Notes: 04-20 P Complete

2026-04-19

Review

- A problem is X-hard if we can reduce all problems in X to it.
- A problem is X-complete if we can reduce problems in X both from and to it.
- We have 3SAT as our standard NP-Complete problem.
- We didn't quite finish the proofs, but Regex Equivalence is PSPACE-complete.
- And there are P-complete problems:

What about the hardest problems in P? (P-Completeness)

If NP-Complete problems are the "hardest problems in NP" (likely impossible to solve fast), **P-Complete** problems are the "hardest problems in P" (likely **impossible to parallelize**). They represent inherently sequential tasks where you *must* wait for step i to finish before starting step $i + 1$.

Technical Note: To prove P-Completeness, we can't use polynomial-time reductions. If we did, *every* problem in P would reduce to *every* other problem in P (because the reduction itself could just solve the problem!). Instead, we use **Log-Space Reductions**. The translation algorithm is only allowed a logarithmic amount of memory (just enough for a few loop counters/pointers), ensuring the reduction can't accidentally solve the problem.

1. The Canonical P-Complete Problem: Circuit Value Problem (CVP)

The Problem: You are given a boolean circuit made of AND and OR gates (Monotone CVP). The bottom layer has fixed boolean inputs (True/False). *Does the final top-level gate output True?*

Why it's in P: It's trivial to solve. You just do a topological sort and evaluate the gates from bottom to top. It takes $O(n)$ time. **Why it's hard to parallelize:** Look at a deep, narrow circuit. Gate 100 relies on Gate 99, which relies on Gate 98. Even with 1,000 CPU cores, you can't speed this up. It is the ultimate bottleneck.

2. A Thematic Problem: Horn-SAT

Since we are talking about SAT, let's look at a restricted version. **The Problem:** Exactly like Boolean Satisfiability, but **every clause can have *at most one* positive literal.**

- Valid: $(\neg x_1 \vee \neg x_2 \vee x_3)$
- Valid: $(\neg x_4 \vee \neg x_5)$
- Valid: (x_6)
- INVALID: $(x_1 \vee \neg x_2 \vee x_3)$ *(Two positive literals)*

Why it's in P: A Horn clause is just an IF-THEN implication in disguise. By De Morgan's Laws, $(\neg x_1 \vee \neg x_2 \vee x_3)$ is exactly the same as: $(x_1 \wedge x_2) \rightarrow x_3$

We can solve this in linear time using **Unit Propagation**:

1. Find any clause with just a single positive literal (e.g., $x_6 = True$).
2. Plug that into your other clauses (knocking over the dominos).
3. If you ever prove a contradiction, it's Unsatisfiable. Otherwise, Satisfiable.

3. Hardware $\leftrightarrow$ Software (Proving Horn-SAT is P-Complete)

We can prove Horn-SAT is P-Complete by reducing CVP to it, and vice versa!

CVP $\rightarrow$ Horn-SAT (Logic can simulate Hardware) We create a variable for every gate. We want the Horn formula to be *Unsatisfiable* if the circuit outputs True.

- **For an AND gate g with inputs y, z:** We need y AND $z \rightarrow g$. We write the Horn clause: $(\neg x_y \vee \neg x_z \vee x_g)$.
- **For an OR gate g with inputs y, z:** We need $y \rightarrow g$ and $z \rightarrow g$. We write two Horn clauses: $(\neg x_y \vee x_g)$ and $(\neg x_z \vee x_g)$.
- **The Output Trick:** Add one final clause demanding the output wire is False: $(\neg x_{out})$.

If the circuit naturally evaluates to True, our forward implications force x_{out} to True, violating the final clause and creating a contradiction! (Log-space because we just loop over gates and print clauses).

Horn-SAT $\rightarrow$ CVP (Hardware can unroll Algorithms) We can't write a "loop" in a circuit, but we can unroll the Unit Propagation algorithm into physical layers.

- Build n layers of the circuit.
- Wire up AND/OR gates to represent the implications (e.g., if $(x_1 \wedge x_2) \rightarrow x_3$, wire x_1 and x_2 from layer t into an AND gate, and feed it into x_3 at layer $t+1$).
- At the final layer n, check for any contradictions (e.g., a purely negative clause like $\neg x_4 \vee \neg x_7$ failing). Wire all contradiction checks into a massive OR gate.

4. Bridge to NP: Reducing Horn-SAT $\rightarrow$ 3SAT

Can we reduce a problem in P to an NP-Complete problem? **Yes.** Because $P \subseteq NP$, we can always use a hard tool to solve an easy problem.

We just use the standard SAT $\rightarrow$ 3SAT reduction:

- Pad short clauses: $(x_1) \rightarrow (x_1 \vee x_1 \vee x_1)$
- Split long clauses with dummy variables: $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \rightarrow (\neg x_1 \vee \neg x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee x_4)$

Notice that the split clauses **are still valid Horn clauses!** We just proved that Horn-3SAT is also P-Complete. This requires only log-space to track the dummy variable counter.

---

5. Class Exercise: Reduce 3SAT $\rightarrow$ Horn-SAT

For the rest of the lecture, we are going to do the reduction in reverse. We are going to reduce an NP-Complete problem to a P problem. If we do this, we prove $\mathbf{P = NP}$, revolutionize computer science, break all modern encryption, and win a million dollars.

The Goal: Convert this standard 3SAT clause into a set of valid Horn clauses:

$$(x_1 \vee x_2 \vee x_3)$$

Instructor notes / Trap warnings for student ideas:

- **Try De Morgan's Law?** $\neg(\neg x_1 \wedge \neg x_2) \rightarrow x_3$. A Horn clause requires a *conjunction of positive variables* on the left. The left side here is an OR statement in disguise.
- **Try Dummy Variables?** $(x_1 \vee y) \wedge (x_2 \vee \neg y \vee x_3)$. You got rid of a positive literal in the second clause, but look at the first clause: $(x_1 \vee y)$. It still has two positive literals. It's not a Horn clause.

The Conceptual Takeaway: Horn clauses represent **Determinism**. "If A and B, then C." There are no choices. It's a straight line of falling dominos. 3SAT represents **Nondeterminism (Branching)**. "Either A is true, or B is true, or C is true." You have to *guess* which path to pursue.

Trying to reduce 3SAT to Horn-SAT is the act of trying to force a universe of branching, exponential choices into a single, deterministic straight line.