CSE4051: Program Verification

Propositional Logic

2025 Fall

Woosuk Lee

Calculus of Computation

- Calculus: a set of symbols + rules for manipulating the symbols
  - e.g., Differential calculus: rules for manipulating integral symbols over a polynomial
- We may ask questions about computations
  - Does this program terminate?
  - Does this program output a sorted array for a given array?
  - Does this program access unallocated memory?
- We need a calculus to reason about computation to answer these questions.

Propositional Logic and First-Order Logic

- Also known as propositional calculus and predicate calculus
- calculi for reasoning about propositions and predicates
- Propositions: statements that can be true or false
  - e.g., "It is raining", "2 + 2 = 4”
- Predicates: statements that can be true or false depending on the values given to them
  - e.g., "x is greater than 2", "y is a prime number"

Syntax of Propositional Logic

- **Syntax**: a set of symbols and rules for combining them to form "sentences" of a language
- Truth symbols $\top$(true), $\bot$ (false) are propositions.
- Propositional (or Boolean) variables: p, q, r, … are propositions.
- Logical connectives are used to combine propositions to construct propositions.
  - Negation: $\neg$ (not)
  - Disjunction: $\vee$ (or)

Conjunction: $\wedge$ (and)

Implication: $\Rightarrow$ (implies)

$$a \Rightarrow b \equiv \neg a \vee b$$

Syntax of Propositional Logic

- **Atom**: a truth symbol or a propositional variable

$$(\neg p \wedge \top) \vee (q \Rightarrow \bot)$$

Syntax of Propositional Logic

- **Atom**: a truth symbol or a propositional variable
- **Literal**: an atom or its negation

$$(\neg p \wedge \top) \vee (q \Rightarrow \bot)$$

Syntax of Propositional Logic

- **Atom**: a truth symbol or a propositional variable
- **Literal**: an atom or its negation
- **Formula**: a finite sequence of literals combined using logical connectives

$$(\neg p \wedge \top) \vee (q \Rightarrow \bot)$$

Semantics of Propositional Logic

- **Semantics**: rules for providing "meaning" to each sentence
- Meaning is given by the truth values (true and false)
- Rules:
  - "$\top$ means true"
  - "$\bot$ means false"
  - "$\top \land \bot$ means false"
  - …
- We cannot enumerate such rules for infinitely many propositions!
- Also, meaning of a proposition varies depending on meaning of variables.

Interpretation

- **Interpretation** I for a formula F maps every variable in F to a truth value
  - e.g., $I : \{p \mapsto true, q \mapsto false\}$
- We write $I \models F$ if F is true under interpretation I.

  Otherwise, we write $I \not\models F$
- Our goal: given a formula F and an interpretation I, decide if $I \models F$ or $I \not\models F$

  using finitely many rules.

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \wedge F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
  - $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \wedge F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
  - $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models \top$$
$$I \not\models \bot$$

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \land F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \land F_2$ is true if both F_1 and F_2 are true
  - $F_1 \lor F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models P \quad \text{iff } I[P] = \text{true}$$

$$I \not\models P \quad \text{iff } I[P] = \text{false}$$

If and only if
(precisely when)

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \land F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \land F_2$ is true if both F_1 and F_2 are true
  - $F_1 \lor F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models \neg F \quad \text{iff } I \not\models F$$

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \land F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \land F_2$ is true if both F_1 and F_2 are true
  - $F_1 \lor F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models F_1 \land F_2 \quad \text{iff } I \models F_1 \text{ and } I \models F_2$$

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \wedge F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
  - $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models F_1 \vee F_2 \quad \text{iff } I \models F_1 \text{ or } I \models F_2$$

Semantics of Propositional Logic

- We define the meaning of basic elements first
  - $\top$ is true, $\bot$ is false
  - a variable is true if it is assigned true, false if assigned false
- Assuming the meaning of a set of elements is fixed, define a more complex element in terms of these elements ($F_1 \wedge F_2$ is more complex formula than the formulae F_1 or F_2)
  - $\neg F$ is true if F is false, and vice versa
  - $F_1 \wedge F_2$ is true if both F_1 and F_2 are true
  - $F_1 \vee F_2$ is true if at least one of F_1 or F_2 is true
  - $F_1 \Rightarrow F_2$ is false only if F_1 is true and F_2 is false

$$I \models F_1 \rightarrow F_2 \quad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$

true when $I \not\models F_1$

Or

$$I \not\models F_1 \rightarrow F_2 \quad \text{iff } I \models F_1 \text{ and } I \not\models F_2$$

Semantics of Propositional Logic

- Recall the previous formula $F : \; P \wedge Q \;\rightarrow\; P \vee \neg Q$

  and interpretation $I : \; \{P \mapsto \text{true}, \; Q \mapsto \text{false}\}$
- Compute the truth value of F as follows:

$$
\begin{array}{lllll}
1. & I & \models & P & \text{since } I[P] = \text{true} \\
2. & I & \not\models & Q & \text{since } I[Q] = \text{false} \\
3. & I & \models & \neg Q & \text{by 2 and semantics of } \neg \\
4. & I & \not\models & P \wedge Q & \text{by 2 and semantics of } \wedge \\
5. & I & \models & P \vee \neg Q & \text{by 1 and semantics of } \vee \\
6. & I & \models & F & \text{by 4 and semantics of } \rightarrow
\end{array}
$$

