SATs

08 March 2025 00:19

Normal forms

Among the formulas equivalent to a given formula, some are of particular interest (the variables here stand for atoms):

- Conjunctive Normal forms (CNF)
  - $(A \vee B \vee C) \wedge (D \vee X) \wedge (\neg A)$
  - ANDs of ORs of literals (atoms or negations of atoms)
  - A clause in this context is a disjunction of literals
- Disjunctive Normal Form (DNF)
  - $(P \wedge Q \wedge A) \vee (R \wedge \neg Q) \vee (\neg A)$
  - ORs of ANDs of literals
  - A clause in this context is a conjunction of literals

Theorem: Every proposition is equivalent to a formula in CNF!

Theorem: Every proposition is equivalent to a formula in DNF!

Making use of truth tables to convert to DNF

Every proposition can be expressed in DNF (ORs of ANDs)!

Express $(P \to Q) \wedge Q$ in DNF

We do it using a truth table

P	Q	$(P \to Q)$	$(P \to Q) \wedge Q$
T	**T**	**T**	**T**
T	**F**	**F**	**F**
F	**T**	**T**	**T**
F	**F**	**T**	**F**

- Enumerate all the **T** rows from the conclusion column
  - Row 1 gives $P \wedge Q$
  - Row 3 gives $\neg P \wedge Q$
- Take **OR** of these formulas
- **Final answer** is $(P \wedge Q) \vee (\neg P \wedge Q)$

Making use of truth tables to convert to CNF

Every proposition can be expressed in CNF (ANDs of ORs)!

Express $(P \to Q) \wedge Q$ in CNF

We do it by using a truth table

P	Q	$(P \to Q)$	$(P \to Q) \wedge Q$
T	**T**	**T**	**T**
T	**F**	**F**	**F**
F	**T**	**T**	**T**
F	**F**	**T**	**F**

- Enumerate all the **F** rows from the conclusion column
  - Row 2 gives $P \wedge \neg Q$
  - Row 4 gives $\neg P \wedge \neg Q$

- Row 2 gives $P \wedge \neg Q$
- Row 4 gives $\neg P \wedge \neg Q$
- Do **AND** of negations of each of these formulas
- We obtain $\neg(P \wedge \neg Q) \wedge \neg(\neg P \wedge \neg Q)$
- **Finally**: equivalent to $(\neg P \vee Q) \wedge (P \vee Q)$ by De Morgan

Making use of equivalences to convert to CNF/DNF

If $P \leftrightarrow Q$ and P occurs in A, then replacing P by Q in A leads to a proposition B, such that $A \leftrightarrow B$

a=b

f(a)

f(a) leads to g(b)

f(a) = g(b)

Example:

- consider the formula $P \rightarrow Q \rightarrow (P \wedge Q)$
- we know that classically $(Q \rightarrow (P \wedge Q)) \leftrightarrow (\neg Q \vee (P \wedge Q))$
- this is an instance of $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$
- when replacing $Q \rightarrow (P \wedge Q)$ by $\neg Q \vee (P \wedge Q)$ in $P \rightarrow Q \rightarrow (P \wedge Q)$, we obtain $P \rightarrow (\neg Q \vee (P \wedge Q))$
- $P \rightarrow Q \rightarrow (P \wedge Q)$ and $P \rightarrow (\neg Q \vee (P \wedge Q))$ are equivalent

Making use of equivalences to convert to CNF/DNF

We can convert a formula to an equivalent formula in CNF or DNF using "known" equivalences.

Example: express $(P \rightarrow Q) \wedge Q$ in CNF using known equivalences

- $(\boxed{P \rightarrow Q}) \wedge Q$
- $\leftrightarrow (\neg P \vee Q) \wedge Q$ – using $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

Example: express $\neg(P \wedge \neg Q) \wedge \neg(\neg P \wedge \neg Q)$ in CNF using known equivalences

- $\boxed{\neg(P \wedge \neg Q)} \wedge \neg(\neg P \wedge \neg Q)$
- $\leftrightarrow\ (\neg P \vee \neg\neg Q) \wedge \boxed{\neg(\neg P \wedge \neg Q)}$ – using de Morgan
- $\leftrightarrow\ (\neg P \vee \boxed{\neg\neg Q}) \wedge (\neg\neg P \vee \neg\neg Q)$ – using de Morgan
- $\leftrightarrow\ (\neg P \vee Q) \wedge (\boxed{\neg\neg P} \vee \neg\neg Q)$ – using double negation elim.
- $\leftrightarrow\ (\neg P \vee Q) \wedge (P \vee \boxed{\neg\neg Q})$ – using double negation elim.
- $\leftrightarrow (\neg P \vee Q) \wedge (P \vee Q)$ – using double negation elim.

