Propositional Satisfiability

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

Presented by
Dr. Albert Wasef

UNIVERSITY OF WATERLOO

Explaining 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

Assume that F is unsatisfiable. What is a short proof / certificate to establish **UNSATISFIABILITY** without a doubt?

For example, is the following formula SAT or UNSAT? How do you explain your answer?

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

UNIVERSITY OF WATERLOO

From CNF to database of clauses

Assume that all propositional formulas are converted to CNF

Each clause is determined by the set of literals

- e.g., (a ∨ b ∨ ¬c) is same as {a, b, ¬c}

A CNF is a database (a set) of clauses

- (a ∨ b ∨ ¬c) ∧ (c) ∧ (¬ b ∧ d) is represented as
- { {a, b, ¬c}, {c}, {¬b, d} }

Propositional Resolution

Resolution is a simple syntactic transformation applied to formulas. From two given formulas in a resolution step, a third formula is generated.

A collection of such "mechanical" transformation rules we call a calculus.

In the case of resolution there is just one rule which is applied over and over again until a certain "goal formula" is obtained.

The definition of a calculus is sensible only if its correctness and its completeness can be established.

To be more precise in the case of the resolution calculus, the task is to prove unsatisfiability of a given formula.

Propositional Resolution

Correctness means that every formula for which the resolution calculus claims unsatisfiability indeed is unsatisfiable.

Completeness means that for every unsatisfiable formula there is a way to prove this by means of the resolution calculus.

Propositional Resolution

Let A be a clause of the form $C \vee p$

Let B be a clause of the form $D \vee \neg p$

Propositional Resolution:

A clause $(C \vee D)$ is a resolvent of A and B on pivot p

Propositional Resolution In Symbols

$$\mathrm{Res}(\{C, p\}, \{D, \neg p\}) = \{C, D\}$$

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

Notation: Inference Rule

Premise

Side-condition

$$\frac{F_1 \quad \ldots \quad F_n}{G} \quad H$$

Conclusion

Inference Rules

We express the evaluation rules as inference rules for our judgments.

The rules are also called evaluation rules.

An inference rule $\dfrac{F_1 \ldots F_n}{G}$ where H

defines a relation between judgments $F_1,\ldots,F_n$ and G.

- The judgments $F_1,\ldots,F_n$ are the premises of the rule;
- The judgments G is the conclusion of the rule;
- The formula H is called the side condition of the rule.

If n=0 the rule is called an axiom. In this case, the line separating premises and conclusion may be omitted.

UNIVERSITY OF WATERLOO

Propositional Resolution Inferenc

Pivot

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

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

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 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

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

$$B \vee C$$

$$\neg A \vee C$$

$$B \vee \neg C$$

$$\neg C$$

Resolution Proof Example

Show by resolution that the following CNF is UNSAT

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

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

Proof of the Resolution Theorem 1/3

(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

- such n must exist because an empty clause was added at some point

Then, $Res^n(F)$ must contain two unit clauses L and $\neg L$

- because the only way to construct an empty clause is to resolve two unit clauses

Hence, F is UNSAT

- every clause added by resolution is implied (entailed) by F
- hence, $F \rightarrow L$ and $F \rightarrow \neg L$
- Therefore, $F \rightarrow (L \land \neg L)$, and $F \rightarrow \text{False}$

Proof of the Resolution Theorem 2/3

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

(base case) if F has 0 atomic propositions and has a clause, then F contains an empty clause

- empty clause is the only clause without any atomic propositions

Proof of the Resolution Theorem 3/3

(inductive case):

Assume F is UNSAT and F has atomic propositions $A_1, \ldots A_{n+1}$

Let F_0 be the result of replacing atomic proposition A_{n+1} by 0, and for every occurrence of the negative literal $(\neg A_{n+1})$ within a clause, the entire clause is canceled.

Let F_1 be the result of replacing atomic proposition A_{n+1} by 1

Since F is UNSAT, so are F_0 and F_1

- e.g., if F_0 is SAT with assignment M, then extend M to $A_{n+1} \rightarrow 0$, …

By IH, both F_0 and F_1 derive an empty clause

- Hence, Res*(F) contains (A_{n+1}) (or empty clause) and Res*(F) contains $(\neg A_{n+1})$ (or empty clause)

Therefore, Res*(F) contains an empty clause!

Example for the last step of Pf of Res Theorem

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

$F_0 = (a) \wedge (\neg a) \wedge (\neg c)$

- Res*(F_0) contains an empty clause
- By following the same resolution steps in F, we show that Res*(F) contains the clause (b)

$F_1 = (a) \wedge (c) \wedge (\neg c)$

- Res*(F_1) contains an empty clause
- By following the same resolution steps in F, we show that Res*(F) contains the clause (¬ b)

Therefore, Res*(F) contains an empty clause!

Proof System

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

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*, the tuple (parents(n), n) is an inference rule in P

