Satisfiability

Verifying cyberphysical systems

Sayan Mitra

mitras@illinois.edu

Some of the slides for this lecture are adapted from slides by Clark Barrett

Readings

- Chapter 7
- Appendix C

Outline

- Propositional Satisfiability problem
- Normal forms
- DPLL algorithm

Boolean *satisfiability* problem

Given a *well-formed formula* in propositional logic, determine whether there exists a satisfying solution

Example: $\alpha(x_1, x_2, \dots, x_n) \equiv (x_1 \wedge x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3 \vee x_2)$

Set of variables: $X = \{x_1, x_2, \dots, x_n\}$,

Each variable is Boolean: $type(x_i) = \{0,1\}$

Formula α is *well-formed* if it uses propositional operators, and ∧, or ∨, not ¬, iff ↔ etc., properly

Recall, a valuation **x** of X maps each x_i to a value 0 or 1

A valuation **x** of X *satisfies* α is each each x_i in α replaced by the corresponding value in **x** evaluates to *true*. We write this as $\boldsymbol{x} \vDash \alpha$

Otherwise, we write $\boldsymbol{x} \nvDash \alpha$

Example: with $\boldsymbol{x} \equiv \langle x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0 \rangle$; $\boldsymbol{x} \vDash \alpha$

Boolean *satisfiability* problem (SAT)

Given a well-formed formula in propositional logic, determine whether there exists a satisfying solution

Restatement: $\exists \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha$?

If the answer is "No" then α is said to be *unsatisfiable*

Aside. If $\forall \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha$ then α is said to be *valid* or *a tautology*

If α is valid then $\neg\alpha$ is unsatisfiable

α and α' are *tautologically equivalent* if they have the same truth tables

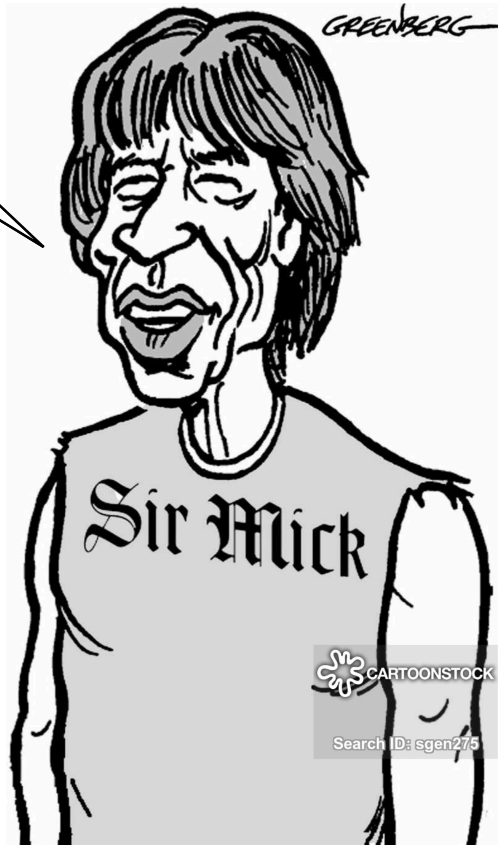
$$\forall \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha \leftrightarrow \boldsymbol{x} \vDash \alpha'$$

What is a naïve method for solving SAT?

What is the complexity of this approach? How many evaluations of $\alpha(x_1, x_2, \ldots, x_n)$?

I Don’t Get No ***Satisfaction***, but I try, try, try,…

I can prove why.



Stephen A. Cook:
The Complexity of Theorem-Proving Procedures. STOC 1971: 151-158

Slide by Sayan Mitra using pictures from Wikipedia and cartoonstock.com

SAT is NP-complete

SAT was the first problem shown to be NP-complete [Cook 71]

