# Propositional Logic

ECE 650
Methods & Tools for Software Engineering (MTSE)
Fall 2023

Presented by
Dr. Albert Wasef

Used by permission from Prof. Arie Gurfinkel

UNIVERSITY OF WATERLOO

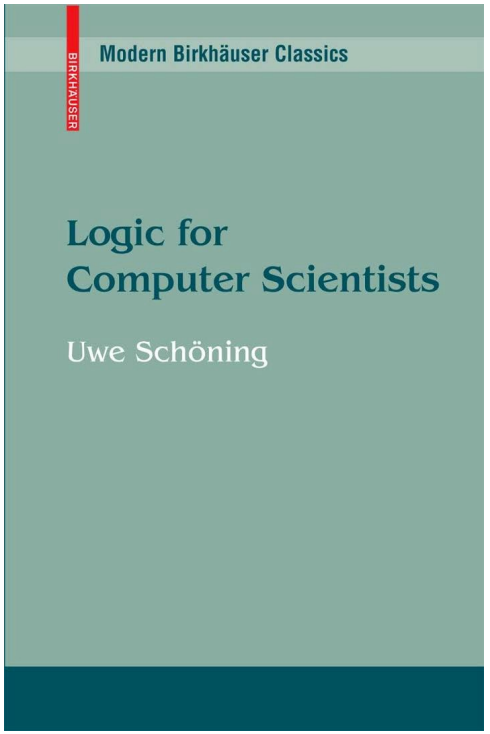
# References

Chpater 1 of Logic for Computer Scientists

- https://link.springer.com/chapter/10.1007/978-0-8176-4763-6_2

Library link:

- https://link-springer-com.proxy.lib.uwaterloo.ca/chapter/10.1007/978-0-8176-4763-6_2



# What is Logic

According to Merriam-Webster dictionary logic is:

**a** (1) : a science that deals with the principles and criteria of validity of inference and demonstration

**d** :the arrangement of circuit elements (as in a computer) needed for computation; *also*: the circuits themselves

# What is Formal Logic

Formal Logic consists of

- syntax – what is a legal sentence in the logic
- semantics – what is the meaning of a sentence in the logic
- proof theory – formal (syntactic) procedure to construct valid/true sentences

Formal logic provides

- a language to precisely express knowledge, requirements, facts
- a formal way to reason about consequences of given facts rigorously

# Where is Formal Logic used in SE?

Programming Languages

- conditional statements
- meaning (semantics) of programs

Requirements and Specification

- rigorous definition of what is to be constructed
- e.g., if we used formal logic for assignments, there would be no questions on what is required, what is optional, and no questions

Computer Hardware

- computers are built out of simple logical gates
- most computer hardware can be specified and understood in propositional logic

Testing and Verification

- rigorously validate that software satisfies its specifications

Algorithms and Optimization

- many complex problems can be reduced to logic and solved effectively using automated decision procedures

# Propositional Logic (or Boolean Logic)

Explores simple grammatical connections such as *and*, *or*, and *not* between simplest “atomic sentences”

A = “Paris is the capital of France”

B = “mice chase elephants”

The subject of propositional logic is to declare formally the truth of complex structures from the truth of individual atomic components

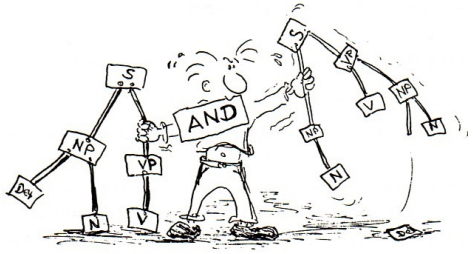
A and B

A or B

if A then B

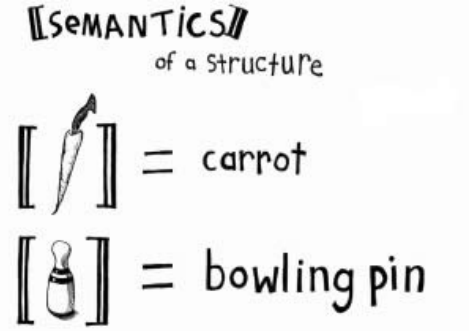
A and not A

# Syntax and Semantics




## Syntax

- MW: the way in which linguistic elements (such as words) are put together to form constituents (such as phrases or clauses)
- Determines and restricts how things are written




## Semantics

- MW: the study of meanings
- Determines how syntax is interpreted to give meaning

# Syntax of Propositional Logic

An *atomic formula* has a form $A_i$ , where i = 1, 2, 3 …

*Formulas* are defined inductively as follows:

- All atomic formulas are formulas
- For every formula F, ¬F (called not F) is a formula
- For all formulas F and G, F ∧ G (called and) and F ∨ G (called or) are formulas

Abbreviations

- use A, B, C, … instead of $A_1$, $A_2$, …
- use $F_1 \rightarrow F_2$ instead of $\neg F_1 \vee F_2$ (implication)
- use $F_1 \leftrightarrow F_2$ instead of $(F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$ (iff)

# Formal Syntax of Propositional Logic

**constant** ::= true | false | 0 | 1 | ⊤ | ⊥

**variable** ::= p | q | r | A | B | C | $A_0$ | $A_1$ | …

**atom** ::= constant | variable

**literal** ::= atom | ¬ atom

**formula** ::= literal |
¬ formula |
formula ∧ formula |
formula ∨ formula

# Example

$$F = \neg((A_5 \wedge A_6) \vee \neg A_3)$$

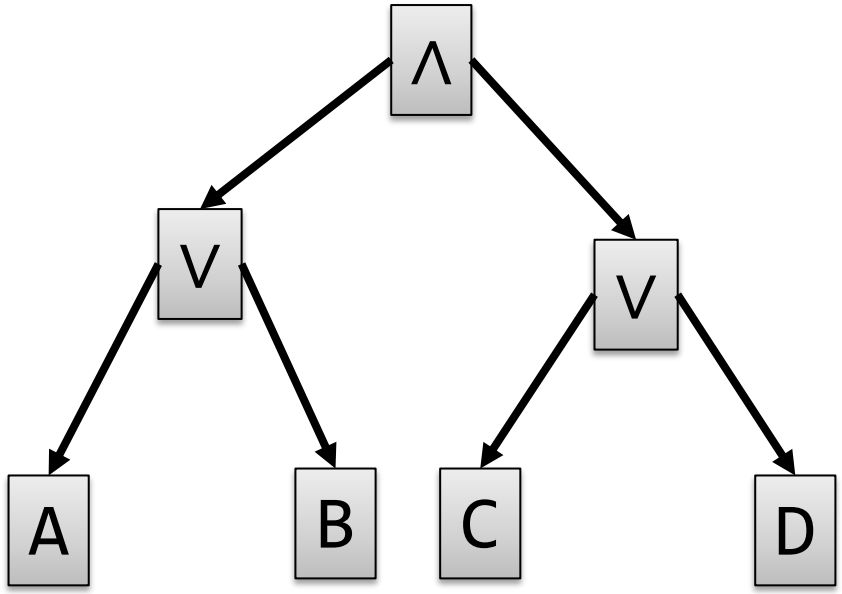
Sub-formulas are

$$F, ((A_5 \wedge A_6) \vee \neg A_3),$$
$$A_5 \wedge A_6, \neg A_3,$$
$$A_5, A_6, A_3$$