Propositional Resolution as an Inference Rule

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

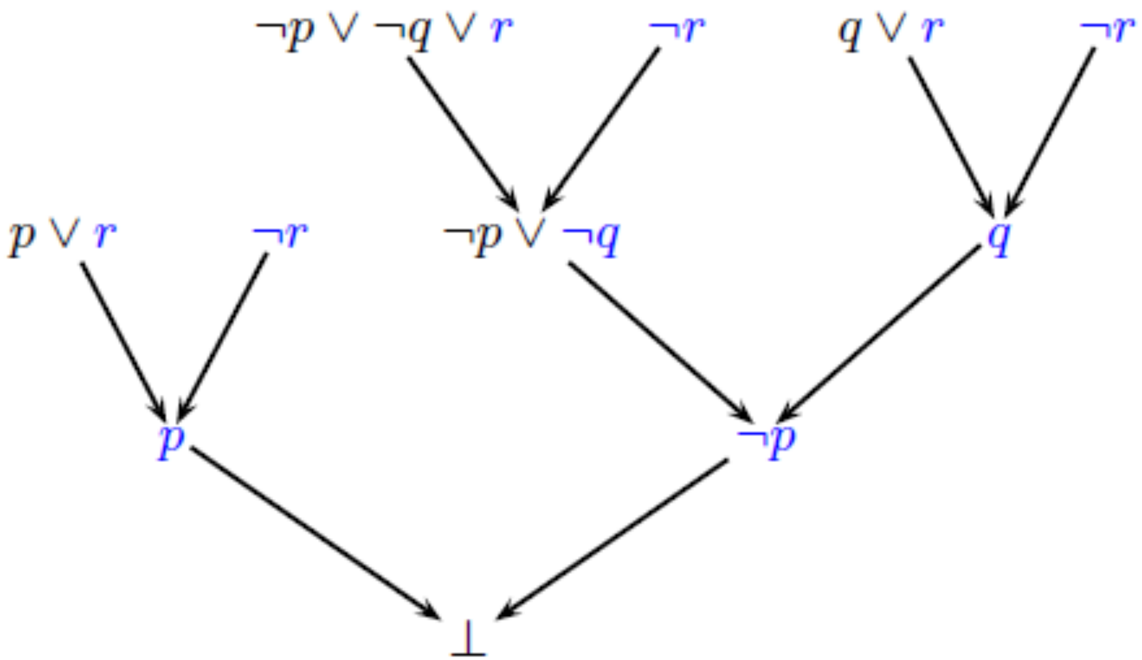
Propositional resolution is a sound inference rule

Proposition resolution proof system consists of a single propositional resolution rule

UNIVERSITY OF WATERLOO

A Resolution Proof Example

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



Another Resolution Pf Example

Show by resolution that the following CNF is UNSAT

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

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

Book: Exercise 33

Using resolution show that

$$A \wedge B \wedge C$$

is a consequence of

$$\neg A \vee B$$

$$\neg B \vee C$$

$$A \vee \neg C$$

$$A \vee B \vee C$$

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

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

PR: Soundness and Completeness

Theorem: Propositional resolution is sound and complete for propositional logic

Proof:

Follows immediately from the Resolution Theorem!

Exercise 34

Show using resolution that F is valid

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

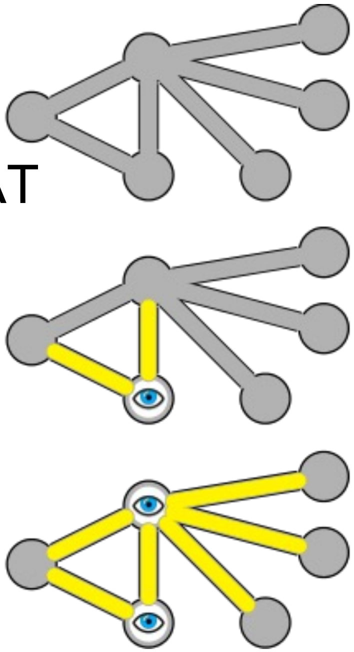
$$\neg F = (B \lor C \lor \neg D) \land (B \lor D) \land (\neg C \lor \neg D) \land \neg B$$

ENCODING PROBLEMS TO SAT

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 assignment4.pdf for details of reduction to CNF-SAT



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

https://git.uwaterloo.ca/ece650-f23/assignments_pdf/-/blob/master/a4_encoding.pdf

https://git.uwaterloo.ca/ece650-f23/assignments_pdf/-/blob/master/a4_example_cnf.txt

Vertex Cover

A cover is a list of k vertices: $c_1, \ldots, c_k$ such that

- every c_i corresponds to at least one vertex
- only one vertex corresponds to each c_i
- no vertex appears more than once in each c_i
- every edge (u, v) is adjacent to a vertex in the cover
  - i.e., either u is c_i or v is c_i for some i

Propositional Encoding

- Let n be the number of vertices
- Each c_i is represented by n Boolean variables
  - $x_{i,j}$ is true iff vertex i is at position j in the cover, i.e., $v_i = c_j$

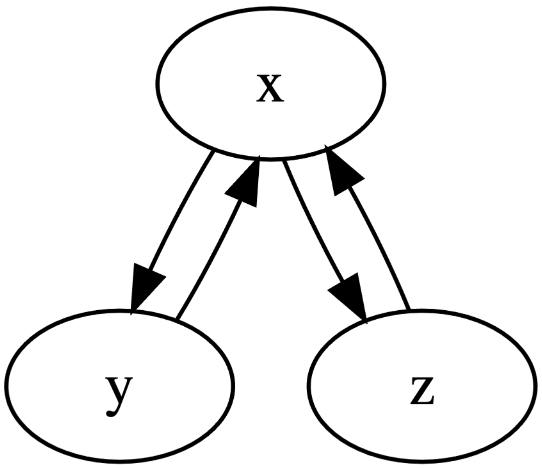
- See constraints in A4 handout

Is there a vertex cover of size 1?

Adjacency Matrix

$$\begin{array}{cccc} & x & y & z \\ x & 0 & 1 & 1 \\ y & 1 & 0 & 0 \\ z & 1 & 0 & 0 \end{array}$$

Graph



3 variables: x, y, z

$x \lor y \lor z$

$\neg x \lor \neg y$

$\neg x \lor \neg z$

$\neg y \lor \neg z$

$y \lor x$

$z \lor x$

sat assignment: x=1, y=0, z=0

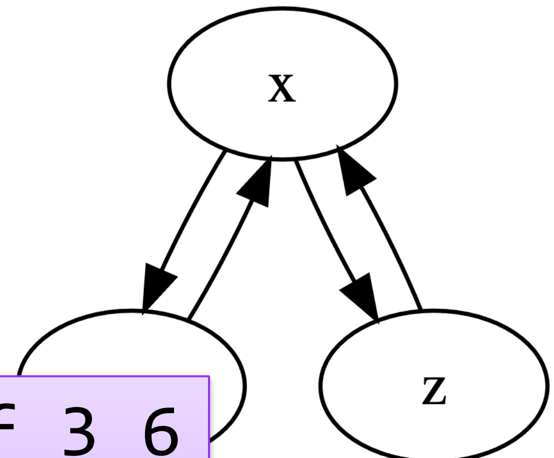
UNIVERSITY OF WATERLOO

Is there a vertex cover of size 1?

