

Practice Problem Set 3

Instructions:

- The following problems are meant for you to practice, so that your understanding of the topic improves.
- You must solve all problems to get the maximum benefit from practice problems.
- You must not submit your solutions to these problems. These are not going to be graded.
- A problem may have multiple solution techniques. Discussion among students is strongly encouraged in order to understand different perspectives.
- Questions marked * are comparably more difficult than the rest.

1. Resolving Pigeonhole Principle

If $n + 1$ pigeons are put in n pigeonholes, it is clear that some pigeonhole must contain more than 1 pigeon. This is referred to as the *Pigeonhole Principle*. Here, we wish to apply resolution to see how a computer may prove this when $n = 2$.

Problem: Prove, using resolution, that it is impossible to place 3 pigeons into 2 holes such that no two pigeons share the same hole. Proceed as follows:

1. Choose suitable variables to encode whether some pigeon is in some hole
2. Formulate the condition that every pigeon has to be in some hole as a conjunction of CNF clauses
3. Formulate the condition(s) that no hole contains two pigeons as a conjunction of CNF clauses
4. Put all the clauses together to get a formula which is satisfiable if and only if the statement of the pigeonhole principle is false

Now apply resolution on the clauses (may be a lot, but try to reason intuitively to get an idea of which clauses to use) to show this formula is unsatisfiable, thus proving the pigeonhole principle for $n = 2$

2. Resolution and DPLL

A unit propagation step in the DPLL algorithm can be thought of as a resolution of a clause with another (unit) clause (which may be assumed or derived) to get the derived clause. Turns out you can extend this intuition further.

Show that the following set of clauses gives p_6 as a learnt (unit) clause in a run of DPLL with CDCL. You may assume that the first branching decision is $p_6 = 0$. Show the implication graph that leads to a conflict, and show how p_6 is learnt as a clause from this

conflict.

$$\begin{aligned}c_1 &= \neg p_1 \vee p_2 \\c_2 &= \neg p_1 \vee p_3 \vee p_5 \\c_3 &= \neg p_2 \vee p_4 \\c_4 &= \neg p_3 \vee \neg p_4 \\c_5 &= p_1 \vee p_5 \vee \neg p_2 \\c_6 &= p_2 \vee p_3 \\c_7 &= p_2 \vee \neg p_3 \vee p_7 \\c_8 &= p_6 \vee \neg p_5 \\c_9 &= p_6 \vee \neg p_1 \\c_{10} &= p_4 \vee \neg p_7\end{aligned}$$

Write the conflict clause (this may not be unique). A conflict clause is one such that every literal in this clause has been derived or assumed to be false; hence this clause has become false. Let these literals be OR'ed to get the formula F (same as conflict clause for now). Now pick any derived literal l such that $\neg l$ is in F and look at the clause c_l which derived it. Resolve c_l with F wrt to l to get the new F . Note that the negation of all literals in F are still derived/assumed (why?). Keep repeating this process. What do you observe?

General problem (try for fun): Let us suppose we run DPLL on an unsatisfiable formula. Give a linear time algorithm in terms of the number of steps in the run to generate resolution proof of unsatisfiability from the run of DPLL.

3. Horn and 2-cnf

Let φ be any Boolean formula in Conjunctive Normal Form (CNF). Consider the claim: From any CNF formula φ , one can compute in polynomial time an equisatisfiable formula $\psi_1 \wedge \psi_2$ where ψ_1 is a Horn formula and ψ_2 is a 2-CNF formula.

If you think statement is true, explain how to compute ψ_1 and ψ_2 and show that $\psi_1 \wedge \psi_2$ is equisatisfiable with φ . Otherwise give a counterexample and prove that in this case no ψ_1 and ψ_2 can exist.

Hint : The property of Horn formula is that each clause has atmost one positive literal. Now some general clause may have more than one positive literal think how can we rename some variable with constraints satisfied to convert that Follow up question : Satisfiability of Horn formula and 2-CNF formula are both in polytime. Suppose the above claim were true does that imply that satisfiability of any CNF formula is polytime as well? Justify your answer

4. Horn is in P

In class, we established that the satisfiability of a Horn formula \mathbb{F} can be determined in polynomial time using the Horn Algorithm. It is an instructive exercise to show that propositional resolution—specifically its restricted form known as unit resolution—is sufficient to achieve the same efficiency.