11/30

Satisfiability of CNF formulas

Problem definition: Given a CNF formula can we set **T** or **F** value to each variable to satisfy the formula?

- **Example**: Consider the formula $(A \vee \neg B) \wedge (C \vee B)$
- Is it satisfiable?
- Satisfiable by setting $A =$ **T**, $B =$ **F** and $C =$ **T**
- Known as **CNF Satisfiability** or simply **SAT**

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$: the class of problems which we can solve in polynomial time
$\mathcal{NP}$: the class of problems where we can verify a potential solution/answer in polynomial time

Clearly, $\mathcal{P} \subseteq \mathcal{NP}$ (solving is a (hard) way of verifying)

What about the other direction? Is $\mathcal{P} = \mathcal{NP}$?

- Status unknown!
- Million dollar question

What do most people believe?

- $\mathcal{P}$ is not equal to $\mathcal{NP}$

Why haven't we been able to prove it then?

- Hard to rule out all possible polytime algorithms?

Hardness for the class $\mathcal{NP}$

$\mathcal{NP}$: the class of problems where we can verify a potential solution/answer in polynomial time

Definition: A problem is $\mathcal{NP}$-hard if it is **at least as hard as** any problem in $\mathcal{NP}$.

More precisely, a problem X is $\mathcal{NP}$-hard if any problem $Y \in \mathcal{NP}$ can be solved

- using an oracle for solving X
- plus a polynomial overhead for translating between X and Y

If $\mathcal{P} \neq \mathcal{NP}$ then a problem being $\mathcal{NP}$-hard means it cannot be solved in polynomial time!

Great, except no one knew how to show existence of a single $\mathcal{NP}$-hard problem!

More Precise Definition:

- A problem X is **NP-hard** if every problem Y in **NP** can be:
  1. **Reduced** to X using an **oracle** for X (an oracle is like a magical black box that solves X instantly).

More Precise Definition:

- A problem X is $\textbf{NP-hard}$ if every problem Y in $\textbf{NP}$ can be:
  1. **Reduced** to X using an **oracle** for X (an oracle is like a magical black box that solves X instantly).
  2. The reduction process should add only **polynomial overhead**, meaning it doesn't make things much harder computationally.

Why Does This Matter?

If you could solve an $\textbf{NP-hard}$ problem in polynomial time, you could also solve every problem in $\textbf{NP}$ in polynomial time. This connects to the famous $\mathbf{P} \neq \mathbf{NP}$ question:

- If $\mathbf{P} \neq \mathbf{NP}$, then $\textbf{NP-hard}$ problems **cannot** be solved in polynomial time.

Cook-Levin Theorem (1971/1973):
CNF-Satisfiability (**SAT**) is $\mathcal{NP}$-hard

How do you show a problem, say X, is $\mathcal{NP}$-hard?

- A polytime reduction from any of the known $\mathcal{NP}$-hard problems, say SAT, to X
- That is, show how you can solve SAT using an oracle for X
- Plus a polynomial overhead for the translation

Tens of thousands of problems known to be $\mathcal{NP}$-hard

Significance of SAT

Many practical problems can be encoded into SAT
(e.g., formal verification, planning/scheduling, etc.)

A possible solution (valuation) can be verified "efficiently" NP Hard

No known algorithm to solve the problem "efficiently" in all cases

In practice, SAT solvers are very efficient
($\mathcal{NP}$-hardness is the worst case)

Why not consider DNF instead of CNF?

Theorem: Any propositional formula can be expressed in CNF

Theorem: Any propositional formula can be expressed in DNF

Theorem: CNF satisfiability is $\mathcal{NP}$-hard

How hard is DNF satisfiability?

- Example of a DNF formula:
  $(A \wedge \neg B \wedge C) \vee (\neg X \wedge Y) \vee (Z)$
- Is it satisfiable?
- **Trivial** to check in polytime!
- Just pick any clause, and set variables to **T** or **F**.

Why not use DNFs then?

Because changing a formula from CNF to DNF can cause exponential blowup!

