Practice Problem Set 3 Solutions

1. **Resolving Pigeonhole Principle**

Solution: Let $P_{i,j}$ denote the proposition "Pigeon i is in Hole j", where pigeons $i \in \{1, 2, 3\}$ and holes $j \in \{1, 2\}$.

The Clause Set (CNF)

1. Existence Constraints (Every pigeon must be in a hole):

$$C_1 : P_{1,1} \vee P_{1,2}$$
$$C_2 : P_{2,1} \vee P_{2,2}$$
$$C_3 : P_{3,1} \vee P_{3,2}$$

2. Disjoint Constraints (No two pigeons in Hole 1):

$$C_4 : \neg P_{1,1} \vee \neg P_{2,1}$$
$$C_5 : \neg P_{1,1} \vee \neg P_{3,1}$$
$$C_6 : \neg P_{2,1} \vee \neg P_{3,1}$$

3. Disjoint Constraints (No two pigeons in Hole 2):

$$C_7 : \neg P_{1,2} \vee \neg P_{2,2}$$
$$C_8 : \neg P_{1,2} \vee \neg P_{3,2}$$
$$C_9 : \neg P_{2,2} \vee \neg P_{3,2}$$

Resolution:
The high level idea is to show that if two pigeons are occupying 2 holes without being together, there would be no space left for the third pigeon. So first we prove that pigeon 3 cannot reside in hole 2, then we prove it cannot reside in hole 1. Finally, we use the fact that it must be in some hole to get the empty clause.

The procedure may seem stretched out, but will feel tractable once you try to see the meaning of each step.

1: Prove Pigeon 3 cannot be in Hole 2 ($\neg P_{3,2}$)

1. Resolve C_1 ($P_{1,1} \vee P_{1,2}$) and C_4 ($\neg P_{1,1} \vee \neg P_{2,1}$) on $P_{1,1}$:

$$R_1 : P_{1,2} \vee \neg P_{2,1}$$

2. Resolve R_1 and C_8 ($\neg P_{1,2} \vee \neg P_{3,2}$) on $P_{1,2}$:

$$R_2 : \neg P_{2,1} \vee \neg P_{3,2}$$

3. Resolve R_2 and C_2 $(P_{2,1} \vee P_{2,2})$ on $P_{2,1}$:

$$R_3 : P_{2,2} \vee \neg P_{3,2}$$

4. Resolve R_3 and C_9 $(\neg P_{2,2} \vee \neg P_{3,2})$ on $P_{2,2}$:

$$R_4 : \neg P_{3,2}$$

2: Prove Pigeon 3 cannot be in Hole 1 ($\neg P_{3,1}$)

