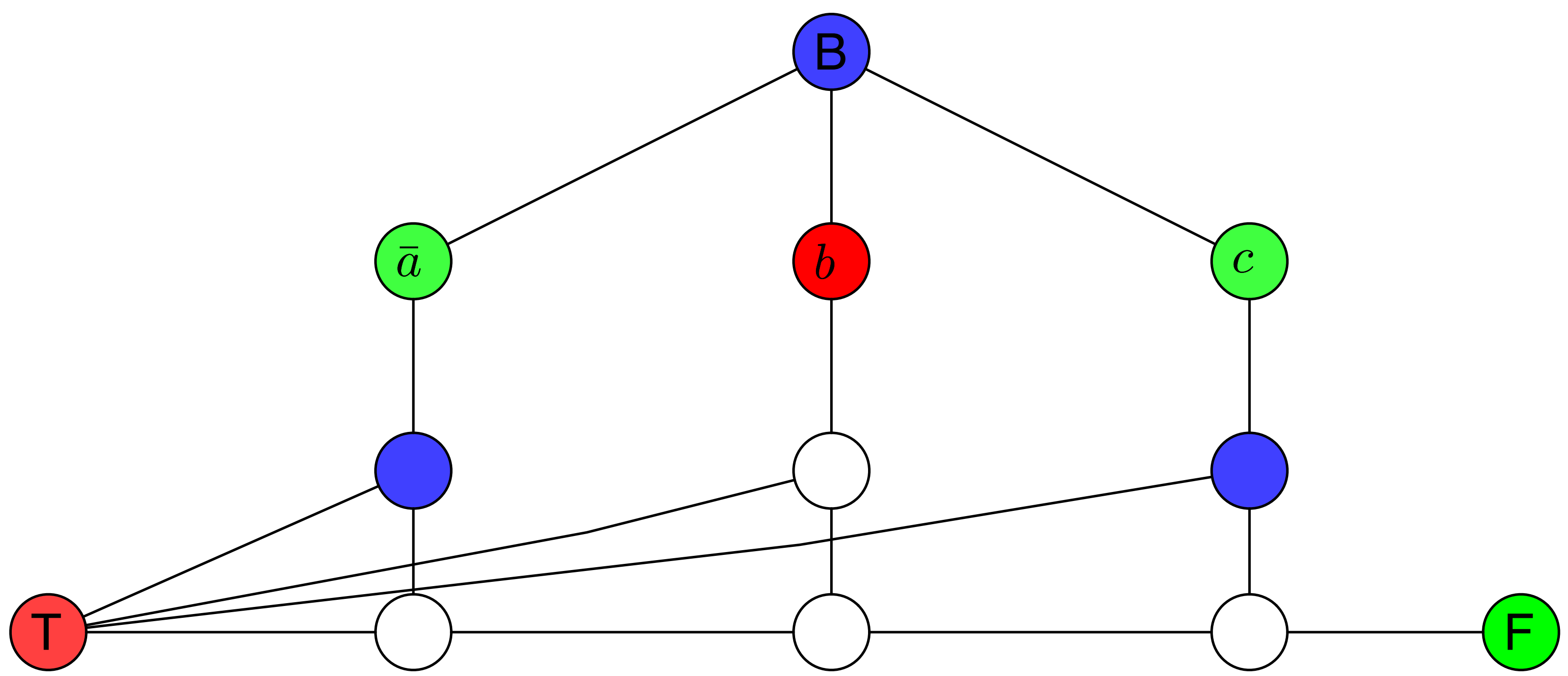
Graph 3-Colorability

- Assume that we have an assignment to the variables that satisfies the formula and all of the clauses
  - Clause is $\neg a \vee b \vee c$ and a and b are true and c is false
  - Color the variable nodes accordingly:



