Propositional Logic

Methods & Tools for Software Engineering (MTSE)
Fall 2018

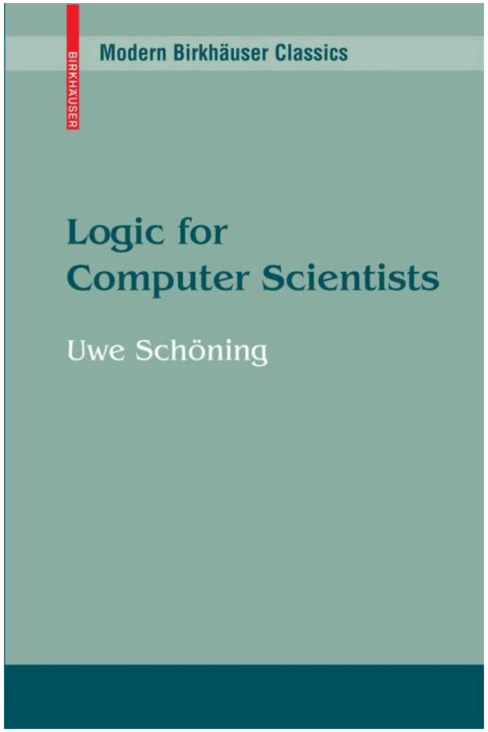
Alireza Sharifi

Slides from Prof. Arie Gurfinkel

UNIVERSITY OF WATERLOO

References

- Chpater 1 of Logic for Computer Scientists
  http://www.springerlink.com/content/978-0-8176-4762-9/



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 build 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”

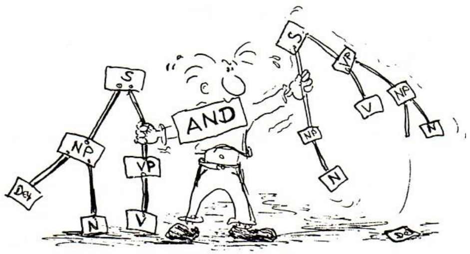
The subject of propositional logic is to declare formally the truth of complex structures from the truth of individual atomic components

A and B

A or B

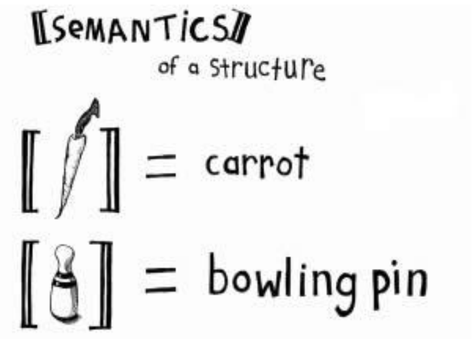
if A then B

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, $\neg F$ (called not F) is a formula
- For all formulas F and G, $F \wedge G$ (called and) and $F \vee G$ (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)

Syntax of Propositional Logic (PL)

$$
\begin{aligned}
\text{truth\_symbol} &::= \top(\text{true}) \mid \bot(\text{false}) \\
\text{variable} &::= p, q, r, \ldots \\
\text{atom} &::= \text{truth\_symbol} \mid \text{variable} \\
\text{literal} &::= \text{atom} | \neg \text{atom} \\
\text{formula} &::= \text{literal} \mid \\
&\quad\; \neg\text{formula} \mid \\
&\quad\; \text{formula} \land \text{formula} \mid \\
&\quad\; \text{formula} \lor \text{formula} \mid \\
&\quad\; \text{formula} \rightarrow \text{formula} \mid \\
&\quad\; \text{formula} \leftrightarrow \text{formula}
\end{aligned}
$$

Example

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

Sub-formulas are

$$\begin{aligned} F, ((A_5 \wedge A_6) \vee \neg A_3), \\ A_5 \wedge A_6, \neg A_3, \\ A_5, A_6, A_3 \end{aligned}$$

Semantics of propositional logic

Truth values: {0, 1}

D is any subset of the atomic formulas

An assignment **A** is a map **D** $\rightarrow$ {0, 1}

E $\supseteq$ **D** set of formulas built from **D**

An extended assignment **A**': **E** $\rightarrow$ {0, 1} is defined on the next slide

