Logic: SAT, FOL, SMT

Testing, Quality Assurance, and Maintenance
Winter 2017

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

CNF Examples

CNF 1

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

CNF 2

- ~b
- ~a || b || ~c
- a
- ~a || c
- unsat

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

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:



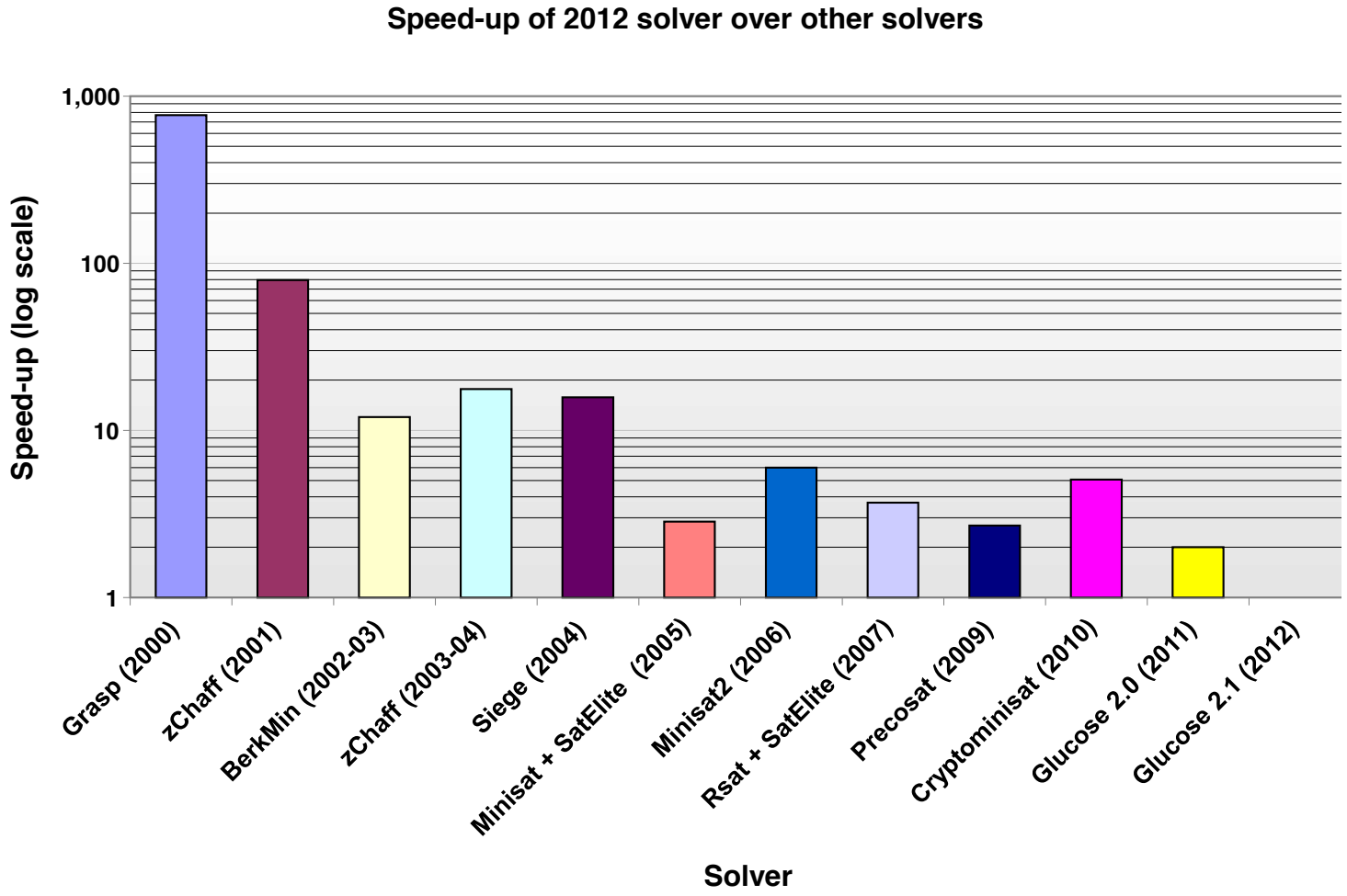
There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner 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