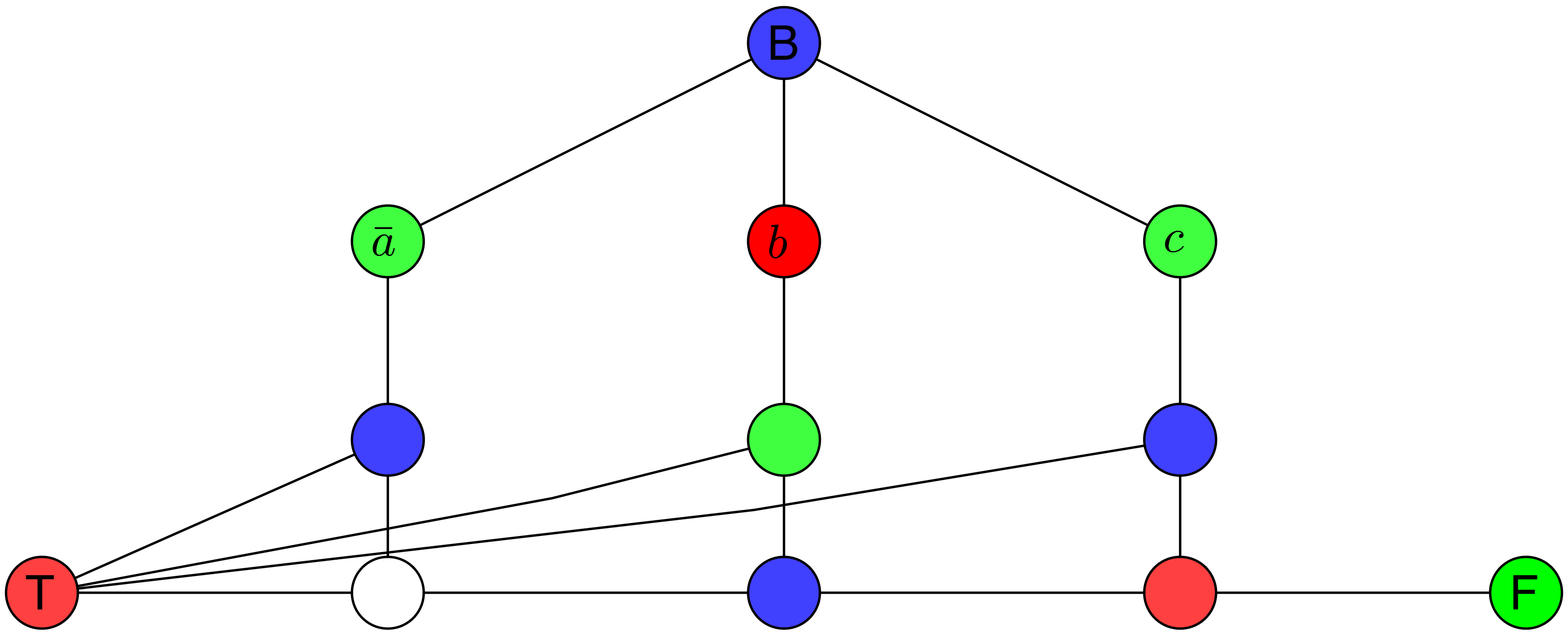
Graph 3-Colorability

- Some node colors now follow by necessity



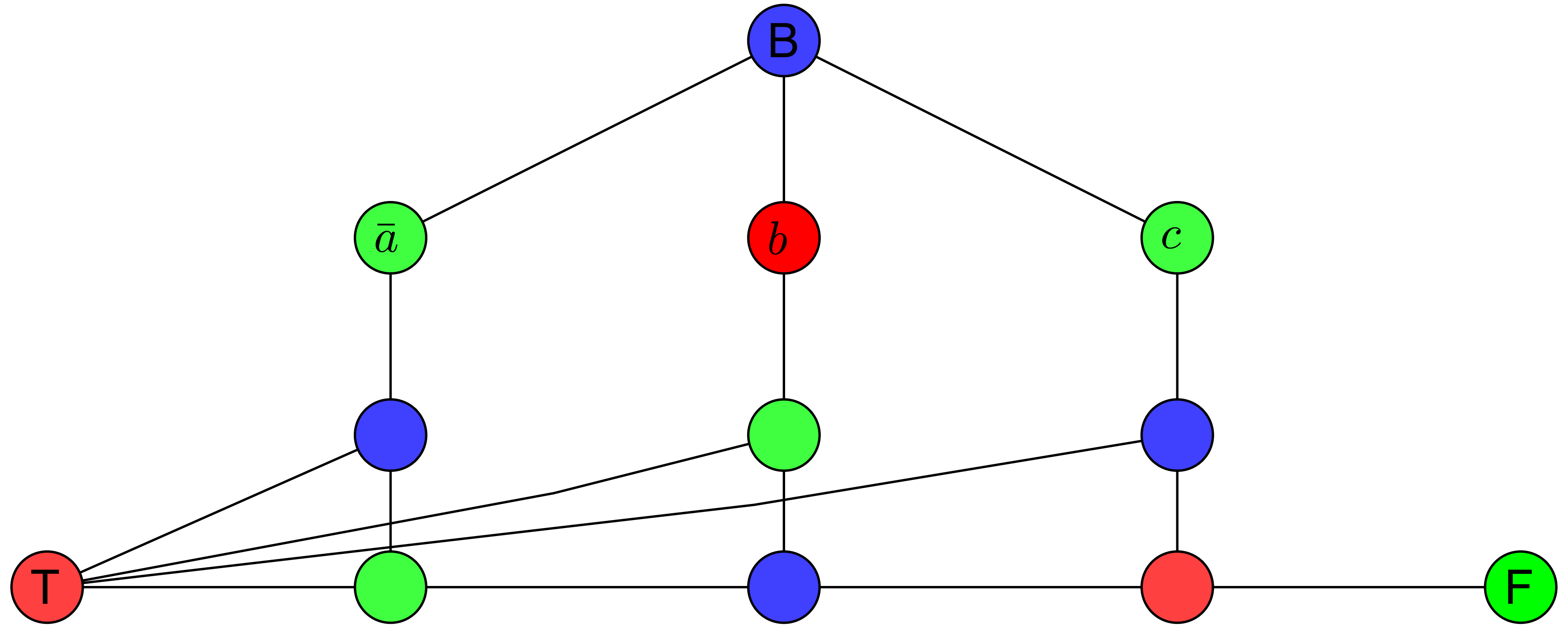
Graph 3-Colorability

- Color the node under a True-colored vertex with the False color and the one underneath with Blue



