SAT Solving

Testing, Quality Assurance, and Maintenance
Winter 2018

Prof. Arie Gurfinkel

based on slides by Prof. Ruzica Piskac, Nikolaj Bjorner, and others

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

UNIVERSITY OF WATERLOO

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

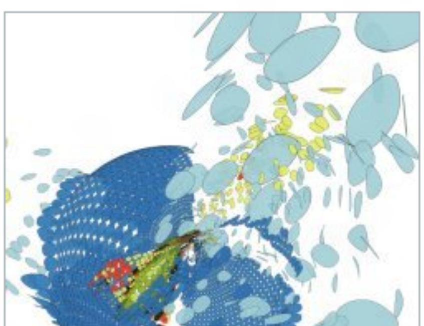
Boolean Satisfiability: From Theoretical Hardness to Practical Success

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

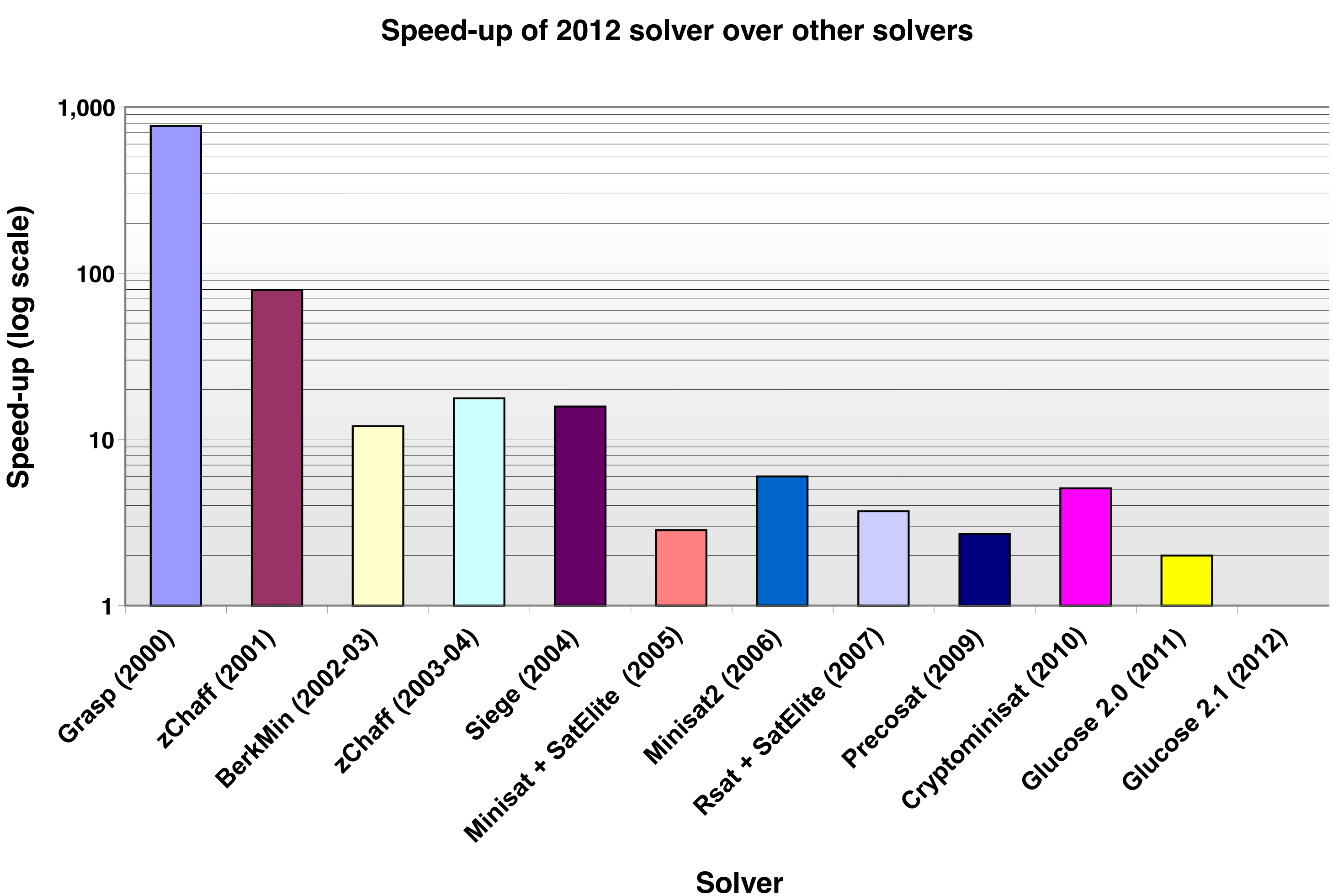
» Forgot Password?