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

Some Experience with SAT Solving



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

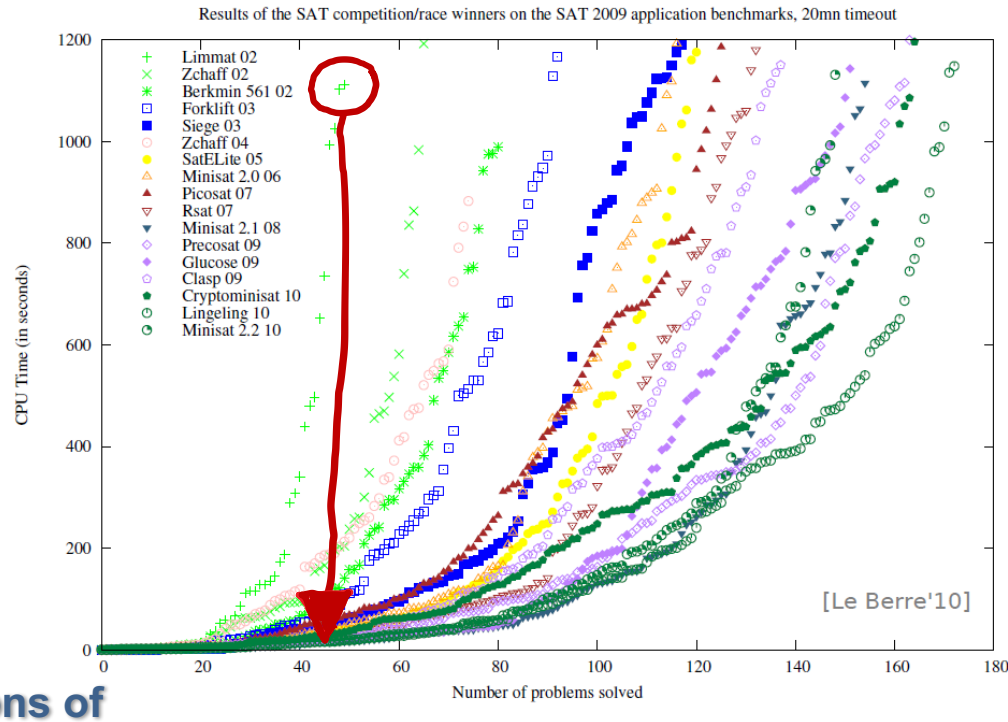
SAT - Milestones

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

2002 2010



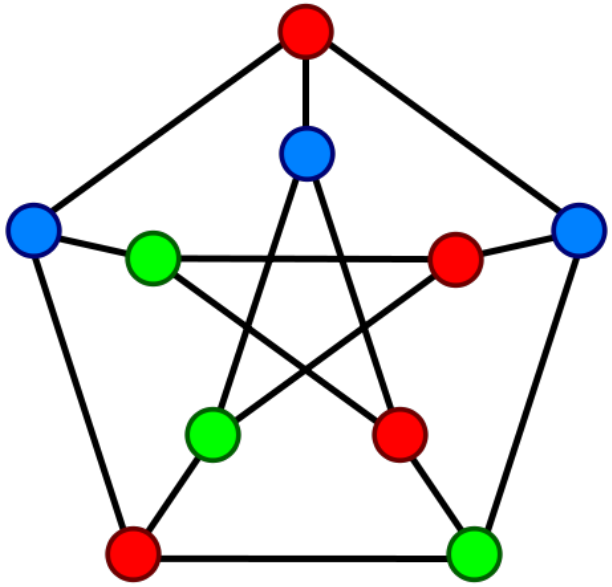
Millions of variables from HW designs

Courtesy Daniel le Berre

ENCODING PROBLEMS INTO SAT

Graph k-Coloring

Given a graph *G = (V, E)*, and a natural number *k > 0* is it possible to assign colors to vertices of *G* such that no two adjacent vertices have the same color.