Graph 3-Colorability

- The last vertex color follows

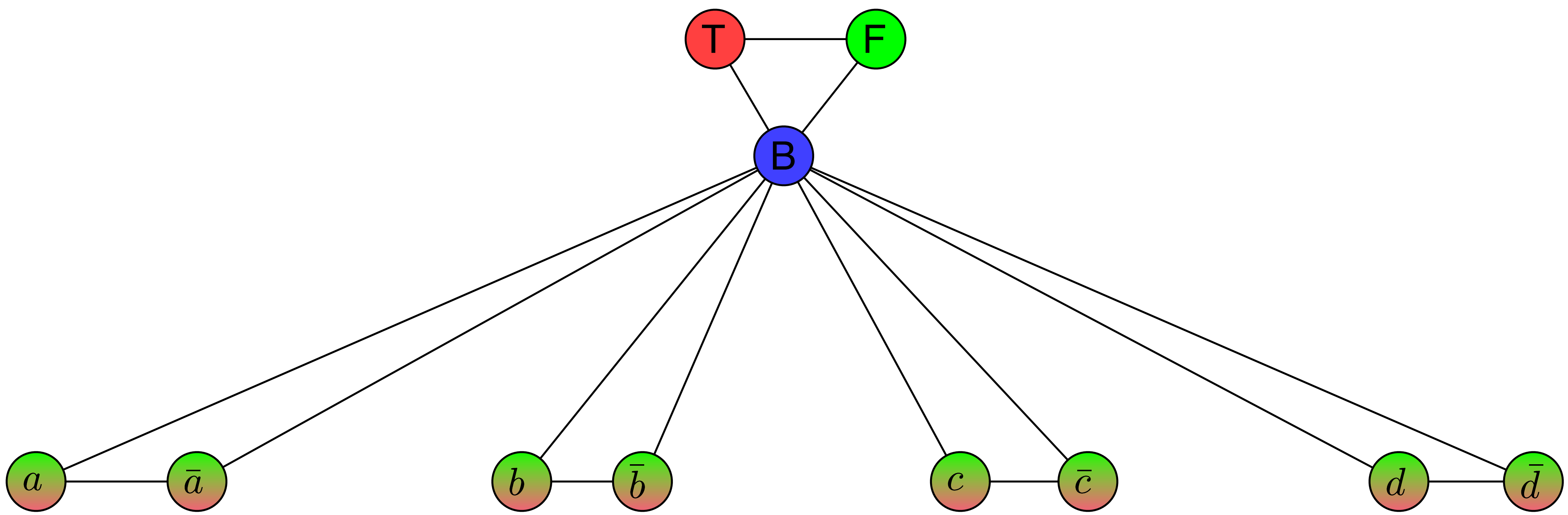


Graph 3-Colorability

- This gadget is now three-colored
  - Do the same thing for all gadgets
  - Get a three coloring of the graph