# Abstract Syntax Tree (AST)

AST is an abstract tree representation of formulas

- each node represents a syntactic construct occurring in the formula
  - e.g., variable, operator, etc.
- called *"**abstract**"* because some details of concrete syntax are omitted
  - AST normalizes (provides common representation) of irrelevant differences in syntax (e.g., white space, order of operations)
- `example AST: (A ∨ B) ∧ (C ∨ D)`



# Semantics of propositional logic 1/2

Start with two truth values: {0, 1}

- 0 stands for false, and 1 stands for true

Let **D** be any subset of the *atomic* formulas

An *assignment* **A** is a map **D** → {0, 1}

- **A** assigns true/false to every atomic in **D**

Let **E** ⊇ **D** be a set of formulas built from **D** using propositional connectives

*Extended assignment* **A**': **E** → {0, 1} extends **A** from atomic formulas to all formulas

# Semantics of propositional logic 2/2

For an atomic formula $A_i$ in **D**: $\mathbf{A'}(A_i) = \mathbf{A}(A_i)$

$\mathbf{A'}(F \wedge G) = 1$ if $\mathbf{A'}(F) = 1$ and $A'(G) = 1$
$= 0$ otherwise

$\mathbf{A'}(F \vee G) = 1$ if $\mathbf{A'}(F) = 1$ or $A'(G) = 1$
$= 0$ otherwise

$\mathbf{A'}(\neg F) = 1$ if $\mathbf{A'}(F) = 0$
$= 0$ otherwise

# Exercise: Define Extended Assignment

$$F = \neg((A \wedge B) \vee C)$$

$$\mathcal{A}(A) = 1$$
$$\mathcal{A}(B) = 1$$
$$\mathcal{A}(C) = 0$$

Is F true or false under A'?

# Truth Tables for Basic Operators

| $\mathcal{A}(F)$ | $\mathcal{A}(G)$ | $\mathcal{A}'(F \wedge G)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $\mathcal{A}(F)$ | $\mathcal{A}(G)$ | $\mathcal{A}'(F \vee G)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

An extended assignment A' extends the truth table from atomic propositions to propositional formulas

| $\mathcal{A}(F)$ | $\mathcal{A}'(\neg F)$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

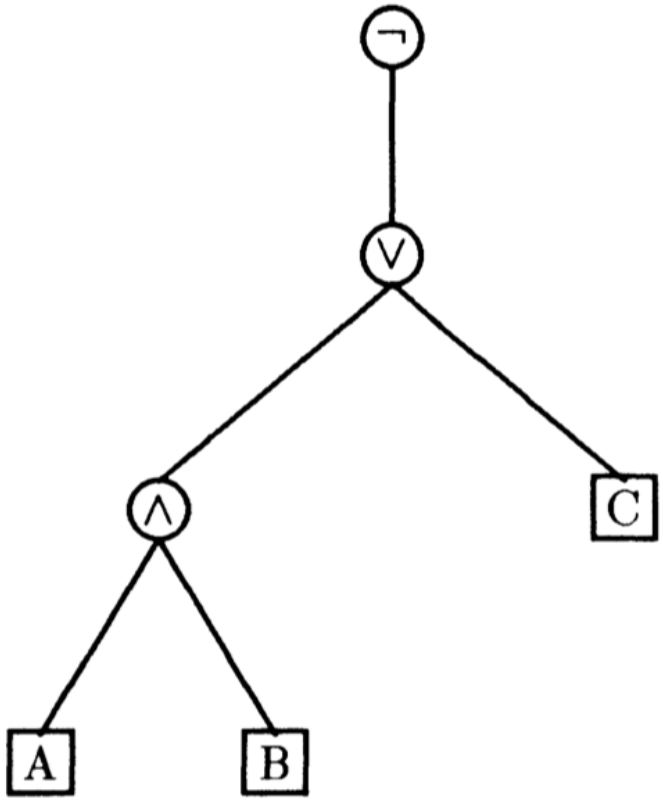
## Formula

$$F = \neg((A \wedge B) \vee C)$$

## Assignment

$$\mathcal{A}(A) = 1$$
$$\mathcal{A}(B) = 1$$
$$\mathcal{A}(C) = 0$$

## Abstract Syntax Tree (AST)



# Propositional Logic: Semantics

An assignment A is *suitable* for a formula F if A assigns a truth value to every atomic proposition of F

An assignment A is a *model* for F, written A⊨ F, iff

- A is suitable for F
- A'(F) = 1, i.e., F *evaluates to true (or holds)* under A

A formula F is *satisfiable* iff F has a model, otherwise F is *unsatisfiable* (or contradictory)

A formula F is *valid* (or a tautology), written ⊨ F, iff every suitable assignment for F is a model for F

# Determining Satisfiability via a Truth Table

A formula F with n atomic sub-formulas has $2^n$ suitable assignments

Build a truth table enumerating all assignments

F is satisfiable iff there is at least one entry with 1 in the output

| | $A_1$ | $A_2$ | $\cdots$ | $A_{n-1}$ | $A_n$ | $F$ |
|---|---|---|---|---|---|---|
| $\mathcal{A}_1$: | 0 | 0 | | 0 | 0 | $\mathcal{A}_1(F)$ |
| $\mathcal{A}_2$: | 0 | 0 | | 0 | 1 | $\mathcal{A}_2(F)$ |
| $\vdots$ | | | $\ddots$ | | | $\vdots$ |
| $\mathcal{A}_{2^n}$: | 1 | 1 | | 1 | 1 | $\mathcal{A}_{2^n}(F)$ |

# Problem: Is formula F SAT?

$$F = (\neg A \rightarrow (A \rightarrow B))$$

| $A$ | $B$ | $\neg A$ | $(A \rightarrow B)$ | $F$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |

# Validity and Unsatisfiability

**Theorem:**

A formula F is valid if and only if ¬F is unsatifsiable

**Proof:**

F is valid ⇔ every suitable assignment for F is a model for F

⇔ every suitable assignment for ¬ F is not a model for ¬ F

⇔ ¬ F does not have a model

⇔ ¬ F is unsatisfiable

# Book: Exercise #10

Prove of give a counterexample

Valid

(a) If (F ⇒ G) is *valid* and F is *valid*, then G is *valid*

(b) If (F ⇒ G) is *sat* and F is *sat*, then G is *sat*

Not Valid

(c) If (F ⇒ G) is *valid* and F is *sat*, then G is *sat*

Valid

# Semantic Equivalence

Two formulas F and G are *(semantically) equivalent*, written **F ≡ G**, iff for every assignment A that is suitable for both F and G, A'(F) = A'(G)

For example, (F ∧ G) is equivalent to (G ∧ F)

Formulas with different atomic propositions can be equivalent

- e.g., all tautologies are equivalent to true
- e.g., all unsatisfiable formulas are equivalent to false

# Substitution Theorem

**Theorem:** Let F and G be equivalent formulas. Let H be a formula in which F occurs as a sub-formula. Let H' be a formula obtained from H by replacing every occurrence of F by G. Then, H and H' are equivalent.

In symbols:

$$F \equiv G \quad \Rightarrow \quad H \equiv H[F \rightarrow G]$$

**Proof:**

(Let's talk about proof by induction first…)

# Aside: Notation for Substitution

We need a concise notation for “Find-Replace”

- Replace every occurrence of **variable** (string, value, …) **X** with **variable** (string, value, …) **Y** in a **formula** (string, object, state) **F**, and return the new formula (string, object, state) without changing F in place.