5. Resolve C_1 $(P_{1,1} \vee P_{1,2})$ and C_7 $(\neg P_{1,2} \vee \neg P_{2,2})$ on $P_{1,2}$:

$$R_5 : P_{1,1} \vee \neg P_{2,2}$$

6. Resolve R_5 and C_5 $(\neg P_{1,1} \vee \neg P_{3,1})$ on $P_{1,1}$:

$$R_6 : \neg P_{2,2} \vee \neg P_{3,1}$$

7. Resolve R_6 and C_2 $(P_{2,1} \vee P_{2,2})$ on $P_{2,2}$:

$$R_7 : P_{2,1} \vee \neg P_{3,1}$$

8. Resolve R_7 and C_6 $(\neg P_{2,1} \vee \neg P_{3,1})$ on $P_{2,1}$:

$$R_8 : \neg P_{3,1}$$

3: Final Contradiction

9. Resolve R_4 $(\neg P_{3,2})$ and C_3 $(P_{3,1} \vee P_{3,2})$ on $P_{3,2}$:

$$R_9 : P_{3,1}$$

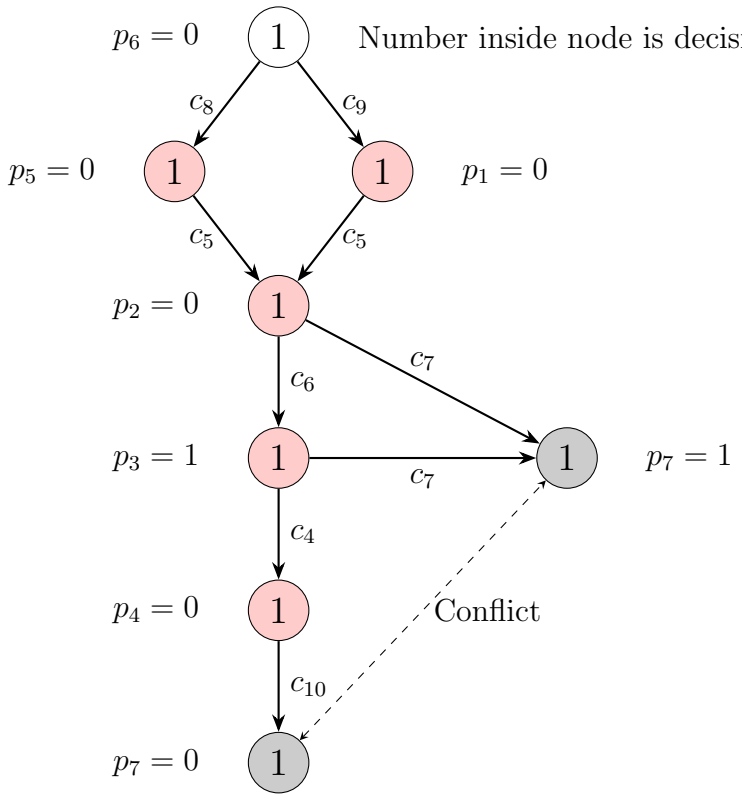
10. Resolve R_9 $(P_{3,1})$ and R_8 $(\neg P_{3,1})$ on $P_{3,1}$:

$$\emptyset$$

Resolution to the empty clause ($\emptyset$) proves that the set of clauses is unsatisfiable. Therefore, 3 pigeons cannot fit into 2 holes without overlap.

2. **Resolution and DPLL**

Solution: Let's draw the implication graph corresponding to the branching decision $p_6 = 0$.



1. There is only one decision node in this implication graph. So, we can indeed learn the clause p_6 (draw a cut intercepting the two edges coming out of the unique root of the implication graph in this case).

   Food for thought: If you use the first unique implication point in this implication graph, you'd obtain the learnt clause: p_2

2. Let's start with the conflict clause c_7. This is a clause all of whose literals become false if we follow the chain of unit propagations $p_2 = 0, p_3 = 1, p_4 = 0, p_7 = 0$. So $F = p_7 \vee p_2 \vee \neg p_3$.

   Food for thought: You can also consider c_{10} *as a conflict clause, and repeat this exercise.*

3. It is derived from $\neg p_7$, $\neg p_2$ and p_3.

4. Pick a derived literal with its negation in F, say p_3. It is derived using clause c_6. Now resolve F with c_6 to get $F' = p_7 \vee p_2$. Update $F := F'$. Note that negation of every literal in the new F is derived/assumed.

5. Pick a derived literal with its negation in F, say $\neg p_2$. It is derived using c_5. Resolve F with c_5 to get $F' = p_7 \vee p_5 \vee p_1$. Update $F := F'$. Note that negation of every literal in the new F is derived/assumed.

6. Repeat the same procedure with p_1 and p_5 to eventually get $F = p_7 \vee p_6$.

Observe that the final F was derived purely by resolution of the given clauses. And it is precisely the learnt clause. You can try to see this by noting that we are 'de-deriving' our way through the DPLL tree at each step. Eventually, F is exhausted of the derived variables and contains only assumed variables (decision variables). So the disjunction of negation of assumed literals is the final resolved clause, which is precisely the learnt clause.

3. **Horn and 2-cnf**

Solution: The claim is indeed true.
For each positive literal x in φ, introduce a new variable q and replace x by $\neg q$. Then enforce that q behaves like $\neg x$ by adding the equivalence:

$$q \leftrightarrow \neg x.$$

This equivalence is same as:

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

This is 2-cnf clause form.
After replacing all positive literals like this the original formula will only contain negative literals and is thus horn formula and along with that added conditions are 2-cnf form so original formula is now converted to Horn formula $\wedge$ 2-CNF formula.
Follow up question:
The intuition written is wrong since even though Horn Sat and 2-Sat separately return sat for formula we can still not be sure of satisfiability of full formula since there is and between these two formulas.

4. **Horn is in P**

Solution: If there are no unit clauses, the horn formula must be satisfiable.

Why?

Set all variables to be false. Any clause with a positive literal x must also contain negative literals. Since all variables are false, the negative literals $\neg x$ are true. Hence, every clause is satisfied.