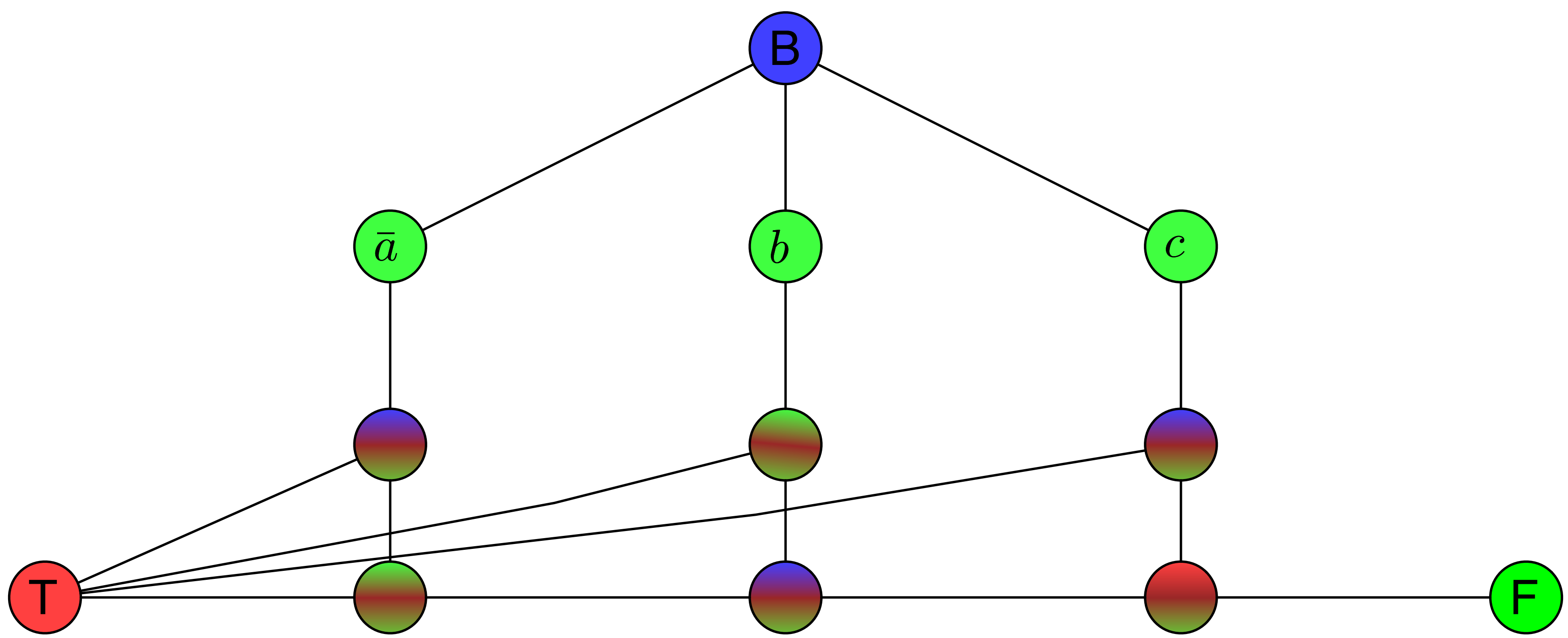
Graph 3-Colorability

- Vice versa, assume that you have a three coloring
- The base part of the graph assures that each literal is colored either with Red (for True) or with Green (for False)