In Python, this looks like

- `string.replace(old, new [, count])`
- `F.replace(X, Y)`

In books, logic, slides, this course

- F[X / Y] – “F with X replaced by Y”
- F[X / Y] – ”F with X replacing Y”
- F[X → Y] – “F with X replacing Y”
- F[X ← Y] – “F with X replaced by Y”
- `s/X/Y/g` – sed syntax

# Mathematical Induction (over Natural Numbers)

To proof that a property P(n) holds for all natural numbers n

1. Show that P(0) is true

2. Assume that P(k) is true for some natural number k
   - This assumption is called an **Inductive Hypothesis (IH)**

3. Show that P(k+1) is true using IH from the previous step

4. Conclude that P(n) holds for all natural numbers n
   - P(n) is proven (or established, true) by mathematical induction

# Example: Mathematical Induction

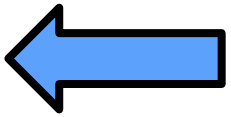
Show by induction that the formula for arithmetic series is correct

$$0 + \cdots + n = \frac{n(n+1)}{2}$$

Base Case: P(0) is $0 = \frac{0(0+1)}{2}$

IH: Assume P(k), show P(k+1)

$$\begin{aligned} & 0 + \cdots + k + (k+1) \\ = \; & \frac{k(k+1)}{2} + (k+1) \\ = \; & \frac{k(k+1)+2(k+1)}{2} \\ = \; & \frac{(k+1)((k+1)+1)}{2} \end{aligned}$$



IH is used in this step

# Structural Induction on the formula structure

The definition of a syntax of a formula is an *inductive* definition

- first, define atomic formulas; second, define more complex formulas from simple ones, each next definition uses previous definition recursively

The definition of the semantics of a formula is also inductive

- first, determine value of atomic propositions; second, define values of more complex formulas

The same principle works for proving properties of formulas!

- To show that every formula F satisfies some property S:
- (base case) show that S holds for atomic formulae
- (induction step) assume S holds for an arbitrary fixed formulas F and G. Show that S holds for $(F \wedge G)$, $(F \vee G)$, and $(\neg F)$

# Substitution Theorem (back from detour)

**Theorem:** Let F and G be equivalent formulas. Let H be a formula in which F occurs as a sub-formula. Let H' be a formula obtained from H by replacing every occurrence of F by G. Then, H and H' are equivalent.

**Proof:** by induction on formula structure

(base case) if H is atomic, then F = H, H' = G, and F ≡ G

(inductive step)

(case 1) H = ¬ $H_1$

(case 2) H = $H_1 \wedge H_2$

(case 3) H = $H_1 \vee H_2$

# Useful Equivalences (1/ 2)

$$
\begin{array}{rcll}
(F \wedge F) & \equiv & F & \\
(F \vee F) & \equiv & F & \text{(Idempotency)} \\
 & & & \\
(F \wedge G) & \equiv & (G \wedge F) & \\
(F \vee G) & \equiv & (G \vee F) & \text{(Commutativity)} \\
 & & & \\
((F \wedge G) \wedge H) & \equiv & (F \wedge (G \wedge H)) & \\
((F \vee G) \vee H) & \equiv & (F \vee (G \vee H)) & \text{(Associativity)} \\
 & & & \\
(F \wedge (F \vee G)) & \equiv & F & \\
(F \vee (F \wedge G)) & \equiv & F & \text{(Absorption)} \\
 & & & \\
(F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) & \\
(F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) & \text{(Distributivity)} \\
 & & & \\
\neg\neg F & \equiv & F & \text{(Double Negation)}
\end{array}
$$

# Useful Equivalences (2/ 2)

$$\neg(F \wedge G) \equiv (\neg F \vee \neg G)$$
$$\neg(F \vee G) \equiv (\neg F \wedge \neg G) \qquad \text{(deMorgan's Laws)}$$

$$(F \vee G) \equiv F, \text{ if } F \text{ is a tautology}$$
$$(F \wedge G) \equiv G, \text{ if } F \text{ is a tautology} \qquad \text{(Tautology Laws)}$$

$$(F \vee G) \equiv G, \text{ if } F \text{ is unsatisfiable}$$
$$(F \wedge G) \equiv F, \text{ if } F \text{ is unsatisfiable} \qquad \text{(Unsatisfiability Laws)}$$

Don't believe in these laws. Prove them using structural induction!

# Bool: Exercise 18: Children and Doctors

Formalize and show that the two statements are equivalent

- If the child has temperature or has a bad cough and we reach the doctor, then we call him

$$((T \vee C) \wedge R) \Rightarrow D$$

- If the child has temperature, then we call the doctor, provided we reach him, and, if we reach the doctor then we call him, if the child has a bad cough

$$(R \Rightarrow (T \Rightarrow D)) \wedge (C \Rightarrow (R \Rightarrow D))$$

$((T \vee C) \wedge R) \Rightarrow D$

$( (T \wedge R) \vee (C \wedge R) \Rightarrow D)$

$((T \wedge R) \Rightarrow D) \wedge ((C \wedge R) \Rightarrow D)$

$(R \Rightarrow (T \Rightarrow D)) \wedge (C \Rightarrow (R \Rightarrow D))$

Law:

$(a \vee b) \Rightarrow c$

$(a \Rightarrow c) \wedge (b \Rightarrow c)$

Extra Slides

# END OF LECTURE

# Book Example: Secret to long life

"What is the secret of your long life?" a centenarian was asked.

"I strictly follow my diet: If I don't drink beer for dinner, then I always have fish. Any time I have both beer and fish for dinner, then I do without ice cream. If I have ice cream or don't have beer, then I never eat fish."

The questioner found this answer rather confusing. Can you simplify it?



Centenarian – a person who lives to or beyond an age 100.


SINCE 1828

JOIN MWU | GAMES | BROWSE THESAURUS | WORD OF THE DAY | WORDS AT PLAY

normal form

DICTIONARY | THESAURUS

# normal form noun

## Definition of *normal form*

*logic*

: a canonical or standard fundamental form of a statement to which others can be reduced

*especially* : a compound statement in the propositional calculus consisting of nothing but a conjunction of disjunctions whose disjuncts are either elementary statements or negations thereof

https://www.merriam-webster.com/dictionary/normal%20form

# Negation Normal Form (NNF)

A formula F is in Negation Normal Form (NNF) if every occurrence of negation (¬) in F is applied to an atomic sub-formula of F.

For example,

- ¬ a ∨ ¬ b ∨ c  is in NNF
- ¬ (a ∧ b ∧ ¬ c)  is NOT in NNF (why?)

**Theorem (NNF):** For every formula F there is a semantically equivalent form G in NNF. In symbols

- For every F, exists G in NNF, such that F ≡ G

# Proof of NNF Theorem

By structural induction on the structure of a formula F

**(base case):** atomic formulas A and ¬A are in NNF

**(IH):** Assume that the theorem is true for every sub-formula of F: For every sub-formula H of F there exists J in NNF such that J ≣ H. Show that there exists G in NNF such that G ≣ F

**(¬ case):** Assume F = ¬H. Then, F ≣ ¬(¬J) ≣ J

**(∨ case):** Assume F = $H_1 \vee H_2$. Then, F ≣ $J_1 \vee J_2$

**(∧ case):** Assume F = $H_1 \wedge H_2$. Then, F ≣ $J_1 \wedge J_2$

# Normal Form: DNF

A *literal* is either an atomic proposition v or its negation ¬v

A *cube* is a conjunction of literals

- e.g., (v1 ∧ ¬ v2 ∧ v3)

A formula F is in *Disjunctive Normal Form* (DNF) if F is a disjunction of conjunctions of literals

$$\bigvee_{i=1}^{n} \left( \bigwedge_{j=1}^{m_i} L_{i,j} \right)$$

**(Fun) Fact:** determining whether a DNF formula F is satisfiable is easy

- easy == linear in the size of the formula

# Normal Form: CNF

A *literal* is either an atomic proposition v or its negation ¬v

A *clause* is a disjunction of literals

- e.g., (v1 ∨ ¬v2 ∨ v3)

A formula F is in *Conjunctive Normal Form* (CNF) if F is a conjunction of disjunctions of literals

$$\bigwedge_{i=1}^{n} \left( \bigvee_{j=1}^{m_i} L_{i,j} \right)$$

**(Fun) Fact:** determining whether a CNF formula F is satisfiable is hard

- hard == NP-complete

# Normal Form Theorem

**Theorem:** For every formula F, there is an equivalent formula $F_1$ in CNF, and an equivalent formula $F_2$ in DNF.

That is, CNF and DNF are normal forms:

- Every propositional formula can be converted to CNF and to DNF without affecting its meaning (i.e., semantics)!