Formally:

- does there exists a function f : V → [0..k) such that
- for every edge (u, v) in E, f(u) != f(v)

Graph coloring for k > 2 is NP-complete

Problem: Encode k-coloring of G into CNF

- construct CNF *C* such that C is SAT iff G is k-colorable

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

k-coloring as CNF

Let a Boolean variable $f_{v,i}$ denote that vertex *v* has color *i*

- if $f_{v,i}$ is true if and only if f(v) = i

Every vertex has at least one color

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

No vertex is assigned two colors

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

No two adjacent vertices have the same color

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

Davis Putnam Logemann Loveland

DPLL PROCEDURE

Syntax of Propositional Logic (PL)

$$
\begin{aligned}
\text{truth_symbol} &::= \top(\text{true}) \mid \bot(\text{false}) \\
\text{variable} &::= p, q, r, \ldots \\
\text{atom} &::= \text{truth_symbol} \mid \text{variable} \\
\text{literal} &::= \text{atom} \mid \neg\text{atom} \\
\text{formula} &::= \text{literal} \mid \\
&\quad \neg\text{formula} \mid \\
&\quad \text{formula} \wedge \text{formula} \mid \\
&\quad \text{formula} \vee \text{formula} \mid \\
&\quad \text{formula} \rightarrow \text{formula} \mid \\
&\quad \text{formula} \leftrightarrow \text{formula}
\end{aligned}
$$

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)

Conjuctive Normal Form

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

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

Clauses – CNF conversion

cnf(φ) = **let** (q,F) = cnf'(φ) **in** q $\wedge$ F

cnf'(a) = (a, *true*)

cnf'($\varphi \wedge \varphi'$) = **let** (q,F_1) = cnf'(φ)
(r, F_2) = cnf'(φ')
p = fresh Boolean variable
in
(p, $F_1 \wedge F_2 \wedge (\neg p \vee q) \wedge$
$(\neg p \vee r) \wedge$
$(p \vee \neg q \vee \neg r)$)

Exercise: cnf'($\varphi \vee \varphi'$), cnf'($\varphi \leftrightarrow \varphi'$), cnf'($\neg \varphi$)

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)