Semantics of propositional logic

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

$$\mathbf{A'}((F \wedge G)) = \begin{cases} 1 & \text{if } \mathbf{A'}(F) = 1 \text{ and } A'(G) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{A'}((F \vee G)) = \begin{cases} 1 & \text{if } \mathbf{A'}(F) = 1 \text{ or } A'(G) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{A'}(\neg F) = \begin{cases} 1 & \text{if } \mathbf{A'}(F) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Example

$$F = \neg(A \land B) \lor C$$

$$\mathcal{A}(A) = 1$$

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

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

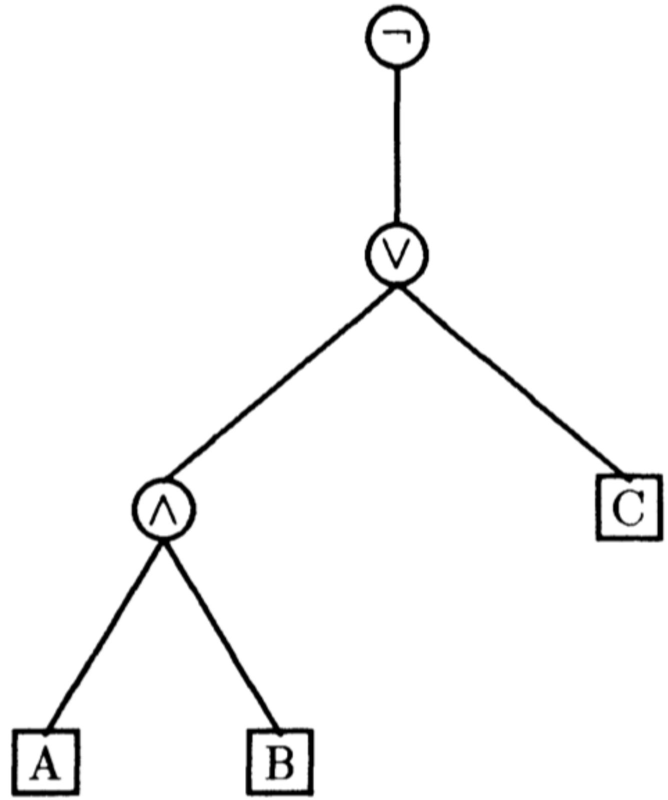
Truth Tables for Basic Operators

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

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

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

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



$$\mathcal{A}(A) = 1$$

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

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

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$\models$ F, iff

- A is suitable for F
- A(F) = 1, i.e., F *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 $\models$ 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	$\mathcal{A}_{2^n}(F)$

UNIVERSITY OF WATERLOO

An example

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

A	B	$\neg A$	$(A \to B)$	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 $\neg$F is unsatifsiable

Proof:

F is valid $\Leftrightarrow$ every suitable assignment for F is a model for F

$\Leftrightarrow$ every suitable assignment for $\neg$ F is not a model for $\neg$ F

$\Leftrightarrow$ $\neg$ F does not have a model

$\Leftrightarrow$ $\neg$ F is unsatisfiable

Exercise 10

Prove or give a counterexample

(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

(c) If (F->G) is valid and F is sat, then G is sat

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.

Proof:

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

Mathematical Induction

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

1. Show that P(0) is true

2. Show that P(k+1) is true for some natural number k, using an Inductive Hypothesis that P(k) is true

Example: Mathematical Induction

Show by induction that P(n) is true

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

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

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

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

UNIVERSITY OF WATERLOO

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

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 $(\neg F)$

UNIVERSITY OF WATERLOO

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.