**Proof:** (by induction on the structure of the formula F)

Details are left as an exercise!

# Converting a formula to CNF

Given a formula F

1. Substitute in F every occurrence of a sub-formula of the form
   ¬¬G by G
   ¬(G ∧ H) by (¬G ∨ ¬H)
   ¬(G ∨ H) by (¬G ∧ ¬H)
   The result is a formula in Negation Normal Form (NNF)

2. Substitute in F each occurrence of a sub-formula of the form
   (F ∨ (G ∧ H)) by ((F ∨ G) ∧ (F ∨ H))
   ((F ∧ G) ∨ H) by ((F ∨ H) ∧ (G ∨ H))

The resulting formula F is in CNF

- the result in CNF might be exponentially bigger than original formula F

# Example: From Truth Table to CNF and DNF

**DNF**

$$(\neg A \wedge \neg B \wedge \neg C) \vee$$
$$(A \wedge \neg B \wedge \neg C) \vee$$
$$(A \wedge \neg B \wedge C)$$

**CNF**

$$(A \vee B \vee \neg C) \wedge$$
$$(A \vee \neg B \vee C) \wedge$$
$$(A \vee \neg B \vee \neg C) \wedge$$
$$(\neg A \vee \neg B \vee C) \wedge$$
$$(\neg A \vee \neg B \vee \neg C)$$

**Truth table**

| $A$ | $B$ | $C$ | $F$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

see the book for detailed algorithm

# From Truth Table to DNF / CNF

From a truth table T to DNF formula F

- For every row i of T with value 1, construct a conjunction $F_i$ that characterizes the row. That is, $F_i$ is a formula that is true **exactly** for the assignment of row I
- Let $F = \vee F_i$, then F is equivalent to T and is in DNF
- Fact: F has as many disjuncts as rows with value 1 in T
- Question: How big can F be? How small can F be?

From a truth table T to CNF formula G

- For every row i of T with value 0, construct a conjunction $F_i$ that characterizes the row
- Let $G_i$ be NNF of $\neg F_i$
- Let $G = \wedge G_i$, then G is equivalent to T and is in CNF
- Fact: F has as many disjuncts as rows with value 0 in T
- Question: How big can F be? How small can F be?

# 2-CNF Fragment

A formula F is in 2-CNF iff

- F is in CNF
- every clause of F has at most 2 literals

**Theorem:** There is a polynomial algorithm for deciding wither a a 2-CNF formula F is satisfiable

# Horn Fragment

A formula F is in Horn fragment iff

- F is in CNF
- in every clause, at most one literal is positive

$$(A \vee \neg B) \wedge (\neg C \vee \neg A \vee D) \wedge (\neg A \vee \neg B) \wedge D \wedge \neg E$$

- Note that each clause can be written as an implication
  - e.g. C ∧ A ⇒ D , A ∧ B ⇒ False, True ⇒ D

$$(B \rightarrow A) \wedge (A \wedge C \rightarrow D) \wedge (A \wedge B \rightarrow 0) \wedge (1 \rightarrow D) \wedge (E \rightarrow 0)$$

**Theorem:** There is a polynomial time algorithm for deciding satisfiability of a Horn formula F

# Horn Satisfiability

**Input**: a Horn formula F

**Output**: UNSAT or SAT + satisfying assignment for F

**Step 1**: Mark every occurrence of an atomic formula *A* in F if there is an occurrence of sub-formula of the form *A* in F

**Step 2**: pick a formula G in F of the form $A1 \wedge \ldots \wedge An \rightarrow B$ such that all of $A_1, \ldots, A_n$ are already marked

- if B = 0, return UNSAT
- otherwise, mark B and go back to Step 2

**Step 3**: Construct a suitable assignment S such that S(Ai) = 1 iff Ai is marked. Return SAT with a satisfying assignment S.

# Exercise 21

Apply Horn satisfiability algorithm on a formula

$$(\neg A \vee \neg B \vee \neg D)$$

$$\neg E$$

$$(\neg C \vee A)$$

$$C$$

$$B$$

$$(\neg G \vee D)$$

$$G$$

# 3-CNF Fragment

A formula F is in 3-CNF iff

- F is in CNF
- every clause of F has at most 3 literals

**Theorem**: Deciding whether a 3-CNF formula F is satisfiable is at least as hard as deciding satisfiability of an arbitrary CNF formula G

**Proof:** by effective *reduction* from CNF to 3-CNF

Let G be an arbitrary CNF formula. Replaced every clause of the form

$$(\ell_0 \vee \cdots \vee \ell_n)$$

with 3-literal clauses

$$(\ell_0 \vee b_0) \wedge (\neg b_0 \vee \ell_1 \vee b_1) \wedge \cdots \wedge (\neg b_{n-1} \vee \ell_n)$$

where $\{b_i\}$ are fresh atomic propositions not appearing in F

# Complexity of 3-CNF Satisfiability

**Theorem (Cook-Levin):** The Boolean Satisfiability Problem is NP-complete

**Consequences**

- If a formula F is satisfiable, then there exists a certificate for satisfiability that can be checked in P (polynomial) time.
  - That is, checking solutions is easy
- Any other problem that has polynomial certificates is polynomial reducible to Boolean Satisfiability
  - That is, such problems can be solved by writing a loop-free program, compiling it to a Boolean circuit, and checking whether the circuit ever accepts some input
- **MANY MANY MANY OPTIMIZATION PROBLEMS ARE LIKE THAT**
- Boolean Satisfiability is easy iff P = NP
  - i.e., Boolean satisfiability today is a VERY VERY VERY HARD problem!