» 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

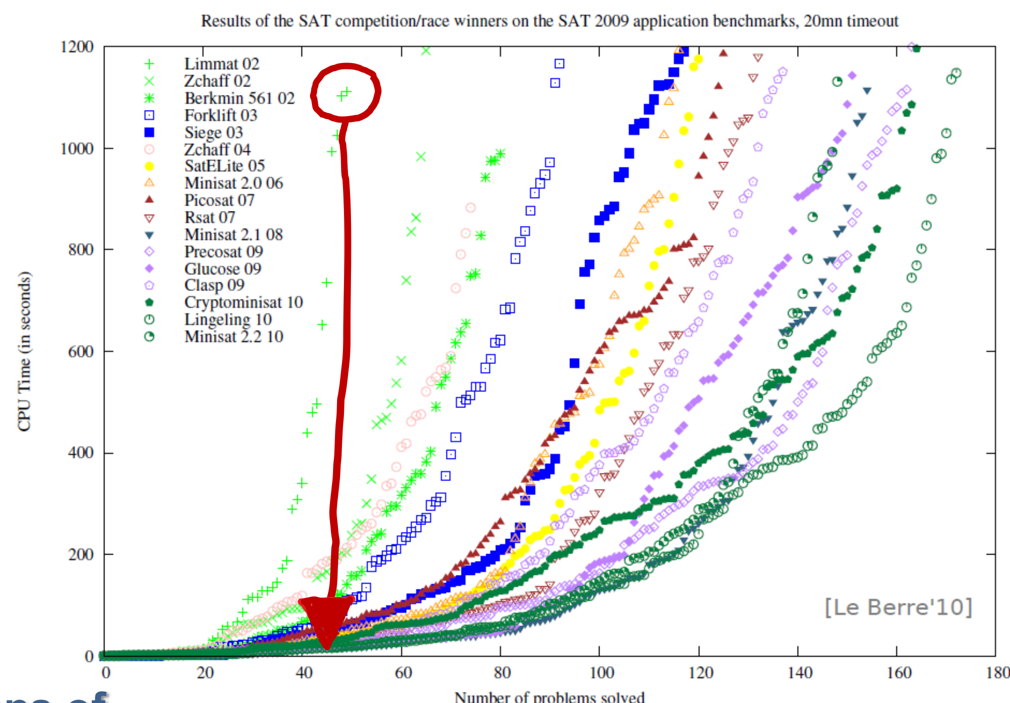
SAT - Milestones

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

2002 2010



Millions of variables from HW designs

Courtesy Daniel le Berre

Davis Putnam Logemann Loveland

DPLL PROCEDURE

Decision Procedure for Satisfiability

Algorithm that in some finite amount of computation decides if a given propositional logic (PL) formula F is satisfiable

- NP-complete problem

Modern decision procedures for PL formulae are called SAT solvers

Naïve approach

- Enumerate models (i.e., truth tables)
- Enumerate resolution proofs

Modern SAT solvers

- DPLL algorithm
  - Davis-Putnam-Logemann-Loveland
- Combines model- and proof-based search
- Operates on Conjunctive Normal Form (CNF)

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

UNIVERSITY OF WATERLOO

SAT solving by resolution (DP)

Assume that input formula F is in CNF

1. Pick two clauses C_1 and C_2 in F that can be resolved
2. If the resolvent C is an empty clause, return UNSAT
3. Otherwise, add C to F and go to step 1
4. If no new clauses can be resolved, return SAT

Termination: finitely many derived clauses

DPLL: David Putnam Logemann Loveland

Combines pure resolution-based search with case splitting on decisions

Proof search is restricted to unit resolution

- can be done very efficiently (polynomial time)

