Propositional Logic

Methods & Tools for Software Engineering (MTSE)
Fall 2019

Prof. Arie Gurfinkel
References

• Chapter 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”

\[
A = \text{“Paris is the capital of France”} \\
B = \text{“mice chase elephants”}
\]

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 $i = 1, 2, 3 \ldots$

*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 $G$, $F \land G$ (called and) and $F \lor G$ (called or) are formulas

*Abbreviations*

- use $A, B, C, \ldots$ instead of $A_1, A_2, \ldots$
- use $F_1 \rightarrow F_2$ instead of $\neg F_1 \lor F_2$ (implication)
- use $F_1 \leftrightarrow F_2$ instead of $(F_1 \rightarrow F_2) \land (F_2 \rightarrow F_1)$ (iff)
Syntax of Propositional Logic (PL)

\[
\begin{align*}
\text{truth\_symbol} & \ ::= \top (\text{true}) \mid \bot (\text{false}) \\
\text{variable} & \ ::= p, q, r, \ldots \\
\text{atom} & \ ::= \text{truth\_symbol} \mid \text{variable} \\
\text{literal} & \ ::= \text{atom} \mid \neg \text{atom} \\
\text{formula} & \ ::= \text{literal} \mid \\
& \quad \neg \text{formula} \mid \\
& \quad \text{formula} \land \text{formula} \mid \\
& \quad \text{formula} \lor \text{formula} \mid \\
& \quad \text{formula} \rightarrow \text{formula} \mid \\
& \quad \text{formula} \leftrightarrow \text{formula}
\end{align*}
\]
Example

\[ F = \neg((A_5 \land A_6) \lor \neg A_3) \]

Sub-formulas are

\[ F, ((A_5 \land A_6) \lor \neg A_3), A_5 \land A_6, \neg A_3, A_5, A_6, A_3 \]
We 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 \( D \rightarrow \{0, 1\} \)
- \( A \) assigns True or False to every atomic in \( D \)

Let \( E \supseteq D \) be set of formulas built from \( D \) using propositional connectives

*Extended assignment* \( A' : E \rightarrow \{0, 1\} \) extends \( A \) from atomic formulas to all formulas
- continued on the next slide
Semantics of propositional logic

For an atomic formula $A_i$ in $D$: $A'(A_i) = A(A_i)$

$A'(F \land G) = 1$ if $A'(F) = 1$ and $A'(G) = 1$
$= 0$ otherwise

$A'(F \lor G) = 1$ if $A'(F) = 1$ or $A'(G) = 1$
$= 0$ otherwise

$A'(?F) = 1$ if $A'(F) = 0$
$= 0$ otherwise
Exercise: Define Extended Assignment

\[ F = \neg (A \land B) \lor C \]

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

Is \( F \) true or false under \( A' \)?
# Truth Tables for Basic Operators

<table>
<thead>
<tr>
<th>$A(F)$</th>
<th>$A(G)$</th>
<th>$A'(F \land G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A(F)$</th>
<th>$A(G)$</th>
<th>$A'(F \lor G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$A(F)$</th>
<th>$A'(\neg F)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Formula

\[ F = \neg (A \land B) \lor C \]

Assignment

\[ \mathcal{A}(A) = 1 \]
\[ \mathcal{A}(B) = 1 \]
\[ \mathcal{A}(C) = 0 \]
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., $F$ 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 $\models F$, 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.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$\cdots$</th>
<th>$A_{n-1}$</th>
<th>$A_n$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$A_1(F)$</td>
</tr>
<tr>
<td>$A_2$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A_2(F)$</td>
</tr>
<tr>
<td>$A_{2^n}$:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$A_{2^n}(F)$</td>
</tr>
</tbody>
</table>
Problem: Is formula $F$ SAT?

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

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$\neg A$</th>
<th>$(A \rightarrow B)$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Validity and Unsatisfiability

**Theorem:**
A formula $F$ is valid if and only if $\neg F$ is unsatisfiable

**Proof:**
- $F$ is valid $\iff$ every suitable assignment for $F$ is a model for $F$
  $\iff$ every suitable assignment for $\neg F$ is not a model for $\neg F$
  $\iff \neg F$ does not have a model
  $\iff \neg F$ is unsatisfiable
Prove or give a counterexample

(a) If \((F \Rightarrow G)\) is valid and \(F\) is valid, then \(G\) is valid

(b) If \((F \Rightarrow G)\) is sat and \(F\) is sat, then \(G\) is sat

(c) If \((F \Rightarrow G)\) is valid and \(F\) is sat, then \(G\) is sat
Semantic Equivalence

Two formulas $F$ and $G$ are \textit{(semantically) equivalent}, written $F \equiv G$, iff for every assignment $A$ that is suitable for both $F$ and $G$, $A'(F) = A'(G)$