Satisfiability and Validity

- Q is *satisfiable* if and only if
  - A satisfying interpretation of Q exists (i.e., I ⊧ Q for some I)

- Q is *valid* if and only if
  - All interpretations of Q are satisfying (i.e., I ⊧ Q for all I)
  - Otherwise, *invalid* (i.e., there exists I such that $I \not\models Q$)

- Satisfiability and validity are dual
  - "Q is valid" ≡ "¬ Q is *unsatisfiable*"

Methods for Deciding Satisfiability & Validity

- Truth-table method (a.k.a. proof by enumeration)
  - Enumerate all interpretations and check if a formula is satisfiable in every case
- Semantic argument method (a.k.a. proof by deduction)
  - Assuming the formula is invalid (i.e., there exists a falsifying interpretation I such that $I \not\models F$, check if the assumption leads to a contradiction.

Truth-Table Method

- Consider formula $F: P \wedge Q \rightarrow P \vee \neg Q$
- Truth table (0 corresponds to the value false, 1 to true)

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

- F is valid because it is true under every possible interpretation.

Truth-Table Method

- Consider formula $F: \; P \vee Q \;\rightarrow\; P \wedge Q$
- Truth table

P	Q	$P \vee Q$	$P \wedge Q$	F
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

- F is invalid because the second and third rows are false.

Semantic Argument Method

- Assume a formula is invalid, and check if it leads to a contradiction by applying *proof rules.*
- A proof rule has one or more premises (assumed facts) and deductions (deduced facts)

$$\frac{\text{Assumed fact1 , …, Assumed fact n}}{\text{Deduced fact'1, … , Deduced fact' n}}$$

- Read as "If fact1, …, fact n are true, then fact'1, …, fact' n are also true.

Semantic Argument Method

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \land G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \qquad \frac{I \not\models F \land G}{I \not\models F \quad \mid \quad I \not\models G} \qquad \frac{I \models F \lor G}{I \models F \quad \mid \quad I \models G} \qquad \frac{I \not\models F \lor G}{\begin{array}{l} I \not\models F \\ I \not\models G \end{array}}$$

AND

OR

$$\frac{I \models F \to G}{I \not\models F \quad \mid \quad I \models G} \qquad \frac{I \not\models F \to G}{\begin{array}{l} I \models F \\ I \not\models G \end{array}} \qquad \frac{\begin{array}{l} I \models F \\ I \not\models F \end{array}}{I \models \bot}$$

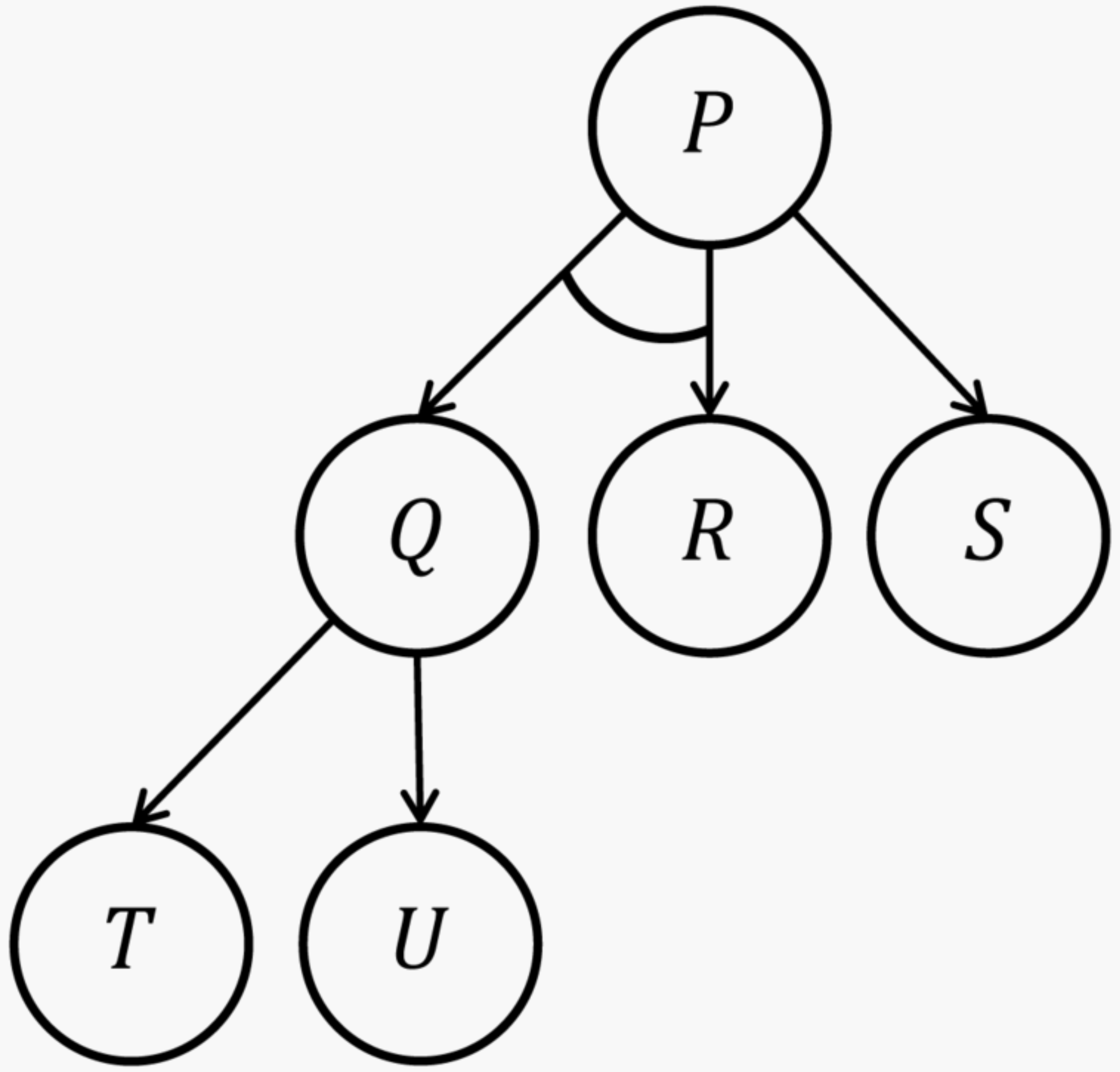
Contradiction!