Adjacency Matrix

	x	y	z
x	0	1	1
y	1	0	0
z	1	0	0

Graph



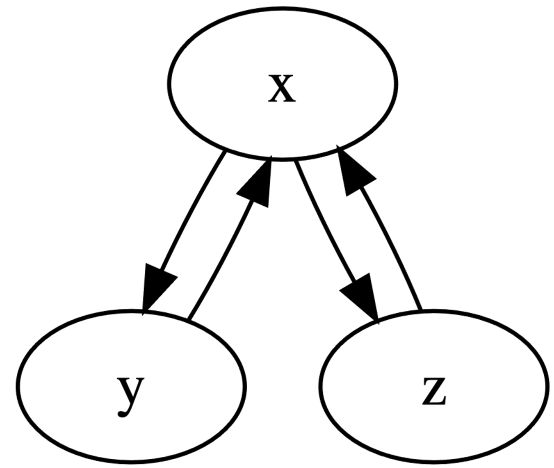
DIMACS Format:

```
p cnf 3 6
1 2 3 0
-1 -2 0
-1 -3 0
-2 -3 0
2 1 0
3 1 0
```

Is there a vertex cover of size 2?

6 variables: x_1, x_2 y_1, y_2, z_1, z_2

Graph



$x_1 \lor y_1 \lor z_1$

$x_2 \lor y_2 \lor z_2$

$\neg x_1 \lor \neg x_2$

$\neg y_1 \lor \neg y_2$

$\neg z_1 \lor \neg z_2$

$\neg x_1 \lor \neg y_1$ $\quad$ $\neg x_2 \lor \neg y_2$

$\neg x_1 \lor \neg z_1$ $\quad$ $\neg x_2 \lor \neg z_2$

$\neg y_1 \lor \neg z_1$ $\quad$ $\neg y_2 \lor \neg z_2$

$y_1 \lor x_1 \lor y_2 \lor x_2$

$z_1 \lor x_1 \lor z_2 \lor x_2$

https://git.uwaterloo.ca/ece650-f23/assignments_pdf/-/blob/master/a4_example_cnf.txt

SAT SOLVING ALGORITHMS

Algorithms for SAT

SAT is NP-complete

- solution can be checked in polynomial time
- no polynomial algorithms for finding a solution are known

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

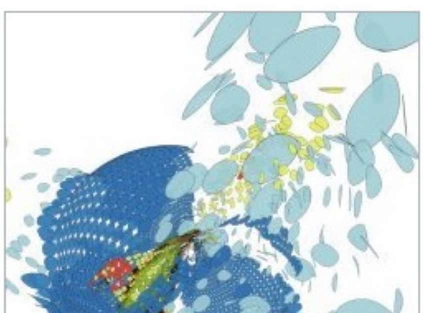
Boolean Satisfiability: From Theoretical Hardness to Practical Success

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

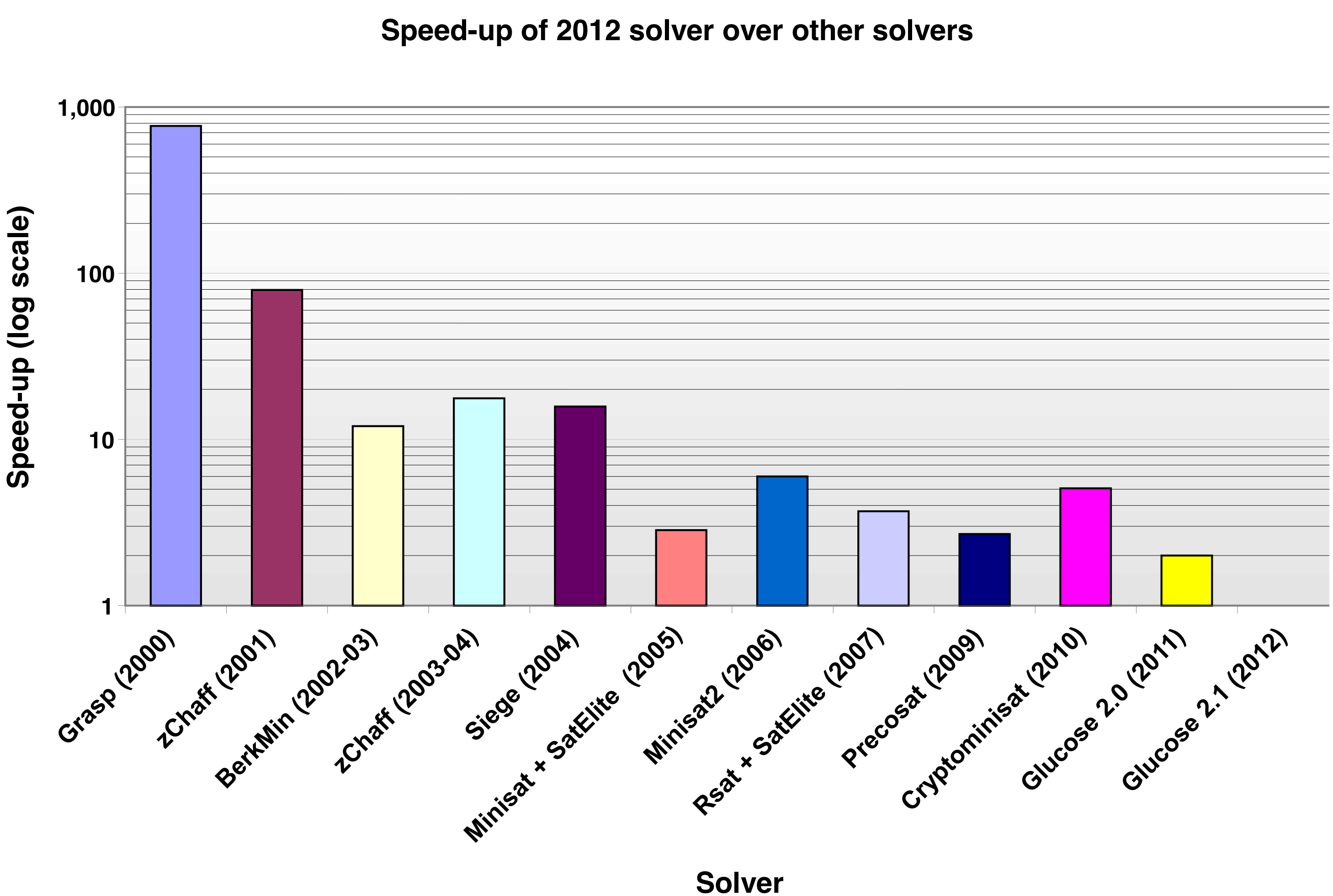
ARTICLE CONTENTS:

Introduction

Boolean Satisfiability

Theoretical hardness: SAT and NP-Completeness

Some Experience with SAT Solving



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

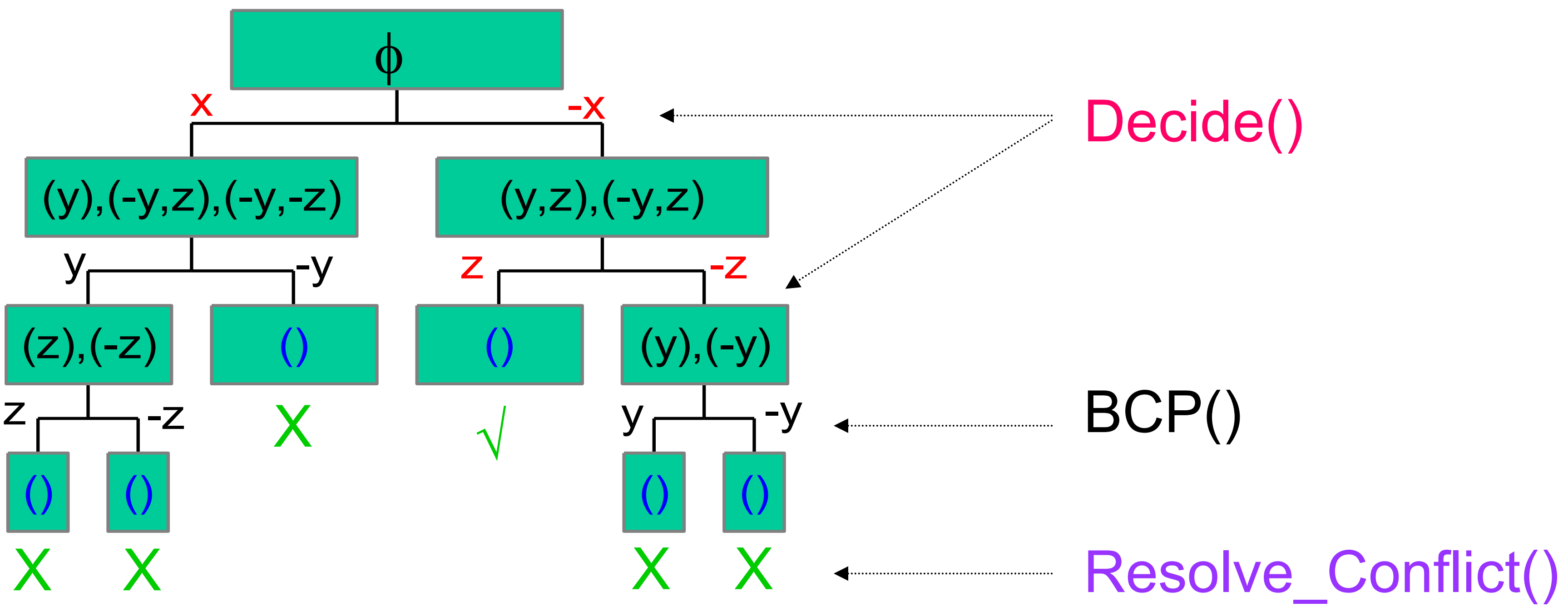
UNIVERSITY OF WATERLOO

Conflict-Driven Clause Learning

CDCL SAT SOLVER

A Basic SAT algorithm

Given ϕ in CNF: (x,y,z),(-x,y),(-y,z),(-x,-y,-z)



DPLL: Davis Putnam Logeman Loveland

DPLL is a combination of two rules

- **Split** – pick an atomic proposition p and try setting p to 0 and 1
- **Unit** – if there is a clause with a single literal p; set the variable accordingly; remove all clauses in which p is true, and erase ¬p from all clauses