If there is a unit clause, say L, remove all clauses containing L, they are trivially satisfied. Any clause containing a literal $\neg L$ loses that literal. If a clause becomes empty, the formula must be UNSAT.

Each resolution step strictly simplifies the formula.

Let n, m be number of variables and number of clauses in a Horn Formula. Each variable can be set to true atmost once. Each clause is processed only when a literal is removed. Thus, total work is bounded by the number of literal occurences, which is polynomial.

(Note: anytime there are no unit clauses, the formula becomes satisfiable as noted above.)

(Turns out we only need unit resolution for Horn formulas)

5. **ChatGPT breaks?!?**

Answer(i)

Solution:

- Error: Distribute OR over AND requires exponential time (worst-case).
- Fix 1: Hence, overall algorithm given by ChatGPT for converting into CNF is exponential (worst-case) in time.

- Fix 2: There doesn't exist any linear time algorithm to convert a given propositional formula into CNF without Tseitin encoding.

Answer(ii)

Solution:

- Error: Distribute AND over OR requires exponential time (worst-case).
- Fix 1: Hence, overall algorithm given by ChatGPT for converting into DNF is exponential(worst-case) in time.
- Fix 2: There doesn't exist any linear time algorithm to convert a given propositional formula into DNF.

Answer(iii)

Solution: Correct. No errors.

Answer (iv)

Solution: Correct. No errors.

Answer (v)

Solution:

- Notice that: ChatGPT discards and corrects its own answer it gave in Question (iv) (where it gave a linear time algorithm to convert a propositional formula into CNF in linear time, which is incorrect). ChatGPT now accepts that there is no linear time algorithm to convert into CNF.
- However, it still doesn't correct with one error that it made in Question (vi) (where it gave a linear time algorithm to check satisfiability of a propositional formula in CNF, which is also incorrect)
- Fix: There does not exist linear time algorithm to check satisfiability of a given propositional formula in CNF.
- Even though the answer it gave that checking satisfiability of any formula requires exponential time(worst-case) is correct, but reasoning is incorrect. The correct answer should be: Not only is converting into CNF formula going to take exponential time(worst-case), but checking satisfiability of CNF formula also requires exponential time(worst-case).

6. **Minimal Horn Model**

Solution:

1. Let ϕ be a Horn formula, and let M_1 and M_2 be two models of ϕ. We must show that the assignment corresponding to $M = M_1 \cap M_2$ satisfies every clause $C \in \phi$.

   By definition, a Horn clause C contains at most one positive literal. It can be written in the general form:

   $$C = \neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_k \vee q$$

   where $k \geq 0$, and q is an optional positive literal (if q is absent, the clause is a negative/goal clause).

   We consider how M evaluates C. There are two main cases based on why M_1 and M_2 satisfy C:

   - **Case 1: At least one model satisfies C by falsifying a premise.** Suppose, without loss of generality, that M_1 satisfies C because it assigns False to some p_i. This means $p_i \notin M_1$. Since $p_i \notin M_1$, it follows strictly from set intersection that $p_i \notin M_1 \cap M_2$. Therefore, the intersection model M also assigns False to p_i, satisfying the negative literal $\neg p_i$, and thus satisfying the clause C.
   - **Case 2: Both models satisfy C by making the conclusion true.** Suppose neither M_1 nor M_2 falsifies any p_i. This means $\{p_1, \ldots, p_k\} \subseteq M_1$ and $\{p_1, \ldots, p_k\} \subseteq M_2$. In order for both models to satisfy C, they must both assign True to the positive literal q (note: if q were absent, this case would be impossible as the models would falsify the clause, contradicting that they are models). Since $q \in M_1$ and $q \in M_2$, we have $q \in M_1 \cap M_2$. Thus, M assigns True to q, satisfying the clause C.

   Since M satisfies an arbitrary Horn clause $C \in \phi$ in all possible cases, M satisfies all clauses in ϕ. Therefore, $M = M_1 \cap M_2$ is a model of ϕ.

2. Let ϕ be a satisfiable Horn formula defined over a finite set of variables V. Let $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ be the set of all models of ϕ. Because ϕ is satisfiable, $\mathcal{M}$ is non-empty ($n \geq 1$).

