3/30 - SAT solving

Welcome the wonderful wacky world of SAT solving

- Suppose we have a boolean formula *f* with variables x1, x2, x3, …
- Is *f* true?
  - It depends on the values we give to x1, x2, x3, …
- Instead we can ask two different questions:
  - Is it true for all variable assignments?
  - Is it true for some variable assignment? (i.e is it **satisfiable**?)
- We want to answer the second question.
- Naive method: just try out all assignments with a big truth table
  - Very slow: there are exponential number of possible assignments
  - In fact, we don't have a solver that's sub-exponential in the worst case
  - But, modern solvers do much better in the average case

Some vocabulary:

- *Literal* - a variable or its negation
- *Clause* - a disjunction (big or) of literals
  - Notice: the empty clause is false
- *Conjunctive Normal Form (CNF)* - a conjunction (big and) of clauses
  - Notice: the empty CNF is true

A new method: going through a **search tree (DPLL algorithm)**

- Use some heuristic to pick what variable and value to split on
- Search and branch on some variable and assign it a value
  - If that doesn't work, backtrack and try the other value
- We want to prune some subtrees so that we don't even have to search them
  - One method: if we have unit clauses (**unit propagation/unit clause elim**)
    - Just assign vars in unit clauses to match their sign
    - Need to satisfy the unit clause - only way to do that is by satisfying that literal
  - Another method: if a variable always has the same sign (**pure literal elimination**)
    - Assign the var to be consistent with its sign
    - This can only help satisfy things
- If we ever produce the empty clause: **Unsat**
- If we ever produce an empty CNF: **SAT**

We can imagine during our search:

- Delete literals that can't help us satisfy the clause anyway
- Delete clauses that are already satisfied (we're done with them)

- If we derive the empty clause, false
- If we derive a consistent set of literals, SAT
- Otherwise, ask the heuristic how to branch out, and explore