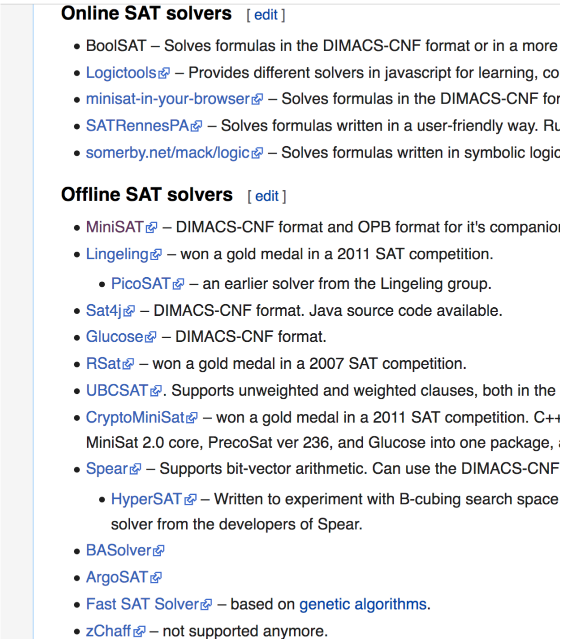
2-SAT can be solved in polynomial time (Exercise)

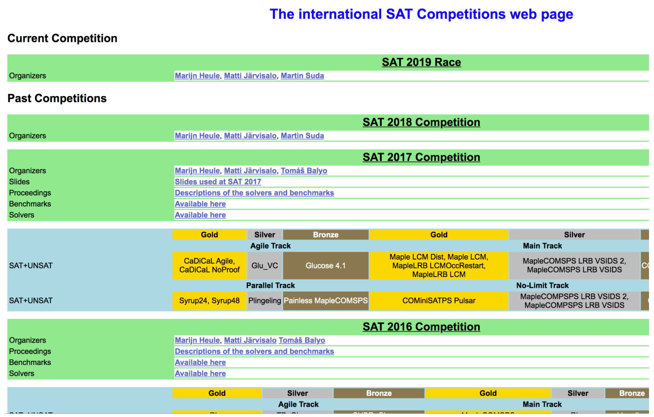
(Read definition of NP: Nondeterministic Polytime in Appendix C)

This has real implications

1. Essentially we don't know better than the naïve algorithm
2. A solver for SAT can be used to solve any other problem in the NP class with only polytime slowdown. i.e., makes a lot of sense to build SAT solvers
3. SAT/SMT solving is the cornerstone of *many* verification procedures

Stephen Cook, The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on theory of computing. STOC '71.





Details

We will assume α to be in *conjunctive normal form (CNF)*

literals: variable or its negation, e.g., x_3, $\neg x_3$

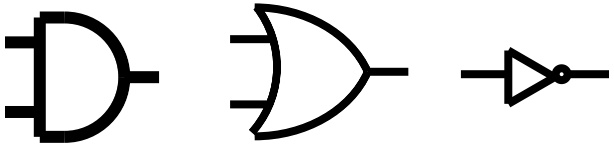
clause: disjunction (or) of literals, e.g., $(x_1 \vee x_2 \vee \neg x_3)$

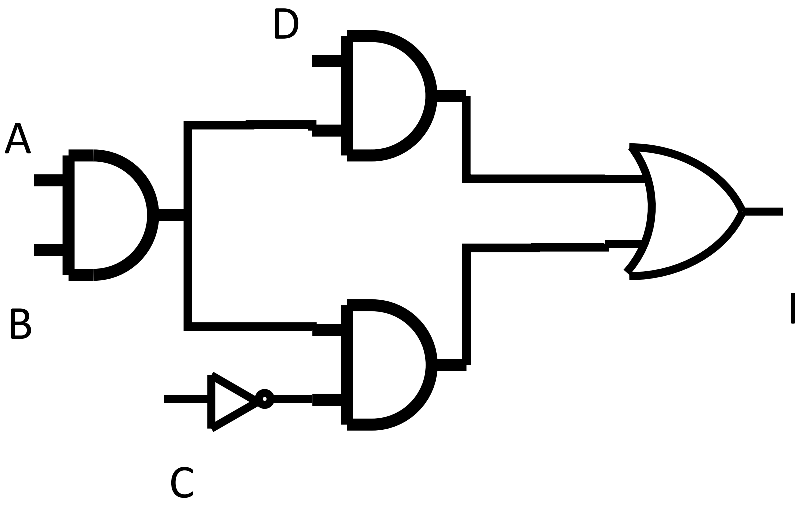
CNF formula: conjunction (and) of clauses,

e.g., $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_1)$

A variable may appear *positively* or *negatively* in a clause

Logic and circuits





$$I \equiv \big(D \land (A \land B)\big) \lor \big(\neg C \land (A \land B)\big)$$