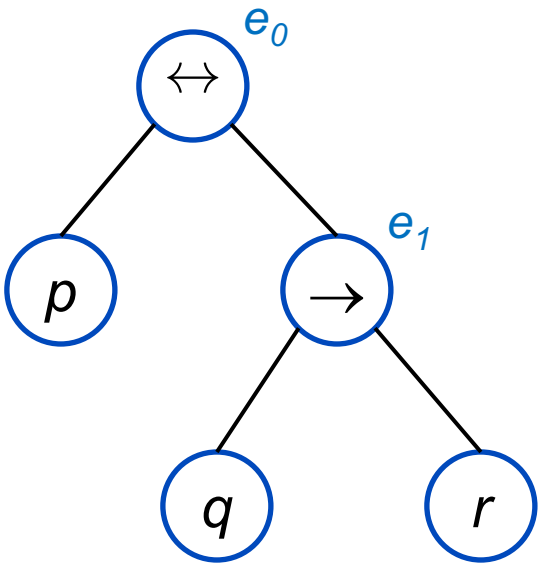
    p = mk_fresh_var ()
    # C is CNF for p↔(q∧r)
    C = (¬p∨q)∧(¬p∨r)∧(p∨¬q∨¬r)
    return (p, F1∧F2∧C)
  elif ϕ == ψ∨ξ:
    …
```

Exercise: Complete cases for

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

Tseitin Transformation: Example

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

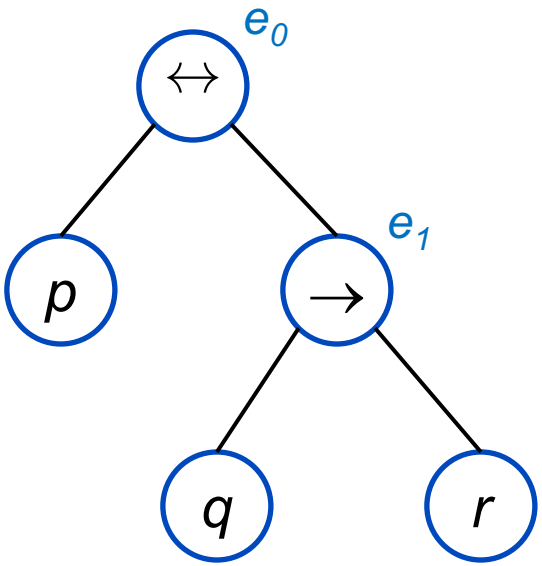


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

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

Tseitin Transformation: Example

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

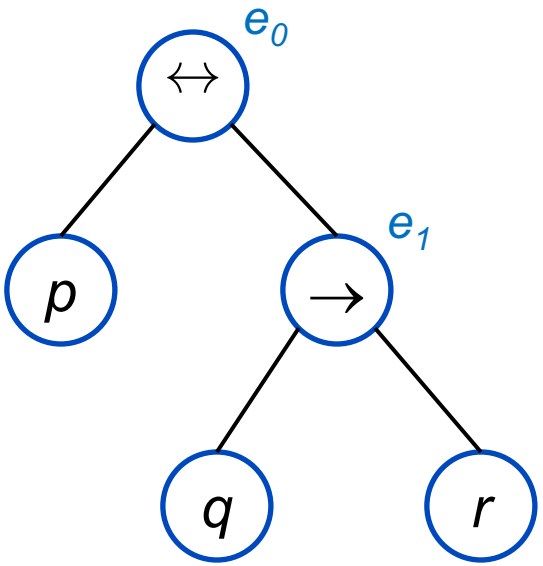


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