# Background Reading: SAT



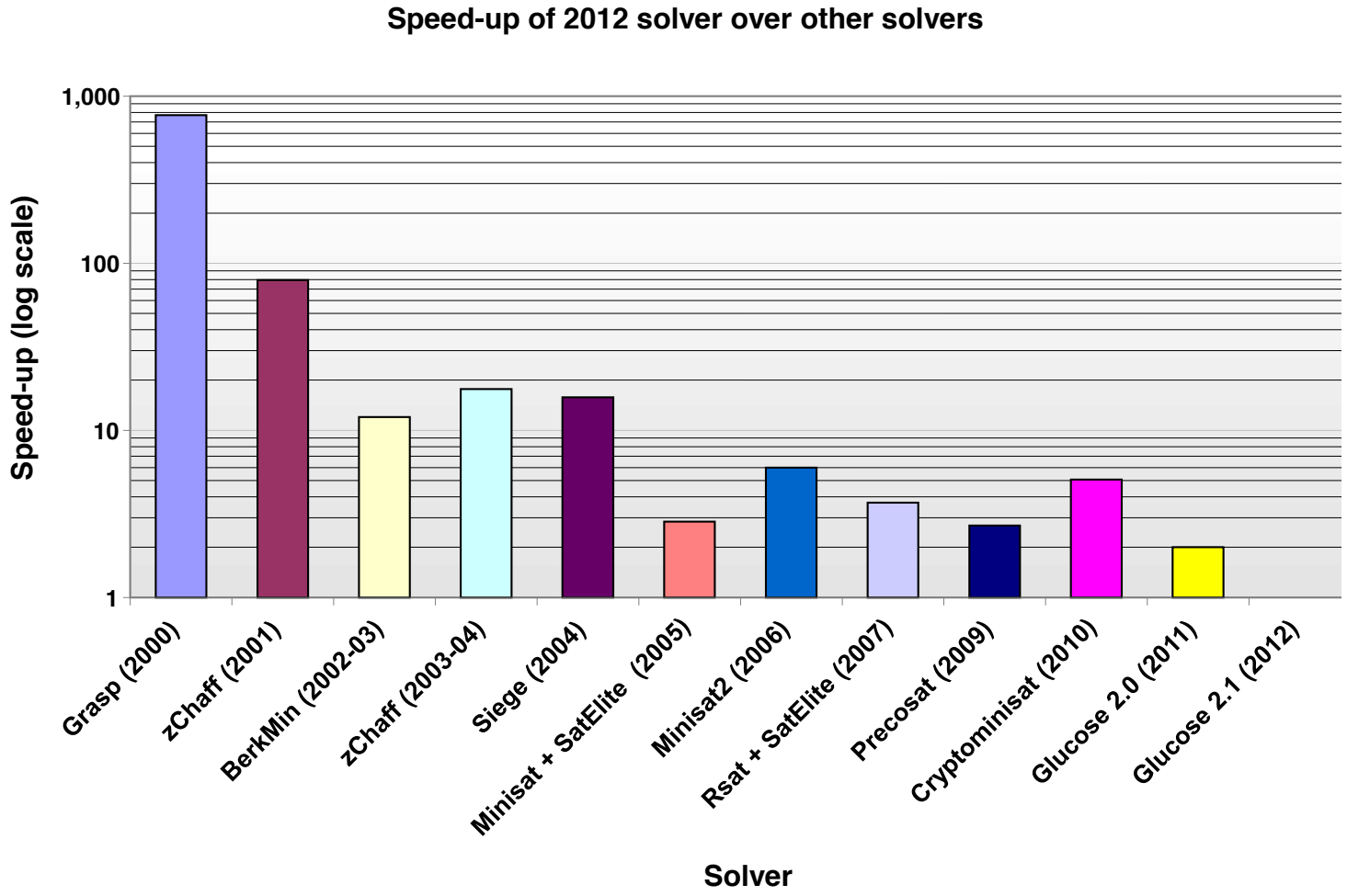
REVIEW ARTICLES

# Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang
Communications of the ACM, Vol. 52 No. 8, Pages 76-82
10.1145/1536616.1536637


There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner consistent with various rules the host would like to impose. This also applies to applications in computing, for example, ensuring that a hardware/software system functions correctly with its overall behavior constrained by the behavior of its components and their

# Some Experience with SAT Solving




from M. Vardi, https://www.cs.rice.edu/~vardi/papers/highlights15.pdf

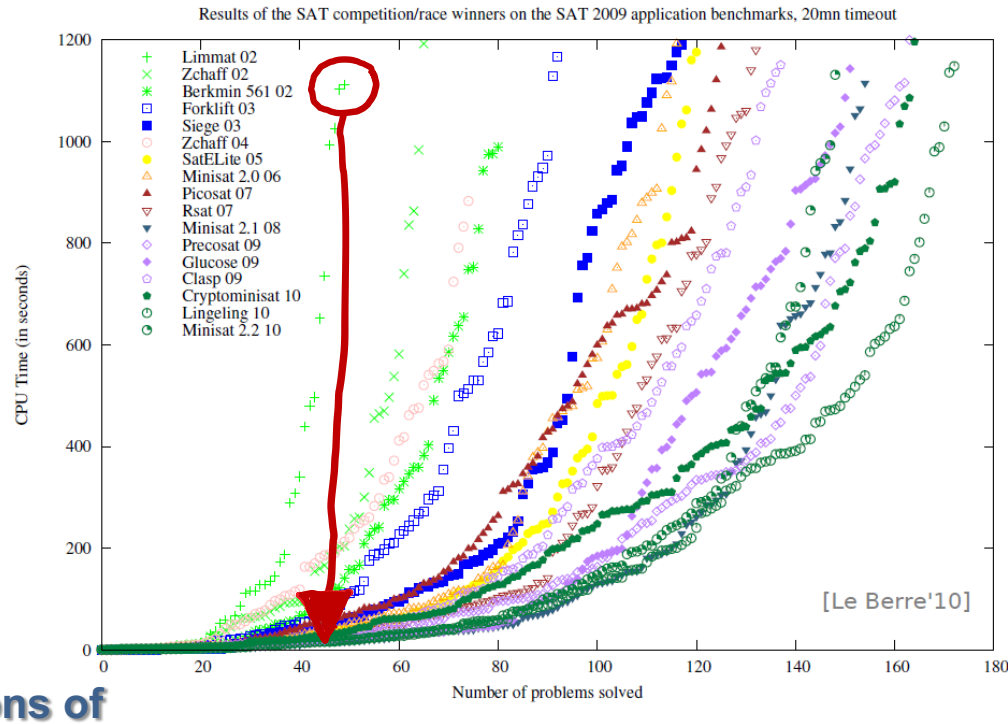
# SAT - Milestones

Problems impossible 10 years ago are trivial today

| year | Milestone |
|---|---|
| 1960 | Davis-Putnam procedure |
| 1962 | Davis-Logeman-Loveland |
| 1984 | Binary Decision Diagrams |
| 1992 | DIMACS SAT challenge |
| 1994 | SATO: clause indexing |
| 1997 | GRASP: conflict clause learning |
| 1998 | Search Restarts |
| 2001 | zChaff: 2-watch literal, VSIDS |
| 2005 | Preprocessing techniques |
| 2007 | Phase caching |
| 2008 | Cache optimized indexing |
| 2009 | In-processing, clause management |
| 2010 | Blocked clause elimination |

**Concept**

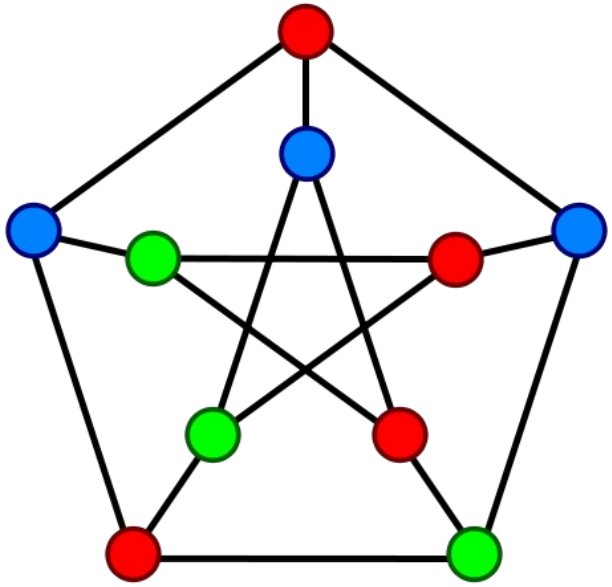
2002 2010




**Millions of variables from HW designs**

Courtesy Daniel le Berre

# Graph k-Coloring



Given a graph *G = (V, E)*, and a natural number *k > 0* is it possible to assign colors to vertices of *G* such that no two adjacent vertices have the same color.