Repeated subexpression is inefficient

Solution: rename $(A \land B) \leftrightarrow E$

$I' \equiv (D \land E) \lor (\neg C \land E) \land \big((A \land B) \leftrightarrow E\big)$

I and I' are **not** *tautologically equivalent*

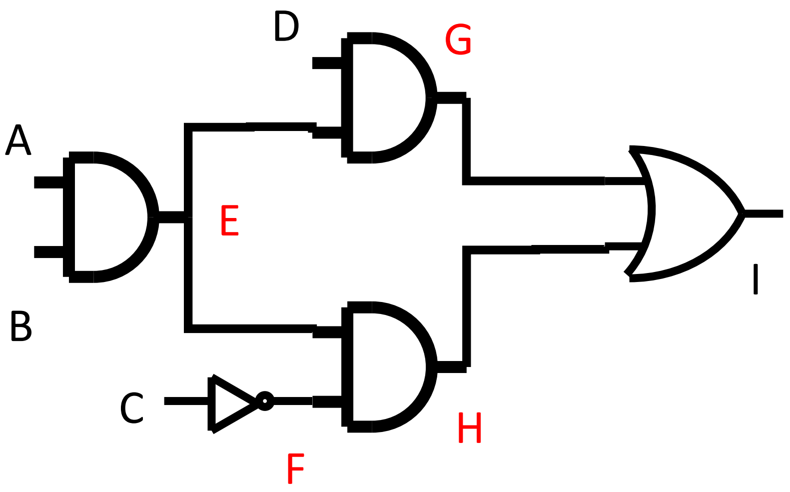
$C = 0, A = B = 1, E = 0$ satisfies I

But they are *equisatisfiable,* i.e., I is satisfiable iff I' is also satisfiable

Converting to CNF

- View the formula as a graph
- Give new names (variables) to non-leafs
- Relate the inputs and the outputs of the nonleafs and add this as a new clause
- Take conjunction of all of this

Converting to CNF



- $F \leftrightarrow \neg C$
  - $F \rightarrow \neg C \wedge \neg C \rightarrow F$
  - $(\neg F \vee C) \wedge (\neg C \vee F)$
- $(A \wedge B) \leftrightarrow E$
  - $\big((A \wedge B) \rightarrow E\big) \wedge \big(E \rightarrow (A \wedge B)\big)$
  - $(\neg(A \wedge B) \vee E) \wedge \big(\neg E \vee (A \wedge B)\big)$
  - $(\neg A \vee \neg B \vee E) \wedge \big(\neg E \vee A) \wedge (\neg E \vee B)\big)$
- $(G \vee H) \leftrightarrow I$
  - $\big((G \vee H) \rightarrow I\big) \wedge \big(I \rightarrow (G \vee H)\big)$
  - $(\neg G \wedge \neg H \vee I) \wedge (\neg I \vee G \vee H)$
  - $(\neg G \vee I) \wedge (\neg H \vee I) \wedge (\neg I \vee G \vee H)$
- $(D \wedge E) \leftrightarrow G$
  - $(\neg D \vee \neg E \vee G) \wedge \big(\neg G \vee D) \wedge (\neg G \vee E)\big)$
- $(F \wedge E) \leftrightarrow H$
  - $(\neg F \vee \neg E \vee H) \wedge \big(\neg H \vee F) \wedge (\neg H \vee E)\big)$

Standard representations of CNF

- $(\neg A \vee \neg B \vee E) \wedge \big(\neg\, E \vee A) \wedge (\neg E \vee B)\big)$
- $(A' + B' + E)(E' {+} A)(E' + B)$
- $(-1\ \ -2\ \ 5)(-5\ 1)(-5\ \ \ 2)$ DIMACS