Proof: by induction on formula structure

(base case) if H is atomic, then $F = H$, $H' = G$, and $F \equiv G$

(inductive step)

(case 1) $H = \neg H_1$

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

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

UNIVERSITY OF WATERLOO

Useful Equivalences (1/ 2)

$$
\begin{array}{rcll}
(F \wedge F) & \equiv & F & \\
(F \vee F) & \equiv & F & \text{(Idempotency)} \\[1em]
(F \wedge G) & \equiv & (G \wedge F) & \\
(F \vee G) & \equiv & (G \vee F) & \text{(Commutativity)} \\[1em]
((F \wedge G) \wedge H) & \equiv & (F \wedge (G \wedge H)) & \\
((F \vee G) \vee H) & \equiv & (F \vee (G \vee H)) & \text{(Associativity)} \\[1em]
(F \wedge (F \vee G)) & \equiv & F & \\
(F \vee (F \wedge G)) & \equiv & F & \text{(Absorption)} \\[1em]
(F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) & \\
(F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) & \text{(Distributivity)} \\[1em]
\neg\neg F & \equiv & F & \text{(Double Negation)}
\end{array}
$$

Useful Equivalences (2/ 2)

$$\begin{array}{rcll}
\neg(F \wedge G) & \equiv & (\neg F \vee \neg G) & \\
\neg(F \vee G) & \equiv & (\neg F \wedge \neg G) & \text{(deMorgan's Laws)} \\
 & & & \\
(F \vee G) & \equiv & F, \text{ if } F \text{ is a tautology} & \\
(F \wedge G) & \equiv & G, \text{ if } F \text{ is a tautology} & \text{(Tautology Laws)} \\
 & & & \\
(F \vee G) & \equiv & G, \text{ if } F \text{ is unsatisfiable} & \\
(F \wedge G) & \equiv & F, \text{ if } F \text{ is unsatisfiable} & \text{(Unsatisfiability Laws)}
\end{array}$$

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
- 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

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?

UNIVERSITY OF WATERLOO

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

Normal Forms: CNF and DNF

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 is in *Conjunctive Normal Form* (CNF) if it is a conjunction of disjunctions of literals (i.e., a conjunction of clauses):

- e.g., (v1 || ~v2) && (v3 || v2)

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

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

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

Normal Form Theorem

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

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

Converting a formula to CNF

Given a formula F

1. Substitute in F every occurrence of a sub-formula of the form

   $\neg\neg G$ by G

   $\neg(G \wedge H)$ by $(\neg G \vee \neg H)$

   $\neg(G \vee H)$ by $(\neg G \wedge \neg H)$

   This is called Negation Normal Form (NNF)

2. Substitute in F each occurrence of a sub-formula of the form

   $(F \vee (G \wedge H))$ by $((F \vee G) \wedge (F \vee H))$

   $((F \wedge G) \vee H)$ by $((F \vee H) \wedge (G \vee H))$

The resulting formula F is in CNF

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

From Truth Table to CNF and DNF

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

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

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

UNIVERSITY OF WATERLOO

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 whether 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 $A_1 \wedge \ldots \wedge A_n$ -> 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 an suitable assignment S such that S(Ai) = 1 iff Ai is marked. Return SAT with a satisfying assignment S.