Formally:

- does there exists a function f : V → [0..k) such that
- for every edge (u, v) in E, f(u) != f(v)

Graph coloring for k > 2 is NP-complete

**Problem**: Encode k-coloring of G into CNF

- construct CNF *C* such that C is SAT iff G is k-colorable

https://en.wikipedia.org/wiki/Graph_coloring

# *k*-coloring as CNF

Let a Boolean variable $f_{v,i}$ denote that vertex *v* has color *i*

- if $f_{v,i}$ is true if and only if f(v) = i

Every vertex has at least one color

$$\bigvee_{0 \leq i < k} f_{v,i} \qquad (v \in V)$$

No vertex is assigned two colors

$$\bigwedge_{0 \leq i < j < k} (\neg f_{v,i} \vee \neg f_{v,j}) \qquad (v \in V)$$

No two adjacent vertices have the same color

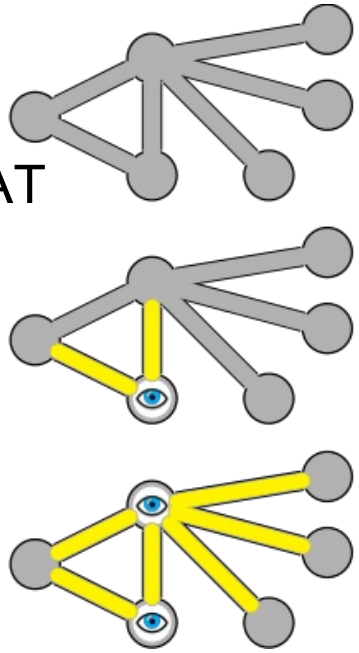
$$\bigwedge_{0 \leq i < k} (\neg f_{v,i} \vee \neg f_{u,i}) \qquad ((v,u) \in E)$$

# Vertex Cover

Given a graph G=(V,E). A vertex cover of G is a subset C of vertices in V such that every edge in E is incident to at least one vertex in C

see a4_encoding.pdf for details of reduction to CNF-SAT

- will be given together with assignment 4



https://en.wikipedia.org/wiki/Vertex_cover

# Compactness Theorem

**Theorem:**

A (possibly infinite) set M of propositional formulas is satisfiable iff every finite subset of M is satisfiable.

**Corollary**:

A (possibly infinite) set M of propositional formulas is unsatisfiable iff there exists a finite subset U of M such that U is unsatisfiable

**Proof:**

- Section 1.4 in Logic for Computer Scientists by Uwe Schoning

# Satisfiability and Unsatisfiability

Let F be a propositional formula (large)

Assume that F is satisfiable. What is a short proof / certificate to establish satisfiability without a doubt?

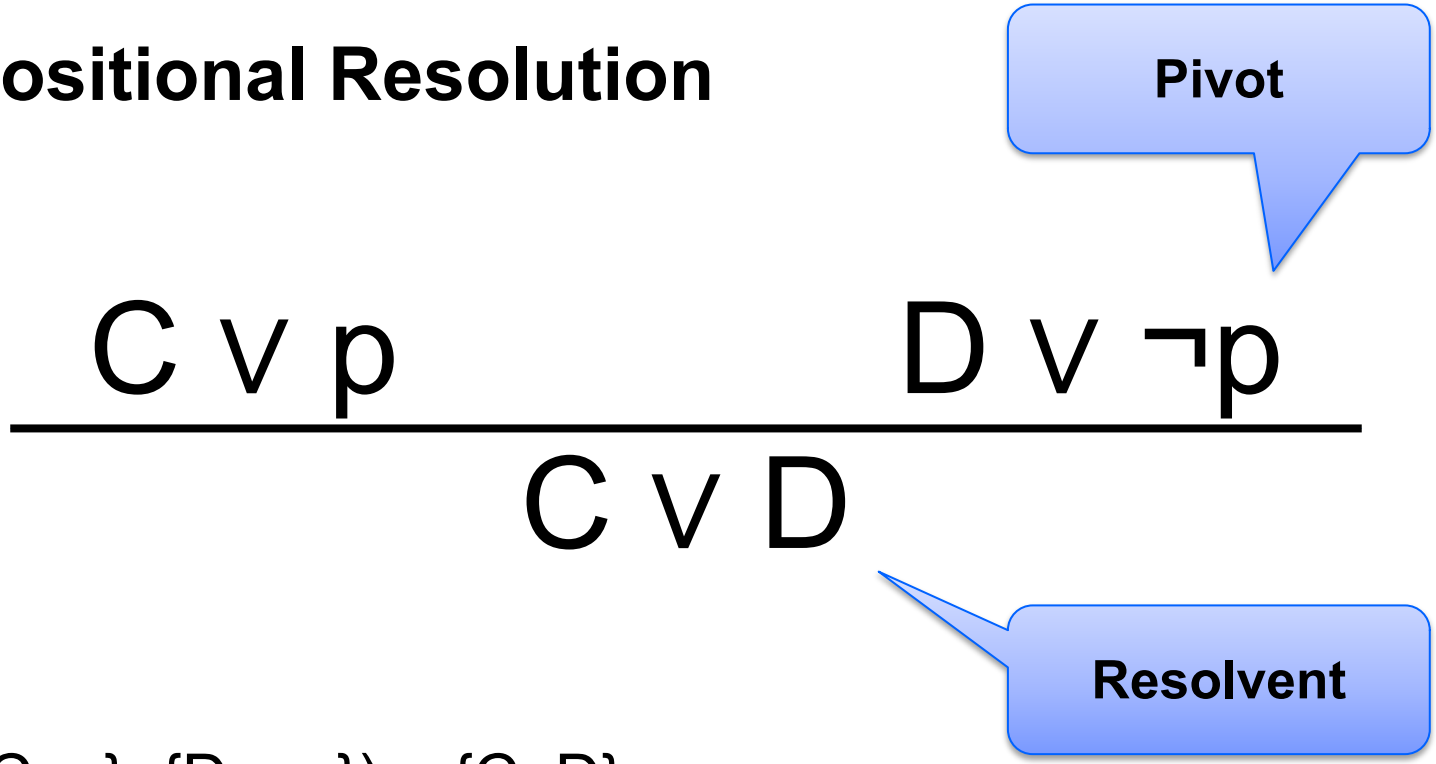
- provide a model. The model is linear in the size of the formula

Now, assume that F is unsatisfiable. What is a short proof / certificate to establish UNSATISFIABILITY without a doubt?

