CS5733 Program Synthesis

#6.SAT Solving

Ashish Mishra, August 16, 2024

Partly based on slides by Roopsha Samata at Purdue

Roadmap

- Previously
  - PL

- Today
  - Normal Forms and Tseitin's Transformation
  - DPLL algorithm for SAT solving
  - One challenge for current SAT solvers
  - Variations of the satisfiability problem (e.g., MaxSAT)

Example : Recap PL formula

formula $F : (P \land Q) \;\rightarrow\; (\top \lor \neg Q)$

atoms: $P,\ Q,\ \top$

literals: $P,\ Q,\ \top,\ \neg Q$

subformulae: $P,\ Q,\ \top,\ \neg Q,\ P \land Q,\ \top \lor \neg Q,\ F$

abbreviation

$$F : P \land Q \;\rightarrow\; \top \lor \neg Q$$

PL Semantics (Meaning)

$$\text{Sentence } F + \text{Interpretation } I = \text{Truth value (true, false)}$$

Interpretation

$$I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \cdots\}$$

Evaluation of F under I:

F	$\neg F$
0	1
1	0

where 0 corresponds to value false
1 corresponds to value true

$I \models F$ if F evaluates to true under I
$I \not\models F$ if F evaluates to false under I

F_1	F_2	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Satisfying and Falsifying Interpretations

PL Semantics (Inductive definitions)

Base Case:

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

Inductive Case:

$$I \models \neg F \quad \text{iff } I \not\models F$$
$$I \models F_1 \wedge F_2 \quad \text{iff } I \models F_1 \text{ and } I \models F_2$$
$$I \models F_1 \vee F_2 \quad \text{iff } I \models F_1 \text{ or } I \models F_2$$
$$I \models F_1 \rightarrow F_2 \quad \text{iff, if } I \models F_1 \text{ then } I \models F_2$$
$$I \models F_1 \leftrightarrow F_2 \quad \text{iff, } I \models F_1 \text{ and } I \models F_2, \text{ or } I \not\models F_1 \text{ and } I \not\models F_2$$

Note:

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

Satisfiability and Validity

F is satisfiable iff there exists $I : I \vDash F$

F is valid iff for all $I : I \vDash F$

Duality:
F is valid iff $\neg F$ is unsatisfiable

Procedure for deciding satisfiability *or* validity suffices!

Normal Forms

Normal Forms

- A normal form for a logic is a syntactical restriction such that for every formula in the logic, there is an equivalent formula in the normal form.

- Three useful normal forms for propositional logic:
  - Negation Normal Form (NNF)
  - Disjunctive Normal Form (DNF)
  - Conjunctive Normal Form (CNF)

Negation Normal Form (NNF)

Atom	T , ⊥ , propositional variables
Literal	Atom | ¬Atom
Formula	Literal | Formula op Formula
op	∨ | ∧

The only logical connectives are ¬, ∧, ∨

Negations appear only in literals

Conversion to NNF:

Eliminate → and ↔

"Push negations in" using DeMorgan's Laws:

$\neg(F_1 \wedge F_2) \Leftrightarrow (\neg F_1 \vee \neg F_2)$

$\neg(F_1 \vee F_2) \Leftrightarrow (\neg F_1 \wedge \neg F_2)$

Example: Convert $F : \neg(P \rightarrow \neg(P \wedge Q))$ to NNF

$F' : \neg(\neg P \vee \neg(P \wedge Q))$ $\rightarrow$ to $\vee$

$F'' : \neg\neg P \wedge \neg\neg(P \wedge Q)$ De Morgan's Law

$F''' : P \wedge P \wedge Q$ $\neg\neg$

F''' is equivalent to F ($F''' \Leftrightarrow F$) and is in NNF

Disjunctive Normal Form (DNF)

Atom	⊤ , ⊥ , propositional variables
Literal	Atom | ¬Atom
Disjunct	Literal ∧ Disjunct
Formula	Disjunct ∨ Formula

Disjunction of conjunctions of literals

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

Conversion to DNF:

First convert to NNF

Distribute ∧ over ∨

$$((F_1 \lor F_2) \land F_3) \Leftrightarrow ((F_1 \land F_3) \lor (F_2 \land F_3))$$

$$(F_1 \land (F_2 \lor F_3)) \Leftrightarrow ((F_1 \land F_2) \lor (F_1 \land F_3))$$

Deciding satisfiability of DNF formulas is trivial
Why not convert all PL formulas to DNF for SAT solving?
Exponential blow-up of formula size in DNF conversion!

Example

Example: Convert

$$F: \; (Q_1 \; \vee \; \neg\neg Q_2) \; \wedge \; (\neg R_1 \; \rightarrow \; R_2) \text{ into DNF}$$

$$\begin{array}{lll} F' & : (Q_1 \vee Q_2) \wedge (R_1 \vee R_2) & \text{in NNF} \\ F'' & : (Q_1 \wedge (R_1 \vee R_2)) \vee (Q_2 \wedge (R_1 \vee R_2)) & \text{dist} \\ F''' & : (Q_1 \wedge R_1) \vee (Q_1 \wedge R_2) \vee (Q_2 \wedge R_1) \vee (Q_2 \wedge R_2) & \text{dist} \end{array}$$

F''' is equivalent to F ($F''' \Leftrightarrow F$) and is in DNF

Conjunctive Normal Form (CNF)

Atom	⊤ , ⊥ , propositional variables
Literal	Atom | ¬Atom
Clause	Literal ∨ Clause
Formula	Clause ∧ Formula

Conjunction of disjunctions of literals

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

Deciding satisfiability of CNF formulas is not trivial
CNF conversion must also exhibit an exponential blow-up of formula size
Yet, almost all SAT solvers convert to CNF first before solving. Why?

Conversion to CNF:

First convert to NNF

Distribute ∨ over ∧

$$((F_1 \land F_2) \lor F_3) \Leftrightarrow ((F_1 \lor F_3) \land (F_2 \lor F_3))$$

$$(F_1 \lor (F_2 \land F_3)) \Leftrightarrow ((F_1 \lor F_2) \land (F_1 \lor F_3))$$

Natural representation because in practice, many formulas arise from multiple constraints that must hold *simultaneously* (AND).