Let n be the number of variables and m be the number of clauses. Suppose we repeatedly apply the following rule: *If \mathbb{F} contains a unit clause $[x]$, perform all possible resolutions with $[x]$ and simplify the formula.*

- Show that if \mathbb{F} contains no unit clauses, then \mathbb{F} is necessarily satisfiable. (Hint: Consider the "minimal" assignment where every variable is set to false.)
- Explain why, in each step where we resolve a unit clause $[x]$ with a clause containing $[\neg x]$, the formula \mathbb{F} remains a Horn formula.
- Observe that each variable x can be derived as a unit clause at most once. Use this to prove that the total number of resolution steps is $O(n)$, and that the total work performed is bounded by a polynomial in the number of literal occurrences.

5. **ChatGPT breaks?!?** ChatGPT was asked to provide linear-time algorithms for specific cases. Snapshots of conversation with ChatGPT are given. There may or may not be error(s) in its algorithm. Please point out the error(s), if you find any, in each of the responses below and report the fix for the same. Question (i): Find error(s)/fix(-es), if any:

Response limit to 50 words. Please give me a linear time algorithm to convert a given propositional formula to CNF. You are not allowed to use Tseitin encoding.



A linear-time algorithm to convert a propositional formula to CNF involves:

1. **Eliminate implications** using: $A \rightarrow B \equiv \neg A \vee B$.
2. **Move negations inward** using De Morgan's laws and double negation.
3. **Distribute OR over AND** (apply distributive laws).
4. **Simplify** any tautologies or contradictions.

This ensures a CNF form in linear time.



Question (ii): Find error(s)/fix(-es), if any:

Response limit to 50 words. Please give me a linear time algorithm to convert a given propositional formula to DNF.



To convert a propositional formula to DNF in linear time:

1. **Eliminate implications** ($A \rightarrow B \equiv \neg A \vee B$).
2. **Move negations inward** using De Morgan's laws.
3. **Distribute AND over OR**.
4. **Simplify** by eliminating redundancies and tautologies.

The result will be a disjunctive normal form.



Question (iii): Find error(s)/fix(-es), if any:

Response limit to 50 words. Please give me a linear time algorithm to check satisfiability of a given propositional formula in DNF.



To check satisfiability of a DNF formula in linear time:

1. **Parse the DNF** formula as a disjunction of conjunctions.
2. For each conjunction, check if it contains no contradictions (e.g., both P and $\neg P$).
3. If at least one conjunction is consistent, the formula is satisfiable.



Question (iv): Find error(s)/fix(-es), if any:

Response limit to 50 words. Please give me a linear time algorithm to check validity of a given propositional formula in CNF.



To check validity of a CNF formula in linear time:

1. **Check each clause:** A formula is valid if every clause contains at least one literal that is always true.
2. **Check for unsatisfiable clauses:** If all clauses are satisfiable and no clause is a contradiction, the formula is valid.



Question (v): Find error(s)/fix(-es), if any:

Response limit to 50 words. Since converting any formula into CNF requires linear time (you proved) and checking satisfiability of any CNF formula requires linear time (you proved), checking satisfiability of any formula is linear. Correct?



Not quite. Converting a formula to CNF isn't always linear in the general case. While checking satisfiability of a CNF formula is linear, converting arbitrary formulas (especially non-CNF ones) often requires exponential time due to the complexity of the transformations, making the overall process not linear.



6. Minimal Horn Model

Let ϕ be a propositional formula over a set of variables V . A **model** (satisfying assignment) is represented as a set $M \subseteq V$ containing exactly the variables assigned to **True**. A **Horn formula** is a set of clauses where each clause contains at most one positive literal.

1. The Closure Property Consider the Horn clause $C = (\neg p \vee \neg q \vee r)$, which can be viewed as the implication $(p \wedge q) \rightarrow r$. Let $M_1 = \{p, q, r\}$ and $M_2 = \{r\}$. Both are models of C .

a **Observation:** Calculate $M_3 = M_1 \cap M_2$. Is M_3 a model of C ? What happens if you try this with $M_1 = \{p, q, r\}$ and $M_2 = \{p\}$?

b **Proof:** Let ϕ be a Horn formula and let M_1, M_2 be two distinct models of ϕ . Prove that their intersection $M_1 \cap M_2$ is also a model of ϕ .