For example, $(F \land G)$ is equivalent to $(G \land F)$

Formulas with different atomic propositions can be equivalent
  \begin{itemize}
  \item e.g., all tautologies are equivalent to True
  \item e.g., all unsatisfiable formulas are equivalent to False
  \end{itemize}
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 (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)

\[
0 + \cdots + k + (k + 1) = \frac{k(k+1)}{2} + (k+1)
\]
\[
= \frac{k(k+1)+2(k+1)}{2}
\]
\[
= \frac{(k+1)((k+1)+1)}{2}
\]

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 $(F \land G)$, $(F \lor G)$, 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 $F \equiv G$
(inductive step)

(case 1) $H = \neg H_1$

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

(case 3) $H = H_1 \lor H_2$
Useful Equivalences (1/ 2)

\[(F \land F) \equiv F\]
\[(F \lor F) \equiv F\]  \hspace{1cm} \text{(Idempotency)}

\[(F \land G) \equiv (G \land F)\]
\[(F \lor G) \equiv (G \lor F)\]  \hspace{1cm} \text{(Commutativity)}

\[((F \land G) \land H) \equiv (F \land (G \land H))\]
\[((F \lor G) \lor H) \equiv (F \lor (G \lor H))\]  \hspace{1cm} \text{(Associativity)}

\[(F \land (F \lor G)) \equiv F\]
\[(F \lor (F \land G)) \equiv F\]  \hspace{1cm} \text{(Absorption)}

\[(F \land (G \lor H)) \equiv ((F \land G) \lor (F \land H))\]
\[(F \lor (G \land H)) \equiv ((F \lor G) \land (F \lor H))\]  \hspace{1cm} \text{(Distributivity)}

\[\neg\neg F \equiv F\]  \hspace{1cm} \text{(Double Negation)}
Useful Equivalences (2/2)

\[-(F \land G) \equiv (\neg F \lor \neg G)\]
\[-(F \lor G) \equiv (\neg F \land \neg G)\]  
(\text{deMorgan’s Laws})

\[(F \lor G) \equiv F, \text{ if } F \text{ is a tautology}\]
\[(F \land G) \equiv G, \text{ if } F \text{ is a tautology}\]  
(Tautology Laws)

\[(F \lor G) \equiv G, \text{ if } F \text{ is unsatisfiable}\]
\[(F \land G) \equiv F, \text{ if } F \text{ is unsatisfiable}\]  
(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

  \[ ((T \lor C) \land R) \Rightarrow D \]

• 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

  \[ (R \Rightarrow (T \Rightarrow D)) \land (C \Rightarrow (R \Rightarrow D)) \]
((T \lor C) \land R) \Rightarrow D
( (T \land R) \lor (C \land R) \Rightarrow D)
((T \land R) \Rightarrow D) \land ((C \land R) \Rightarrow D)
(R \Rightarrow (T \Rightarrow D)) \land (C \Rightarrow (R \Rightarrow D))

Law:
(a \lor b) \Rightarrow c
(a \Rightarrow c) \land (b \Rightarrow c)
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.
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
Normal Form: DNF

A literal is either an atomic proposition \( v \) or its negation \( \neg v \).

A cube is a conjunction of literals

- E.g., \((v1 \land \neg v2 \land v3)\)

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 \( \neg v \)

A clause is a disjunction of literals
- e.g., \((v_1 \lor \neg v_2 \lor v_3)\)

A formula \( F \) is in Conjunction 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. For every formula $F$, there is 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 $F$)
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 \( G \)
   \( \neg (G \land H) \) by \( \neg G \lor \neg H \)
   \( \neg (G \lor H) \) by \( \neg G \land \neg H \)
   This is called Negation Normal Form (NNF)

2. Substitute in F each occurrence of a sub-formula of the form
   \( (F \lor (G \land H)) \) by \( ((F \lor G) \land (F \lor H)) \)
   \( ((F \land G) \lor H) \) by \( ((F \lor H) \land (G \lor H)) \)

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

**Truth table**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**DNF**

\[
(-A \land -B \land -C) \lor \\
(A \land -B \land -C) \lor \\
(A \land -B \land C)
\]

**CNF**

\[
(A \lor B \lor -C) \land \\
(A \lor -B \lor C) \land \\
(A \lor -B \lor -C) \land \\
(-A \lor -B \lor C) \land \\
(-A \lor -B \lor -C)
\]

see the book for detailed algorithm
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

$$\left( A \lor \neg B \right) \land \left( \neg C \lor \neg A \lor D \right) \land \left( \neg A \lor \neg B \right) \land D \land \neg E$$

- Note that each clause can be written as an implication
  - e.g. $C \land A \Rightarrow D$, $A \land B \Rightarrow \text{False}$, $\text{True} \Rightarrow D$

$$\left( B \to A \right) \land \left( A \land C \to D \right) \land \left( A \land B \to 0 \right) \land \left( 1 \to D \right) \land \left( E \to 0 \right)$$

**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 $A_1 \land \ldots \land A_n \rightarrow B$ 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 an suitable assignment $S$ such that $S(A_i) = 1$ iff $A_i$ is marked. Return SAT with a satisfying assignment $S$. 
Exercise 21