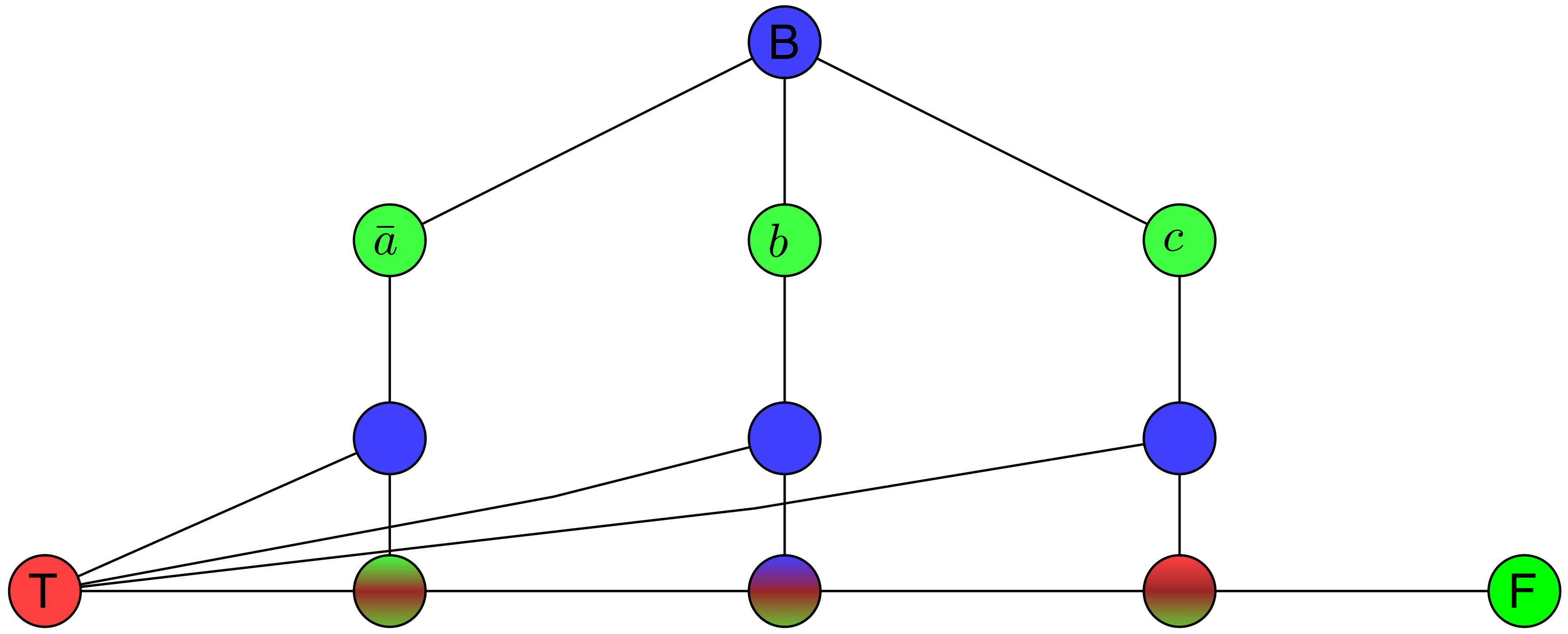
Graph 3-Colorability

- If the three coloring where to assign Green (the color of Falsehood) to the literal nodes, then what would happen



