

# Propositional Satisfiability

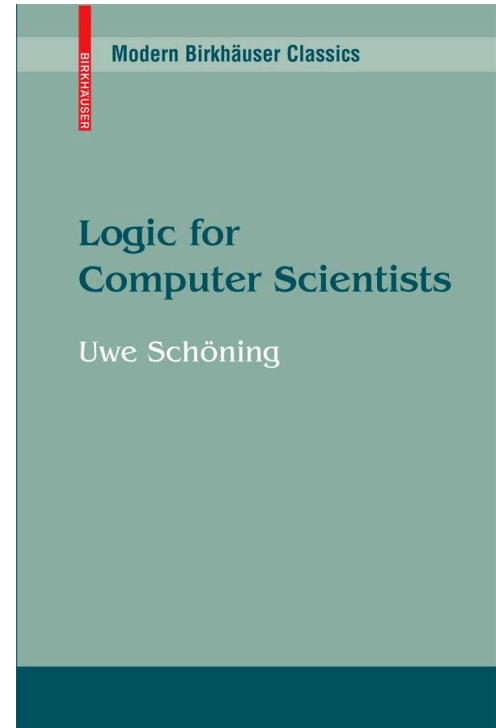
Methods & Tools for Software Engineering (MTSE)  
Fall 2017

Prof. Arie Gurfinkel



# References

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



# Syntax of Propositional Logic

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

*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  $A, B, C, \dots$  instead of  $A_1, A_2, \dots$
- 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)

$\text{truth\_symbol} ::= \top(\text{true}) \mid \perp(\text{false})$

$\text{variable} ::= p, q, r, \dots$

$\text{atom} ::= \text{truth\_symbol} \mid \text{variable}$

$\text{literal} ::= \text{atom} \mid \neg \text{atom}$

$\text{formula} ::= \text{literal} \mid$

$\neg \text{formula} \mid$

$\text{formula} \wedge \text{formula} \mid$

$\text{formula} \vee \text{formula} \mid$

$\text{formula} \rightarrow \text{formula} \mid$

$\text{formula} \leftrightarrow \text{formula}$

# Boolean Satisfiability (CNF-SAT)

Let  $V$  be a set of variables

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

A *clause* is a disjunction of literals

- e.g.,  $(v1 \vee \sim v2 \vee v3)$

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

- e.g.,  $(v1 \vee \sim v2) \wedge (v3 \vee v2)$

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

- $\sim b$
- $\sim a \parallel \sim b \parallel \sim c$
- $a$
- sat:  $s(a) = \text{True}; s(b) = \text{False}; s(c) = \text{False}$

## CNF 2

- $\sim b$
- $\sim a \parallel b \parallel \sim c$
- $a$
- $\sim a \parallel c$
- unsat

# 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

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](#)