AND-OR Tree

- We will use an and-or tree as a graphical representation of a proof.
- The following tree represents

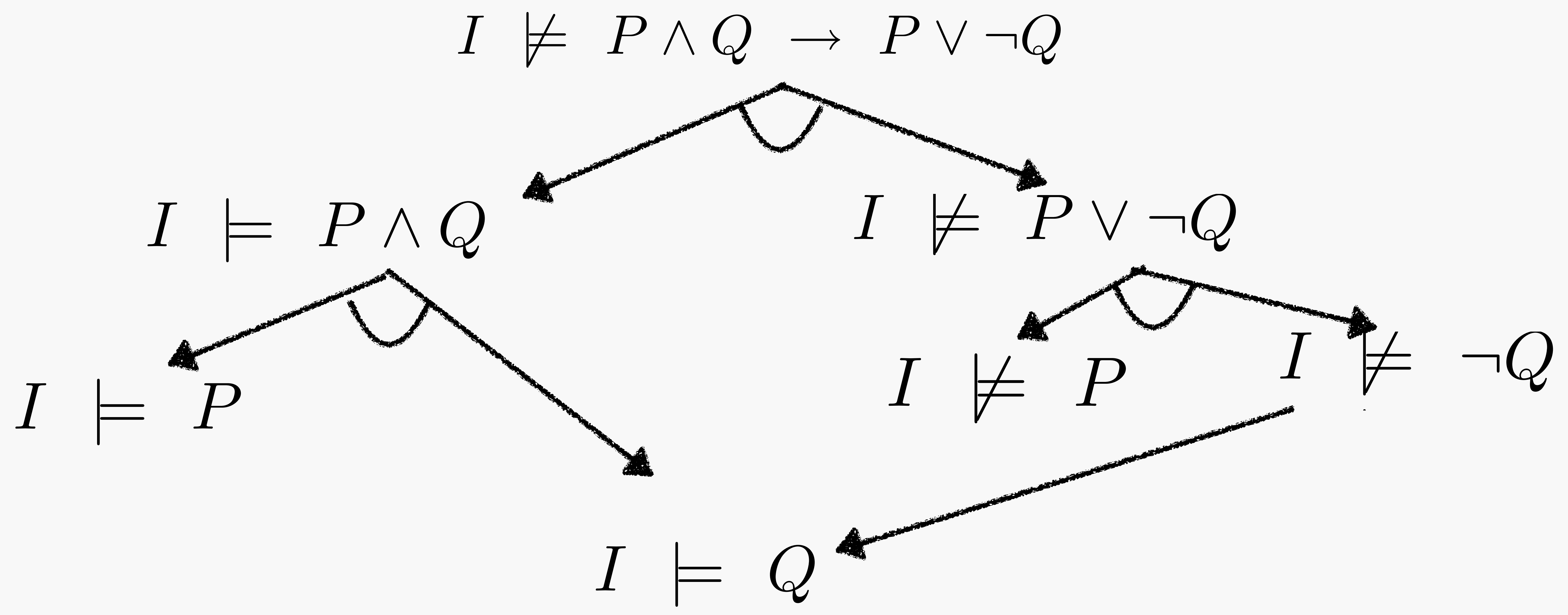
"If (Q and R) or S, then P"

"If T or U, then Q"