Case split restores completeness

DPLL can be described by the following two rules

- F is the input formula in CNF

$$\frac{F}{F, p \quad | \quad F, \neg p}\text{split} \qquad p \textit{ and } \neg p \textit{ are not in } F$$

$$\frac{F,\ C \vee \ell, \neg \ell}{F, C, \neg \ell}\text{unit}$$

Davis, Martin; Logemann, George; Loveland, Donald (1962). "A Machine Program for Theorem Proving". C. ACM. **5** (7): 394–397. doi:10.1145/368273.368557

The original DPLL procedure

Incrementally builds a satisfying truth assignment M for the input CNF formula F

M is grown by

- deducing the truth value of a literal from M and F, or
- guessing a truth value

If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value

UNIVERSITY OF WATERLOO

DPLL: Illustration

M | F

Partial model

Set of clauses

DPLL: Illustration

Guessing

$$p \;\mid\; p \vee q, \neg q \vee r$$

$$\Downarrow$$

$$p, \neg q \mid p \vee q, \neg q \vee r$$

DPLL: Illustration

Deducing

$$p \mid p \vee q, \neg p \vee s$$

$$p, s \mid p \vee q, \neg p \vee s$$

UNIVERSITY OF WATERLOO

DPLL: Illustration

Backtracking

$p, \neg s, \; q \;\; | \;\; p \vee q, s \vee q, \neg p \vee \neg q$

$p, s \; | \; p \vee q, s \vee q, \neg p \vee \neg q$

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\boxed{\neg x_1} \vee x_2) \wedge (\boxed{x_3} \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ and x_3 are pure literals

Pure literal rule :

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

UNIVERSITY OF WATERLOO

DPLL (as a procedure)

- Standard backtrack search
- DPLL(F) :
  - Apply unit propagation
  - If conflict identified, return UNSAT
  - Apply the pure literal rule
  - If F is satisfied (empty), return SAT
  - Select decision variable x
    - If DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

The Original DPLL Procedure – Example

assign

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ \boxed{1}$

Deduce 1

1

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce $\neg 2$

1, 2

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Guess 3

1, 2, 3

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Deduce 4

1, 2, 3, 4

$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Conflict

UNIVERSITY OF WATERLOO

The Original DPLL Procedure – Example

assign	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ \boxed{1}$
Deduce 1	
1	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \boxed{\neg 1 \lor \neg 2,}\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce $\neg 2$	
1, 2	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Guess 3	
1, 2, 3	$1 \lor 2,\ \boxed{2 \lor \neg 3 \lor 4,}\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce 4	
1, 2, 3, 4	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Undo 3	

The Original DPLL Procedure – Example

assign	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce 1	
1	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce $\neg 2$	
1, 2	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Guess $\neg 3$	
1, 2, 3	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Model Found	

UNIVERSITY OF WATERLOO

An Abstract Framework for DPLL

The DPLL procedure can be described declaratively by simple sequent-style calculi

Such calculi, however, cannot model meta-logical features such as backtracking, learning, and restarts

We model DPLL and its enhancements as transition systems instead

A transition system is a binary relation over states, induced by a set of conditional transition rules

An Abstract Framework for DPLL

State

- **fail** or M ‖ F
- where
  - F is a CNF formula, a set of clauses, and
  - M is a sequence of annotated literals denoting a partial truth assignment

Initial State

- ∅ ‖ F, where F is to be checked for satisfiability

Expected final states:

- **fail** if F is unsatisfiable
- M ‖ G
  where
  - M is a model of G
  - G is logically equivalent to F

Transition Rules for DPLL

Extending the assignment:

UnitProp

$$M \parallel F, C \lor l \to M\, l \parallel F, C \lor l \quad \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

Decide

$$M \parallel F, C \to M\, l^d \parallel F, C \quad \begin{cases} l \text{ or } \neg l \text{ occur in } C \\ l \text{ is undefined in } M \end{cases}$$

Notation: l^d is a decision literal

UNIVERSITY OF WATERLOO

Transition Rules for DPLL

Repairing the assignment:

Fail $M \parallel F, C \to fail$ $\begin{cases} M \models \neg C \\ \text{M does not contain decision literals} \end{cases}$

Backtrack $M\ l^d\ N \parallel F, C \to M\ \neg l \parallel F, C$ $\begin{cases} M\ l^d\ N \models \neg C \\ \text{l is the last decision literal} \end{cases}$