Is the following formula SAT or UNSAT? How do you explain your answer?

$$\neg b \wedge (\neg a \vee b \vee \neg c) \wedge a \wedge (\neg a \vee c)$$

# Propositional Resolution




Res({C, p}, {D, ¬p}) = {C, D}

Given two clauses (C, p) and (D, ¬p) that contain a literal p of different polarity, create a new clause by taking the union of literals in C and D

# Resolution Lemma

F is a CNF formula; X and Y are two clauses in F

R be a resolvent of X and Y

Then,

F ∪ { R } is semantically equivalent to F

- R is implied by F
- Any model that makes F true, also makes R true
- Adding R to F does not make F any harder to satisfy

# Resolution Theorem

F be a set of clauses (i.e., a formula in CNF)

$$Res(F) = F \cup \{R \mid R \text{ is a resolvent of two clauses in } F\}$$

$Res^n$ is defined recursively as follows:

$$Res^0(F) = F$$
$$Res^{n+1}(F) = Res(Res^n(F)), \text{ for } n \geq 0$$
$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F)$$

**Theorem**: A CNF F is UNAT iff Res*(F) contains an empty clause

# Exercise from LCS

For the following set of clauses determine $\text{Res}^n$ for n=0, 1, 2

$$A \vee \neg B \vee C$$

$$B \vee C$$

$$\neg A \vee C$$

$$B \vee \neg C$$

$$\neg C$$

# Proof of the Resolution Theorem

*(Soundness)* By Resolution Lemma, F is equivalent to $Res^i(F)$ for any i. Let n be such that $Res^{n+1}(F)$ contains an empty clause, but $Res^n(F)$ does not. Then, $Res^n(F)$ must contain to unit clauses L and ¬L. Hence, it is UNSAT, since there is no model that can make L both true and false.

(Completeness) Assume F is UNSAT. Proof by induction on the number of different atomic propositions in F.

**Base** case is trivial: F contains an empty clause.

**IH**: Assume F only has atomic propositions $A_1, \ldots A_{n+1}$

- Let $F_0$ be the result of replacing $A_{n+1}$ by 0
- Let $F_1$ be the result of replacing $A_{n+1}$ by 1
- Both $F_0$ and $F_1$ are UNSAT, so apply IH to $F_0$ and $F_1$
- Restore replaced literals by re-inserting into every clause removed from
- Combine the two resolutions (if necessary)

# Proof System

$$P_1, \ldots, P_n \vdash C$$

An inference rule is a tuple ($P_1$, …, $P_n$, C)

- where, $P_1$, …, $P_n$, C are formulas
- $P_i$ are called premises and C is called a conclusion
- intuitively, the rules says that the conclusion is true if the premises are

A proof system P is a collection of inference rules

A proof in a proof system P is a tree (or a DAG) such that

- nodes are labeled by formulas
- for each node *n*, (parents(n), n) is an inference rule in P

# Propositional Resolution

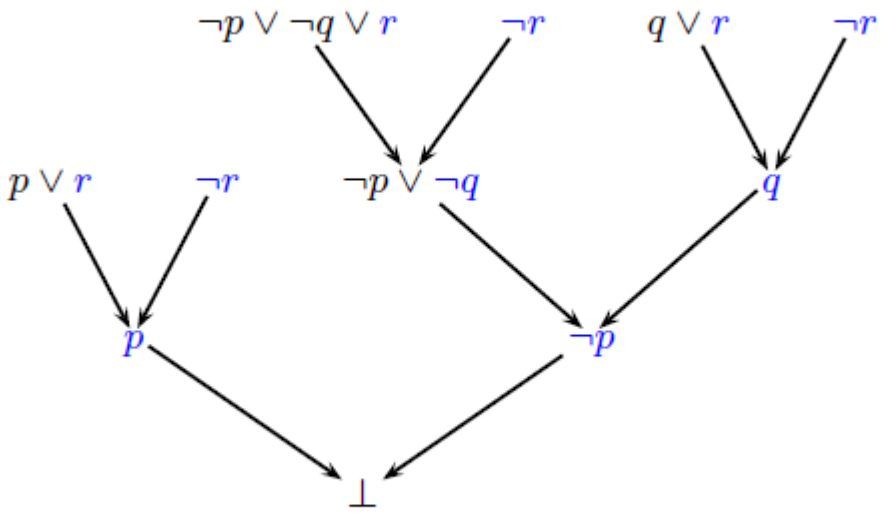
$$\frac{C \vee p \qquad D \vee \neg p}{C \vee D}$$

Propositional resolution is a sound inference rule

Proposition resolution system consists of a single propositional resolution rule

# Example of a resolution proof

A refutation of $\neg p \vee \neg q \vee r,\ p \vee r,\ q \vee r,\ \neg r$:



# Resolution Proof Example

Show by resolution that the following CNF is UNSAT

$$\neg b \wedge (\neg a \vee b \vee \neg c) \wedge a \wedge (\neg a \vee c)$$

$$\dfrac{\dfrac{\dfrac{\neg a \vee b \vee \neg c \qquad a}{b \vee \neg c} \qquad \neg b}{\neg c} \qquad \dfrac{a \qquad \neg a \vee c}{c}}{\bot}$$

# Entailment and Derivation

A set of formulas F entails a set of formulas G iff every model of F and is a model of G

$$F \models G$$

A formula G is derivable from a formula F by a proof system P if there exists a proof whose leaves are labeled by formulas in F and the root is labeled by G

$$F \vdash_P G$$

# Soundness and Completeness

A proof system P is sound iff

$$(F \vdash_P G) \implies (F \models G)$$

A proof system P is complete iff

$$(F \models G) \implies (F \vdash_P G)$$

# Propositional Resolution

**Theorem:** Propositional resolution is sound and complete for propositional logic

**Proof**: Follows from Resolution Theorem

# Book: Exercise 33

Using resolution show that

$$A \wedge B \wedge C$$

is a consequence of

$$\neg A \vee B$$

$$\neg B \vee C$$

$$A \vee \neg C$$

$$A \vee B \vee C$$

# Exercise 34

Show using resolution that F is valid

$$F = (\neg B \wedge \neg C \wedge D) \vee (\neg B \wedge \neg D) \vee (C \wedge D) \vee B$$

$$\neg F = (B \vee C \vee \neg D) \wedge (B \vee D) \wedge (\neg C \vee \neg D) \wedge \neg B$$

# Boolean Satisfiability (CNF-SAT)

Let V be a set of variables

A *literal* is either a variable v in V or its negation ¬v

A *clause* is a disjunction of literals

- e.g., (v1 ∨ ¬v2 ∨ v3)

A Boolean formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

- e.g., (v1 ∨ ¬v2) ∧ (v3 ∨ v2)

An *assignment* *s* of Boolean values to variables *satisfies* a clause *c* if it evaluates at least one literal in *c* to true

An assignment *s* *satisfies* a formula *C* in CNF if it satisfies every clause in *C*

Boolean Satisfiability Problem (CNF-SAT):

- determine whether a given CNF C is satisfiable

# Are the following CNFs SAT or UNSAT

CNF 1 (3 clauses)

- ¬ b
- ¬ a ∨ ¬b \or ¬c
- a
- SAT: s(a) = True;  s(b) = False; s(c) = False

CNF 2 (4 clauses)