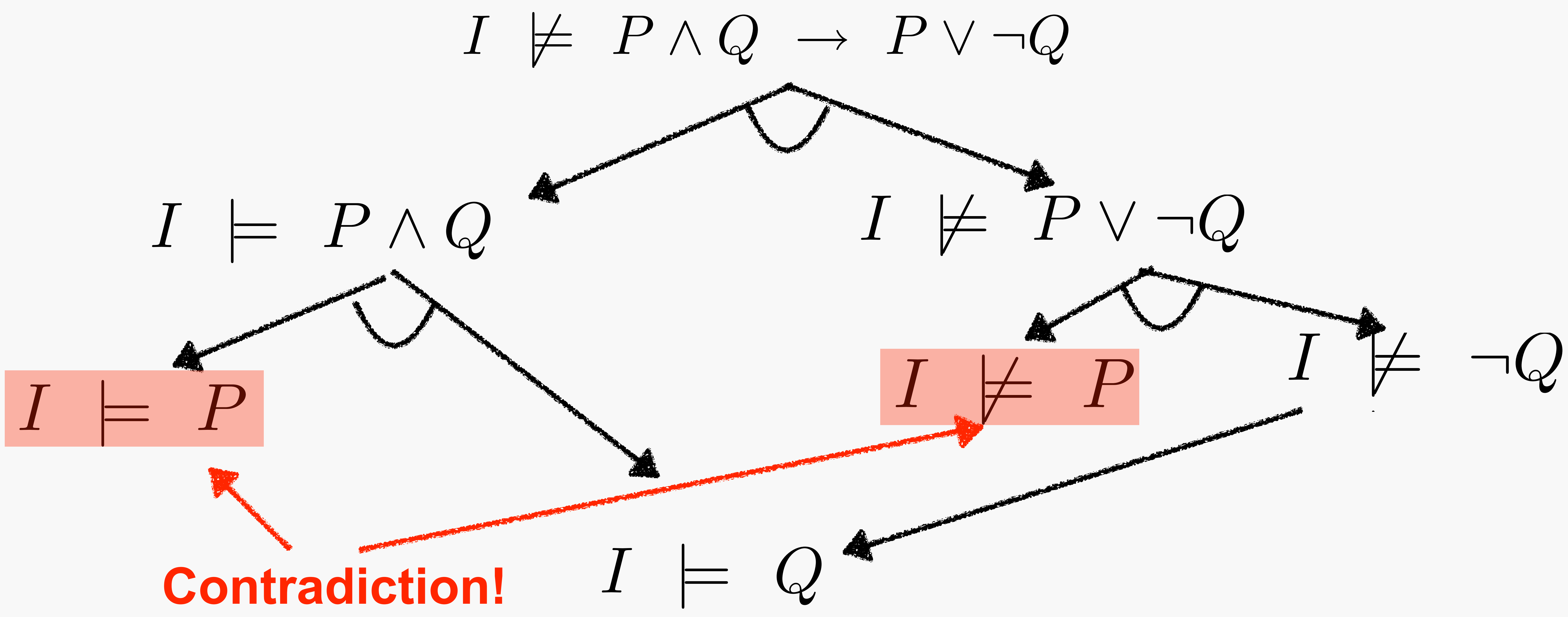
Semantic Argument Method

- To prove formula $F:\ P \wedge Q \ \rightarrow \ P \vee \neg Q$ is valid, assume it is invalid and derives a contradiction (then, the assumption is wrong, which means F is valid).