VIEWS: 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

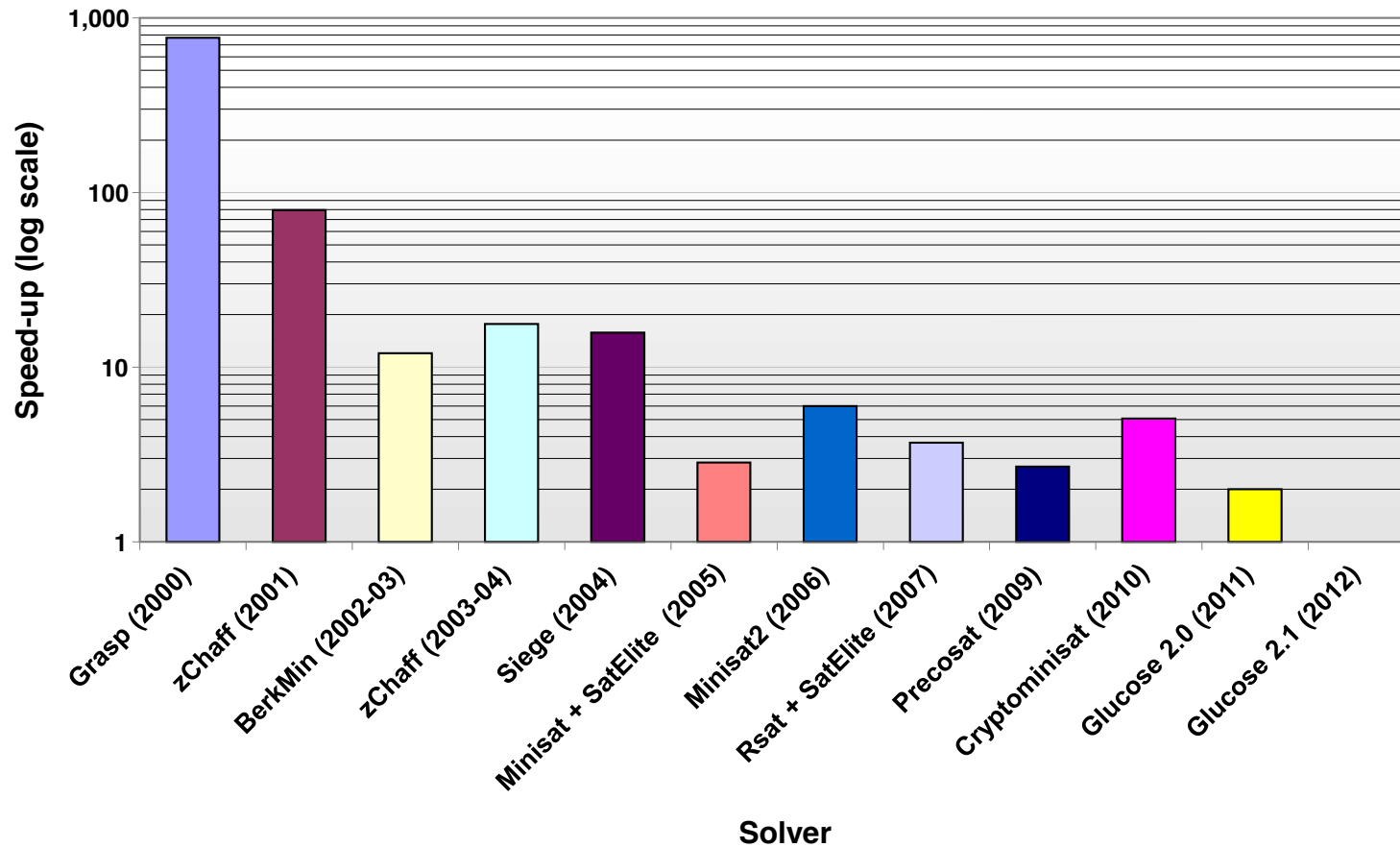
ARTICLE CONTENTS:

[Introduction](#)  
[Boolean Satisfiability](#)  
[Theoretical hardness: SAT and NP Completeness](#)