split

$$\frac{F}{F, p \quad | \quad F, \neg P}$$

unit

$$\frac{F, C \vee p, \neg p}{F, C, \neg p}$$

Introduced in 1961. Still core of modern efficient SAT solvers

Basic CDCL Algorithm

CDCL: Conflict Driven Clause Learning
BCP: Boolean Constraint Propagation (

Choose the next variable and value. Return False if all variables are assigned

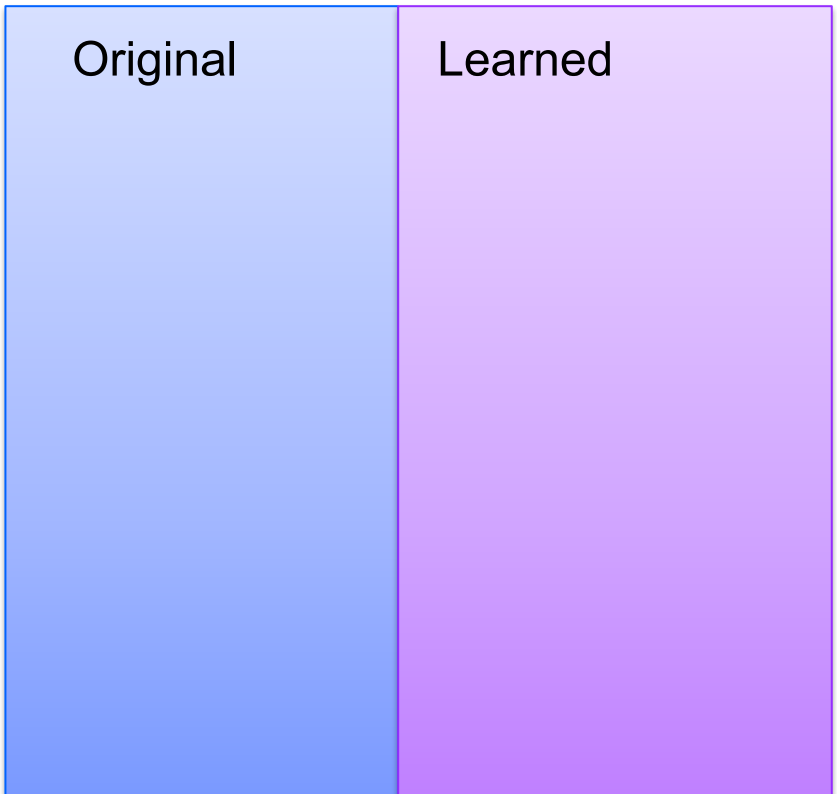
```
while (true)
{
   if (!Decide()) return (SAT);
   while (!BCP())
      if (!Resolve_Conflict()) return (UNSAT);
}
```

Apply repeatedly the *unit clause rule*. Return False if reached a conflict

Backtrack until no conflict. Return False if impossible

Architecture of a SAT Solver

Clause Database



Core of the SAT solver is a clause database

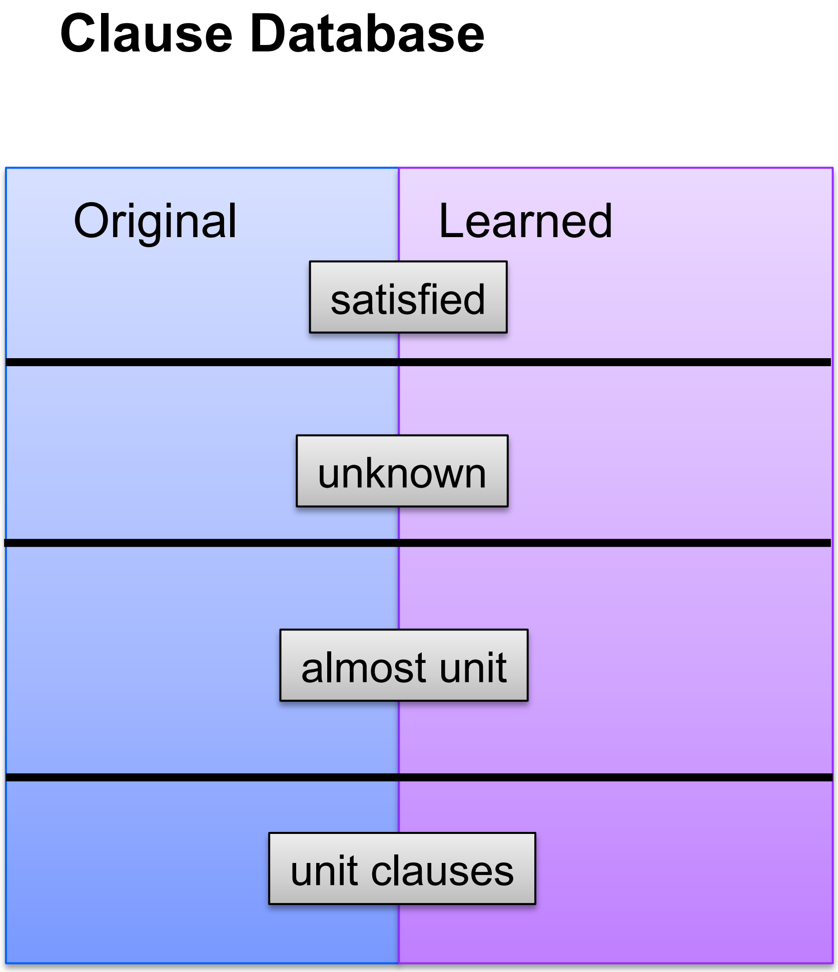
It is divided into **Original** and **Learned** clauses

Original clauses are clauses that are part of the initial problem

Learned clauses are all conflict clauses learned during current run

Each learned clause is implied by Original clauses

Architecture of a SAT Solver



The clauses in the database are classified based on their status in the current partial assignment

- Satisfied – are clauses that have at least one satisfied literal. They are inactive at the moment
- Unknown – clauses that have at least one unset literal
- Almost unit – clauses that have only two unset literals. They can become unit soon
- Unit clauses – clauses that are already unit and must be processed to extend the current partial assignment (i.e., clauses to be used for BCP)

Architecture of a SAT Solver

A partial assignment (called **trail**) keeps track of the current partial assignment and all decisions that have led to the assignment