UNIVERSITY OF WATERLOO

Transition Rules DPLL – Example

$\varnothing \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp $\neg 2$

$1, 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Decide 3

$1, 2, 3^d \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 4

$1, 2, 3^d, 4 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Backtrack 3

UNIVERSITY OF WATERLOO

Transition Rules DPLL – Example

$\emptyset \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp $\neg 2$

$1, 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Decide 3

$1, 2, 3^d \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 4

$1, 2, 3 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Backtrack 3

Transition Rules for DPLL (on one slide)

UnitProp $M \parallel F, C \vee l \rightarrow M\, l \parallel F, C \vee l$ $\begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$

Decide $M \parallel F, C \rightarrow M\, l^d \parallel F, C$ $\begin{cases} l \text{ or } \neg l \text{ occur in } C \\ l \text{ is undefined in } M \end{cases}$

Fail $M \parallel F, C \rightarrow \text{fail}$ $\begin{cases} M \models \neg C \\ M \text{ does not contain decision literals} \end{cases}$

Backtrack $M\, l^d\, N \parallel F, C \rightarrow M \neg l \parallel F, C$ $\begin{cases} M\, l^d\, N \models \neg C \\ l \text{ is the last decision literal} \end{cases}$

UNIVERSITY OF WATERLOO

The DPLL System – Correctness

Some terminology

- Irreducible state: state to which no transition rule applies.
- Execution: sequence of transitions allowed by the rules and starting with states of the form $\emptyset \parallel F$.
- Exhausted execution: execution ending in an irreducible state

Proposition (Strong Termination) Every execution in DPLL is finite

Proposition (Soundness) For every exhausted execution starting with $\emptyset \parallel F$ and ending in $M \parallel F$, $M \models F$

Proposition (Completeness) If F is unsatisfiable, every exhausted execution starting with $\emptyset \parallel F$ ends with fail

Maintained in more general rules + theories

Modern DPLL: CDCL

Conflict **D**riven **C**lause **L**earning

- two watched literals – efficient index to find clauses that can be used in unit resolution
- periodically restart backtrack search
- activity-based decision heuristic to choose decision variable
- **conflict resolution via clausal learning**

We will briefly look at clausal learning

More details on CDCL are available in

- Chapter 2 of Decision Procedures book
- http://gauss.ececs.uc.edu/SAT/articles/FAIA185-0131.pdf

Conflict Directed Clause Learning

Lemma learning

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s$

$\Downarrow$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg s$

$\Downarrow$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg q$

$\Downarrow$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee t$

Learned Clause by Resolution

A new clause is learned by resolving the conflicting clause with clauses deduced from the last decision

$$\dfrac{\dfrac{t \vee \neg p \vee q \qquad \neg q \vee s}{t \vee \neg p \vee s} \qquad \neg p \vee \neg s}{\neg p \vee t}$$

Modern CDCL: Abstract Rules

Initialize	$\epsilon \mid F$	F *is a set of clauses*
Decide	$M \mid F \implies M, \ell \mid F$	ℓ *is unassigned*
Propagate	$M \mid F, C \vee \ell \implies M, \ell^{C \vee \ell} \mid F, C \vee \ell$	C *is false under* M
Sat	$M \mid F \implies M$	F *true under* M
Conflict	$M \mid F, C \implies M \mid F, C \mid C$	C *is false under* M
Learn	$M \mid F \mid C \implies M \mid F, C \mid C$	
Unsat	$M \mid F \mid \emptyset \implies Unsat$	
Backjump	$MM' \mid F \mid C \vee \ell \implies M\ell^{C \vee \ell} \mid F$	$\bar{C} \subseteq M, \neg\ell \in M'$
Resolve	$M \mid F \mid C' \vee \neg\ell \implies M \mid F \mid C' \vee C$	$\ell^{C \vee \ell} \in M$
Forget	$M \mid F, C \implies M \mid F$	C is a learned clause
Restart	$M \mid F \implies \epsilon \mid F$	