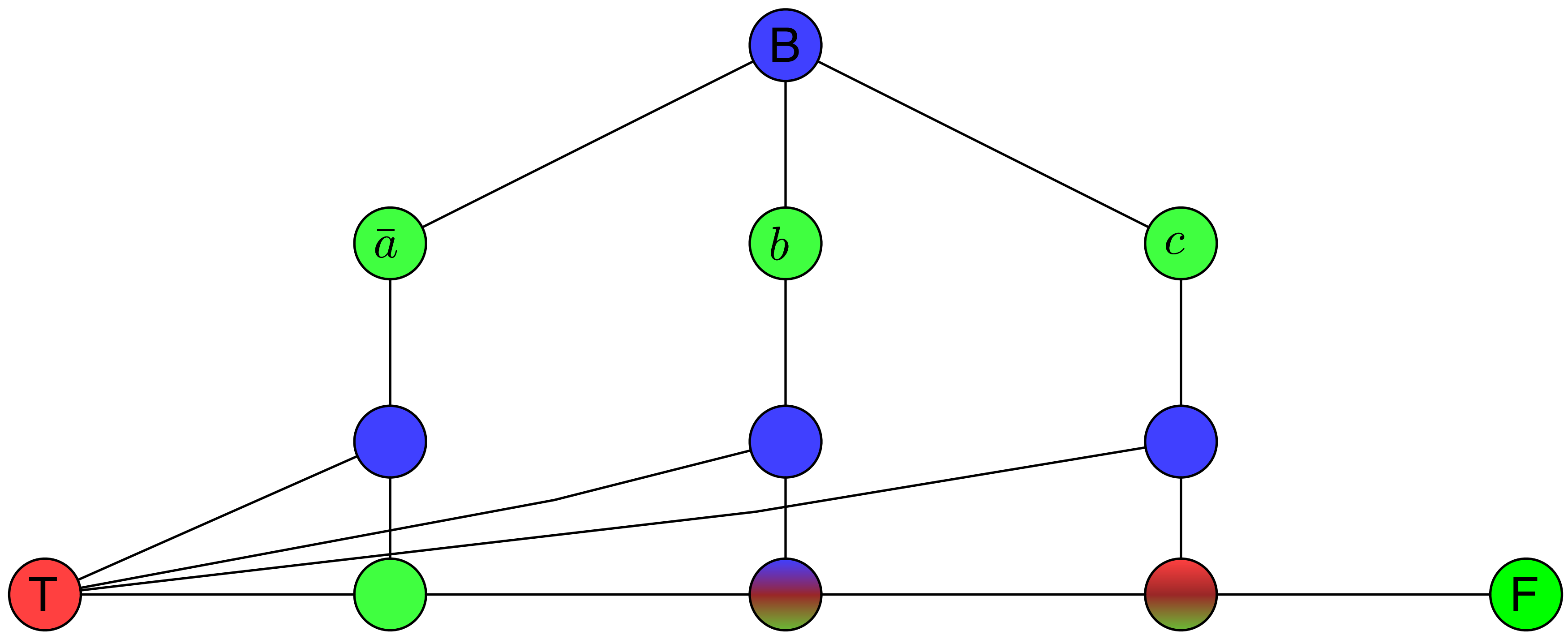
Graph 3-Colorability

- We can deduce some colors
  - The middle layer of nodes has to be blue, because they have a Red neighbor (the true node) and a Green neighbor