Potential Problem with CNF: Size blowup

Distributivity will duplicate entire subformulas

Can happen repeatedly: $(p_1 \wedge p_2 \wedge p_3) \vee (q_1 \wedge q_2 \wedge q_3) =$
$(p_1 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_2 \vee (q_1 \wedge q_2 \wedge q_3)) \wedge (p_3 \vee (q_1 \wedge q_2 \wedge q_3))$
$= (p_1 \vee q_1) \wedge (p_1 \vee q_2) \wedge (p_1 \vee q_3)$
$\wedge (p_2 \vee q_1) \wedge (p_2 \vee q_2) \wedge (p_2 \vee q_3)$
$\wedge (p_3 \vee q_1) \wedge (p_3 \vee q_2) \wedge (p_3 \vee q_3)$

Worst-case blowup? : exponential!

Can't use this transformation for subsequent algorithms (e.g., satisfiability checking) if resulting formula is inefficiently large (possibly too large to represent/process).

Equisatisfiability and Tseitin's Transformation

Two formulas F_1 and F_2 are equisatisfiable iff:
F_1 is satisfiable iff F_2 is satisfiable

Tseitin's transformation converts any PL formula F_1 to equisatisfiable formula F_2 in CNF with only a linear increase in size

Note that equisatisfiability is a much weaker notion than equivalence, but is adequate for checking satisfiability.

Tseitin Transformation

Idea: rather than duplicate subformula:

introduce *new proposition* to represent it

add constraint: *equivalence* of subformula with new proposition

write this equivalence in CNF

Transformation rules for three basic operators

formula	$p \leftrightarrow$ formula	rewritten in CNF
$\neg A$	$(\neg A \rightarrow p) \wedge (p \rightarrow \neg A)$	$(A \vee p) \wedge (\neg A \vee \neg p)$
$A \wedge B$	$(A \wedge B \rightarrow p) \wedge (p \rightarrow A \wedge B)$	$(\neg A \vee \neg B \vee p) \wedge (A \vee \neg p) \wedge (B \vee \neg p)$
$A \vee B$	$(p \rightarrow A \vee B) \wedge (A \vee B \rightarrow p)$	$(A \vee B \vee \neg p) \wedge (\neg A \vee p) \wedge (\neg B \vee p)$

Tseitin's Transformation

1. Introduce an auxiliary variable rep(G) for each subformula $G = G_1 \ op \ G_2$ of formula F_1
2. Constrain auxiliary variable to be equivalent to subformula: rep(G) $\leftrightarrow$ rep(G_1) op rep(G_2)
3. Convert equivalence constraint to CNF: **CNF**(rep(G) $\leftrightarrow$ rep(G_1) op rep(G_2))
4. Let F_2 be rep(F) $\wedge$ $\bigwedge_G$ **CNF**(rep(G) $\leftrightarrow$ rep(G_1) op rep(G_2)). Check if F_2 is satisfiable.

F_1 and F_2 are equisatisfiable!

Tseitin Transformation: Example

Add numbered proposition for each operator:

$(a \overset{1}{\wedge} \neg b) \vee \neg(c \overset{2}{\wedge} d)$

no need to number negations

nor top-level operator $(\ldots) \vee (\ldots)$

New propositions: $p_1 \leftrightarrow a \overset{1}{\wedge} \neg b, \quad p_2 \leftrightarrow c \overset{2}{\wedge} d$.

Rewrite equivalences for new propositions in CNF,
conjunct with top-level operator of formula:

$(p_1 \vee \neg p_2)$

$\wedge (\neg a \vee b \vee p_1) \wedge (a \vee \neg p_1) \wedge (\neg b \vee \neg p_1)$

$\wedge (\neg c \vee \neg d \vee p_2) \wedge (c \vee \neg p_2) \wedge (d \vee \neg p_2)$