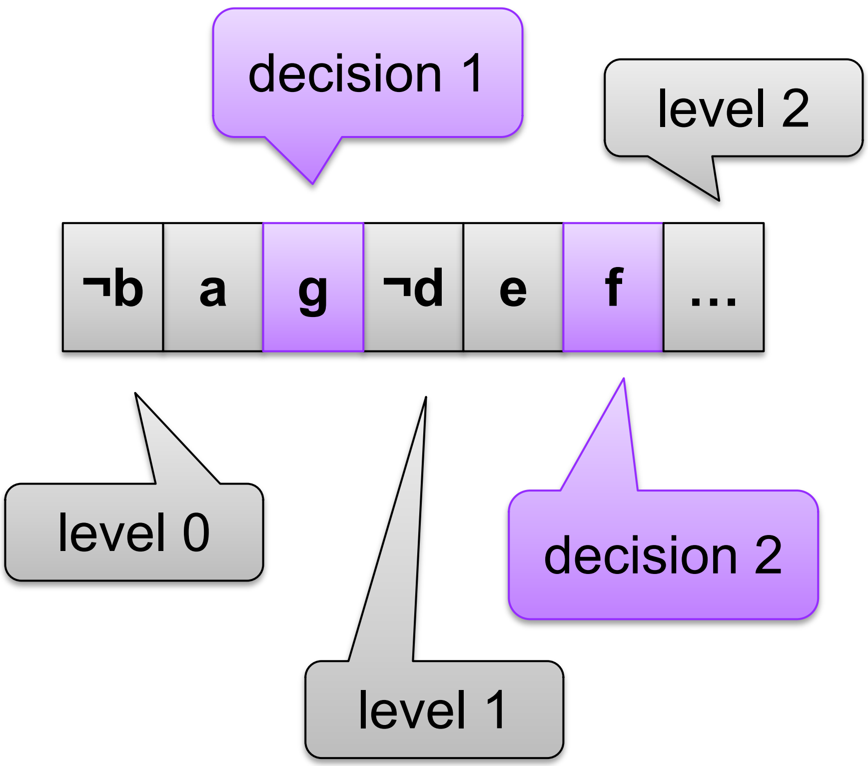
A trail is partitioned into levels

- a level of an assigned literal is the number of decisions before it

Literals at level 0 represent unit clauses that are implied by the database

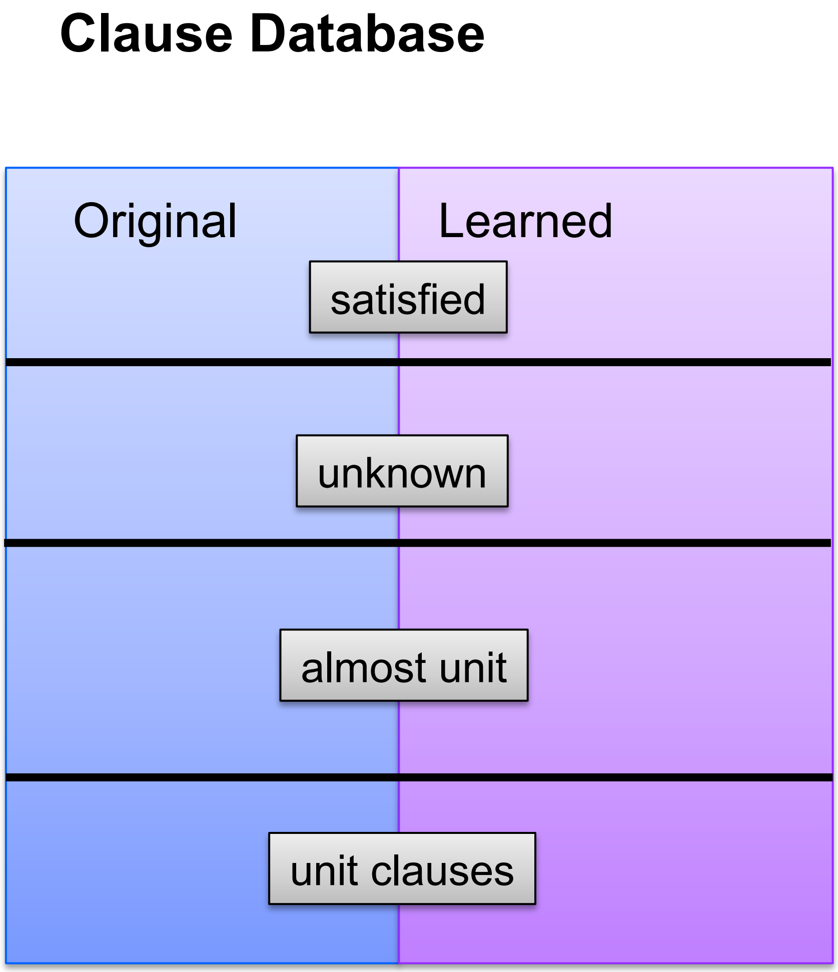
- These are true facts of the database that do not depend on any decision