- SMTLib

Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Transform the given formula α by applying a sequence of satisfiability preserving rules

If final result has an empty clause then *unsatisfiable*

if final result has no clauses then the formula is *satisfiable*

Davis Putnam Algorithm (DP) 1960

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3. **Resolution**

DP 1960

Rule 1. **Unit propagation**

A clause has a single literal

$$\alpha \equiv \cdots \wedge \cdots \wedge p \wedge \cdots \wedge \cdots$$

What choice do we really have?

$$\alpha \equiv \cdots \wedge (x_1 \vee \neg p \vee x_2) \wedge p \wedge \cdots \wedge (\neg x_3 \vee \neg p \vee x_1) \ldots$$

DP 1960

Rule 1. **Unit propagation**

A clause has a single literal

$$\alpha \equiv \cdots \wedge \cdots \wedge p \wedge \cdots \wedge \cdots$$

What choice do we really have?

$$\alpha' \equiv \cdots \wedge (x_1 \vee x_2) \wedge \cdots \wedge (\neg x_3 \vee x_1) \ldots$$

α and α' are equisatisfiable

Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in α

$$\alpha \equiv \cdots \land (x_1 \lor \neg p \lor x_2) \land (x_4 \lor \neg p) \land \cdots \land (\neg x_3 \lor \neg p \lor x_1) \ldots$$

p does not appear anywhere

Makes sense to set $p = 0$ and remove all occurrences of $\neg p$

Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in α

$$\alpha \equiv \cdots \wedge (x_1 \vee \neg p \vee x_2) \wedge (x_4 \vee \neg p) \wedge \cdots \wedge (\neg x_3 \vee x_1) \ldots$$

p does not appear anywhere

Makes sense to set $p = 0$ and remove all clauses in which $\neg p$ occurs

α and α' are equisatisfiable

$$\alpha' \equiv \cdots \wedge \cdots \wedge \cdots \wedge (\neg x_3 \vee x_1) \ldots [p = 0]$$

Davis Putnam Algorithm (DP) 1960

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3. **Resolution**

Choose a literal p that appears with both polarity in α. Suppose $(\ell_1 \vee \ell_2 \vee \cdots \vee p)$ be a clause in which p appears positively, and $(k_1 \vee k_2 \vee \cdots \vee \neg p)$ be a clause in which p appears negatively

Then the resolved clause is $(\ell_1 \vee \ell_2 \vee \cdots \vee k_1 \vee k_2 \vee \cdots k_m)$

Pairwise, resolve each clause in which p appears positively with a clause in which p appears negatively, and take the conjunction of all the results

Why is the result equisatisfiable?

What is the size of the resulting formula?

DPLL modifies resolution in DP with recursive DFS rule

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3'. Let Δ be the current set of clauses. Choose a literal p in Δ.

Check satisfiability of $\Delta \cup \{ p \}$ (guessing $p = 1$)

If satisfiable then return True else

return result of checking satisfiability of $\Delta \cup \{ \neg p \}$

This is essentially a depth first search

A simple greedy algorithm for SAT (GSAT)

Input: Set of clauses *C* over *X*, parameters *max-flips, max-tires*

Output: A satisfying assignment for C, or ∅ if none found

for i = 1 to *max-tries*

v *:=* random truth assignment in *val(X)*

for j = 1 to *max-flips*

if $v \vDash C$ then return v

$p \coloneqq$ variable in C such that flipping its value gives the largest increase in the number of clauses of C that are satisfied by v

$v \coloneqq v$ with the assignment to p flipped

return ∅

Problem	tautology	dptaut	dplltaut
prime 3	0.00	0.00	0.00
prime 4	0.02	0.06	0.04
prime 9	18.94	2.98	0.51
prime 10	11.40	3.03	0.96
prime 11	28.11	2.98	0.51
prime 16	>1 hour	out of memory	9.15
prime 17	>1 hour	out of memory	3.87
ramsey 3 3 5	0.03	0.06	0.02
ramsey 3 3 6	5.13	8.28	0.31
mk_adder_test 3 2	>>1 hour	6.50	7.34
mk_adder_test 4 2	>>1 hour	22.95	46.86
mk_adder_test 5 2	>>1 hour	44.83	170.98
mk_adder_test 5 3	>>1 hour	38.27	250.16
mk_adder_test 6 3	>>1 hour	out of memory	1186.4
mk_adder_test 7 3	>>1 hour	out of memory	3759.9