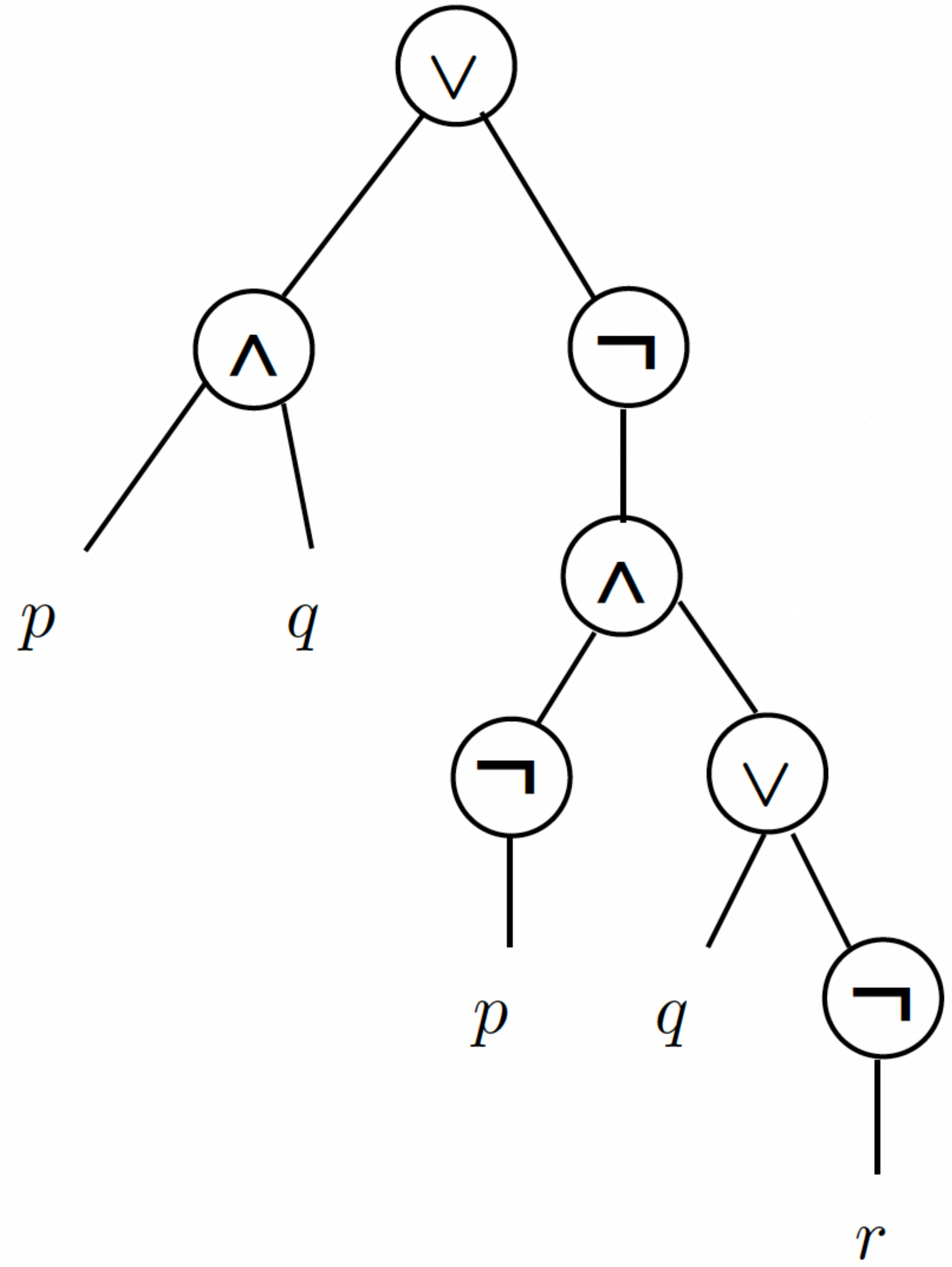
overall formula

$p_1 \leftrightarrow a \wedge \neg b$

$p_2 \leftrightarrow c \wedge d$

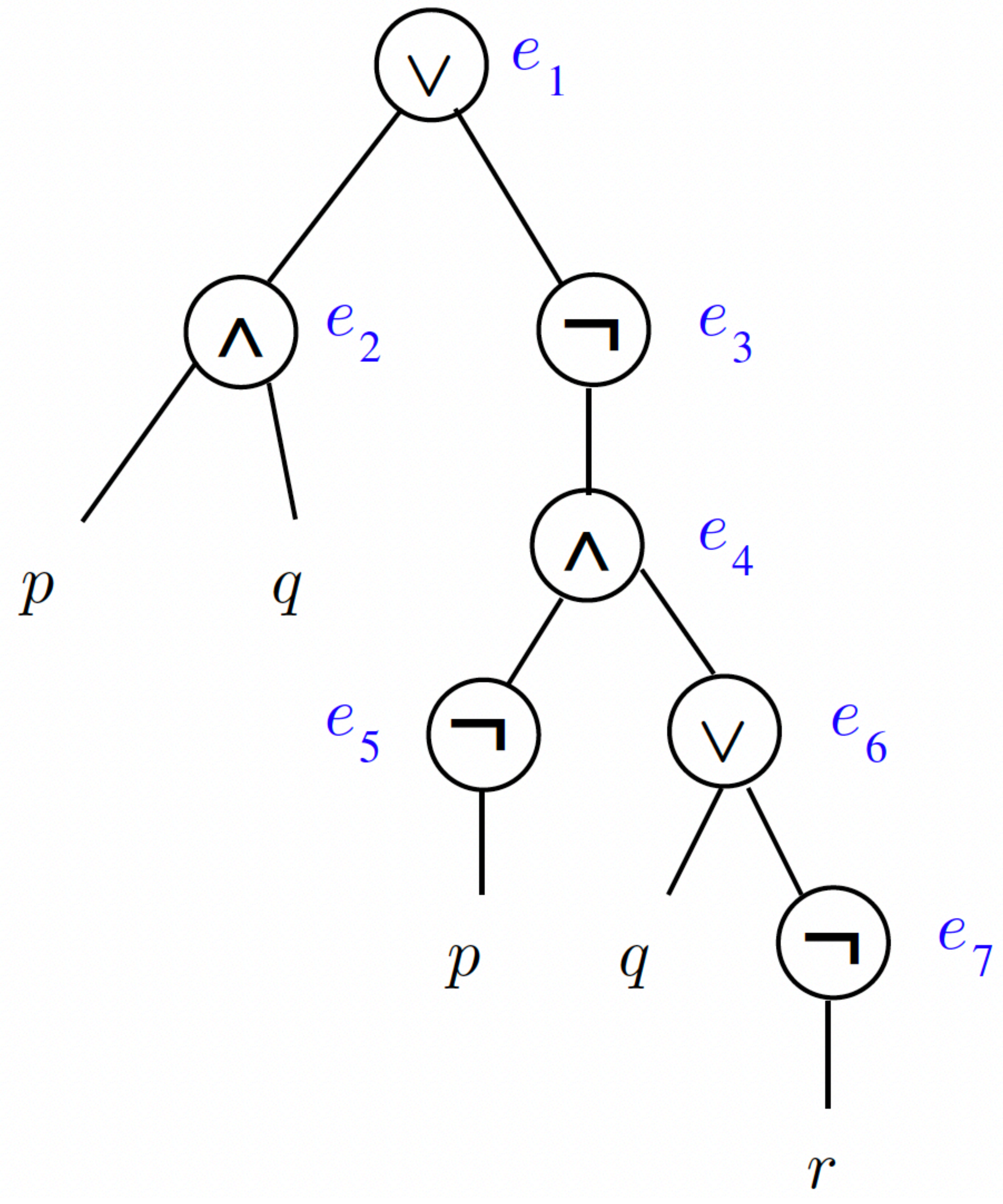
Example

Let F be $(p \wedge q) \vee \neg(\ \neg p \wedge (q \vee \neg r)\ )$



Example

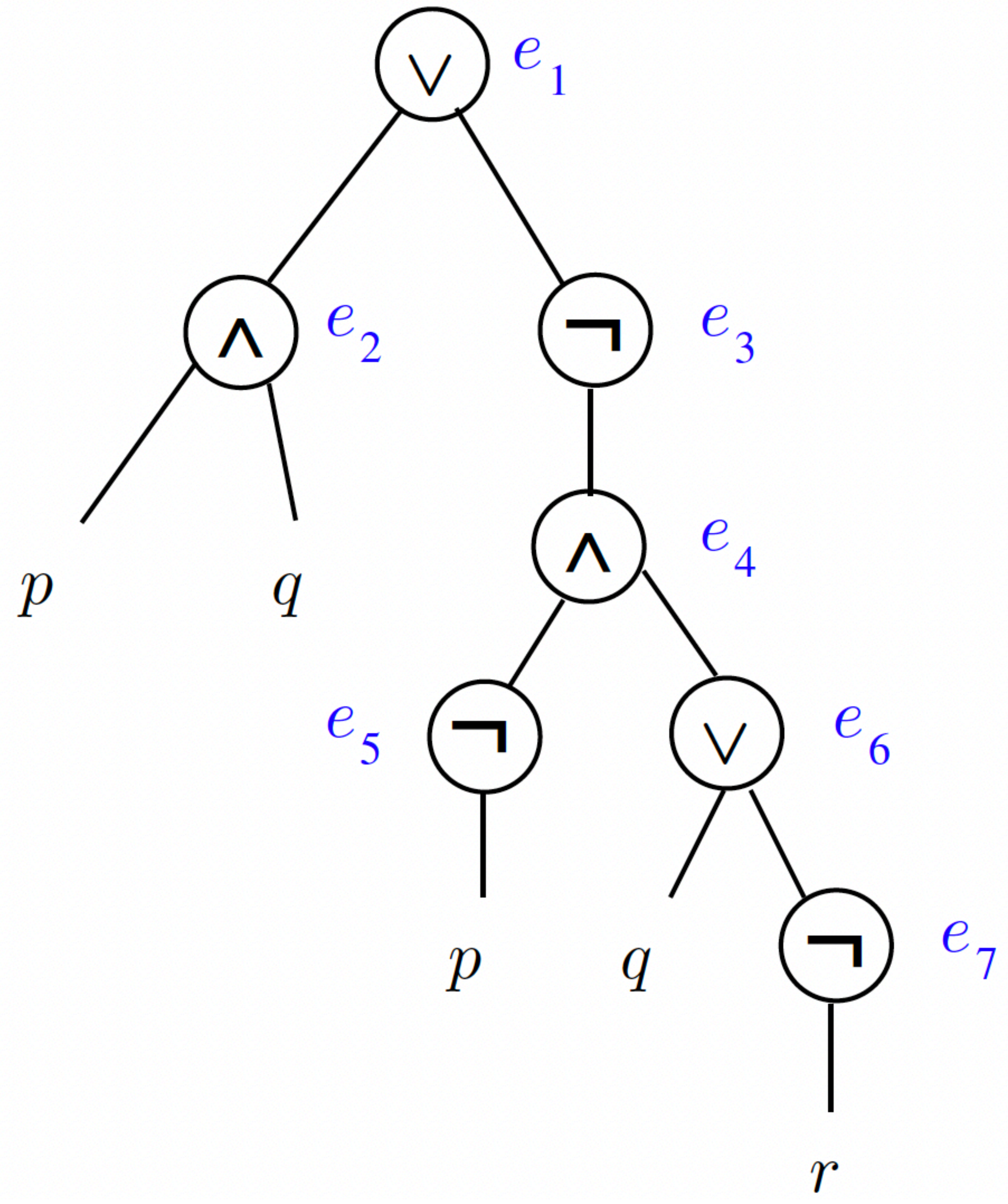
Let F be $(p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$