UNIVERSITY OF WATERLOO

Exercise 21

Apply Horn satisfiability algorithm on a formula

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

$$\neg E$$

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

$$C$$

$$B$$

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

$$G$$

UNIVERSITY OF WATERLOO

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

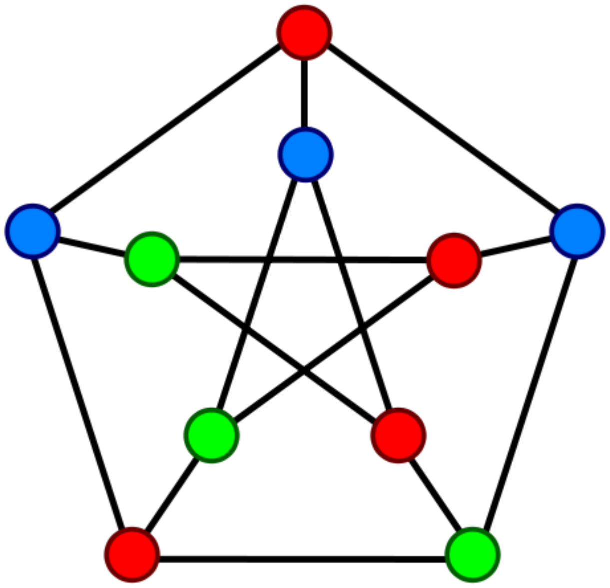
$$(\ell_0 \vee \cdots \vee \ell_n)$$

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

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

UNIVERSITY OF WATERLOO

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

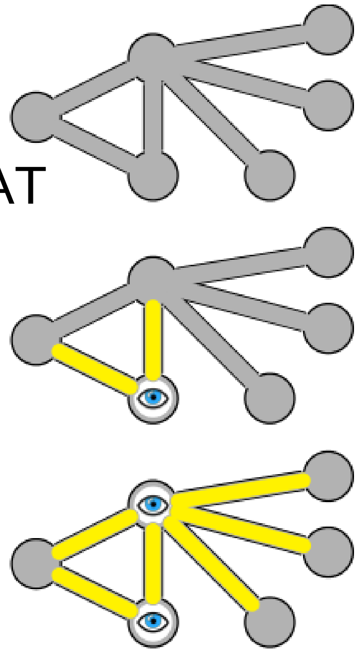
$$\bigwedge_{0 \leq i < k} (\neg f_{v,i} \vee \neg f_{u,i}) \qquad ((v,u) \in E)$$

UNIVERSITY OF WATERLOO

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



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.

Propositional Resolution

Pivot

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

Resolvent

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

Lemma:

Let F be a CNF formula. Let R be a resolvent of two clauses X and Y in F. Then, $F \cup \{R\}$ is equivalent to F

UNIVERSITY OF WATERLOO

Resolution Theorem

Let F be a set of clauses

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

$$Res^0(F) = F$$

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

$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F)$$

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

Exercise from LCS

For the following set of clauses determine Res^n for n=0, 1, 2

$$A \lor \neg B \lor C$$

$$B \lor C$$

$$\neg A \lor C$$

$$B \lor \neg C$$

$$\neg C$$

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.

(Completeness) By induction on the number of different atomic propositions in F.

Base case is trivial: F contains an empty $_{\text{clause}}$.

IH: Assume F has atomic propositions A1, … 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

Apply IH to F_0 and F_1 . Restore replaced literals. Combine the two resolutions.

UNIVERSITY OF WATERLOO

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

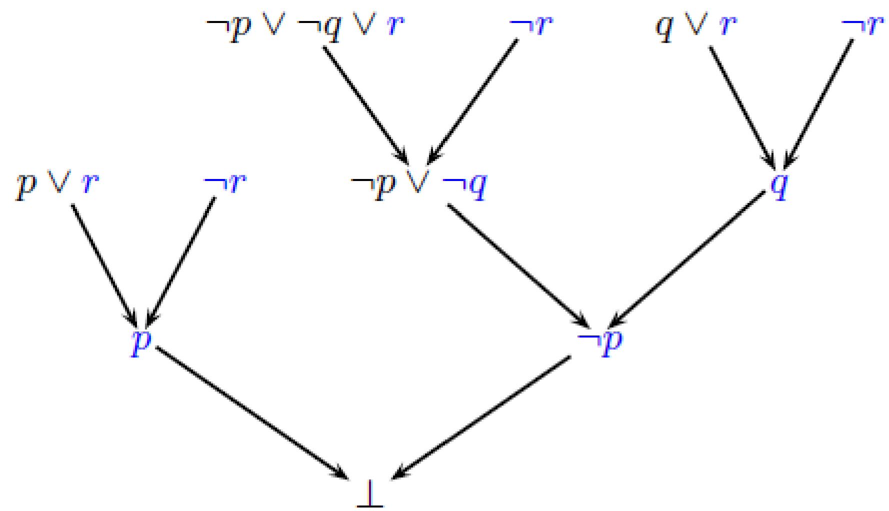
$$\frac{C \lor p \qquad\qquad D \lor \neg p}{C \lor D}$$