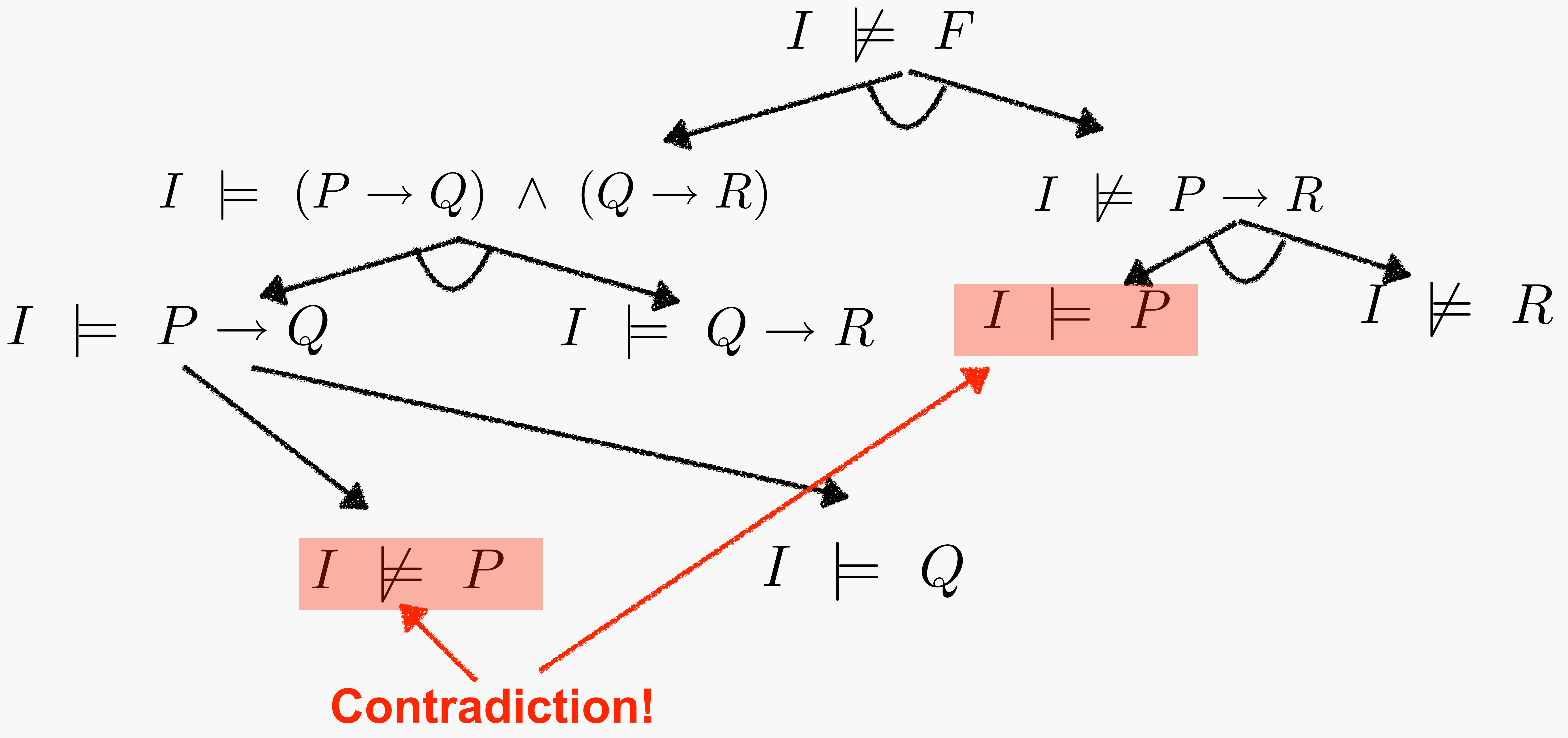
Semantic Argument Method

- To prove formula $F: P \land Q \rightarrow P \lor \neg Q$ is valid, assume it is invalid and derives a contradiction (then, the assumption is wrong, which means F is valid).



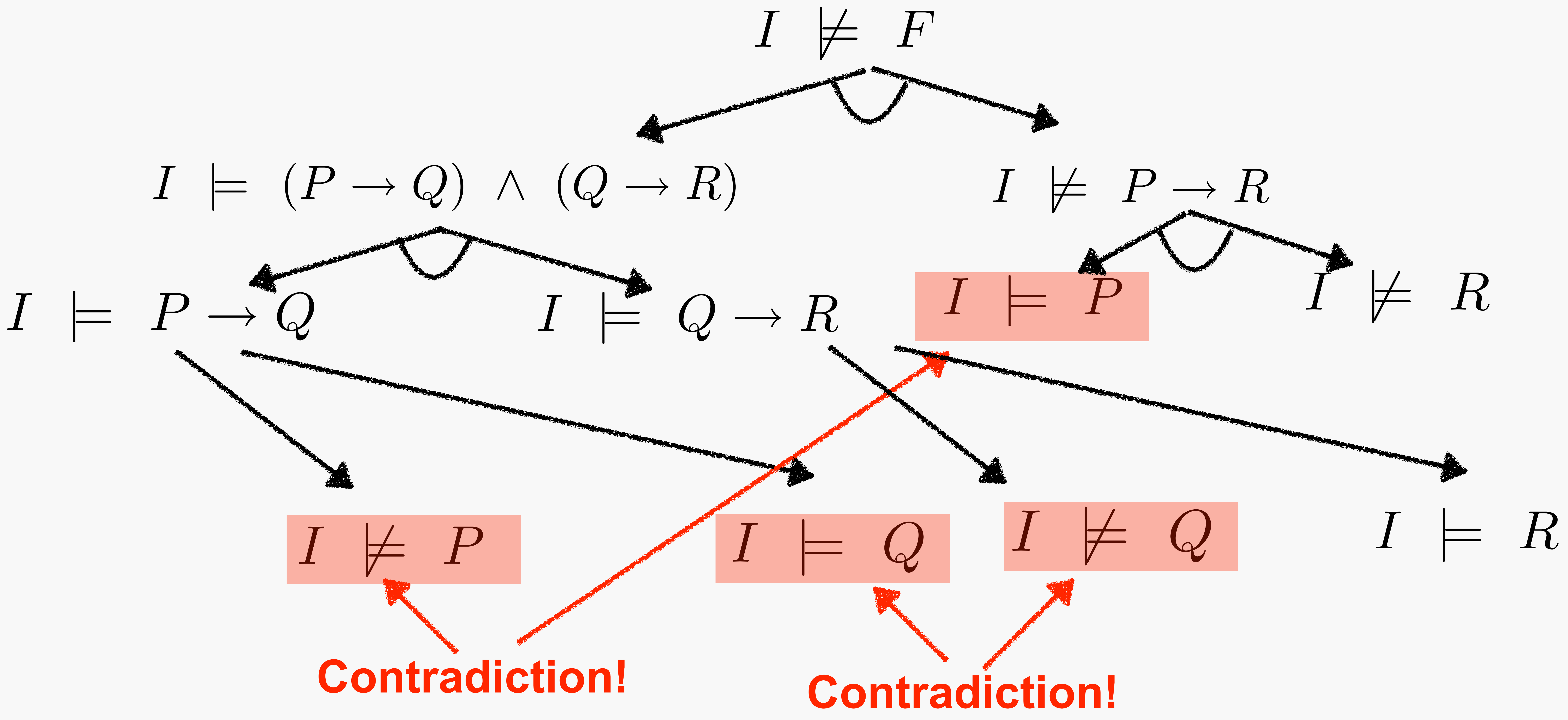
Semantic Argument Method

- To prove formula $F: (P \rightarrow Q) \land (Q \rightarrow R) \rightarrow (P \rightarrow R)$ is valid