- e_1
- $e_1 \leftrightarrow e_2 \vee e_3$
- $e_2 \leftrightarrow p \wedge q$
- $e_3 \leftrightarrow \neg e_4$
- $e_4 \leftrightarrow e_5 \wedge e_6$
- $e_5 \leftrightarrow \neg p$
- $e_6 \leftrightarrow q \vee \neg e_7$
- $e_7 \leftrightarrow \neg r$

Example

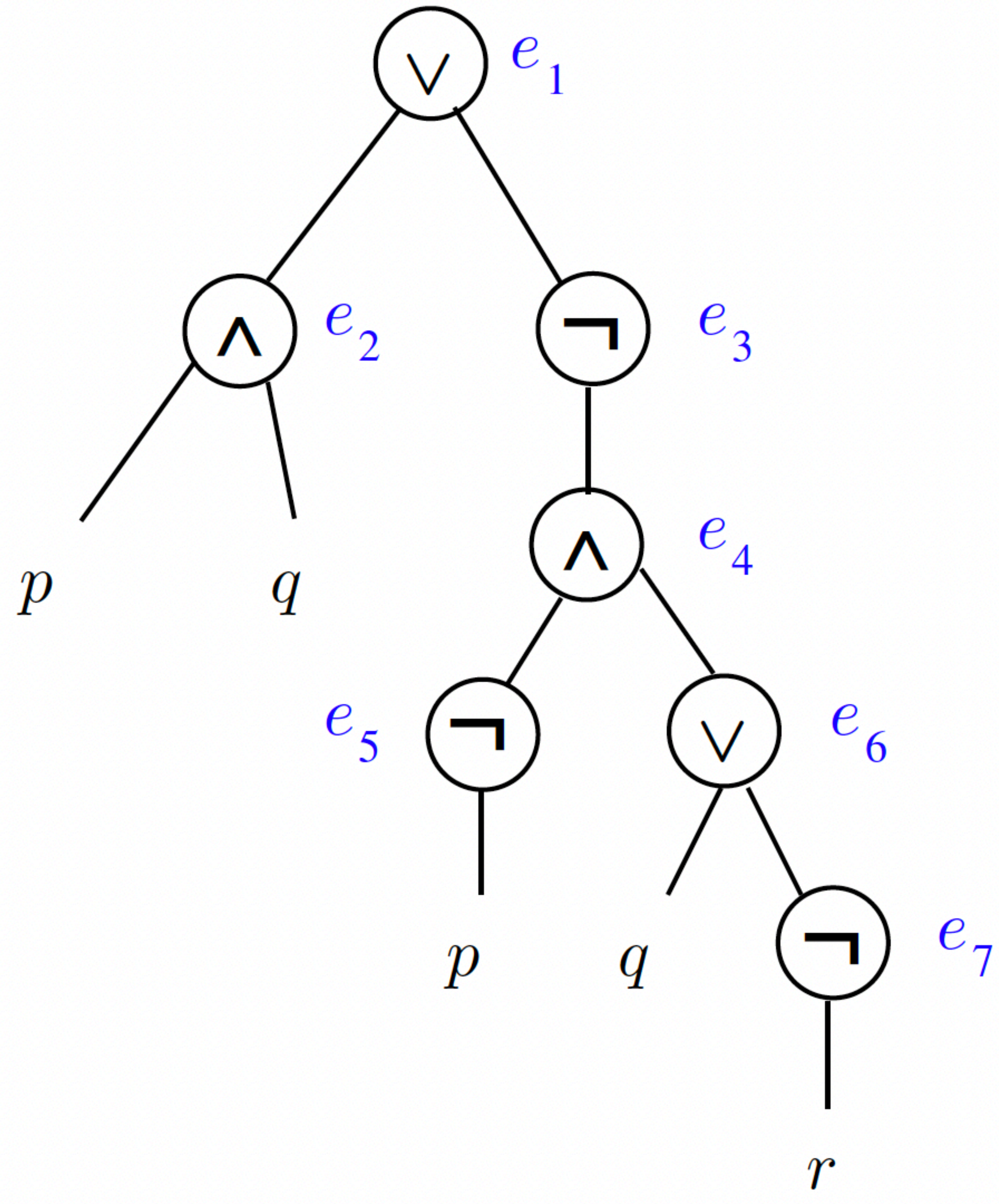
Let F be $(p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$