- ¬b
- ¬a ∨ b ∨  ¬c
- a
- ¬a ∨ c
- UNSAT

# DIMACS CNF File Format

Textual format to represent CNF-SAT problems

```
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Format details

- comments start with `c`
- header line: `p cnf nbvar nbclauses`
  - `nbvar` is # of variables, `nbclauses` is # of clauses
- each clause is a sequence of distinct numbers terminating with 0
  - positive numbers are variables, negative numbers are negations

# Algorithms for SAT

SAT is NP-complete

DPLL (Davis-Putnam-Logemman-Loveland, ‘60)

- smart enumeration of all possible SAT assignments
- worst-case EXPTIME
- alternate between deciding and propagating variable assignments

CDCL (GRASP ‘96, Chaff ‘01)

- conflict-driven clause learning
- extends DPLL with
  - smart data structures, backjumping, clause learning, heuristics, restarts…
- scales to millions of variables
- N. Een and N. Sörensson, “An Extensible SAT-solver”, in SAT 2013.

# Background Reading: SAT



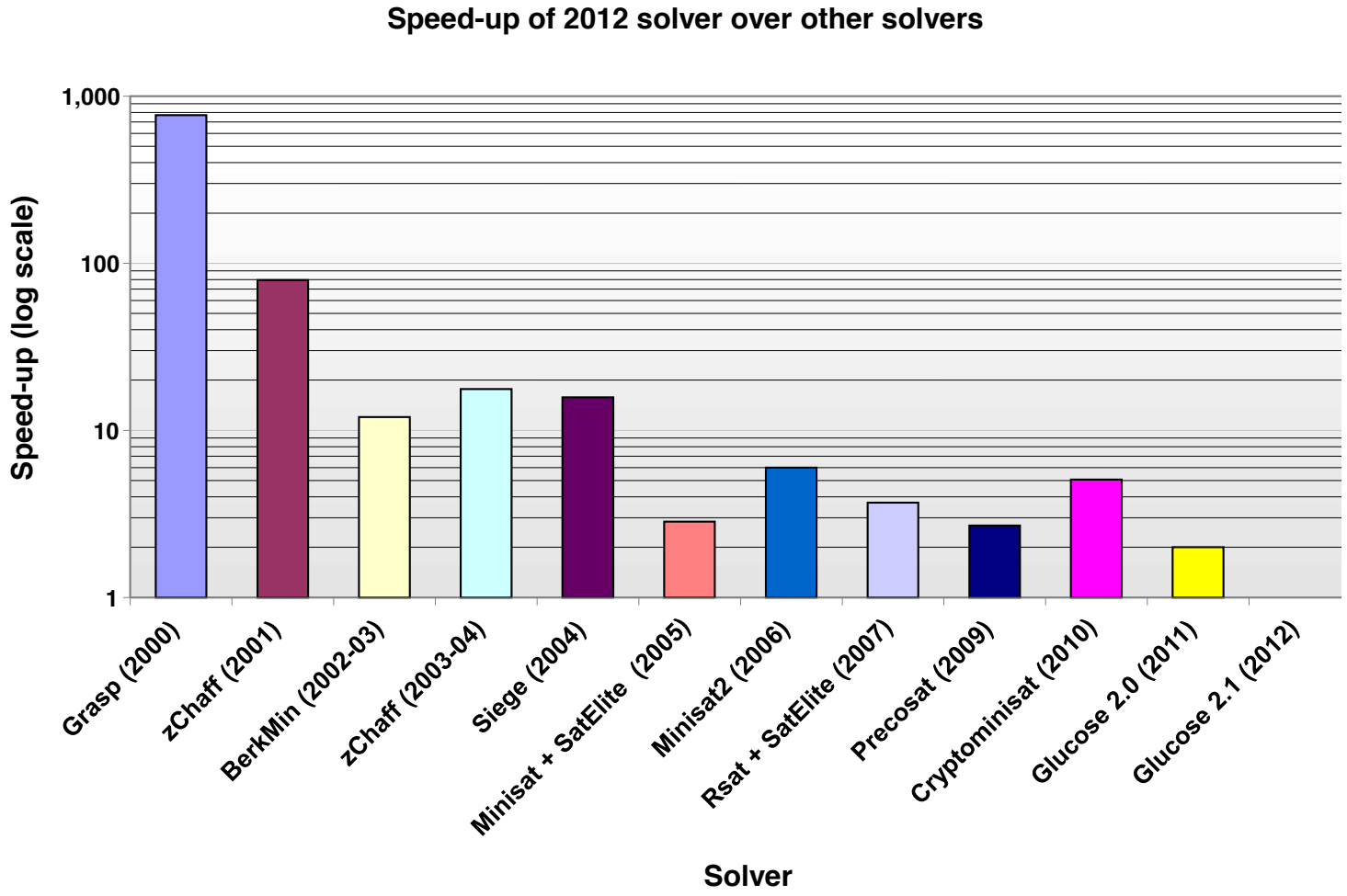
REVIEW ARTICLES

# Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang
Communications of the ACM, Vol. 52 No. 8, Pages 76-82
10.1145/1536616.1536637


There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner consistent with various rules the host would like to impose. This also applies to applications in computing, for example, ensuring that a hardware/software system functions correctly with its overall behavior constrained by the behavior of its components and their

# Some Experience with SAT Solving




from M. Vardi, https://www.cs.rice.edu/~vardi/papers/highlights15.pdf

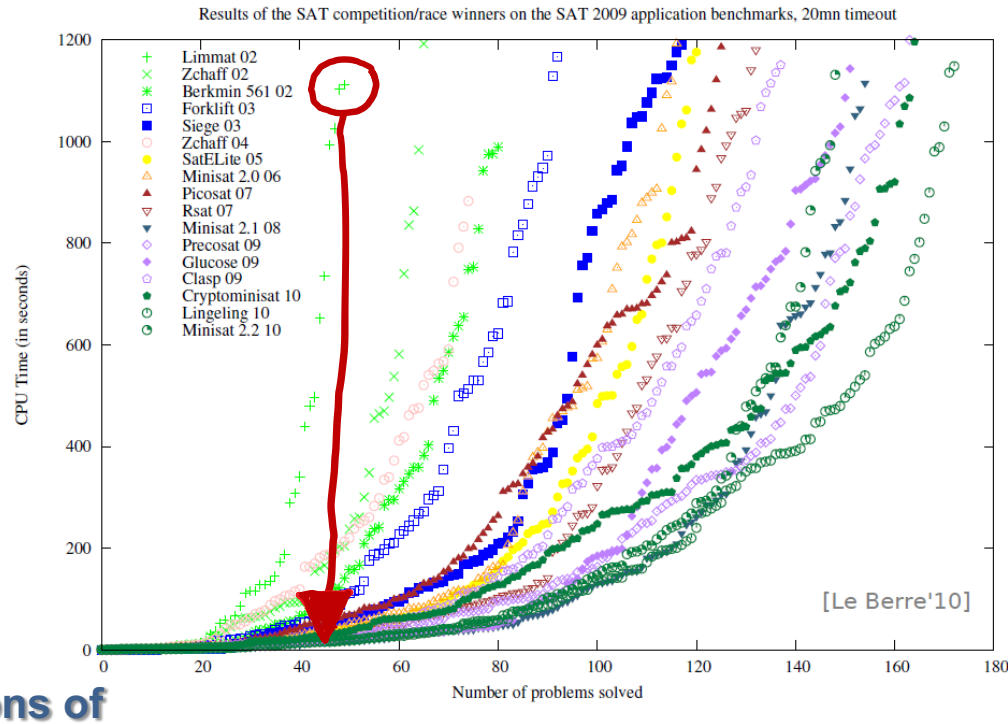
# SAT - Milestones

Problems impossible 10 years ago are trivial today

| year | Milestone |
|---|---|
| 1960 | Davis-Putnam procedure |
| 1962 | Davis-Logeman-Loveland |
| 1984 | Binary Decision Diagrams |
| 1992 | DIMACS SAT challenge |
| 1994 | SATO: clause indexing |
| 1997 | GRASP: conflict clause learning |
| 1998 | Search Restarts |
| 2001 | zChaff: 2-watch literal, VSIDS |
| 2005 | Preprocessing techniques |
| 2007 | Phase caching |
| 2008 | Cache optimized indexing |
| 2009 | In-processing, clause management |
| 2010 | Blocked clause elimination |

**Concept**

2002 2010




**Millions of variables from HW designs**

Courtesy Daniel le Berre