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

Tseitin Transformation: Example

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



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

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

Tseitin Transformation [1968]

Used in practice

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

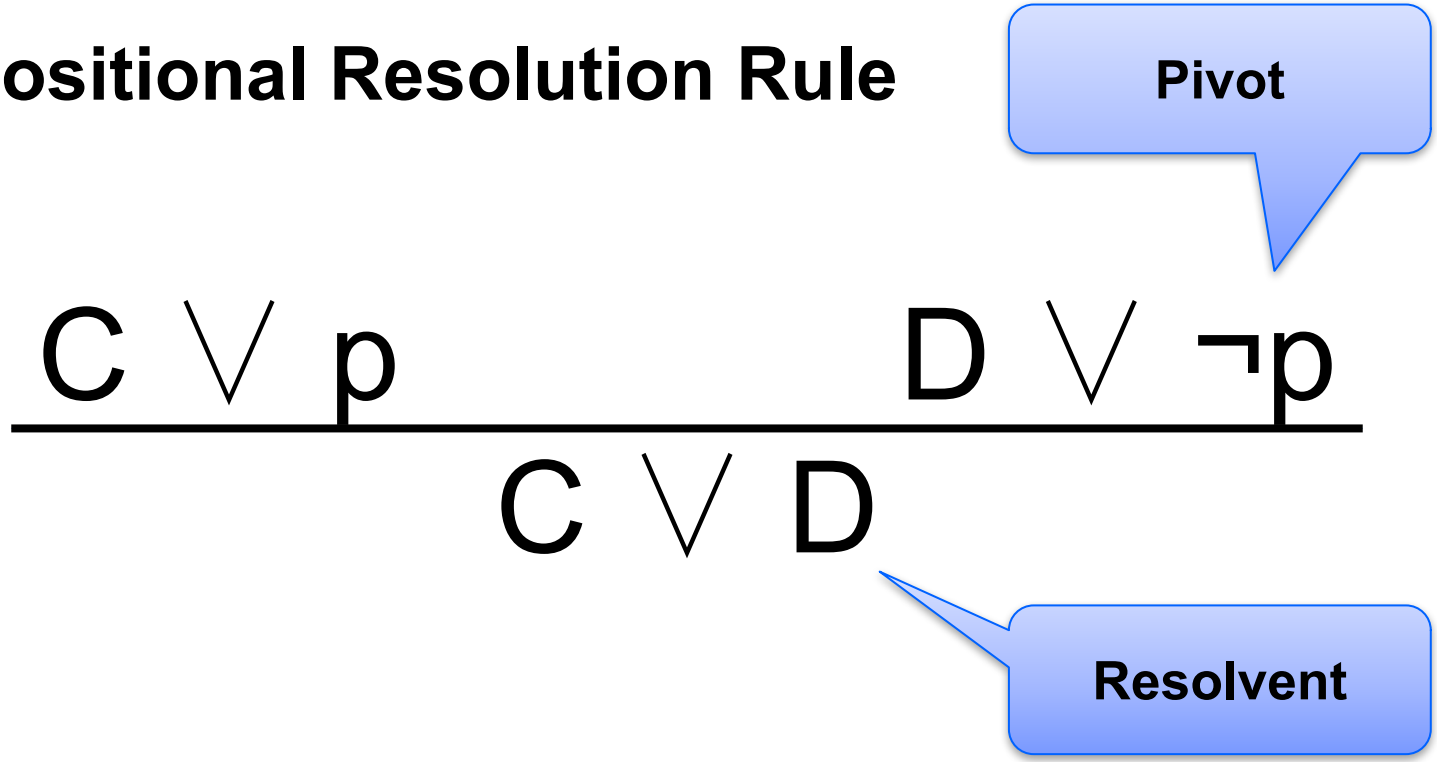
Does not produce an equivalent CNF

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

Propositional Resolution Rule

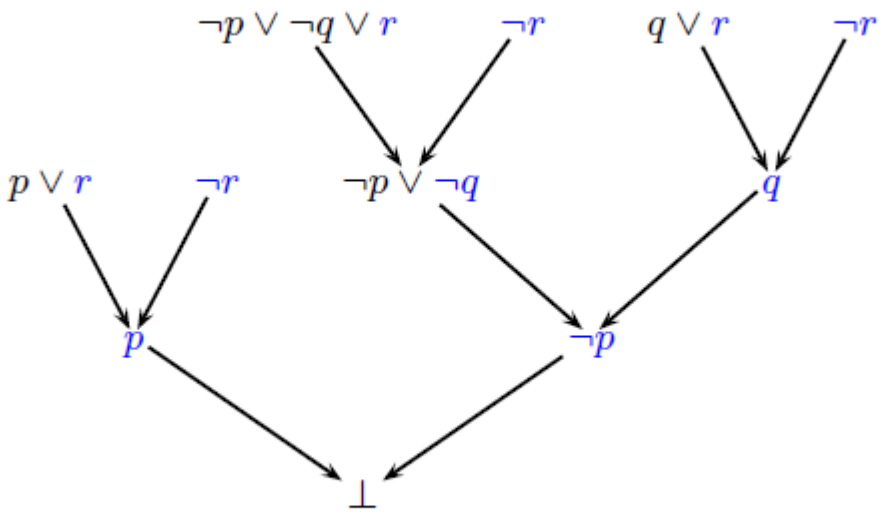


Soundness: Conjunction of the antecedents implies the resolvent.