   Define M_{min} as the intersection of all models of ϕ:

   $$M_{min} = \bigcap_{i=1}^{n} M_i$$

   Since V is finite, n is finite. By applying the result from Part 1 inductively, the intersection of any finite number of models of a Horn formula is also a model. Therefore, M_{min} is a model of ϕ.

   Since M_{min} is the intersection of all $M_i \in \mathcal{M}$, $M_{min} \subseteq M_i$ for all i. This means M_{min} is a minimal model. Furthermore, any minimal model must contain M_{min}, making M_{min} the **unique** minimal model.

3. Consider the standard CNF formula consisting of a single clause with two positive literals:

   $$\phi = x \vee y$$

This formula has three models (sets of true variables):

$$M_1 = \{x\}, \quad M_2 = \{y\}, \quad M_3 = \{x, y\}$$

The intersection of M_1 and M_2 is

$$M = M_1 \cap M_2 = \{x\} \cap \{y\} = \emptyset$$

But the model $M = \emptyset$ corresponds to assigning False to both x and y. Under this assignment, the clause $(x \vee y)$ evaluates to False. Thus, $M_1 \cap M_2$ is not a model of ϕ. This shows that the model intersection property is a special characteristic of Horn clauses and fails when clauses are allowed to have multiple positive literals.

7. **Min, Max Horn**

Solution:

1. The standard Marking Algorithm used to solve Horn-SAT is enough to solve Variation A. The algorithm is:

   (a) Initialize all variables to False.

   (b) While there exists an unsatisfied clause $(p_1 \wedge \cdots \wedge p_m \to q)$ where all p_i are currently True and q is False:

   (c) Set q to True.

   (d) If any purely negative clause $(\neg p_1 \vee \cdots \vee \neg p_n)$ is violated, return Unsatisfiable.

   Since this algorithm never sets a variable to True unless absolutely forced by the clauses, the resulting assignment is has the least number of True propositions. We run this algorithm in $\mathcal{O}(|\phi|)$ time and check if the number of variables set to True is less than or equal to k.

2. The Independent Set problem asks: Given an undirected graph $G = (V, E)$ and an integer k, does there exist a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and no two vertices in S are connected by an edge?

   We construct a Horn formula ϕ_G as follows:

   - **Variables:** For each vertex $v \in V$, create a boolean variable x_v. (Setting x_v = True will correspond to including vertex v in the independent set).
   - **Clauses:** For each edge $(u, v) \in E$, we must enforce that u and v cannot both be in the independent set. We add the clause:

     $$(\neg x_u \vee \neg x_v)$$

     Notice that this is a valid Horn clause (it has zero positive literals).

   Let $\phi_G = \bigwedge_{(u,v)\in E}(\neg x_u \vee \neg x_v)$.

   We claim that G has an independent set of size at least k if and only if ϕ_G has a satisfying assignment with at least k variables set to True.

- ($\Rightarrow$) Suppose G has an independent set S where $|S| \geq k$. Construct an assignment by setting $x_v = \text{True}$ if $v \in S$, and $x_v = \text{False}$ otherwise. Since no two vertices in S share an edge, no edge clause $(\neg x_u \vee \neg x_v)$ will have both literals falsified. Thus, ϕ_G is satisfied, and exactly $|S| \geq k$ variables are True.
- ($\Leftarrow$) Suppose ϕ_G has a satisfying assignment with at least k True variables. Let $S = \{v \in V \mid x_v = \text{True}\}$. Because the assignment satisfies all clauses $(\neg x_u \vee \neg x_v)$ for $(u, v) \in E$, it is impossible for both x_u and x_v to be True simultaneously. Therefore, no two vertices in S share an edge, meaning S is an independent set of size at least k.

8. **CNF Conversion and Resolution Complexity**

9. **Encoding Reachability with Bounded Length**

Solution:

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

(a) Base step

At step 0, only s is reachable:

$$(P_{s,0})$$

For all $v \neq s$:

$$(\neg P_{v,0})$$

(b) Forward propagation

For each edge $(u \to v) \in E$ and each $0 \leq i < k$:

$$P_{u,i} \to P_{v,i+1}$$

CNF form:

$$(\neg P_{u,i} \vee P_{v,i+1})$$

(c) No spontaneous reachability

If v is reached at step $i + 1$, some predecessor must have been reached at step i:

$$P_{v,i+1} \to \bigvee_{u \in In(v)} P_{u,i}$$

CNF form:

$$(\neg P_{v,i+1} \vee \bigvee_{u \in In(v)} P_{u,i})$$

(d) Target condition

Target must be reached at some step:

$$\bigvee_{i=0}^{k} P_{t,i}$$

10. **Removing edges to create a DAG**

Solution:

(a) Consistency constraint

Both $X(u, v)$ and $X(v, u)$ cannot be true because this would imply that u appears before v and v appears before u, which is impossible in a strict ordering.