Convert $(A \vee B) \wedge (C \vee D)$ into DNF

Remember: $P \wedge (Q \vee R) \leftrightarrow (P \wedge Q) \vee (P \wedge R)$

$$\begin{aligned}
& (A \vee B) \wedge (C \vee D) \\
\leftrightarrow\; & ((A \vee B) \wedge C) \vee ((A \vee B) \wedge D) \\
\leftrightarrow\; & (C \wedge (A \vee B)) \vee (D \wedge (A \vee B)) \\
\leftrightarrow\; & (C \wedge A) \vee (C \wedge B) \vee (D \wedge A) \vee (D \wedge B)
\end{aligned}$$

Consider the CNF formula: $(P_1 \vee Q_1) \wedge \cdots \wedge (P_n \vee Q_n)$

Expressing this formula in DNF requires 2^n clauses

Algorithms for SAT?

Brute force for SAT with N variables and M clauses needs $2^N \cdot N \cdot M$ time

- There are 2^N truth assignments
- For each truth assignment and each clause, verify if it is satisfied in N time

Can we solve SAT faster than 2^N? Say 1.999999999^N?

Conjecture (Strong Exponential Time Hypothesis (SETH)): SAT cannot be solved in $(2-\alpha)^N \cdot \text{poly}(N+M)$ time for any constant $\alpha > 0$

SAT solvers

Many state-of-the-art SAT solvers are based on the **Davis-Putman-Logemann-Loveland** algorithm (DPLL)

Basic idea (does a lot of pruning instead of brute force):

1. **Easy** cases
   - Atom p only appears as either p or $\neg p$ (but not both): assign truth value accordingly
2. **Branch** on choosing a variable p and set a truth value to it
   - This choice needs to be done **cleverly**
   - If $p =$ **T**: remove all clauses containing p and remove all literals $\neg p$ from clauses
   - If $p =$ **F**: remove all clauses containing $\neg p$ and remove all literals p from clauses
3. Keep running the above steps **until**
   - All clauses have been removed (all true): return SAT
   - One clause is empty (one is false): **backtrack** in Step 2 and choose a different truth value for p; if it is not possible to backtrack, return UNSAT

Apply the DPLL algorithm to
$(\neg p \vee q \vee r) \wedge (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$

Here is a possible run of the algorithm:

$(\neg p \vee q \vee r) \wedge (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$

$p =$ **T**

$(q \vee r) \wedge (\neg q \vee r)$

$q =$ **T**

(r)

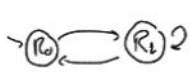
$r =$ **T**

SAT

Seems hard, but read it slowly and

it all makes sense

- A robot can move between 2 rooms R_0 and R_1 as follows:



i.e., ⓐ it starts in room R_0
ⓑ when in room R_0 it can only move to room R_1
ⓒ when in room R_1 it can move to R_1 or R_0

- To model this system, let us consider the following atomic propositions: $p_0, p_1, p_2, \dots$
  such that p_i means that the robot is in room R_0 at step i (i.e., after i moves)
  and $\neg p_i$ means that the robot is in room R_1 at step i
- we can model the initial state of the robot as follows: p_0, i.e., the robot is initially in room R_0 (ⓐ)
- we can model a transition as follows: $(p_i \to \neg p_{i+1}) \wedge (\neg p_i \to (\neg p_{i+1} \vee p_i))$, i.e., ⓑ ∧ ⓒ
- we can capture two moves as follows: $\underbrace{p_0}_{\text{initial position}} \wedge \underbrace{(p_0 \to \neg p_1) \wedge (\neg p_0 \to (\neg p_1 \vee p_1))}_{\text{1st move}} \wedge \underbrace{(p_1 \to \neg p_2) \wedge (\neg p_1 \to (p_2 \vee \neg p_2))}_{\text{2nd move}}$

- Could we prove that when in room R_1, the robot will be in room R_1 next? ← call it P
  N_B: when in R_1, the robot can move to R_0
  How do we formally prove that this property fails? we prove its negation, i.e. $\neg(\neg p_i \to \neg p_{i+1})$
- let's show that this fails after 2 steps, i.e.: $p_0 \wedge (p_0 \to \neg p_1) \wedge (\neg p_0 \to (\neg p_1 \vee p_1)) \wedge (p_1 \to \neg p_2) \wedge (\neg p_1 \to (\neg p_2 \vee p_2))$
  $\wedge \neg(\neg p_1 \to \neg p_2)$

- We'll show this using a SAT solver: (1) convert the formula to a CNF using logical equivalences
  (2) use DPLL to check whether it is satisfiable

