EECS 219C: Computer-Aided Verification

Boolean Satisfiability Solving Part I: Basics

Sanjit A. Seshia

EECS, UC Berkeley

Boolean Functions (Formulas) and Propositional Logic

- Variables: $x_1, x_2, x_3, \ldots, x_n \in \{0, 1\}$ (or {true, false})
- $F(x_1, x_2, x_3, \ldots, x_n) \in \{0,1\}$
- F representable as the output (root) of a circuit (expression DAG) constructed with gates (Boolean operators)
  - Standard Boolean operators: And ($\wedge$, $\cdot$), Or ($\vee$, +), Not ($\neg$, ')
  - Derived operators: Implies ($\rightarrow$) Iff ($\Leftrightarrow$)

The Boolean Satisfiability Problem (SAT)

- Given:

  A Boolean formula $F(x_1, x_2, x_3, \ldots, x_n)$

- Check if F can ever be true (satisfiable)
  - If so, return values to x_i's (satisfying assignment) that make F true

Why is SAT important?

- Theoretical importance:
  - First NP-complete problem (Cook, 1971)
- Many practical applications:
  - Model Checking
  - Automatic Test Pattern Generation
  - Combinational Equivalence Checking
  - Planning in AI
  - Automated Theorem Proving
  - Software Verification
  - …

Terminology

- Literal
- Clause
- Conjunctive Normal Form (CNF)
- Disjunctive Normal Form (DNF)
- Tautology
  - Complexity of tautology checking for propositional logic?

S. A. Seshia 5

An Example

- Inputs to SAT solvers are usually represented in CNF

(a + b + c) (a’ + b’ + c) (a + b’ + c’) (a’ + b + c’)

S. A. Seshia 6

An Example

- Inputs to SAT solvers are usually represented in CNF

(a + b + c) (a’ + b’ + c) (a + b’ + c’) (a’ + b + c’)

Why CNF?

Why CNF?

- Input-related reason
  - Can transform from circuit to CNF in linear time & space (HOW?)
- Solver-related: Most SAT solver variants can exploit CNF
  - Easy to detect a conflict
  - Easy to remember partial assignments that don't work (just add 'conflict' clauses)
  - Other "ease of representation" points?
- Any reasons why CNF might NOT be a good choice?

S. A. Seshia 9

Complexity Issues

- **k-SAT**: A SAT problem with input in CNF with at most k literals in each clause
- Complexity for non-trivial values of k:
  - 2-SAT: ?
  - 3-SAT: ?
  - > 3-SAT: ?

S. A. Seshia 10

2-SAT Algorithm

- Linear-time algorithm (Aspvall, Plass, Tarjan, 1979)
  - Think of clauses as implications
  - Think of a graph with literals as nodes

S. A. Seshia 11

3-SAT: Complexity Bounds (circa 2005)

- Obvious upper bound on run-time?
- Best known deterministic upper bound

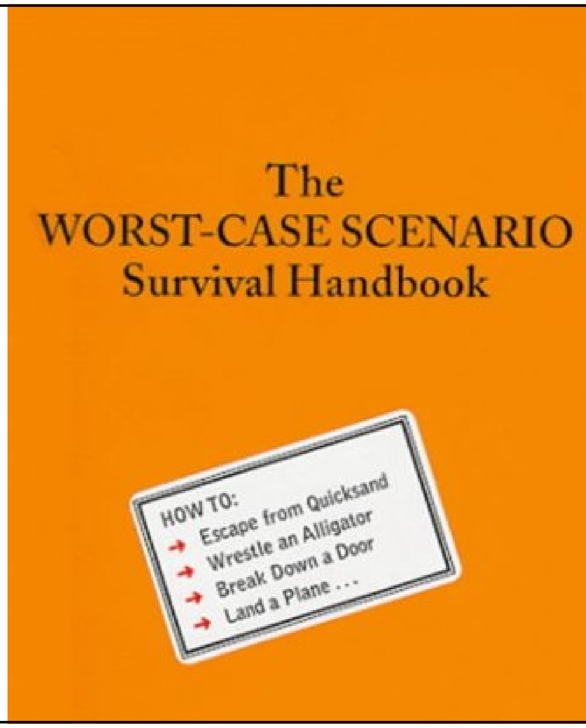
  1.473^n
- Best known randomized upper bound