CNF clause:

$$(\neg X(u, v) \vee \neg X(v, u))$$

(b) Transitivity

Logical implication:

$$(X(u, v) \wedge X(v, w)) \rightarrow X(u, w)$$

CNF form:

$$(\neg X(u, v) \vee \neg X(v, w) \vee X(u, w))$$

(c) Edge consistency

Logical implication:

$$Y(u, v) \rightarrow X(u, v)$$

CNF form:

$$(\neg Y(u, v) \vee X(u, v))$$

(d) Cycles impossible

Suppose the cycle is kept:

$$Y(u_1, u_2), Y(u_2, u_3), \ldots, Y(u_k, u_1)$$

Then constraints imply:

$$X(u_1, u_2), X(u_2, u_3), \ldots, X(u_k, u_1)$$

By transitivity:

$$X(u_1, u_1)$$

which is impossible.

Thus cycles cannot exist.

(e) Optimization objective

Each $Y(u, v) = false$ corresponds to removing an edge.

Thus maximizing the number of $Y(u, v)$ assigned true minimizes the number of removed edges.

11. **Learning with CDCL**

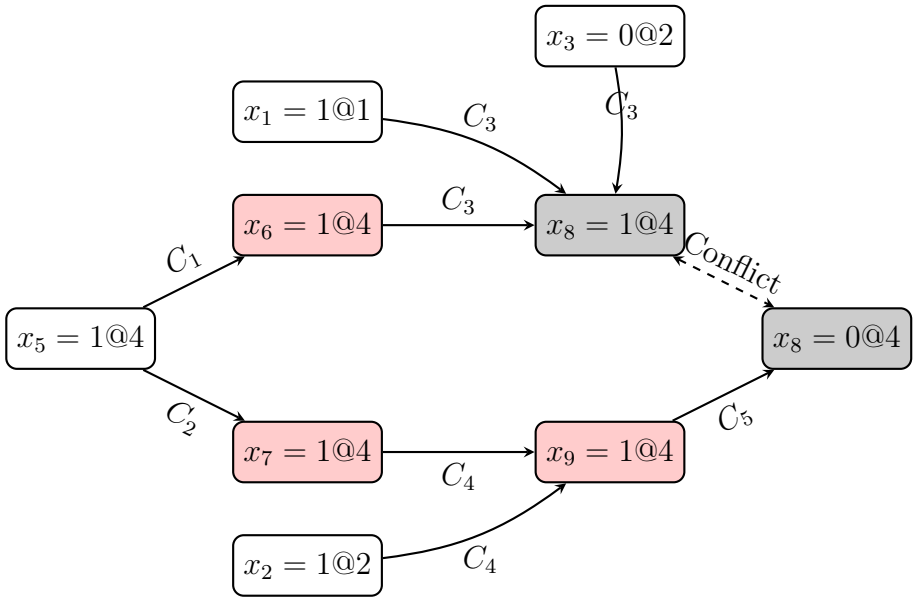
Solution:

Part (a): Unit Propagation and Implication Graph

Starting at Decision Level 4 with the decision $\boldsymbol{x_5 = 1}$, we evaluate the clauses for unit propagation:

1. **Clause** C_1 $(\neg x_5 \vee x_6)$: Since $x_5 = 1$, $\neg x_5$ is False. To satisfy C_1, we must propagate $x_6 = 1$. (Reason: C_1)
2. **Clause** C_2 $(\neg x_5 \vee x_7)$: Since $x_5 = 1$, $\neg x_5$ is False. To satisfy C_2, we must propagate $x_7 = 1$. (Reason: C_2)
3. **Clause** C_3 $(\neg x_1 \vee x_3 \vee \neg x_6 \vee x_8)$:
   - $x_1 = 1$ (Level 1) $\implies$ $\neg x_1$ is False.
   - $x_3 = 0$ (Level 2) $\implies$ x_3 is False.
   - $x_6 = 1$ (Level 4) $\implies$ $\neg x_6$ is False.
   - To satisfy C_3, we must propagate $x_8 = 1$. (Reason: C_3)
4. **Clause** C_4 $(\neg x_2 \vee \neg x_7 \vee x_9)$:
   - $x_2 = 1$ (Level 2) $\implies$ $\neg x_2$ is False.
   - $x_7 = 1$ (Level 4) $\implies$ $\neg x_7$ is False.
   - To satisfy C_4, we must propagate $x_9 = 1$. (Reason: C_4)
5. **Clause** C_5 $(\neg x_8 \vee \neg x_9)$:
   - $x_9 = 1$ (Level 4) $\implies$ $\neg x_9$ is False.
   - To satisfy C_5, we must propagate $x_8 = 0$. (Reason: C_5)
   - **Conflict!** x_8 is now unit-propagated to both 0 and 1.