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

$$\neg b \land (\neg a \lor b \lor \neg c) \land a \land (\neg a \lor c)$$

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

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

$$F \vdash_P G$$

UNIVERSITY OF WATERLOO

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

Exercise 33

Using resolution show that

$$A \wedge B \wedge C$$

is a consequence of

$$\neg A \vee B$$

$$\neg B \vee C$$

$$A \vee \neg C$$

$$A \vee B \vee C$$

UNIVERSITY OF WATERLOO

Exercise 34

Show using resolution that F is valid

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

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

UNIVERSITY OF WATERLOO

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

CNF Examples

CNF 1

- ~b
- ~a || ~b || ~c
- a
- sat: s(a) = True; s(b) = False; s(c) = False

CNF 2

- ~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

http://cacm.acm.org/magazines/2009/8/34498-boolean-satisfiability-from-theoretical-h

Boolean Satisfiability: From ...

Find: currency Previous Next Options

TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS

ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

SIGN IN

COMMUNICATIONS OF THE ACM

Search

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | MAGAZINE ARCHIVE

Home / Magazine Archive / August 2009 (Vol. 52, No. 8) / Boolean Satisfiability: From Theoretical Hardness... / Full Text

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

Comments

VIEW AS: SHARE:

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 composition, or finding a plan for a robot to reach a goal that is

SIGN IN for Full Access

User Name

Password

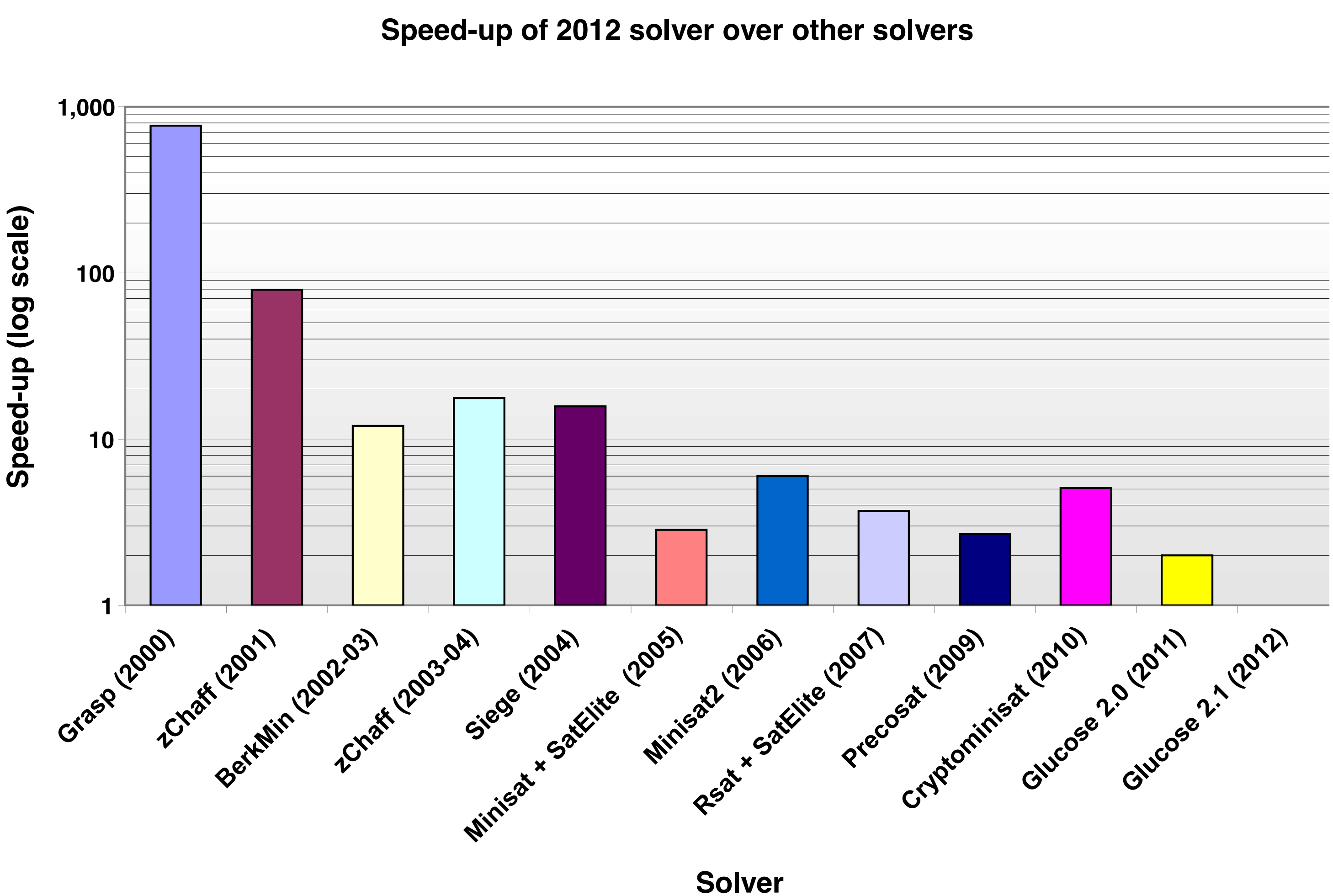
» Forgot Password?