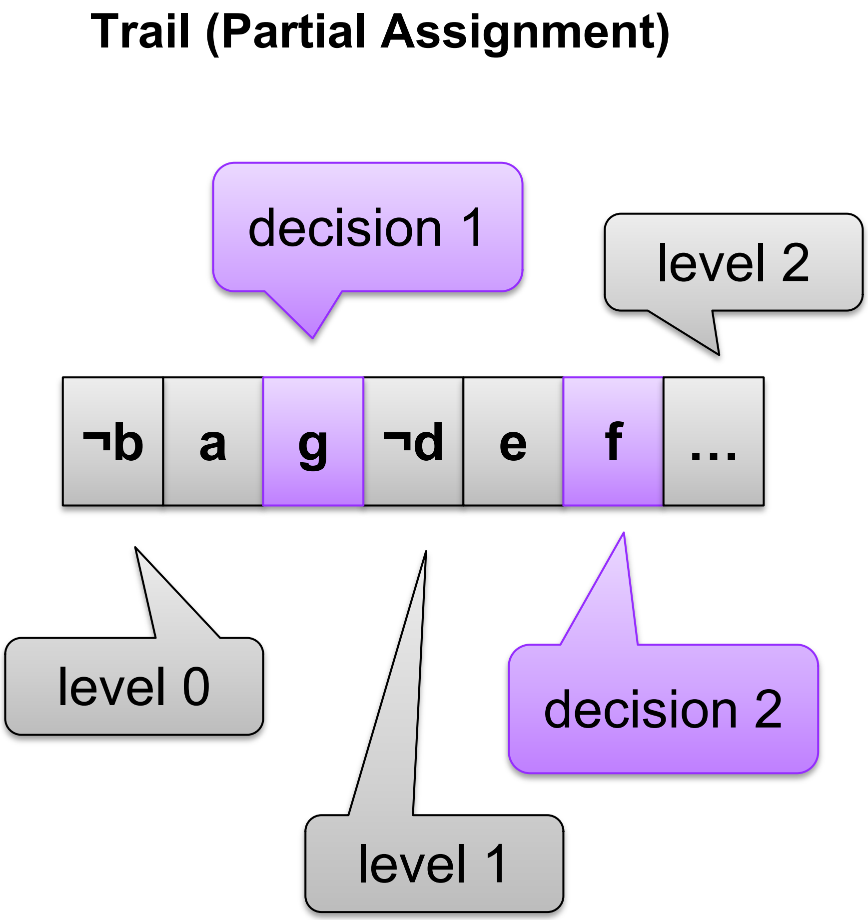
Trail (Partial Assignment)



UNIVERSITY OF WATERLOO

Architecture of a SAT Solver

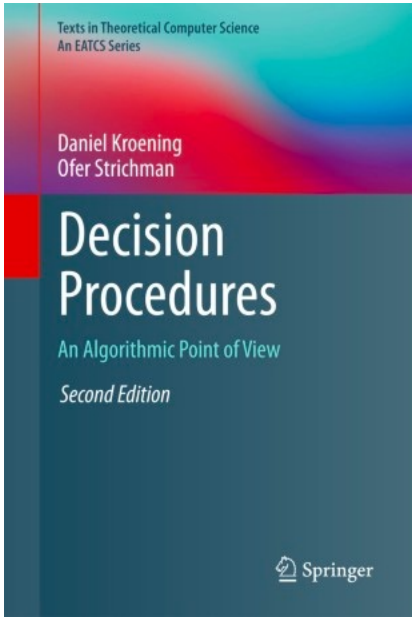




References

- Chapter: Decision Procedures for Propositional Logic from Decision Procedures An Algorithmic Point of View by Daniel Kroening and Ofer Strichman

- https://link-springer-com.proxy.lib.uwaterloo.ca/chapter/10.1007/978-3-662-50497-0_2



SAT - Milestones

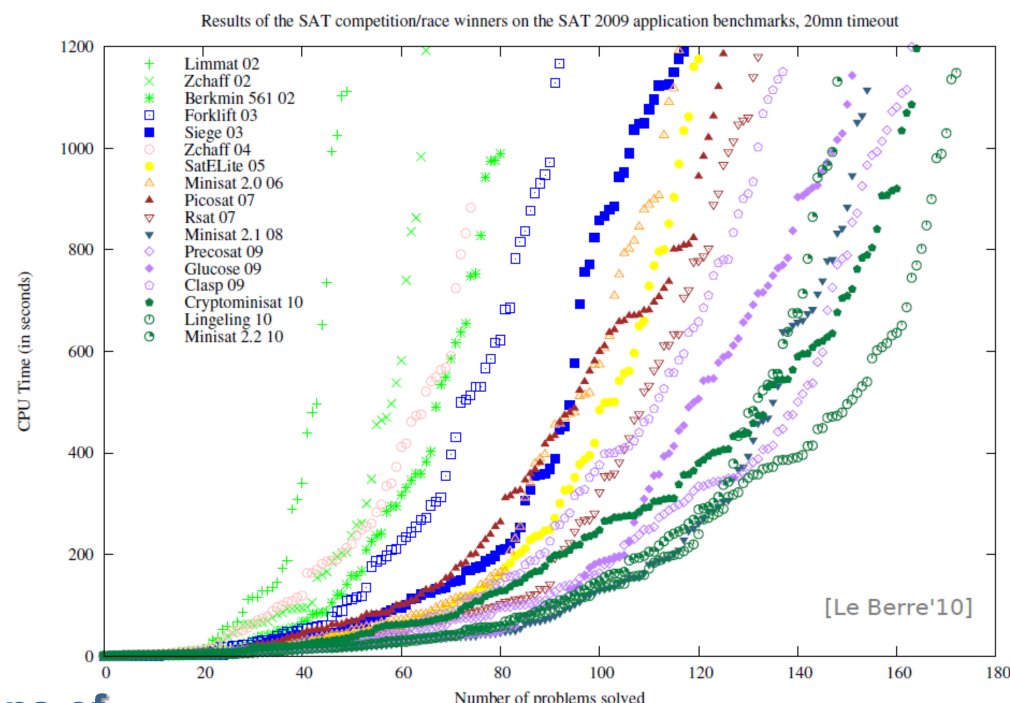
Problems impossible 10 years ago are trivial today

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

CONVERTING TO CNF

Conjuctive Normal Form

$$
\begin{array}{lll}
\varphi \leftrightarrow \psi & \Rightarrow_{\mathrm{CNF}} & \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi \\
\varphi \rightarrow \psi & \Rightarrow_{\mathrm{CNF}} & \neg\varphi \vee \psi \\
\neg(\varphi \vee \psi) & \Rightarrow_{\mathrm{CNF}} & \neg\varphi \wedge \neg\psi \\
\neg(\varphi \wedge \psi) & \Rightarrow_{\mathrm{CNF}} & \neg\varphi \vee \neg\psi \\
\neg\neg\varphi & \Rightarrow_{\mathrm{CNF}} & \varphi \\
(\varphi \wedge \psi) \vee \xi & \Rightarrow_{\mathrm{CNF}} & (\varphi \vee \xi) \wedge (\psi \vee \xi)
\end{array}
$$