Implication Graph:



Here $x_i = b@l$ means, x_i is set to boolean value b at decision level l.

Part (b): 1-UIP and Learned Clause

1. Identifying the 1-UIP:

The First Unique Implication Point (1-UIP) is the node at the current decision level (Level 4) closest to the conflict that dominates all paths from the decision variable to the conflict node. In other words, all paths from the node corresponding to the last decision level (i.e. 4) to the conflict nodes must pass through the 1-UIP. The only node at Level 4 that exists on *all* paths to the conflict ($x_5 \to x_6 \to x_8$ and $x_5 \to x_7 \to x_9 \to x_8$) is x_5. Therefore, x_5 **is the 1-UIP**.

2. Deriving the Learned Clause (via Resolution):

We need to find a cut that separates all the decision nodes (white) from the conflict nodes (grey), and has as few nodes whose outgoing edges cross the cut. To get a cut corresponding to the 1-UIP, i.e, $x_5 = 1@4$, we see that the only possible cut is the one that intercepts the edges going out of all the white nodes. This gives the conflict clause $(x_3 \vee \neg x_1 \vee \neg x_5 \vee \neg x_2)$.

There's another way of arriving at the smallest conflict clause corresponding to a 1-UIP. We work backwards from the conflict clause C_5 (this is the clause that finally had both its literals becoming false), resolving it with the antecedent clauses until we only have the 1-UIP from the current decision level.

$$\begin{aligned}
\text{Start:} \quad & (\neg x_8 \vee \neg x_9) \quad (\text{Clause } C_5) \\
\text{Resolve } x_8 \text{ w/ } C_3: \quad & (\neg x_8 \vee \neg x_9) \otimes_{x_8} (\neg x_1 \vee x_3 \vee \neg x_6 \vee x_8) \\
= \quad & (\neg x_9 \vee \neg x_1 \vee x_3 \vee \neg x_6) \\
\text{Resolve } x_9 \text{ w/ } C_4: \quad & (\neg x_9 \vee \neg x_1 \vee x_3 \vee \neg x_6) \otimes_{x_9} (\neg x_2 \vee \neg x_7 \vee x_9) \\
= \quad & (\neg x_1 \vee x_3 \vee \neg x_6 \vee \neg x_2 \vee \neg x_7) \\
\text{Resolve } x_6 \text{ w/ } C_1: \quad & (\cdots \vee \neg x_6) \otimes_{x_6} (\neg x_5 \vee x_6) \\
= \quad & (\neg x_1 \vee x_3 \vee \neg x_2 \vee \neg x_7 \vee \neg x_5) \\
\text{Resolve } x_7 \text{ w/ } C_2: \quad & (\cdots \vee \neg x_7) \otimes_{x_7} (\neg x_5 \vee x_7) \\
= \quad & (\neg x_1 \vee x_3 \vee \neg x_2 \vee \neg x_5 \vee \neg x_5)
\end{aligned}$$

Simplifying by merging duplicate literals, the final learned clause is: $C_{\textbf{learned}} = (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5)$

Part (c): Backtracking and Solver State

1. Determine the Backtracking (Assertion) Level:

We examine the decision levels of the variables in the newly learned clause: $x_1@1$, $x_2@2$, $x_3@2$, and $x_5@4$. The assertion level is the *second highest* decision level present in the learned clause. The solver will non-chronologically backtrack to **Decision Level 2**.