  1.324^n
- Best known lower bound

  $n^{2.761}$

S. A. Seshia 12

Worst-Case Complexity



Beyond Worst-Case Complexity

- What we really care about is "typical-case" complexity
- But how can one measure "typical-case"?
- Two approaches:
  - Is your problem a restricted form of 3-SAT? That might be polynomial-time solvable
  - Experiment with (random) SAT instances and see how the solver run-time varies with formula parameters (#vars, #clauses, …)

Special Cases of 3-SAT

- You already know one: 2-SAT
  - T. Larrabee observed that many clauses in ATPG tend to be 2-CNF
- Another useful class: Horn-SAT
  - A clause is a Horn clause if at most one literal is positive
  - If all clauses are Horn, then problem is Horn-SAT
  - E.g. Application:- Simulation checking between 2 finite-state systems

Horn-SAT

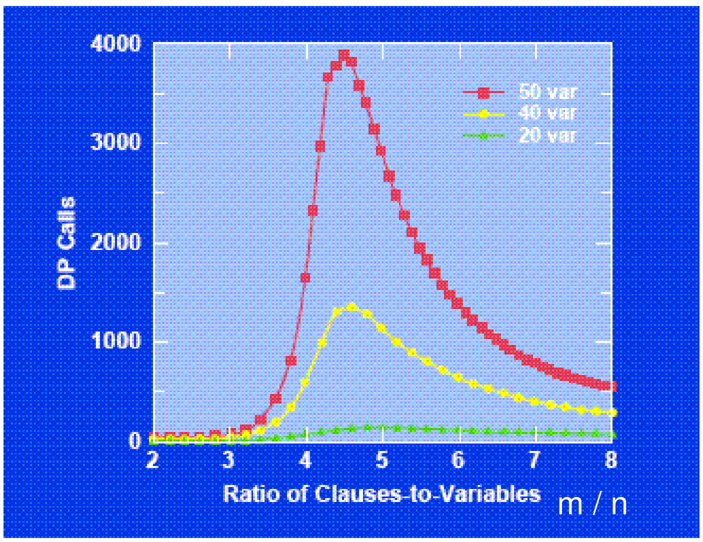
- Can we solve Horn-SAT in polynomial time? How?
  - Hint: view clauses as implications.

- Variants:
  - Negated Horn-SAT: Clauses with at most one literal negative
  - Renamable Horn-SAT: Doesn’t look like a Horn-SAT problem, but turns into one when polarities of some variables are flipped

Phase Transitions in k-SAT

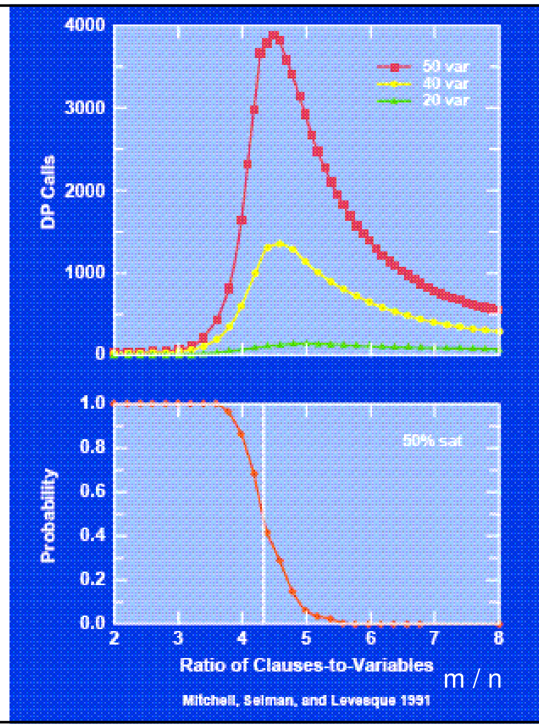
- Consider a fixed-length clause model
  - k-SAT means that each clause contains exactly k literals
- Let SAT problem comprise **m** clauses and **n** variables
  - Randomly generate the problem for fixed k and varying m and n
- Question: How does the problem hardness vary with m/n ?