Every propositional formula can be put in CNF

PROBLEM: (potential) exponential blowup of the resulting formula

Tseitin Transformation – Main Idea

Introduce a fresh variable e_i for every subformula G_i of F

- intuitively, e_i represents the truth value of G_i

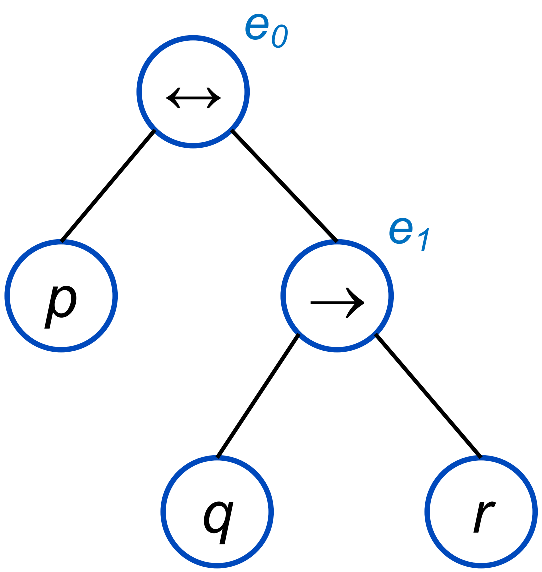
Assert that every e_i and G_i pair are equivalent

- $\mathrm{e_i} \leftrightarrow \mathrm{G_i}$
- convert this to CNF in the naïve way

Conjoin all such assertions in the end

Tseitin Transformation: Example

$G : p \leftrightarrow (q \rightarrow r)$

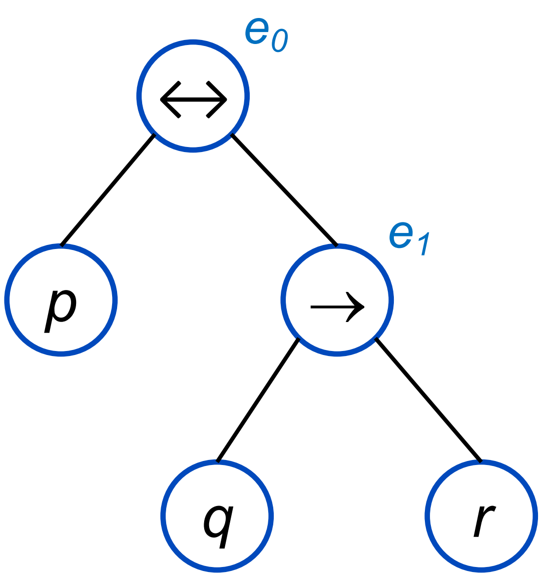


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$$
\begin{aligned}
& e_0 \leftrightarrow (p \leftrightarrow e_1) \\
= \quad & (e_0 \rightarrow (p \leftrightarrow e_1)) \wedge ((p \leftrightarrow e_1)) \rightarrow e_0) \\
= \quad & (e_0 \rightarrow (p \rightarrow e_1)) \wedge (e_0 \rightarrow (e_1 \rightarrow p)) \wedge \\
& (((p \wedge e_1) \vee (\neg p \wedge \neg e_1)) \rightarrow e_0) \\
= \quad & (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee \neg e_1 \vee p) \wedge \\
& (\neg p \vee \neg e_1 \vee e_0) \wedge (p \vee e_1 \vee e_0)
\end{aligned}
$$

Tseitin Transformation: Example

$G : p \leftrightarrow (q \rightarrow r)$



$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$G : e_0 \wedge$
$(\neg e_0 \vee \neg p \vee e_1) \wedge$
$(\neg e_0 \vee p \vee \neg e_1) \wedge$
$(e_0 \vee p \vee e_1) \wedge$
$(e_0 \vee \neg p \vee \neg e_1) \wedge$
$(\neg e_1 \vee \neg q \vee r) \wedge$
$(e_1 \vee q) \wedge (e_1 \vee \neg r)$

UNIVERSITY OF WATERLOO

Formula to CNF Conversion

`mk_fresh_var()` returns a fresh variable not used anywhere before

```
def cnf (ϕ):
  p, F = cnf_rec (ϕ)
  return p ∧ F

def cnf_rec (ϕ):
  if is_atomic (ϕ): return (ϕ, True)
  elif ϕ == ψ ∧ ξ:
    q, F1 = cnf_rec (ψ)
    r, F2 = cnf_rec (ξ)

    p = mk_fresh_var ()
    # C is CNF for p↔(q∧r)
    C = (¬p∨q)∧(¬p∨r)∧(p∨¬q∨¬r)
    return (p, F1∧F2∧C)
  elif ϕ == ψ∨ξ:
    …
```

Exercise: Complete cases for $\phi == \psi \vee \xi$, $\phi == \neg\psi$, $\phi == \psi \leftrightarrow \xi$

Tseitin Transformation [1968]

Used in practice

- No exponential blow-up
- CNF formula size is linear with respect to the original formula

Does not produce an equivalent CNF

However, given F, the following holds for the computed CNF F':

- F' is equisatisfiable to F
- Every model of F' can be translated (i.e., projected) to a model of F
- Every model of F can be translated (i.e., completed) to a model of F'

No model is lost or added in the conversion

Acknowledgement

Many of the slides are due to:

- Nikolaj Bjorner
- Daniel Kroening
- Ofer Strichman

https://www.decision-procedures.org/slides/