Food for thought: Why is the backtracking level the second highest decision level *in the conflict clause, and not the highest? Think about it: what is the highest decision level in the conflict clause if we are using 1-UIP to obtain the conflict clause? Would this be any different from chronological backtracking?*

2. Solver State Immediately After Backtracking:

- All assignments from Level 3 (x_4) and Level 4 (x_5, x_6, x_7, x_8, x_9) are erased.

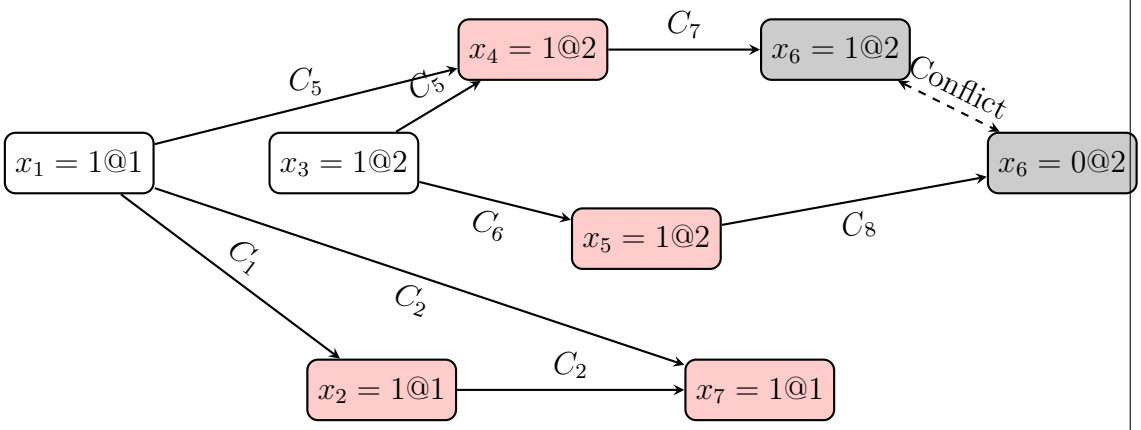
- The solver is back at Level 2, with assignments $x_1 = 1$, $x_2 = 1$, and $x_3 = 0$.
- The learned clause is added to the database: $(\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_5)$.
- Under the Level 2 assignments, this clause evaluates to (False$\vee$False$\vee$False$\vee\neg x_5$).
- **Result:** This becomes a unit clause. The variable x_5 **is immediately propagated and assigned a value of 0** (False) at Decision Level 2.

12. **Conflict-Clause Driven Learning**

Solution:

(i) Suppose the $DL1$ is $x_1 = 1$. C_3, C_4 are satisfied. $x_2 = 1$ at C_1 is forced by unit propagation. $x_7 = 1$ is forced by unit propagation (clause C_2) once we have $x_1 = x_2 = 1$

(ii) $DL2$ is $x_3 = 1$. This further triggers the unit propagations as noted in the below implication graph:



The First Unique Implication Point (1-UIP) is the node at the current decision level (Level 2) closest to the conflict that dominates all paths from the decision variable to the conflict node. In other words, all paths from the node corresponding to the last decision level (i.e. 2) to the conflict nodes must pass through the 1-UIP. The only node at Level 2 that exists on *all* paths to the conflict ($x_3 \to x_4 \to x_6$ and $x_3 \to x_5 \to x_6$) is x_3. Therefore, x_3 **is the 1-UIP**.

We need to find a cut that separates all the decision nodes (white) from the conflict nodes (grey), and has as few nodes whose outgoing edges cross the cut.

To get a cut corresponding to the 1-UIP, i.e, $x_3 = 1@2$, we see that the only possible cut is the one that intercepts the edges going out of all the white nodes. This gives the learnt clause $(\neg x_1 \vee \neg x_3)$.

If $x_6 = 1$ is propagated (due to clause C_7) before $x_6 = 0$ (due to clause C_8), then clause C_8 becomes conflict clause (all literals set to false). Otherwise, clause C_7 is the conflict clause.

Food for thought: Notice the difference between a conflict clause and a learnt clause. A conflict clause is one that was originally present in the formula, and that has all its literals set to false after the assignments (decisions + unit propagation) that lead to the conflict. A learnt clause is invariably not already present in the

original formula, but is implied by the original formula. A learnt clause is also one that becomes false as a consequence of the assignment that leads to the conflict, but it wasn't originally present in the formula – it is obtained after finding a cut in the implication graph leading to the conflict.