- e_1
- $e_1 \leftrightarrow e_2 \vee e_3$
  $\neg e_1 \vee e_2 \vee e_3$
  $\neg e_2 \vee e_1$
  $\neg e_3 \vee e_1$
- $e_2 \leftrightarrow p \wedge q$
- $e_3 \leftrightarrow \neg e_4$
- $e_4 \leftrightarrow e_5 \wedge e_6$
- $e_5 \leftrightarrow \neg p$
- $e_6 \leftrightarrow q \vee \neg e_7$
- $e_7 \leftrightarrow \neg r$

Example

Let F be $(p \wedge q) \vee \neg(\ \neg p \wedge (q \vee \neg r)\ )$



- e_1
- $e_1 \leftrightarrow e_2 \vee e_3$

  $\neg e_1 \vee e_2 \vee e_3$

  $\neg e_2 \vee e_1$

  $\neg e_3 \vee e_1$
- $e_2 \leftrightarrow p \wedge q$

  $\neg p \vee \neg q \vee e_2$

  $\neg e_2 \vee p$

  $\neg e_2 \vee q$
- $e_3 \leftrightarrow \neg e_4$
- $e_4 \leftrightarrow e_5 \wedge e_6$
- $e_5 \leftrightarrow \neg p$
- $e_6 \leftrightarrow q \vee \neg e_7$
- $e_7 \leftrightarrow \neg r$

What do we get?

A new formula with more propositions than the original one
NOT an equivalent formula

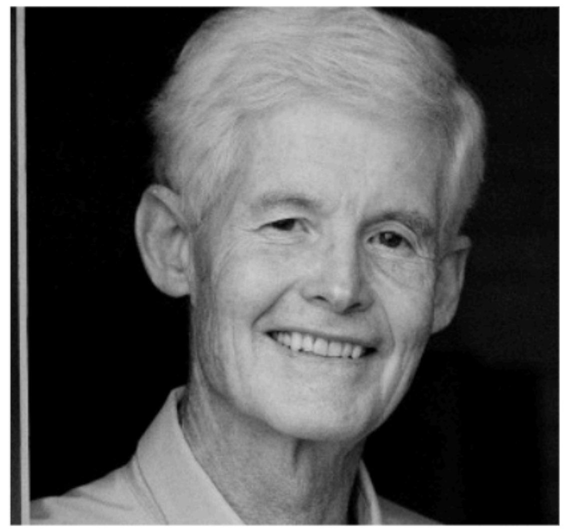
New formula is *satisfiable iff the original is satisfiable*
we call it *equisatisfiable*)

Size of resulting formula: *linear* in original size
good for use in satisfiability checking

The Boolean Satisfiability problem

A bit of history

Cook



Levin

Karp

The SAT problem

For F in CNF, exists $I : I \models F$?