Hint: Assume $M_1 \cap M_2$ fails a clause $C \in \phi$ and derive a contradiction regarding the original models.

2. The Minimal Model The closure property suggests that "shrinking" models by intersection eventually leads to a unique "smallest" model.

a **Observation:** Suppose a formula ϕ has models $M_1 = \{a, b, c\}$, $M_2 = \{a, b\}$, and $M_3 = \{a, d\}$. If ϕ is Horn, what is the smallest possible model it must have based on these three?

b **Proof:** Using the result from Part 1, prove that every satisfiable Horn formula has a **unique minimal model** M_{min} such that M_{min} is a subset of M for every model M of ϕ .

3. Generalization and Counterexamples Does the property of being "closed under intersection" apply to all propositional logic, or is it a special characteristic of Horn formulas?

7. Min, Max Horn

Consider the following two variations of Horn-SAT:

- **Variation A (MIN-HORN):** Given a satisfiable Horn formula ϕ and an integer k , does there exist a satisfying assignment where **at most** k variables are set to True?
- **Variation B (MAX-HORN):** Given a satisfiable Horn formula ϕ and an integer k , does there exist a satisfying assignment where **at least** k variables are set to True?

Prove that:

1. Variation A can be solved in polynomial time.
2. The Independent Set problem (given a graph, find if a large set of vertices exists such that no two vertices in the set are connected by an edge) can be converted to an instance of solving Variation B on a Horn formula in polynomial time.

Note: there is no known polynomial time algorithm for the Independent Set problem. Part 2 shows MAX-HORN is at least as hard as the Independent Set problem, so it too has no known polynomial time algorithm.

8. CNF Conversion and Resolution Complexity

Let p_1, p_2, \dots, p_n be propositional variables. Consider the unsatisfiable formula

$$\psi_n := \phi_n \wedge \neg\phi_n$$

where

$$\phi_n := (p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow \dots \leftrightarrow p_n))).$$

Answer the following:

(a) Blow-up in CNF conversion

Convert

$$\psi_3 := \phi_3 \wedge \neg\phi_3$$

into CNF using only logical equivalences and distributive laws (i.e., without introducing any new variables).

- (a) Write the resulting CNF explicitly.
 - (b) Count the number of clauses obtained.
 - (c) Generalize your argument to show that for ψ_n , the number of clauses produced by such a conversion grows as 2^n .
- (b) **Exponential lower bound for Resolution.**
Consider the CNF obtained in part (a) for $n = 3$, without introducing auxiliary variables. Count the number of resolution steps required to derive \perp . Explain why any Resolution refutation of this CNF will require exponentially many steps in n .
- (c) **Tseitin encoding and polynomial refutation.**
- (a) Introduce fresh auxiliary variables to represent the subformulas of ϕ and write the Tseitin encoding of ϕ_n .
 - (b) Construct the CNF of ψ_n using this encoding.
 - (c) Show that the number of clauses in the encoded formula is linear in n .
 - (d) Explain why a Resolution refutation of this encoded CNF requires only polynomially many steps.

9. Encoding Reachability with Bounded Length

Let $G = (V, E)$ be a directed graph, and let $s, t \in V$. We want to encode the existence of a directed path from s to t of length at most k as a propositional CNF formula.

To do this, we introduce auxiliary propositional variables:

$$P_{v,i} \quad \text{for each } v \in V \text{ and } 0 \leq i \leq k,$$

with the intended meaning:

$$P_{v,i} = \text{“there exists a directed path from } s \text{ to } v \text{ of length exactly } i\text{.”}$$

Using these variables, construct a CNF formula $\Psi(G, s, t, k)$ such that

$$\Psi(G, s, t, k) \text{ is satisfiable} \leftrightarrow \text{there exists a directed path from } s \text{ to } t \text{ of length } \leq k.$$

Answer the following:

(a) **(Base step)**

Write CNF clauses that enforce that the path starts at s at step 0, and no other vertex is reached at step 0.

(b) **(Forward propagation)**

Suppose $(u \rightarrow v) \in E$. Write a CNF clause that enforces:

$$\text{if } u \text{ is reached at step } i, \text{ then } v \text{ can be reached at step } i + 1.$$

Write such clauses for all edges and all steps $0 \leq i < k$.