From Slides of Clark Barrett's lecture.
Summer School on Verification Technology, Systems & Applications, September 17, 2008 – p. 42/98

Stålmarck's algorithm

Breadth first search instead of depth-first

Given a set of clauses Δ and any basic deduction algorithm R, the ***dilemma rule*** performs a case split on some literal p by considering the new sets of clauses $\Delta \cup \{(\neg p)\}$ and $\Delta \cup \{(p)\}$.

R is applied to each of these to get Δ_0 and Δ_1 respectively

The original Δ is augmented with $\Delta_0 \cap \Delta_1$

Abstract DPLL

- Abstract DPLL uses **states** and **transitions** to model the progress of the algorithm
- Most states are of the form $M||F$ where
  - M is a **sequence** of annotated **literals** denoting partial truth assignment
  - F is the CNF formula being checked, represented as a set of **clauses**
- **Initial state:** $\emptyset||F,$ where F is to be checked for satisfiability
- Transitions between states are defined by a set of **conditional transition rules**
- **Final state**
  - *Fail* special state, if F is unsatisfiable, or
  - $M||G$, where G is CNF formula equisatisfiable with original F and $M \vDash G$
- We will write $M \vDash C$ to mean that every truth assignment v, $v(M) = True$ implies $v(C) = True$

Abstract DPLL

UnitProp: $M \| \| F, C \vee \ell$	$\rightarrow M\ \ell \| \| F, C \vee \ell$	If $\begin{cases} M \vDash \neg C \\ \ell \text{ is undefined in } M \end{cases}$
PureLiteral: $M \| \| F$	$\rightarrow M\ \ell \| \| F$	If $\begin{cases} \ell \text{ occurs in some clause of } F \\ \neg\ell \text{ occurs in no clause of } F \\ \ell \text{ is undefined in } M \end{cases}$
Decide: $M \| \| F$	$\rightarrow M\ \ell^d \| \| F$	If $\begin{cases} \ell \text{ or } \neg\ell \text{ occurs in a clause of } F \\ \ell \text{ is undefined in } M \end{cases}$
Backtrack: $M\ \ell^d\ N \| \| F, C$	$\rightarrow M\ \neg\ell \| \| F, C$	If $\begin{cases} M\ \ell^d\ N \vDash \neg C \\ N \text{ contains no decision literals} \end{cases}$
Fail: $M \| \| F, C$	$\rightarrow fail$	If $\begin{cases} M \vDash \neg C \\ M \text{ contains no decision literals} \end{cases}$

Abstract DPLL and Abstract DPLL Modulo Theories Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli, 2006.

An Example: Abstract DPLL

$\phi || 1 \vee \bar{2} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (PureLiteral)

$4 || 1 \vee \bar{2} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad \underline{1 \vee 4}$

$\Rightarrow$ (Decide)

$4\ 1^d\ || \underline{1 \vee \bar{2}} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (UnitProp)

$4\ 1^d\ \bar{2}\ || 1 \vee \bar{2}. \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (UnitProp)

$4\ 1^d\ \bar{2}\ 3\ || 1 \vee \bar{2}. \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$4\ 1^d\ \bar{2}\ 3\ || 1 \vee \bar{2}. \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (Backtrack)

$4\ \bar{1}\ || 1 \vee \bar{2} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (UnitProp)

$4\ \bar{1}\ \bar{2} || 1 \vee \bar{2} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$

$\Rightarrow$ (UnitProp)

$4\ \bar{1}\ \bar{2}\ 3\ || 1 \vee \bar{2} \quad \bar{1} \vee \bar{2} \quad 2 \vee 3 \quad \bar{3} \vee 2 \quad 1 \vee 4$ => Fail

Assignments

- HW1 (due Sept 17th)
  - Install Z3
- Give a 3 sentence pitch for your project in next class

- Next: SMT