# Some Experience with SAT Solving

Speed-up of 2012 solver over other solvers



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

# SAT - Milestones

Problems impossible 10 years ago are trivial today

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

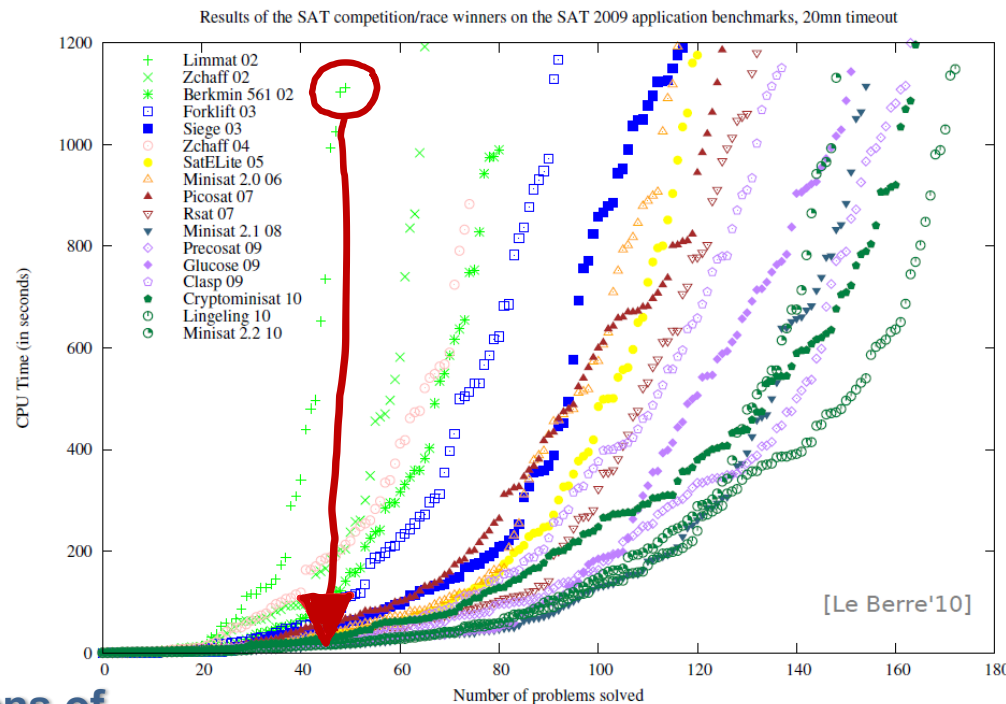
Concept



Millions of  
variables from  
HW designs

2002

2010



Courtesy Daniel le Berre

# USING A SAT SOLVER

# DIMACS interface to a SAT Solver

Input:

- a CNF in DIMACS format

Output:

- SAT/UNSAT + satisfying assignment

We will use a SAT solver called MiniSAT

- available at <https://github.com/agurfinkel/minisat>
- written in C++
- use as a library in Assignment 4
- use via DIMACS interface today in class

# 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

# MiniSat

MiniSat is one of the most famous modern SAT-solvers

- written in C++
- designed to be easily understandable and customizable
- many new SAT-solvers use MiniSAT as their base

Web page: <http://minisat.se/>

We will use a slightly updated version from GitHub:

<https://github.com/agurfinkel/minisat>

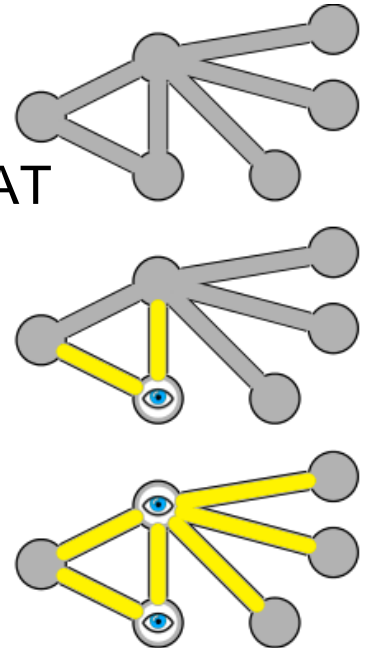
Good references for understanding SAT solving details

- MiniSat architecture: <http://minisat.se/downloads/MiniSat.pdf>
- Donald Knuth's SAT13 (also based on MiniSat)
  - <http://www-cs-faculty.stanford.edu/~knuth/programs/sat13.w>

# 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



## Exercise 33

Using resolution show that

$$A \wedge B \wedge C$$

is a consequence of

$$\neg A \vee B$$

$$\neg B \vee C$$

$$A \vee \neg C$$

$$A \vee B \vee C$$



## Exercise 34

Show using resolution that  $F$  is valid

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

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

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 truth table

Modern SAT solvers

- DPLL algorithm
  - Davis-Putnam-Logemann-Loveland
- Operates on Conjunctive Normal Form (CNF)

# Propositional Resolution

Pivot

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

Resolvent

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

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  $D$

# 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  $F$

# 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 UNAT iff  $Res^*(F)$  contains an empty clause

UNSAT?

# Exercise from LCS

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

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

$$B \vee C$$

$$\neg A \vee C$$

$$B \vee \neg C$$

$$\neg C$$

# Proof of the Resolution Theorem

(*Soundness*) By Resolution Lemma,  $F$  is equivalent to  $\text{Res}^i(F)$  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  $A_1, \dots, A_{n+1}$

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.



# Proof System

$$P_1, \dots, P_n \vdash C$$

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

- where,  $P_1, \dots, P_n, C$  are formulas
- $P_i$  are called **premises** and  $C$  is called a **conclusion**
- intuitively, the rule 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