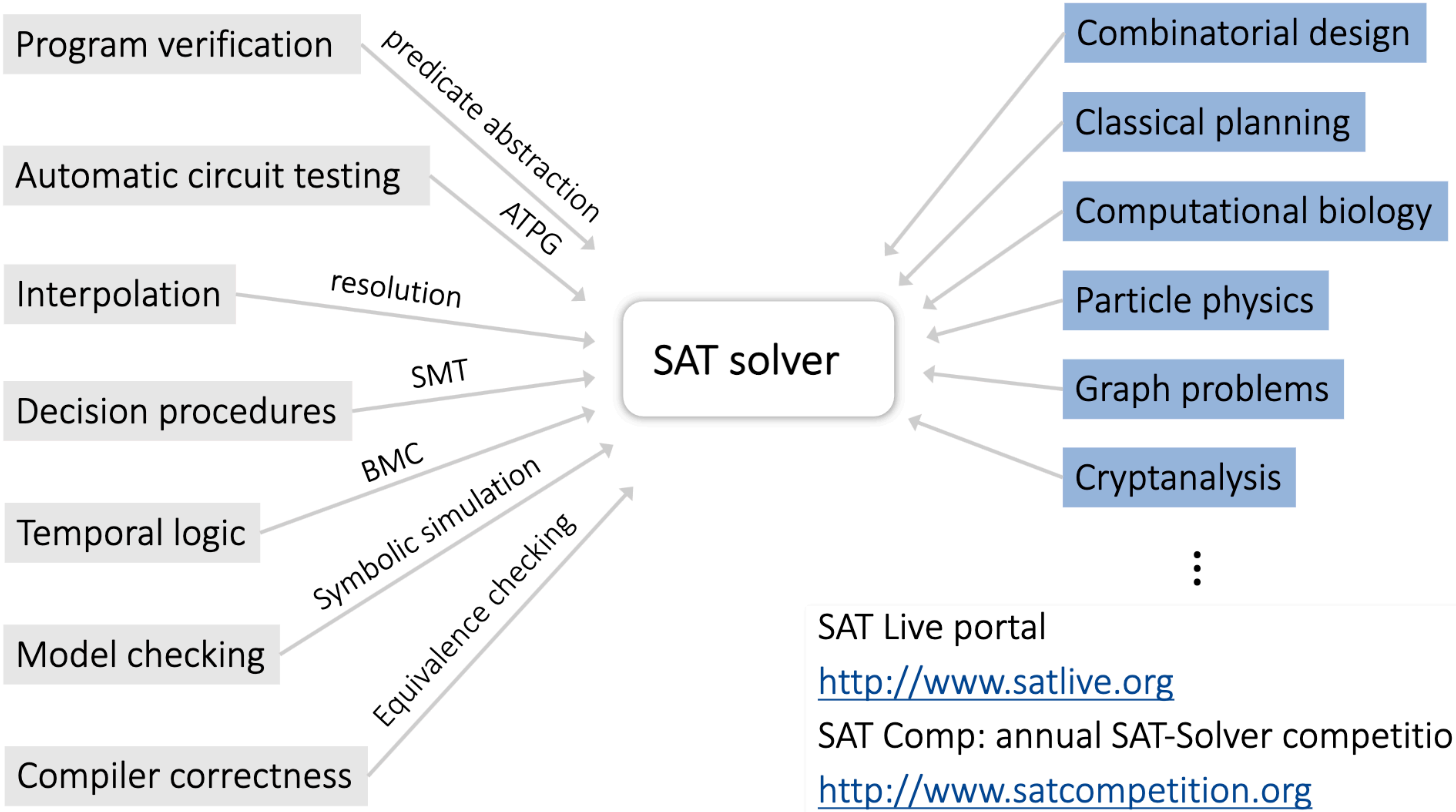
First NP-complete problem!

Cook-Levin Theorem:

SAT is NP-complete

Cook, *The complexity of theorem proving procedures*, 1971

Karp, *Reducibility among combinatorial problems*, 1972

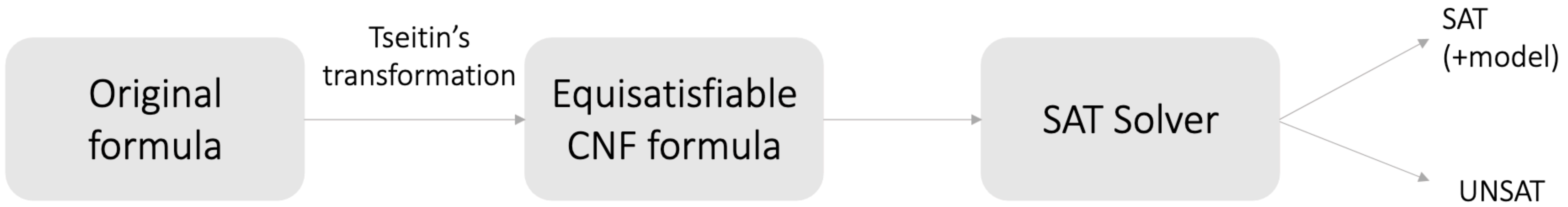


⋮

SAT Live portal
http://www.satlive.org
SAT Comp: annual SAT-Solver competition
http://www.satcompetition.org

A Modern SAT Solver

A Modern SAT Solver



Almost all SAT solvers today are based on DPLL (Davis-Putnam-Logemann-Loveland)

These algorithms are also called "Decision Procedures"

History Again

1962: the original algorithm known as DP (Davis-Putnam)

⇒ "simple" procedure for automated theorem proving

Davis and Putnam hired two programmers, Logemann and Loveland, to implement their ideas on the IBM 704.

Not all of the original ideas worked out as planned

⇒ refined algorithm is what is known today as DPLL

DPLL Insight

Two distinct approaches for the Boolean satisfiability problem

- Search
  - Find satisfying assignment by searching through all possible assignments
  - Example: truth table
- Deduction
  - Deduce new facts from set of known facts, i.e, application of proof rules
  - Example: semantic argument method
- DPLL combines search and deduction in a very effective way!

- Deductive principle underlying DPLL is propositional resolution
- Resolution can only be applied to formulas in CNF
- SAT solvers convert formulas to CNF to be able to perform resolution

Propositional Resolution

Consider two clauses in CNF:

$$C_1 : (l_1 \vee \ldots p \ldots \vee \ l_k\ )$$
$$C_2 : (l'_1 \vee \ldots \neg p \ldots \vee \ l'_n\ )$$

We can deduce a new clause C_3, called resolvent:

$$C_3 : (l_1 \vee \ldots \vee \ l_k \vee \ l'_1 \vee \ldots\ldots \vee \ l'_n\ )$$

Correctness:

1. If p is assigned T : since C_1 is SAT and since $\neg p$ is ⊥, $(l'_1 \vee \ldots\ldots \vee \ l'_n\ )$ must be true
2. If p is assigned ⊥ : since C_2 is SAT and since p is ⊥, $(l_1 \vee \ldots\ldots \vee \ l_k\ )$ must be true
3. Thus, C_3 must be true

Example

$$F : \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q \ .$$

From resolution

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

construct

$$F_1 : \ (\neg P \vee Q) \ \wedge \ P \ \wedge \ \neg Q \ \wedge \ Q \ .$$

From resolution

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

deduce that F, and thus the original formula, is unsatisfiable.