(c) **(No spontaneous reachability)**

We must also ensure that a vertex cannot be reached at step $i + 1$ unless it has a predecessor reached at step i .

For a fixed vertex v , let

$$In(v) = \{u \in V : (u \rightarrow v) \in E\}.$$

Write a CNF clause that enforces:

$$\text{if } v \text{ is reached at step } i + 1, \text{ then some } u \in In(v) \text{ was reached at step } i.$$

(d) **(Target condition)**

Write a clause that enforces that t must be reached at some step $0 \leq i \leq k$.

10. **Removing edges to create a DAG** Let $G = (V, E)$ be a directed graph.

A fundamental fact from graph theory is:

A directed graph is acyclic if and only if there exists a topological ordering of its vertices.

A topological ordering is a strict ordering of the vertices such that for every edge $(u \rightarrow v)$, vertex u appears before vertex v .

We will encode this property using propositional logic.

Auxiliary variables

For every pair of distinct vertices $u, v \in V$, introduce propositional variables:

$$X(u, v)$$

with the intended meaning:

$$X(u, v) = \text{“vertex } u \text{ appears before vertex } v \text{ in the ordering.”}$$

For every directed edge $(u \rightarrow v) \in E$, introduce propositional variables:

$$Y(u, v)$$

with the intended meaning:

$$Y(u, v) = \begin{cases} \mathbf{true} & \text{edge } (u \rightarrow v) \text{ is kept} \\ \mathbf{false} & \text{edge } (u \rightarrow v) \text{ is removed} \end{cases}$$

Our goal is to construct a propositional formula whose satisfying assignments correspond to selecting a set of edges that forms an acyclic graph.

(a) **(Consistency of ordering)**

Explain why, for every pair of distinct vertices u, v , both $X(u, v)$ and $X(v, u)$ cannot be true simultaneously.

Write a CNF clause that enforces this constraint.

(b) **(Transitivity of ordering)**

A valid ordering must satisfy transitivity. That is, for all distinct vertices u, v, w :

if $X(u, v)$ and $X(v, w)$ are true, then $X(u, w)$ must be true.

Write this constraint as:

- a logical implication, and
- its equivalent CNF clause.

(c) **(Edge consistency constraint)**

To ensure that the kept edges form an acyclic graph, we enforce:

If edge $(u \rightarrow v)$ is kept, then u must appear before v .

Write this constraint as:

- a logical implication involving $Y(u, v)$ and $X(u, v)$, and
- its equivalent CNF clause.

(d) **(Why cycles are impossible)**

Consider a cycle of kept edges:

$$u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow \cdots \rightarrow u_k \rightarrow u_1$$

Using the constraints from part (c), write the ordering constraints that must hold.

Explain why these constraints contradict transitivity.

Conclude that the kept edges cannot contain a cycle.

(e) **(Optimization objective)**

Our goal is to remove the minimum number of edges so the graph becomes acyclic.

Explain why maximizing the number of variables $Y(u, v)$ assigned **true** achieves this goal.

11. Learning with CDCL

Consider a CDCL solver processing variables x_1, \dots, x_9 . The solver is currently at Decision Level 4 with the following partial assignment history:

- Level 1: $x_1 = 1$ (Decision)
- Level 2: $x_2 = 1$ (Decision), $x_3 = 0$ (Propagated)

- Level 3: $x_4 = 0$ (Decision)

At Level 4, the solver makes the decision $x_5 = 1$. The clause database contains the following clauses, where C_1, \dots, C_5 represent the indices of the clauses in the database:

$$\begin{aligned} C_1 &:= (\neg x_5 \vee x_6) \\ C_2 &:= (\neg x_5 \vee x_7) \\ C_3 &:= (\neg x_1 \vee x_3 \vee \neg x_6 \vee x_8) \\ C_4 &:= (\neg x_2 \vee \neg x_7 \vee x_9) \\ C_5 &:= (\neg x_8 \vee \neg x_9) \end{aligned}$$

- Perform Unit Propagation starting from the decision $x_5 = 1$. Construct the Implication Graph showing all nodes, directed edges labeled with the causing clause index, and identify the conflict node κ .
- Identify the First Unique Implication Point (1-UIP). By analyzing the graph cuts or performing resolution backwards from the conflict to the 1-UIP, derive the learned clause.
- Using the learned clause, determine the backtracking level (assertion level). Describe the state of the solver (which variable is assigned and what value) immediately after the non-chronological backtracking occurs.