Apply Horn satisfiability algorithm on a formula

\[(\neg A \lor \neg B \lor \neg D) \land \neg E \land (\neg C \lor A) \land C \land B \land (\neg G \lor D) \land G\]
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

$$(\ell_0 \lor \cdots \lor \ell_n)$$

with 3-literal clauses

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

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 VERY HARD problem!
Background Reading: SAT

COMMUNICATIONS OF THE ACM

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 composition, or finding a plan for a robot to reach a goal that is
Some Experience with SAT Solving

Speed-up of 2012 solver over other solvers

### SAT - Milestones

Problems impossible 10 years ago are trivial today

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

2002 - 2010

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20min timeout

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 $f : V \rightarrow [0..k)$ such that
- for every edge $(u, v)$ in $E$, $f(u) \neq f(v)$

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](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

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

No vertex is assigned two colors

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

No two adjacent vertices have the same color

$$\bigwedge_{0 \leq i < k} (\neg f_{v,i} \lor \neg f_{u,i}) \quad ((v, u) \in E)$$
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?

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

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

\[
\begin{align*}
C \lor p & \quad \quad \quad D \lor \neg p \\
\hline
C \lor D
\end{align*}
\]

\[
\text{Res}([C, p], [D, \neg p]) = [C, D]
\]
Resolution 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.

• i.e., R is implied by F. Adding it to F does not change the meaning of F
Resolution Theorem

Let $F$ be a set of clauses

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

Define $Res^n$ 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 $n=0, 1, 2$

\[
\begin{align*}
A & \lor \neg B \lor C \\
B & \lor C \\
\neg A & \lor C \\
B & \lor \neg C \\
\neg C & 
\end{align*}
\]
Proof of the Resolution Theorem

(Soundness) By Resolution Lemma, $F$ is equivalent to $\text{Res}^i(F)$ for any $i$. Let $n$ be such that $\text{Res}^{n+1}(F)$ contains an empty clause, but $\text{Res}^n(F)$ does not. Then $\text{Res}^n(F)$ must contain to unit clauses $L$ and $\neg L$. Hence, it is UNSAT.

(Completeness) By induction on the number of different atomic propositions in $F$.
Base case is trivial: $F$ contains an empty clause.
IH: Assume $F$ 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
Apply IH to $F_0$ and $F_1$. Restore replaced literals. Combine the two resolutions.
An inference rule is a tuple \((P_1, \ldots, P_n, C)\)

- where, \(P_1, \ldots, P_n, C\) 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 \(n\), \((\text{parents}(n), n)\) is an inference rule in \(P\)
Propositional Resolution

\[ \frac{C \lor p \quad D \lor \neg p}{C \lor D} \]

Propositional resolution is a sound inference rule

Propositional resolution system consists of a single propositional resolution rule
Example of a resolution proof

A refutation of \( \neg p \lor \neg q \lor r, p \lor r, q \lor r, \neg r \):
Resolution Proof Example

Show by resolution that the following CNF is UNSAT

\[ \neg b \land (\neg a \lor b \lor \neg c) \land a \land (\neg a \lor c) \]

\[ \begin{array}{c|c|c|c}
\neg a & b & \neg c & a \\
\hline
b & \neg c & b & \neg a \lor c \\
\hline
\neg c & c & \\
\hline
\end{array} \]
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

\[
(F \not\vdash_P G) \implies (F \models G)
\]

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
Using resolution show that

\[ A \land B \land C \]

is a consequence of

\[ \neg A \lor B \]
\[ \neg B \lor C \]
\[ A \lor \neg C \]
\[ A \lor B \lor C \]
Exercise 34

Show using resolution that $F$ is valid

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

$$\neg F = (B \lor C \lor \neg D) \land (B \lor D) \land (\neg C \lor \neg D) \land \neg B$$
Boolean Satisfiability (CNF-SAT)

Let $V$ be a set of variables

A literal is either a variable $v$ in $V$ or its negation $\neg v$

A clause is a disjunction of literals

- e.g., $(v_1 \lor \neg v_2 \lor v_3)$

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

- e.g., $(v_1 \lor \neg v_2) \land (v_3 \lor v_2)$

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)
- \( \neg b \)
- \( \neg a \lor \neg b \lor \neg c \)
- a
- SAT: \( s(a) = \text{True}; \ s(b) = \text{False}; \ s(c) = \text{False} \)

CNF 2 (4 clauses)
- \( \neg b \)
- \( \neg a \lor b \lor \neg c \)
- a
- \( \neg a \lor c \)
- 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-Logemann-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
Background Reading: SAT

COMMUNICATIONS
OF THE
ACM

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 composition, or finding a plan for a robot to reach a goal that is
Some Experience with SAT Solving

Speed-up of 2012 solver over other solvers

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

Problems impossible 10 years ago are trivial today

**Concept**

2002 2010

![Graph showing SAT competition results](image)

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20 minutes timeout

**Millions of variables from HW designs**

Courtesy Daniel le Berre