(iii) We examine the decision levels of the variables in the newly learned clause: x_1@1, x_3@2. The assertion level is the *second highest* decision level present in the learned clause. The solver will non-chronologically backtrack to **Decision Level 1**.

All assignments from level 2 are erased.

Backtracking at level 1 forces the choice $x_1 = 0$. This, in turn, satisfies C_1, C_2. Further $x_2 = 1$ is forced by unit propagation due to C_3 and $x_2 = 0$ is forced by unit propagation due to C_4, which straight away leads to conflict.

Turns out the formula is unsatisfiable.

13. **DPLL run**

Solution: In part (a), no unit clauses or pure literals are obtained until all of x_0 through x_{n-1} are assigned the value 1, one at a time. Once x_{n-1} is assigned 1, UP leads to a conflict—x_n must be assigned both 1 and 0 by UP. This causes a backtrack, which ends up setting x_{n-1} to 0. Once this happens, UP assigns the value 0 to x_n, resulting in the partial assignment $x_{n-1} = x_n = 0$, which satisfies all clauses. Therefore, there is exactly one backtrack.

In part (b), every time a variable x_i for $0 \leq i \leq n-1$ is assigned 1, UP causes x_{n+i} to be in conflict (must be assigned both 0 and 1). This induces a backtrack that sets x_i to 0, followed by UP setting x_{n+i} to 0. The DPLL algorithm will then choose x_{i+1} as the next decision variable, assign it 1, and the above process repeats. So, in this case, DPLL will incur n backtracks, one for each of $x_0, \ldots, x_{n-1}$.

14. **Renamable Horn**

Solution:

Let F be a CNF formula over variables $x_1, \ldots, x_n$. We show how to reduce the problem of deciding whether F is *renamable Horn* to solving a 2–SAT instance.

Step 1: Renaming variables. For every variable x_i introduce a new Boolean variable r_i. The meaning of r_i is:

$$r_i = \begin{cases} 0 & \text{do not negate } x_i, \\ 1 & \text{negate all occurrences of } x_i. \end{cases}$$

Thus, an assignment to the r_i's uniquely determines a renaming of the variables of F.

Step 2: Characterising Horn clauses. A clause is Horn if it contains at most one positive literal. After renaming, a clause becomes non-Horn exactly when it contains *two literals that both evaluate to positive form.* Therefore, for each clause it suffices to ensure that no pair of its literals is simultaneously positive after renaming.

Step 3: Pairwise constraints. Consider a clause C of F and two distinct literals ℓ_i and ℓ_j in C, involving variables x_i and x_j. We add one 2–CNF clause that forbids both ℓ_i and ℓ_j from becoming positive at the same time.

There are four possibilities:

- $\ell_i = x_i$, $\ell_j = x_j$: forbid $(r_i = 0 \wedge r_j = 0)$ by adding $(r_i \vee r_j)$.
- $\ell_i = x_i$, $\ell_j = \neg x_j$: forbid $(r_i = 0 \wedge r_j = 1)$ by adding $(r_i \vee \neg r_j)$.
- $\ell_i = \neg x_i$, $\ell_j = x_j$: forbid $(r_i = 1 \wedge r_j = 0)$ by adding $(\neg r_i \vee r_j)$.
- $\ell_i = \neg x_i$, $\ell_j = \neg x_j$: forbid $(r_i = 1 \wedge r_j = 1)$ by adding $(\neg r_i \vee \neg r_j)$.

Perform this for every unordered pair of literals in every clause of F. Let G be the conjunction of all these 2–clauses. Clearly G is a 2–CNF formula whose size is polynomial in $|F|$.

Step 4: Correctness. *($\Rightarrow$)* If F is renamable Horn, there exists a set of variables to negate that makes all clauses Horn. Let $r_i = 1$ for each negated variable and 0 otherwise. Since no clause has two positive literals after this renaming, none of the forbidden pairs occurs, so all clauses in G are satisfied. Hence G is satisfiable.

($\Leftarrow$) If G is satisfiable, take a satisfying assignment $(r_1, \ldots, r_n)$ and negate exactly those variables x_i with $r_i = 1$. Because all pairwise "both positive" situations were forbidden, no clause can end up with two positive literals. Thus every clause has at most one positive literal, and the renamed formula is Horn. Hence F is renamable Horn.

Recovering the renaming. A satisfying assignment of G directly specifies the renaming: negate x_i iff $r_i = 1$.