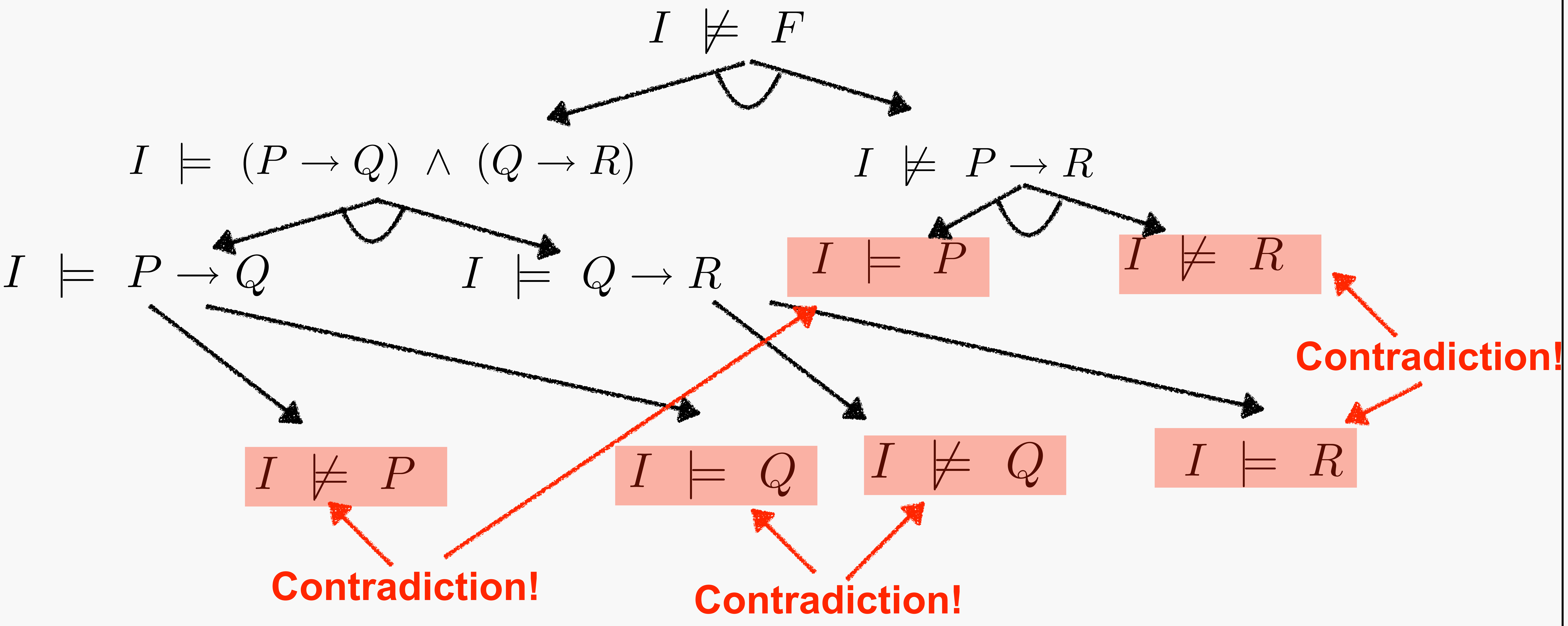
Semantic Argument Method

- To prove formula $F: \ (P \rightarrow Q) \wedge (Q \rightarrow R) \ \rightarrow \ (P \rightarrow R)$ is valid



Semantic Argument Method

- To prove formula $F : (P \to Q) \wedge (Q \to R) \to (P \to R)$ is valid



Checking Satisfiability is Hard

- Boolean satisfiability problem (SAT) : for a given formula, determine if there exists an interpretation that makes the formula true
- **NP-complete**
  - NP: a class of problems that are solvable in polynomial time when you are very lucky (P: a class of problems that are always solvable in polynomial time)
  - NP-complete: hardest ones in NP
  - general SAT algorithms are probably exponential in time

Semantic Equivalence

- Two formulas F_1 and F_2 are equivalent if they evaluate to the same truth value under all interpretations.
- In other words, $(F_1 \implies F_2) \land (F_2 \implies F_1)$ is valid (in short $F_1 \Leftrightarrow F_2$)
  - $P \Leftrightarrow \neg\neg P$
  - $P \rightarrow Q \Leftrightarrow \neg P \lor Q$

Normal Forms

- A normal form of formulae is a *syntactic restriction* such that for every formula of the logic, there is an equivalent formula in the normal form.
- Three important normal forms for propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

- NNF requires that ¬, ∧, and ∨ be the only connectives and that negations appear only in literals. First step before converting to other normal forms
- Transforming into an NNF form can be done using the following equivalences:

$$\begin{aligned}
\neg\neg F_1 &\Leftrightarrow F_1 \\
\neg\top &\Leftrightarrow \bot \\
\neg\bot &\Leftrightarrow \top \\
\neg(F_1 \wedge F_2) &\Leftrightarrow \neg F_1 \vee \neg F_2 \\
\neg(F_1 \vee F_2) &\Leftrightarrow \neg F_1 \wedge \neg F_2 \\
F_1 \rightarrow F_2 &\Leftrightarrow \neg F_1 \vee F_2 \\
F_1 \leftrightarrow F_2 &\Leftrightarrow (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)
\end{aligned}$$

- For transformation, the equivalences should be applied left-to-right.