Completeness: A set of clauses F is UNSAT if and only if there is a resolution derivation of an empty clause from it.

Resolution Proof

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



Completeness Theorem. A set of clauses F is UNSAT iff there is a resolution derivation of an empty clause from it

Resolution Proof Example

Show by resolution that the following CNF is UNSAT

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

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

SAT solving by resolution

Formula must be in CNF

Resolution rule: $\dfrac{C \vee p \qquad D \vee \neg p}{C \vee D}$

Example: $\dfrac{q \vee t \vee p \qquad q \vee r \vee \neg p}{q \vee t \vee r}$

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: $\dfrac{C \vee \ell \qquad \neg\ell}{C \qquad \neg\ell}$ ($C \vee \ell$ is subsumed by C)

Input resolution: $\dfrac{C \vee \ell \qquad D \vee \neg\ell}{C \vee D}$ ($C \vee \ell$ 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.

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

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

Ingredient of most efficient SAT solvers

The original DPLL procedure

Tries to build incrementally a satisfying truth assignment M for a CNF formula F

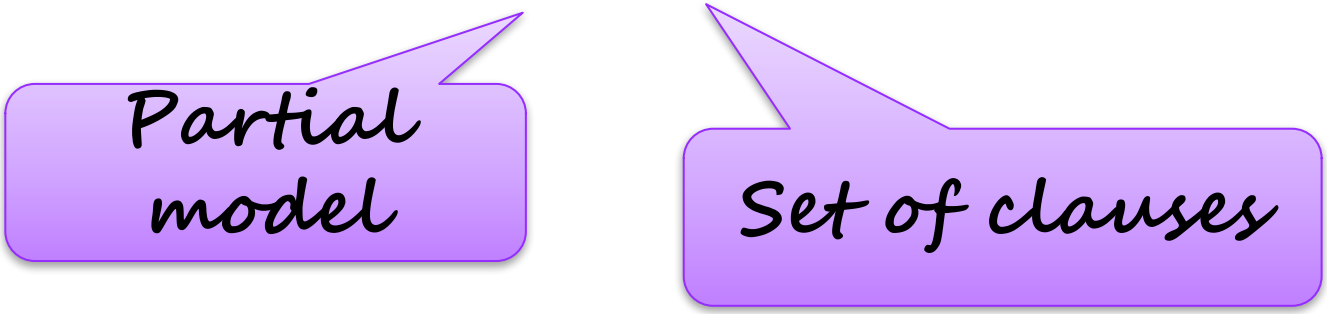
M is grown by

- 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

$$M \mid F$$



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)