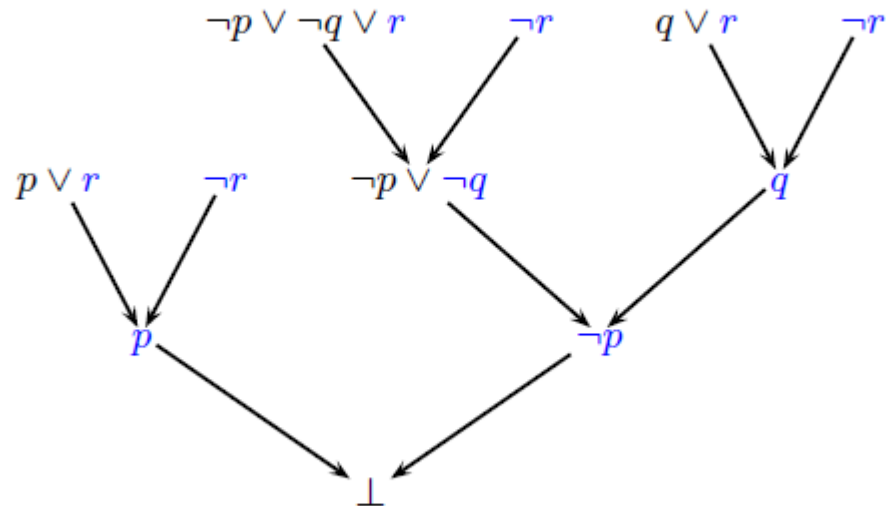
$$\frac{C \vee p \quad D \vee \neg p}{C \vee D}$$

Propositional resolution is a sound inference rule

Proposition resolution system consists of a single propositional resolution rule

# Example of a resolution proof

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



# Resolution Proof Example

Show by resolution that the following CNF is UNSAT

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

$$\begin{array}{c} \frac{\neg a \vee b \vee \neg c \quad a}{b \vee \neg c} \quad \textcolor{red}{b} \quad \frac{a \quad \neg a \vee c}{c} \\ \hline \neg c \quad c \\ \hline \perp \end{array}$$

# 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

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

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

$$\neg A \vee B$$

$$\neg B \vee C$$

$$A \vee \neg C$$

$$A \vee B \vee C$$



## Exercise 34

Show using resolution that  $F$  is valid

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

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

# SAT solving by resolution (DP)

Formula must be in CNF

**Resolution rule:** 
$$\frac{C \vee p \quad D \vee \neg p}{C \vee D}$$

**Example:** 
$$\frac{q \vee t \vee p \quad q \vee r \vee \neg p}{q \vee t \vee r}$$

The result of resolution is the resolvent (clause).  
Original clauses are kept (not deleted).  
Duplicate literals are deleted from the resolvent.

**Note:** No branching.

**Termination:** Only finite number of possible derived clauses.

# Unit & Input Resolution

**Unit resolution:**  $\frac{C \vee \ell \quad \neg \ell}{C}$  ( $C \vee \ell$  is subsumed by  $C$ )

**Input resolution:**  $\frac{C \vee \ell \quad D \vee \neg \ell}{C \vee D}$  ( $C \vee \ell$  member of input  $F$ ).

(Optional) Exercise:

Set of clauses  $F$ :

$F$  has an input refutation iff  $F$  has a unit refutation.

# DPLL

DPLL: David Putnam Logeman Loveland = Unit resolution + split rule.

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

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

Ingredient of most efficient SAT solvers

# The original DPLL procedure

Tries to **build** incrementally a satisfying truth assignment  $M$  for a 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

# DPLL

$M \mid F$

*Partial  
model*

*Set of clauses*

# DPLL

## Guessing

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



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

# DPLL

## Deducing

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



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



# DPLL

## Backtracking

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



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

# Pure Literals

A literal is **pure** if only occurs positively or negatively.

Example :

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

$\neg x_1$  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 !

# 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  $\text{DPLL}(F \wedge x) = \text{SAT}$  return SAT
    - ▶ return  $\text{DPLL}(F \wedge \neg x)$

## The Original DPLL Procedure – Example

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

## The Original DPLL Procedure – Example

assign	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
Deduce 1	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
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 $\neg 2$	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
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	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
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	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
1, 2, 3, 4	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
Undo 3	

## The Original DPLL Procedure – Example

assign	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
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 $\neg 3$	
1, 2, 3	$1 \vee 2, 2 \vee \neg 3 \vee 4, \neg 1 \vee \neg 2, \neg 1 \vee \neg 3 \vee \neg 4, 1$
Model Found	

# 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  $M \parallel F$
- 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

- $\emptyset \parallel F$ , where  $F$  is to be checked for satisfiability

## Expected final states:

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



# Transition Rules for the Original DPLL

Extending the assignment:

$$\text{UnitProp} \quad M \parallel F, C \vee I \rightarrow M \mid I \parallel F, C \vee I \quad \left\{ \begin{array}{l} M \models \neg C \\ I \text{ is undefined in } M \end{array} \right.$$

$$\text{Decide} \quad M \parallel F, C \rightarrow M \mid^d \parallel F, C \quad \left\{ \begin{array}{l} I \text{ or } \neg I \text{ occur in } C \\ I \text{ is undefined in } M \end{array} \right.$$