(1) $p_0 \wedge (p_0 \to \neg p_1) \wedge (\neg p_0 \to (\neg p_1 \vee p_1)) \wedge (p_1 \to \neg p_2) \wedge (\neg p_1 \to (\neg p_2 \vee p_2)) \wedge \neg(\neg p_1 \to \neg p_2)$
$\Leftrightarrow p_0 \wedge (\neg p_0 \vee \neg p_1) \wedge (\neg\neg p_0 \vee \neg p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_2) \wedge (\neg\neg p_1 \vee \neg p_2 \vee p_2) \wedge \neg(\neg\neg p_1 \vee \neg p_2)$ — elim →
$\Leftrightarrow p_0 \wedge (\neg p_0 \vee \neg p_1) \wedge (\neg\neg p_0 \vee \neg p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_2) \wedge (\neg\neg p_1 \vee \neg p_2 \vee p_2) \wedge \neg\neg\neg p_1 \wedge \neg\neg p_2$ — de Morg.
$\Leftrightarrow p_0 \wedge (\neg p_0 \vee \neg p_1) \wedge (p_0 \vee \neg p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee \neg p_2 \vee p_2) \wedge \neg p_1 \wedge p_2$ — ¬¬ elim
CNF ↗

(2) use DPLL: $\cancel{p_0} \wedge \cancel{(\neg p_0 \vee \neg p_1)} \wedge \cancel{(p_0 \vee \neg p_1 \vee p_1)} \wedge \cancel{(\neg p_1 \vee \neg p_2)} \wedge \cancel{(p_1 \vee \neg p_2 \vee p_2)} \wedge \cancel{\neg p_1} \wedge \cancel{p_2}$
$p_0 = T$ $p_1 = F$ $p_2 = T$: the formula is satisfiable
This gives us a run of our robot: $R_0 \to R_1 \to R_0$, which shows that the property P does not hold, i.e. we obtained a counterexample.

A robot can move between two rooms R_0 and R_1 as follows:

1. It starts in room R_0.
2. When in room R_0, it can only move to room R_1.
3. When in room R_1, it can move to R_1 or R_0.

To model this system, consider the following atomic propositions: $p_0, p_1, p_2, \dots$ where p_i means the robot is in room R_0 at step i (i.e., after i moves) and $\neg p_i$ means the robot is in room R_1 at step i.

We can model the initial state of the robot as follows: p_0 - the robot is initially in room R_0. (i)

We can model a transition as follows:

$$(p_i \to \neg p_{i+1}) \wedge (\neg p_i \to (p_{i+1} \vee \neg p_{i+1}))$$

i.e., (ii) & (iii)

We can capture two moves as follows:

i.e., (ii) & (iii)

We can capture two moves as follows:

$$p_2 = (p_0 \rightarrow \neg p_1) \wedge (\neg p_0 \rightarrow (p_1 \vee \neg p_1)) \wedge (p_1 \rightarrow \neg p_2) \wedge (\neg p_1 \rightarrow (p_2 \vee \neg p_2))$$

Can we prove that when in room R_1, the robot will be in room R_1 next? i.e., when in R_1, the robot can move to R_0. Call it P.

How do we formally prove that this property fails? We prove its negation, i.e., $\neg(p_i \rightarrow p_{i+1})$.

Let's show that this fails after 2 steps:

$$p_2 = (p_0 \rightarrow \neg p_1) \wedge (\neg p_0 \rightarrow (p_1 \vee \neg p_1)) \wedge (p_1 \rightarrow \neg p_2) \wedge (\neg p_1 \rightarrow (p_2 \vee \neg p_2)) \wedge \neg(p_1 \rightarrow p_2)$$

We'll show this using a SAT solver:

1. Convert the formula to a CNF using logical equivalences.
2. Use DPLL to check whether it is satisfiable.

1

$$(p_0 \rightarrow \neg p_1) \wedge (\neg p_0 \rightarrow (p_1 \vee \neg p_1)) \wedge (p_1 \rightarrow \neg p_2) \wedge (\neg p_1 \rightarrow (p_2 \vee \neg p_2)) \wedge \neg(p_1 \rightarrow p_2)$$

$$\Leftrightarrow (\neg p_0 \vee \neg p_1) \wedge (\neg p_0 \vee p_1 \vee \neg p_1) \wedge (p_1 \vee \neg p_1) \wedge (\neg p_1 \vee p_2 \vee \neg p_2) \wedge (p_1 \wedge \neg p_2)$$

$$\Leftrightarrow (\neg p_0 \vee \neg p_1) \wedge (\neg p_0 \vee p_1) \wedge (\neg p_0 \vee \neg p_1) \wedge (p_1 \vee \neg p_1) \wedge (\neg p_1 \vee p_2) \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \wedge \neg$$