Unit Resolution or BCP

Consider two clauses in CNF:

$C_1 : \ p$

$C_2 : \ (l_1 \lor \ldots \neg p \ldots \lor l_n)$

We can deduce a new resolvent:

$C_3: \ (l_1 \lor \ldots \lor \ l_n)$

Unit clause: literal

- DPLL uses unit resolution
- Boolean Constraint Propagation: all possible applications of unit resolution on input

Restricted Resolution: BCP

Boolean Constraint Propagation (BCP)

If a clause contains one literal ℓ,

Set ℓ to $\top$:	$\cdots \land \overset{\top}{\not\ell} \land \cdots$
Remove all clauses containing ℓ:	$\cdots \land \cancel{(\cdots \lor \ell \lor \cdots)} \land \cdots$
Remove $\neg\ell$ in all clauses:	$\cdots \land (\cdots \lor \cancel{\neg\ell} \lor \cdots) \land \cdots$

based on the unit resolution

$$\frac{\ell \qquad \neg\ell \lor C}{C} \quad \leftarrow \text{clause}$$

Example:

$$F:\ (P)\ \land\ (\neg P \lor Q)\ \land\ (R \lor \neg Q \lor S)$$

(P) is a unit clause. Therefore, applying unit resolution

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

$$F':\ (Q)\ \land\ (R \lor \neg Q \lor S)\ .$$

Applying unit resolution again

$$\frac{Q \qquad R \lor \neg Q \lor S}{R \lor S}$$

$$F'':\ (R \lor S)$$

Basic DPLL (with BCP)

```
// returns SAT if CNF formula F is satisfiable; //
otherwise returns UNSAT

DPLL(F)
   G = BCP(F)
   if (G = ⊤) then return SAT
   else if (G = ⊥) then return UNSAT
   p = choose_var(G)
   if (DPLL(G[p ↦ ⊤])) then return SAT;
   else return (DPLL(G[p ↦ ⊥]));
```

Boolean constraint propagation

Decision heuristics

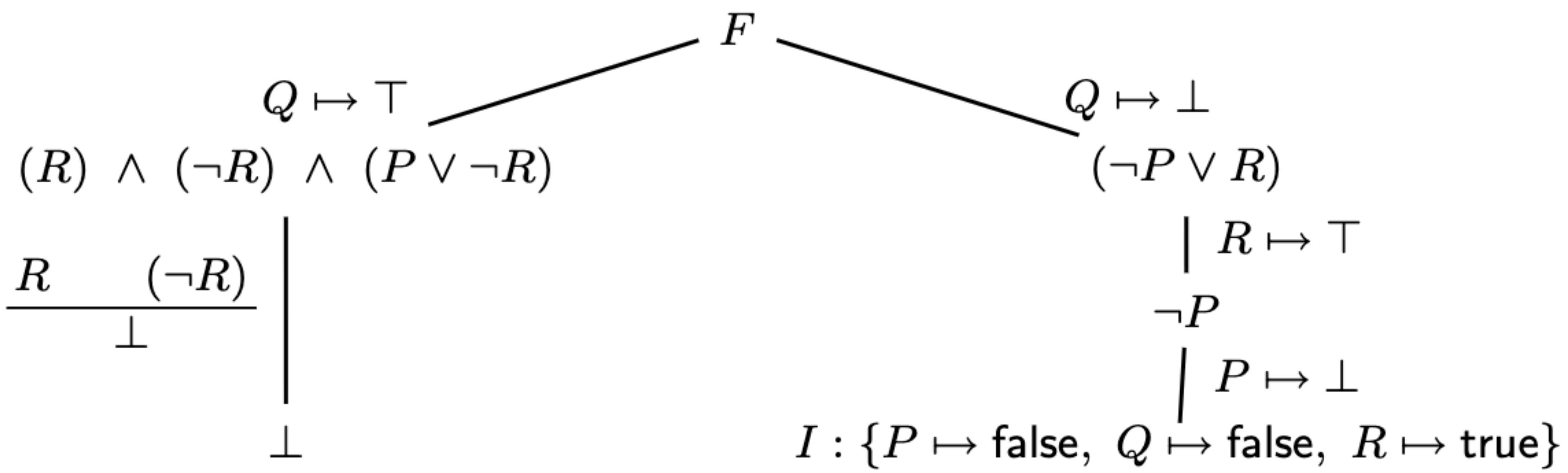
Backtracking

Example

$$F: \; (\neg P \vee Q \vee R) \;\wedge\; (\neg Q \vee R) \;\wedge\; (\neg Q \vee \neg R) \;\wedge\; (P \vee \neg Q \vee \neg R)\;.$$

No BCP Possible

On the first level of recursion, DPLL must branch. Branching on Q or R will result in unit clauses; choose Q.



Unit Resolution, optimized => PLP

Consider two clauses in CNF:

$C_1 : \; p$

$C_2 : \; (l_1 \vee \ldots \neg p \ldots \vee l_n)$

We can deduce a new resolvent:

$C_3 : \; (l_1 \vee \ldots \vee \; l_n)$

Unit clause: literal

if variable p appears only positively or only negatively in F , it should not be chosen by choose vars(F ′).

- DPLL uses unit resolution
- Boolean Constraint Propagation: all possible applications of unit resolution on input

Pure Literal Propagation (PLP)

If P occurs only positive (without negation), set it to $\top$.
If P occurs only negative set it to $\bot$.
Then do the simplifications as in Boolean Constraint Propagation