Negation Normal Form (NNF)

- NNF requires that ¬, ∧, and ∨ be the only connectives and that negations appear only in literals. First step before converting to other normal forms
- Transforming into an NNF form can be done using the following equivalences:

$$\begin{aligned}
\neg\neg F_1 &\Leftrightarrow F_1 \\
\neg\top &\Leftrightarrow \bot \\
\neg\bot &\Leftrightarrow \top \\
\neg(F_1 \wedge F_2) &\Leftrightarrow \neg F_1 \vee \neg F_2 \\
\neg(F_1 \vee F_2) &\Leftrightarrow \neg F_1 \wedge \neg F_2 \\
F_1 \rightarrow F_2 &\Leftrightarrow \neg F_1 \vee F_2 \\
F_1 \leftrightarrow F_2 &\Leftrightarrow (F_1 \rightarrow F_2) \;\wedge\; (F_2 \rightarrow F_1)
\end{aligned}$$

De Morgan's Law

- For transformation, the equivalences should be applied left-to-right.

QUIZ

- Convert the formula $F:\ \neg(P \rightarrow \neg(P \wedge Q))$ to NNF.

Disjunctive Normal Form (DNF)

- A formula is in disjunctive normal form (DNF) if it is a disjunction of conjunctions of literals:

$$\bigvee_i \bigwedge_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

Clause

- For conversion, use the following equivalences:

$$\begin{aligned}(F_1 \vee F_2) \wedge F_3 &\Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3)\\ F_1 \wedge (F_2 \vee F_3) &\Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3)\end{aligned}$$

QUIZ

- Convert the formula $F: (Q_1 \vee \neg\neg Q_2) \wedge (\neg R_1 \rightarrow R_2)$ to DNF
  - You should first transform it into NNF

Disjunctive Normal Form (DNF)

- Deciding satisfiability of a DNF formula is trivial. Why?
  - Given $C_1 \vee C_2 \vee \cdots \vee C_n$, find one clause C_i that is satisfiable
  - Each clause is of form $l_1 \wedge l_2 \cdots \wedge l_m$
  - A clause is satisfiable if there is no contradiction (e.g., $A \wedge \neg A$)
  - This can be done in linear time per clause.

```
for clause in disjuncts:
    if clause is internally consistent:
        return SAT
return UNSAT
```

- Complexity : O(size of formula)

Disjunctive Normal Form (DNF)

- Why don't we just convert formula to DNF and do the simple check?
  - Then, can checking satisfiability be done in linear time?
- No because of the exponential blowup!
  - A formula $(F_1 \vee F_2) \wedge (F_3 \vee F_4)$ is in DNF:

    $(F_1 \wedge F_3) \vee (F_1 \vee F_4) \vee (F_2 \wedge F_3) \vee (F_2 \wedge F_4)$
  - Whenever we distribute, formula size doubles!
- Checking satisfiability by converting to DNF is almost as bad as truth tables.

Conjunctive Normal Form (CNF)

- A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunction of literals:

$$\bigwedge_i \bigvee_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

Clause

- For conversion, use the following equivalences

$$\begin{aligned}(F_1 \wedge F_2) \vee F_3 &\Leftrightarrow (F_1 \vee F_3) \wedge (F_2 \vee F_3) \\ F_1 \vee (F_2 \wedge F_3) &\Leftrightarrow (F_1 \vee F_2) \wedge (F_1 \vee F_3)\end{aligned}$$

QUIZ

- Convert the formula $F: (Q_1 \vee \neg\neg Q_2) \wedge (\neg R_1 \rightarrow R_2)$ to CNF
  - You should first transform it into NNF

Conjunctive Normal Form (CNF)

- Solving CNF is not as easy as solving DNF.
- Conversion to CNF does not explode as DNF.
  - Many formulas that would be very large in DNF can be small in CNF.
- SAT solvers use CNF as their input language.
  - CNF gives a uniform input format for solvers.
  - DIMACS (standard SAT input format)

Conversion to an Equisatisfiable Formula in CNF

- Two formulas F and G are **equisatisfiable** if they are both satisfiable or both unsatisfiable.
- Tseitin's transformation converts a formula F into an equisatisfiable CNF formula with only a *linear increase* in size.

Tseitin's Transformation

- For example, given $F : x \implies (y \land z)$
- For every sub formula G of F (unless G is an atom), introduce a new variable representing G
  - $v_1 \Leftrightarrow (x \implies v_2)$
  - $v_2 \Leftrightarrow (y \land z)$
- Formula: $v_1 \land (v_1 \Leftrightarrow (x \implies v_2)) \land (v_2 \Leftrightarrow (y \land z))$

Tseitin's Transformation (contd.)

- Convert each $v_i \Leftrightarrow G$ into CNF
  - $(v_1 \implies \neg x \vee v_2) \wedge (\neg x \vee v_2 \implies v_1) \rightarrow (\neg v_1 \vee \neg x \vee v_2) \wedge (\neg(\neg x \vee v_2) \vee v_1)$

    $\rightarrow (\neg v_1 \vee \neg x \vee v_2) \wedge (\neg(\neg x \vee v_2) \vee v_1)$

    $\rightarrow (\neg v_1 \vee \neg x \vee v_2) \wedge ((x \wedge \neg v_2) \vee v_1)$

    $\rightarrow (\neg v_1 \vee \neg x \vee v_2) \wedge (x \vee v_1) \wedge (\neg v_2 \vee v_1)$
  - $(v_2 \implies y \wedge z) \wedge (y \wedge z \implies v_2) \rightarrow (\neg v_2 \vee y \wedge z) \wedge (\neg(y \wedge z) \vee v_2)$

    $\rightarrow (\neg v_2 \vee y \wedge z) \wedge (\neg y \vee \neg z \vee v_2)$
- Final result:

  $v_1 \wedge (\neg v_1 \vee \neg x \vee v_2) \wedge (x \vee v_1) \wedge (\neg v_2 \vee v_1) \wedge (\neg v_2 \vee y \wedge z) \wedge (\neg y \vee \neg z \vee v_2)$

DPLL Algorithm

- The two naive methods for satisfiability
  - Truth-table method (a.k.a. proof by **enumeration**)
  - Semantic argument method (a.k.a. proof by **deduction**)
- DPLL algorithm combines enumeration and deduction in an effective way.
- Any given formula is transformed into CNF before fed into the DPLL algorithm.