Notation:  $I^d$  is a decision literal

# Transition Rules for the Original DPLL

Repairing the assignment:

Fail

$M \parallel F, C \rightarrow \text{fail}$

$M \models \neg C$

M does not contain  
decision literals

Backtrack

$M \mid^d N \parallel F, C \rightarrow M \neg I \parallel F, C$

$M \mid^d N \models \neg C$

I is the last decision  
literal

# 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

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

Backtrack  
3

# 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 the Original DPLL

**UnitProp**  $M \parallel F, C \vee I \rightarrow M I \parallel F, C \vee I$   $\left\{ \begin{array}{l} M \models \neg C \\ I \text{ is undefined in } M \end{array} \right.$

**Decide**  $M \parallel F, C \rightarrow M I^d \parallel F, C$   $\left\{ \begin{array}{l} I \text{ or } \neg I \text{ occur in } C \\ I \text{ is undefined in } M \end{array} \right.$

**Fail**  $M \parallel F, C \rightarrow \text{fail}$   $\left\{ \begin{array}{l} M \models \neg C \\ M \text{ does not contain} \\ \text{decision literals} \end{array} \right.$

**Backtrack**  $M I^d N \parallel F, C \rightarrow M \neg I \parallel F, C$   $\left\{ \begin{array}{l} M I^d N \models \neg C \\ I \text{ is the last decision literal} \end{array} \right.$

# The Basic 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 Basic 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

# Modern DPLL: CDCL

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

Two watched literals

Periodically restarting backtrack search

Clausal learning

More details at

<http://gauss.ececs.uc.edu/SAT/articles/FAIA185-0131.pdf>

# Conflict Directed Clause Learning

## Lemma learning

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



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



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



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



# CDCL: learns clauses by resolution

Every learned clause is obtained by resolution from clauses learned before it

$$\frac{\frac{t \vee \neg p \vee q \quad \neg q \vee s}{t \vee \neg p \vee s} \quad \neg p \vee \neg s}{\neg p \vee t}$$

# Modern CDCL

Initialize  $\epsilon \mid F$   $F$  is a set of clauses

Decide  $M \mid F \Rightarrow M, \ell \mid F$   $\ell$  is unassigned

Propagate  $M \mid F, C \vee \ell \Rightarrow M, \ell^{C \vee \ell} \mid F, C \vee \ell$   $C$  is false under  $M$

Sat  $M \mid F \Rightarrow M$   $F$  true under  $M$

Conflict  $M \mid F, C \Rightarrow M \mid F, C \mid C$   $C$  is false under  $M$

Learn  $M \mid F \mid C \Rightarrow M \mid F, C \mid C$

Unsat  $M \mid F \mid \emptyset \Rightarrow \text{Unsat}$

Backjump  $MM' \mid F \mid C \vee \ell \Rightarrow M \ell^{C \vee \ell} \mid F$   $\bar{C} \subseteq M, \neg \ell \in M'$

Resolve  $M \mid F \mid C' \vee \neg \ell \Rightarrow M \mid F \mid C' \vee C$   $\ell^{C \vee \ell} \in M$

Forget  $M \mid F, C \Rightarrow M \mid F$   $C$  is a learned clause

Restart  $M \mid F \Rightarrow \epsilon \mid F$

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

# Conjunctive Normal Form

$\varphi \leftrightarrow \psi$	$\Rightarrow \text{CNF}$	$\varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
$\varphi \rightarrow \psi$	$\Rightarrow \text{CNF}$	$\neg \varphi \vee \psi$
$\neg(\varphi \vee \psi)$	$\Rightarrow \text{CNF}$	$\neg \varphi \wedge \neg \psi$
$\neg(\varphi \wedge \psi)$	$\Rightarrow \text{CNF}$	$\neg \varphi \vee \neg \psi$
$\neg \neg \varphi$	$\Rightarrow \text{CNF}$	$\varphi$
$(\varphi \wedge \psi) \vee \xi$	$\Rightarrow \text{CNF}$	$(\varphi \vee \xi) \wedge (\psi \vee \xi)$