DPLL with PLP

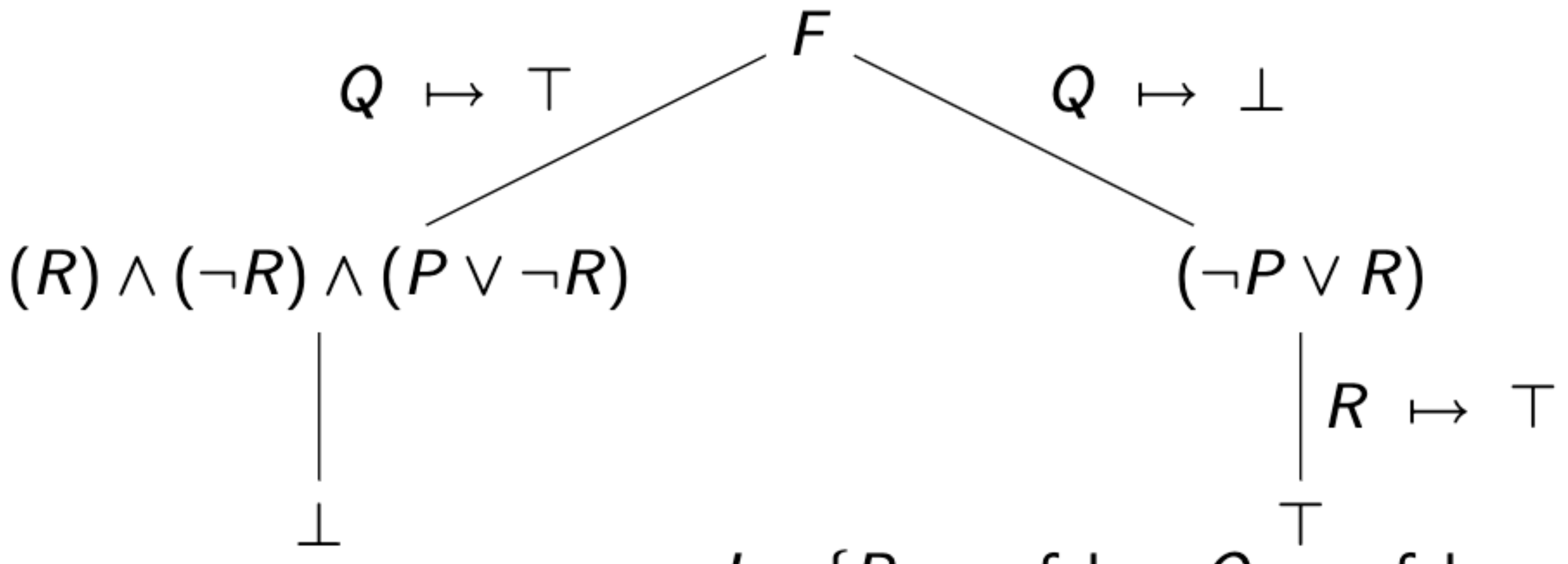
Decides the satisfiability of PL formulae in CNF

Decision Procedure DPLL: Given F in CNF

```
let rec DPLL F =
  let F′ = BCP F in
  let F″ = PLP F′ in
  if F″ = ⊤ then true
  else if F″ = ⊥ then false
  else
     let P = CHOOSE vars(F″) in
     (DPLL F″{P ↦ ⊤}) ∨ (DPLL F″{P ↦ ⊥})
```

Example

$$F: \; (\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg Q \lor \neg R)$$



$$I: \; \{P \mapsto \text{false}, \; Q \mapsto \text{false}, \; R \mapsto \text{true}\}$$

(No matter what P is)

Beyond DPLL

Learning conflict clauses that summarize conflicts and augmenting F with them

Non-chronological backtracking to earlier decision levels based on cause of conflict

Decision heuristics choose the next literal to add to the current partial assignment based on the state of the search.

Conflict-Driven Clause Learning (CDCL)

Further details: Reading for todays Class: Chapter 2.

Decision Procedures: An Algorithmic Point of View Authors: Daniel Kroening, Ofer Strichman

SAT solving landscape today

- CDCL based solvers routinely solve problems with hundred of thousands or even millions of variables.
- But still possible to create very small instances that take very long

Not every small SAT problem is easy

- An example: the pigeonhole problem
- Is it possible to place n pigeons into m holes?
- Obvious for humans!
- But turns out to be very difficult to solve for SAT solvers!

Encoding the Pigeon hole problem in PL

Let's encode this for $m = n - 1$.

- Let $p_{i,j}$ stand for "pigeon i placed in j 'th hole"
- Given we have $n - 1$ holes, how to say i 'th pigeon must be placed in some hole?
- Given we have n pigeons, how to say every pigeon must be placed in some hole?

$$
\begin{aligned}
& p_{1,1} \vee p_{1,2} \vee \ldots p_{1,n-2} \vee p_{1,n-1} \\
\wedge\ & p_{2,1} \vee p_{2,2} \vee \ldots p_{2,n-2} \vee p_{2,n-1} \\
& \vdots \\
\wedge\ & p_{n,1} \vee p_{n,2} \vee \ldots p_{n,n-2} \vee p_{n,n-1}
\end{aligned}
$$

Pigeon hole problem, cont.

More concise way of writing this:

$$\bigwedge_{0 \leq k < n} \left(\bigvee_{0 \leq l < n-1} p_{k,l} \right)$$

We also need to state that multiple pigeons cannot be placed into same hole:

$$\bigwedge_{k} \bigwedge_{i} \bigwedge_{j \neq i} \neg p_{ik} \vee \neg p_{jk}$$

With $n > 25$, this formula cannot be solved by competitive SAT solvers!

Problem: Conflict clauses talk about specific holes/pigeons, but problem is symmetric!

Research on *symmetry breaking*

Variations of the Boolean Satisfiability problem

Maximum Satisfiability (MaxSAT)

Given CNF formula F, find assignment maximizing the number of satisfied clauses of F

- If F is satisfiable, the solution to the MaxSAT problem is the number of clauses in F.
- If F is unsatisfiable, we want to find a maximum subset of F's clauses whose conjunction is satisfiable.

Partial MaxSAT

Given CNF formula F where each clause is marked as **hard** or **soft**, find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses

- Similar to MaxSAT, but we distinguish between two kinds of clauses
- **Hard clauses**: clauses that must be satisfied
- **Soft clauses**: clauses that we would like to, but do not have to, satisfy

- In normal SAT, all clauses are implicitly hard clauses
- In MaxSAT, all clauses are implicitly soft clauses
- In this sense, Partial MaxSAT is a generalization over both SAT and MaxSAT

Partial Weighted MaxSAT

Given CNF formula F where each clause is marked as **hard** or **soft** and is assigned a **weight**, find an assignment that satisfies all hard clauses and maximizes the sum of the weights of satisfied soft clauses

Partial MaxSAT is an instance of partial weighted MaxSAT where all clauses have equal weight

Summary

Today

- DPLL algorithm for SAT solving
- One challenge for current SAT solvers
- Variations of the satisfiability problem (e.g., MaxSAT)

Next

- First-order logic