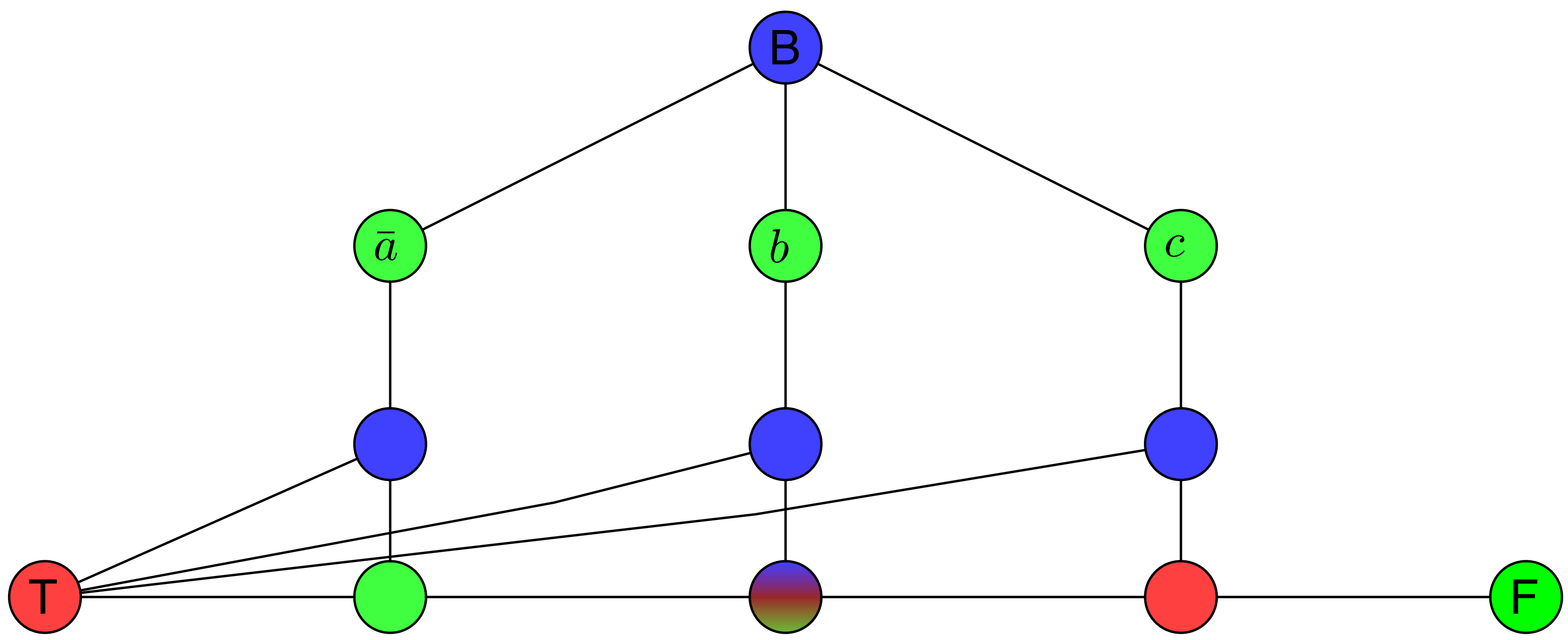
Graph 3-Colorability

- The vertex on the lower row and to the left needs to be Green because it has a Blue and a Red neighbor



Graph 3-Colorability

- The vertex on the lower row on the right has to be Red



Graph 3-Colorability

- But now the middle node has a Red, a Green, and a Blue neighbor
- Thus the graph is not 3-colorable
- Therefore: The graph is 3-colorable implies that in the gadget, one vertex is colored Red (the color of Truth)
    - Not to be confused with https://www.youtube.com/watch?v=8epG4AezdYg which is just propaganda but the color of the Holy Spirit

Graph 3-Colorability

- So, the complete graph is 3-colorable if in all gadgets at least one of the literal nodes is colored Red
- Then all clauses are satisfied
- Then the formula is satisfied

Consequences

- There is a large set of known NP-complete problems
  - Many problems that appear in practice can be shown to be NP-complete
    - You look through the catalogue of NP-complete problems and reduce your problem to one of them
    - (If I can solve my problem polynomially, then I can solve that problem polynomially, but that would mean that $\mathscr{P} = \mathscr{NP}$, which we believe to be false)
  - If this is the case:
    - You cannot expect a solution that scales well
    - But it might be perfectly solvable for the instance sizes that you need to solve