3-SAT Hardness



As n increases hardness transition grows sharper

Transition at m/n $\simeq$ 4.3



Threshold Conjecture

- For every k, there exists a c* such that
  - For m/n < c*, as n → ∞ , problem is satisfiable with probability 1
  - For m/n > c*, as n → ∞ , problem is unsatisfiable with probability 1
- Conjecture proved true for k=2 and c*=1
- For k=3, current status is that c* is in the range 3.42 – 4.51

The (2+p)-SAT Model

- We know:
  - 2-SAT is in P
  - 3-SAT is in NP
- Problems are (typically) a mix of binary and ternary clauses
  - Let p ∈ {0,1}
  - Let problem comprise (1-p) fraction of binary clauses and p of ternary
  - So-called (2+p)-SAT problem

Experimentation with random (2+p)-SAT

- When p < ~0.41
  - Problem behaves like 2-SAT
  - Linear scaling
- When p > ~0.42
  - Problem behaves like 3-SAT
  - Exponential scaling

- Nice observations, but don't help us predict behavior of problems in practice

Backbones and Backdoors

- Backbone [Parkes; Monasson et al.]
  - Subset of literals that must be true in every satisfying assignment (if one exists)
  - Empirically related to hardness of problems
- Backdoor [Williams, Gomes, Selman]
  - Subset of variables such that once you've given those a suitable assignment (if one exists), the rest of the problem is poly-time solvable
  - Also empirically related to hardness
- But no easy way to find such backbones / backdoors! ☹

S. A. Seshia 23

A Classification of SAT Algorithms

- Davis-Putnam (DP)
  - Based on **resolution**
- Davis-Logemann-Loveland (DLL/DPLL)
  - Search-based
  - Basis for current most successful solvers
- Stalmarck's algorithm
  - "Different" kind of search, proprietary algorithm
- Stochastic search
  - Local search, hill climbing, etc.
  - Unable to prove unsatisfiability (incomplete)

S. A. Seshia 24

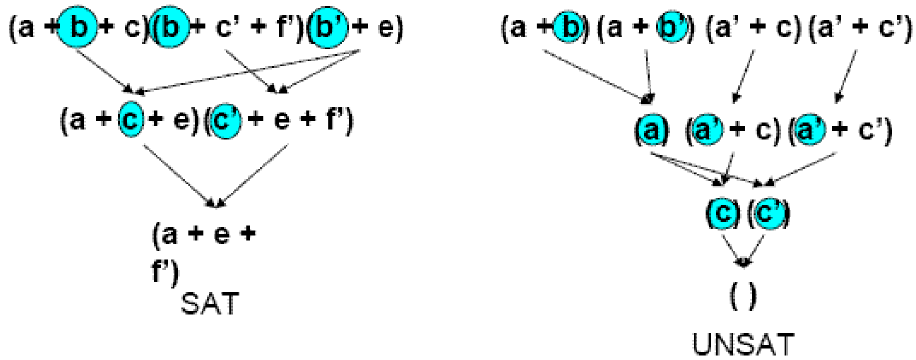
Resolution

- Two CNF clauses that contain a variable x in opposite phases (polarities) imply a new CNF clause that contains all literals except x and x'
- (a + b) (a' + c) = (a + b)(a' + c)(b + c)
- Why is this true?

The Davis-Putnam Algorithm

- Iteratively select a variable x to perform resolution on
- Retain only the newly added clauses and the ones not containing x
- Termination: You either
  - Derive the empty clause (conclude UNSAT)
  - Or all variables have been selected

Resolution Example



How many clauses can you end up with?
(at any iteration)

Next Class

- How DLL algorithm works in current SAT solvers