Unit Resolution

- Suppose we have two clauses C_1 and C_2 that share a variable P but disagrees on its value (e.g., C_1 contains P and C_2 contains $\neg P$)
- Either the rest of C_1 or the rest of C_2 must be satisfied.
  - If P is true, literals other than $\neg P$ in C_2 should be true
  - If P is false, literals other than P in C_1 should be true

Unit Resolution (contd.)

- More formally, suppose we have two clauses C_1 and C_2 that share a variable P such that

  $C_1 = \alpha_1 \vee \cdots \vee \alpha_n \vee P$ and $C_2 = \beta_1 \vee \cdots \vee \beta_m \vee \neg P$
- Then, unit resolution is stated as the following rule:

$$\frac{\alpha_1 \vee \cdots \vee \alpha_n \vee P \qquad \beta_1 \vee \cdots \vee \beta_m \vee \neg P}{\alpha_1 \vee \cdots \vee \alpha_n \vee \beta_1 \vee \cdots \vee \beta_m}$$

Example

Suppose we have $F: \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q$

From resolution

$$\frac{(\neg P \vee Q) \qquad P}{Q}$$

We construct $F_1: \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q \ \wedge \ Q$

From resolution

$$\frac{\neg Q \qquad Q}{\bot}$$

F is unsatisfiable.

DPLL Algorithm

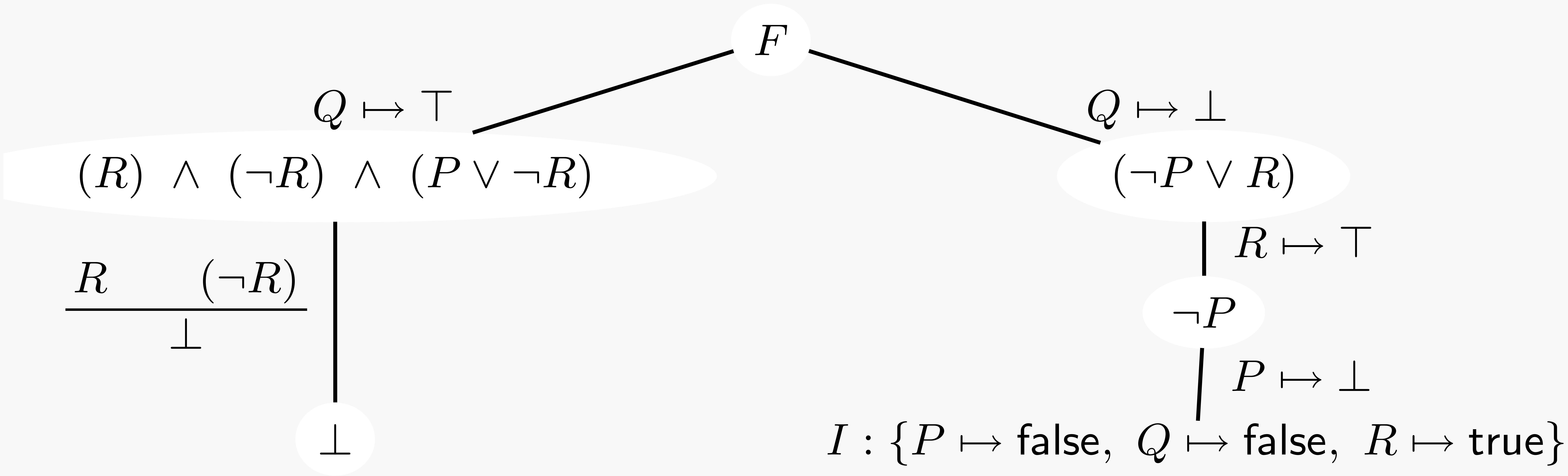
- The process of applying unit resolution as much as possible (i.e., until no more resolution is possible) is called *Boolean constraint propagation* (**BCP**).
- The DPLL algorithm (return true : SAT, return false : UNSAT):

```
function DPLL (F) {
    F′ = BCP(F);
    if F′ = ⊤ then return true
    else if F′ = ⊥ then return false
    else
        P = Choose_var(F′);
        return (DPLL(F′{P ↦ ⊤}) or DPLL(F′{P ↦ ⊥}))
}
```

Replace every occurrence of P in F' with ⊥

DPLL Example

- Consider $F: (\neg P \vee Q \vee R) \wedge (\neg Q \vee R) \wedge (\neg Q \vee \neg R) \wedge (P \vee \neg Q \vee \neg R)$



Summary

- SAT problem
- NNF, DNF, CNF
- Tseitin’s transformation
- Boolean constraint propagation (BCP)
- DPLL