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  $G_i$

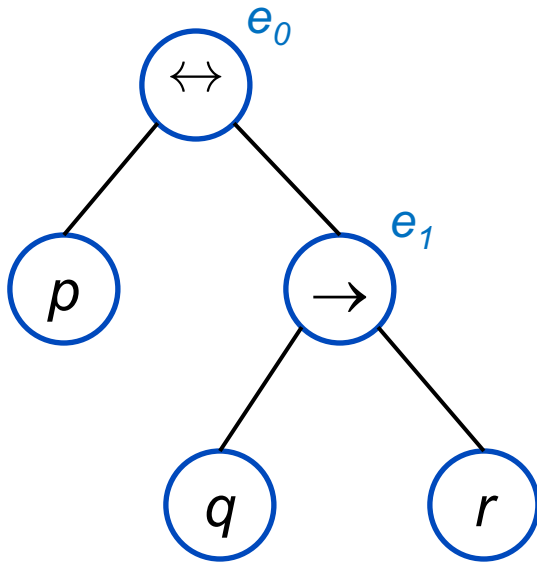
Assert that every  $e_i$  and  $G_i$  pair are equivalent

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

Conjoin all such assertions in the end

# Tseitin Transformation: Example

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

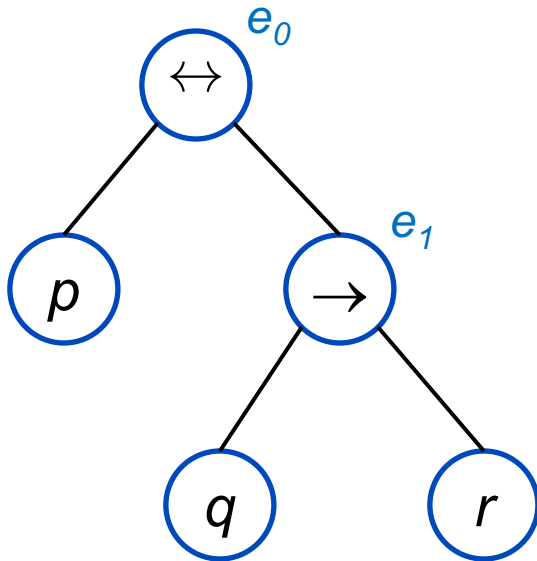


$$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$$

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

# Tseitin Transformation: Example

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

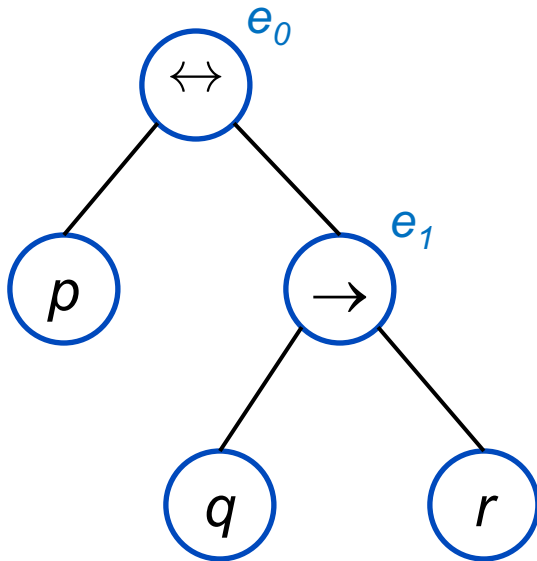


$$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$$

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

# Tseitin Transformation: Example

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



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

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

# Formula to CNF Conversion

```
def cnf ( $\phi$ ):  
    p, F = cnf_rec ( $\phi$ )  
    return p  $\wedge$  F
```

mk\_fresh\_var() returns a fresh variable not used anywhere before

```
def cnf_rec ( $\phi$ ):  
    if is_atomic ( $\phi$ ): return ( $\phi$ , True)  
    elif  $\phi == \psi \wedge \xi$ :  
        q, F1 = cnf_rec ( $\psi$ )  
        r, F2 = cnf_rec ( $\xi$ )  
  
        p = mk_fresh_var ()  
        # C is CNF for  $p \leftrightarrow (q \wedge r)$   
        C = ( $\neg p \vee q$ )  $\wedge$  ( $\neg p \vee r$ )  $\wedge$  ( $p \vee \neg q \vee \neg r$ )  
        return (p, F1  $\wedge$  F2  $\wedge$  C)  
    elif  $\phi == \psi \vee \xi$ :  
        ...
```

**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