» Create an ACM Web Account

SIGN IN

ARTICLE CONTENTS:

Some Experience with SAT Solving



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

UNIVERSITY OF WATERLOO

SAT - Milestones

Problems impossible 10 years ago are trivial today

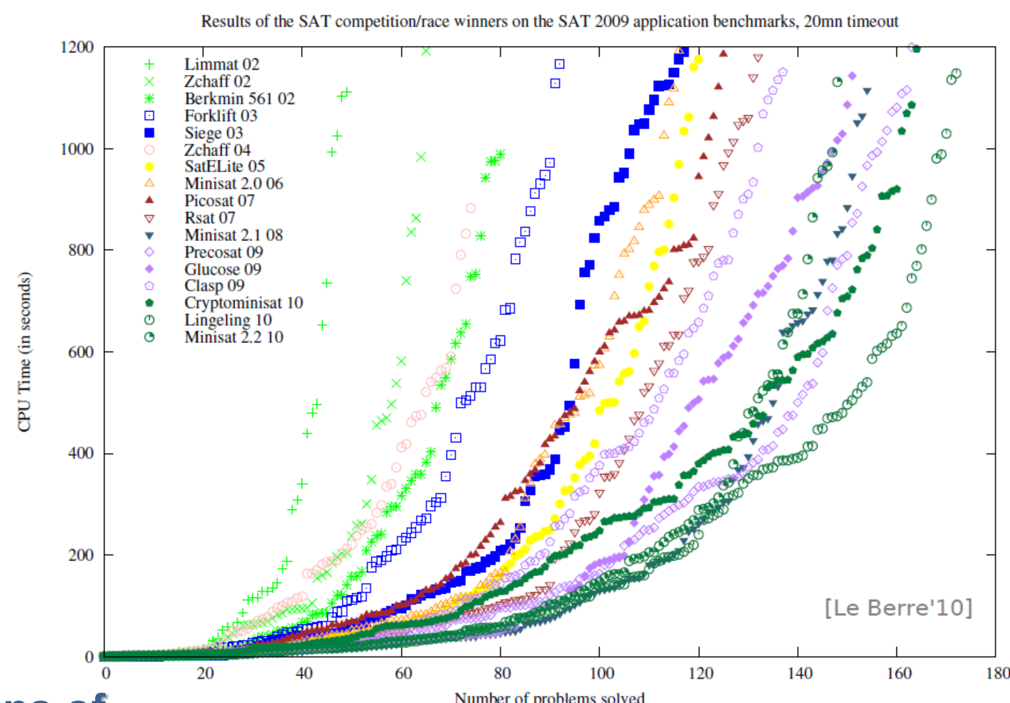
year	Milestone
1960	Davis-Putnam procedure
1962	Davis-Logeman-Loveland
1984	Binary Decision Diagrams
1992	DIMACS SAT challenge
1994	SATO: clause indexing
1997	GRASP: conflict clause learning
1998	Search Restarts
2001	zChaff: 2-watch literal, VSIDS
2005	Preprocessing techniques
2007	Phase caching
2008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

Concept

Millions of variables from HW designs

2002

2010



Courtesy Daniel le Berre