12. **Conflict-Clause Driven Learning** Consider a CNF formula \mathcal{F} consisting of the following eight clauses over the variables $\{x_1, x_2, x_3, x_4, x_5, x_6\}$:

$$\begin{aligned} (C_1) & (\neg x_1 \vee x_2) \\ (C_2) & (\neg x_1 \vee \neg x_2 \vee x_7) \\ (C_3) & (x_1 \vee x_2) \\ (C_4) & (x_1 \vee \neg x_2) \\ (C_5) & (\neg x_1 \vee \neg x_3 \vee x_4) \\ (C_6) & (\neg x_3 \vee x_5) \\ (C_7) & (\neg x_4 \vee x_6) \\ (C_8) & (\neg x_5 \vee \neg x_6) \end{aligned}$$

- Unit Propagation** Suppose the solver begins at Decision Level 1 (*DL1*). Show that the subset of clauses $\{C_1, C_2, C_3, C_4\}$ forces a specific assignment for x_7 and x_2 via unit propagation alone. List the assignments and the order of the clauses triggered.
- The Conflict at Depth.** With the assignments from part (i) fixed at *DL1*, assume the solver makes the following decision: **Set $x_3 = 1$ at Decision Level 2 (*DL2*)**. Trace the resulting unit propagations.
 - Identify the conflicting clause where all literals evaluate to false.
 - Draw the Implication Graph for this conflict. Label each node with its assignment ($x_i = v$) and its decision level (*DL*).
- Clause Learning and Backtracking** Analyze the implication graph from part (ii):
 - Identify the First Unique Implication Point (1UIP).
 - Derive the learned clause by analyzing the graph cuts.
 - If the solver uses non-chronological backtracking, to which decision level will it backtrack after learning this clause? Explain why this is more efficient than chronological backtracking in this specific instance.

13. **DPLL run** Let's try to solve the HornSAT problem using DPLL. We can convert each Horn clause into a CNF clause by simply rewriting $(a \rightarrow b)$ as $(\neg a \vee b)$, which preserves the semantics of the formula. The resultant formula is in CNF.

Suppose our version of DPLL always assigns 1 to a decision variable before assigning 0 (if needed). How many backtrackings are needed if we run this version of DPLL on the (CNF-ised version of) the following Horn formulae, assuming DPLL always chooses the unassigned variable with the smallest subscript when choosing a decision variable?

(a)

$$\left(\left(\bigwedge_{i=0}^{n-1} x_i \right) \rightarrow x_n \right) \wedge \bigwedge_{i=0}^{n-1} (x_n \rightarrow x_i) \wedge \left(\left(\bigwedge_{i=0}^n x_i \right) \rightarrow \perp \right)$$

(b)

$$\bigwedge_{i=0}^{n-1} ((x_i \rightarrow x_{n+i}) \wedge (x_{n+i} \rightarrow x_i) \wedge (x_i \wedge x_{n+i} \rightarrow \perp))$$

14. **Renamable Horn** A *renamable Horn formula* is a CNF formula that can be turned into a Horn formula by negating (all occurrences of) some of its variables. For example,

$$(p_1 \vee \neg p_2 \vee \neg p_3) \wedge (p_2 \vee p_3) \wedge (\neg p_1)$$

can be turned into a Horn formula by negating p_1 and p_2 .

Let F be a CNF formula over variables x_1, \dots, x_n .

- (a) Explain why deciding whether F is renamable Horn can be viewed as choosing, for each variable x_i , whether to negate all its occurrences or leave them unchanged. Introduce a Boolean variable r_i to represent this choice and state its intended meaning.
- (b) Let C be a clause of F . Argue that after renaming, C is Horn if and only if no two literals in C are simultaneously positive. Using this idea, describe how to construct a 2-CNF formula G over the variables r_1, \dots, r_n that forbids such pairs for every clause.
- (c) Prove that G is satisfiable if and only if F is a renamable Horn formula. Explain how a satisfying assignment for G yields an explicit renaming that turns F into a Horn formula.