Model

Proof

Conflict Resolution

[Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized

Conjuctive Normal Form

$$\begin{aligned}
\varphi \leftrightarrow \psi &\Rightarrow_{\text{CNF}} & \varphi \to \psi \wedge \psi \to \varphi \\
\varphi \to \psi &\Rightarrow_{\text{CNF}} & \neg\varphi \vee \psi \\
\neg(\varphi \vee \psi) &\Rightarrow_{\text{CNF}} & \neg\varphi \wedge \neg\psi \\
\neg(\varphi \wedge \psi) &\Rightarrow_{\text{CNF}} & \neg\varphi \vee \neg\psi \\
\neg\neg\varphi &\Rightarrow_{\text{CNF}} & \varphi \\
(\varphi \wedge \psi) \vee \xi &\Rightarrow_{\text{CNF}} & (\varphi \vee \xi) \wedge (\psi \vee \xi)
\end{aligned}$$

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

- $e_i \leftrightarrow G_i$
- and express the assertion as CNF

Conjoin all such assertions in the end

Formula to CNF Conversion

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

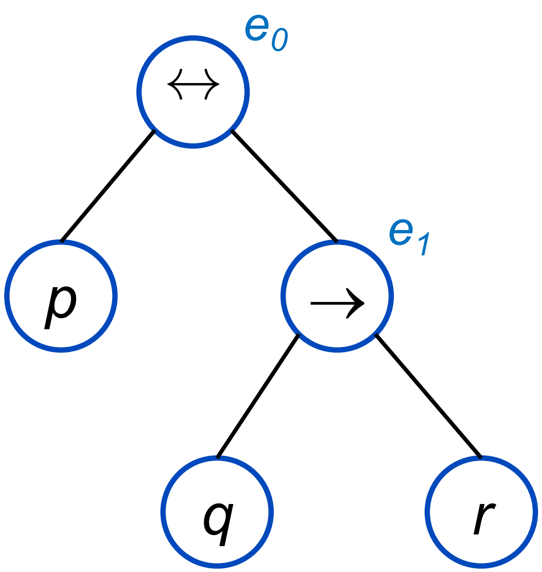
```
def cnf (ϕ):
  p, F = cnf_rec (ϕ)
  return p ∧ F

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 $\phi == \psi \vee \xi$, $\phi == \neg\psi$, $\phi == \psi \leftrightarrow \xi$

UNIVERSITY OF WATERLOO

Tseitin Transformation: Example

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

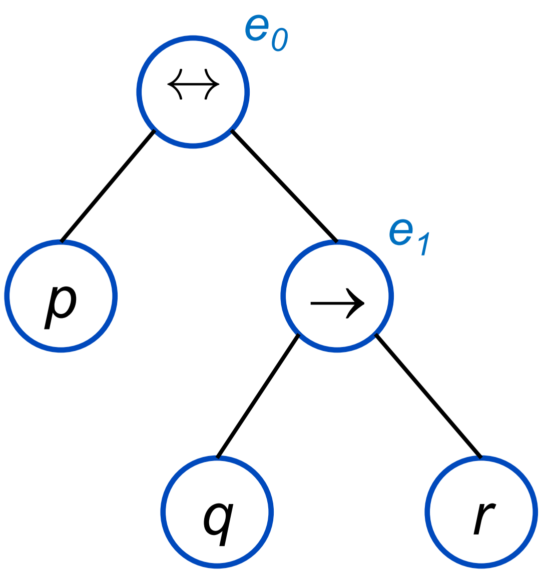


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

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

Tseitin Transformation: Example

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

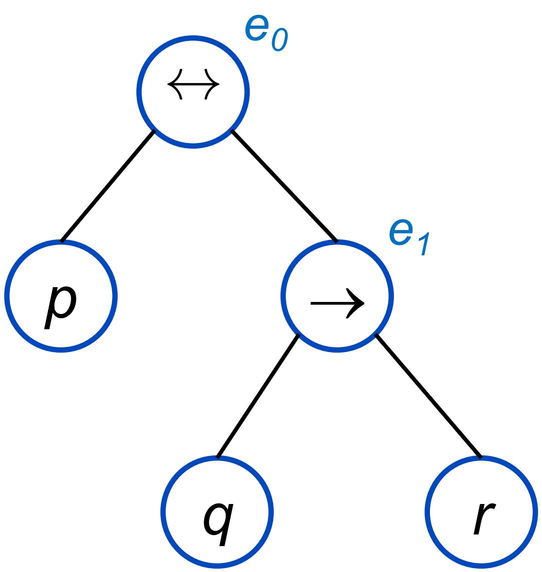


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

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

Tseitin Transformation: Example

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



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

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

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