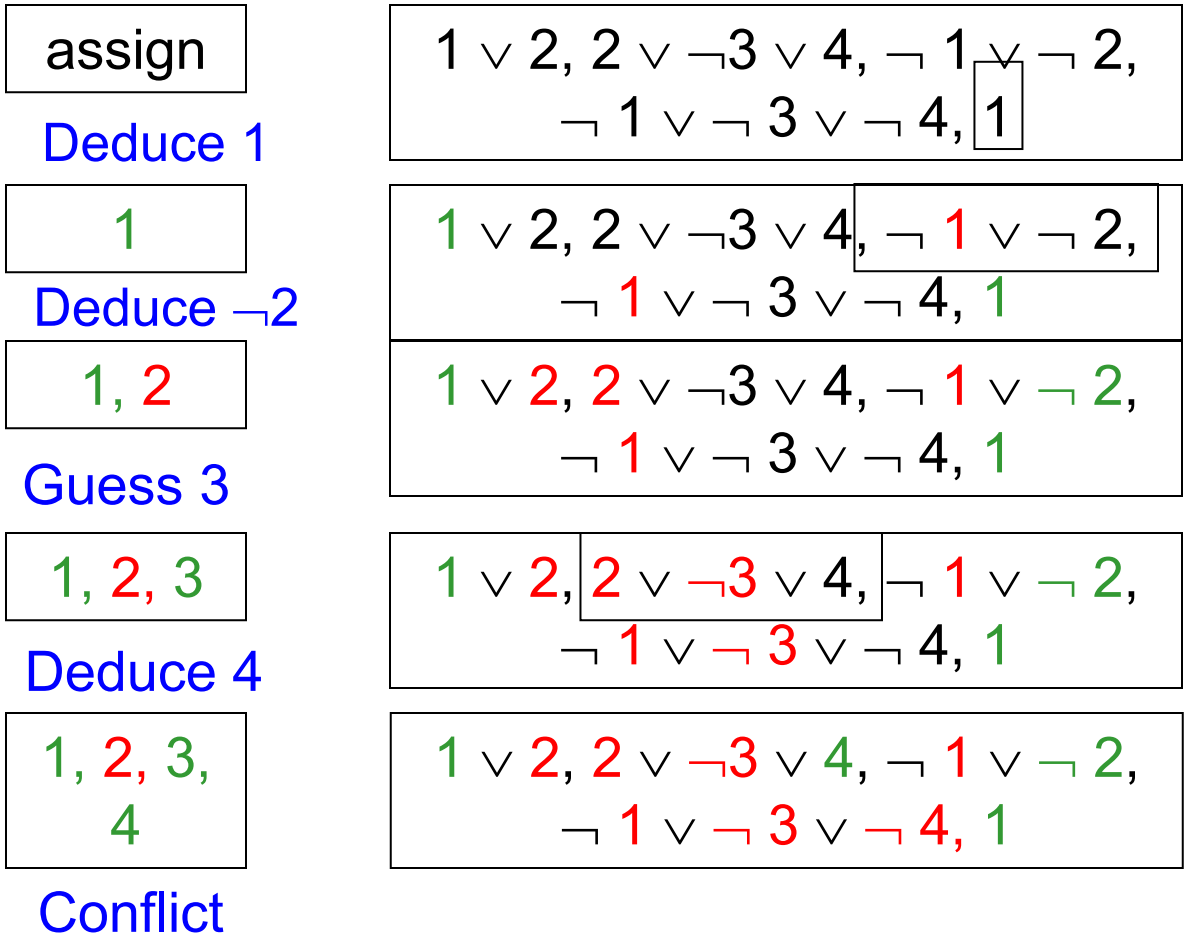
$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

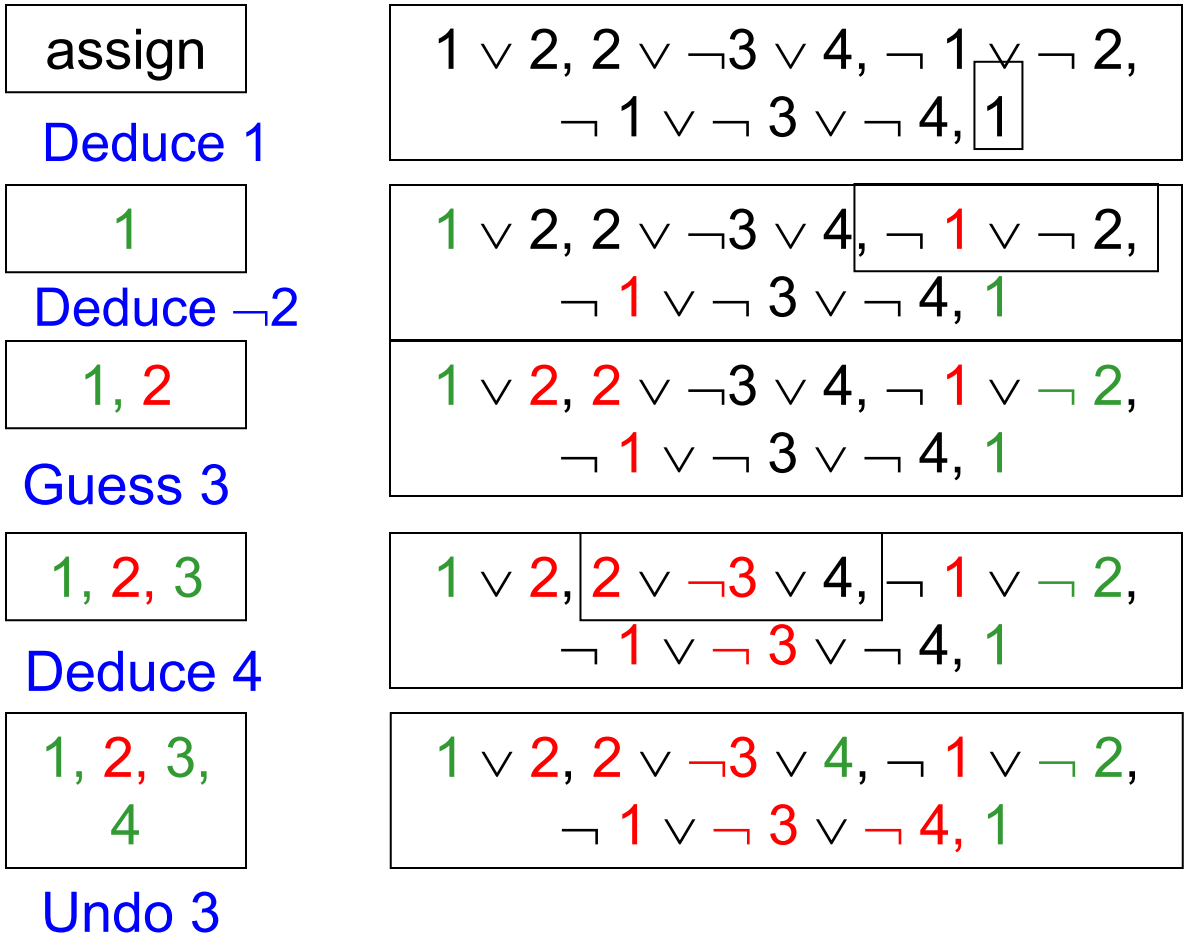
DPLL (as a procedure)

- 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 DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

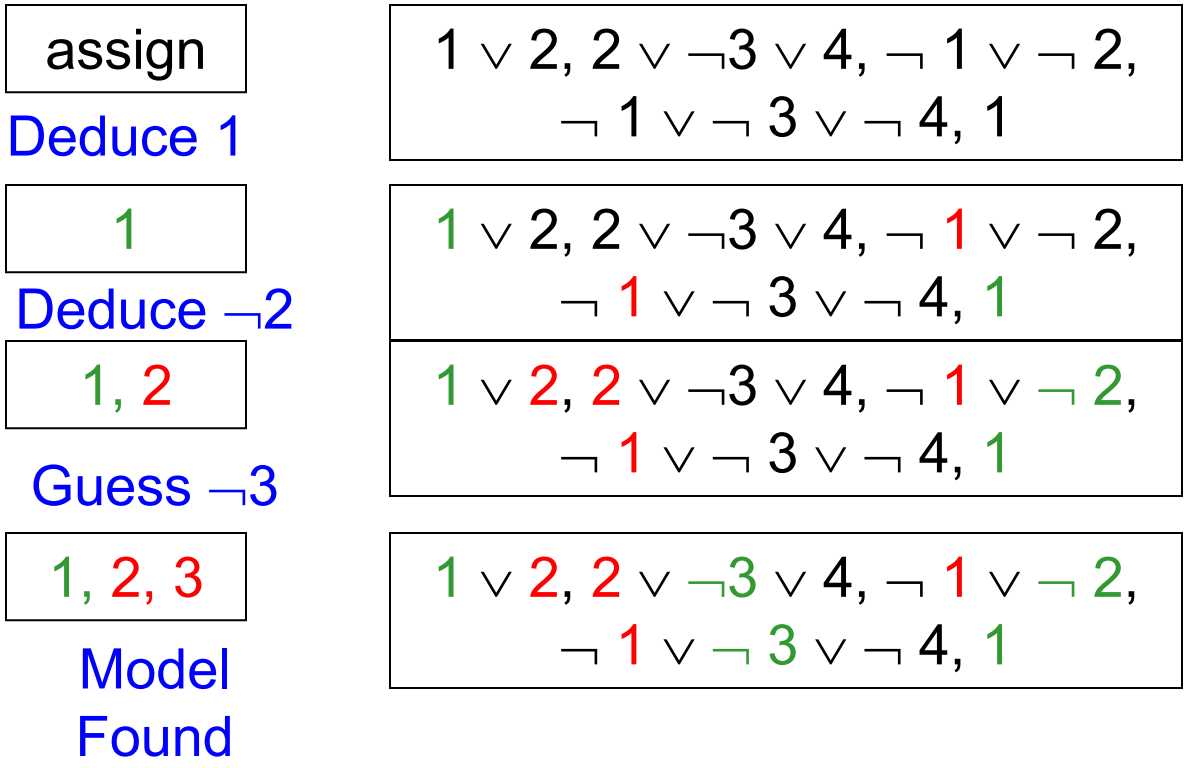
The Original DPLL Procedure – Example



The Original DPLL Procedure – Example



The Original DPLL Procedure – Example



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 the Original DPLL

Extending the assignment:

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

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

Notation: l^d is a decision literal

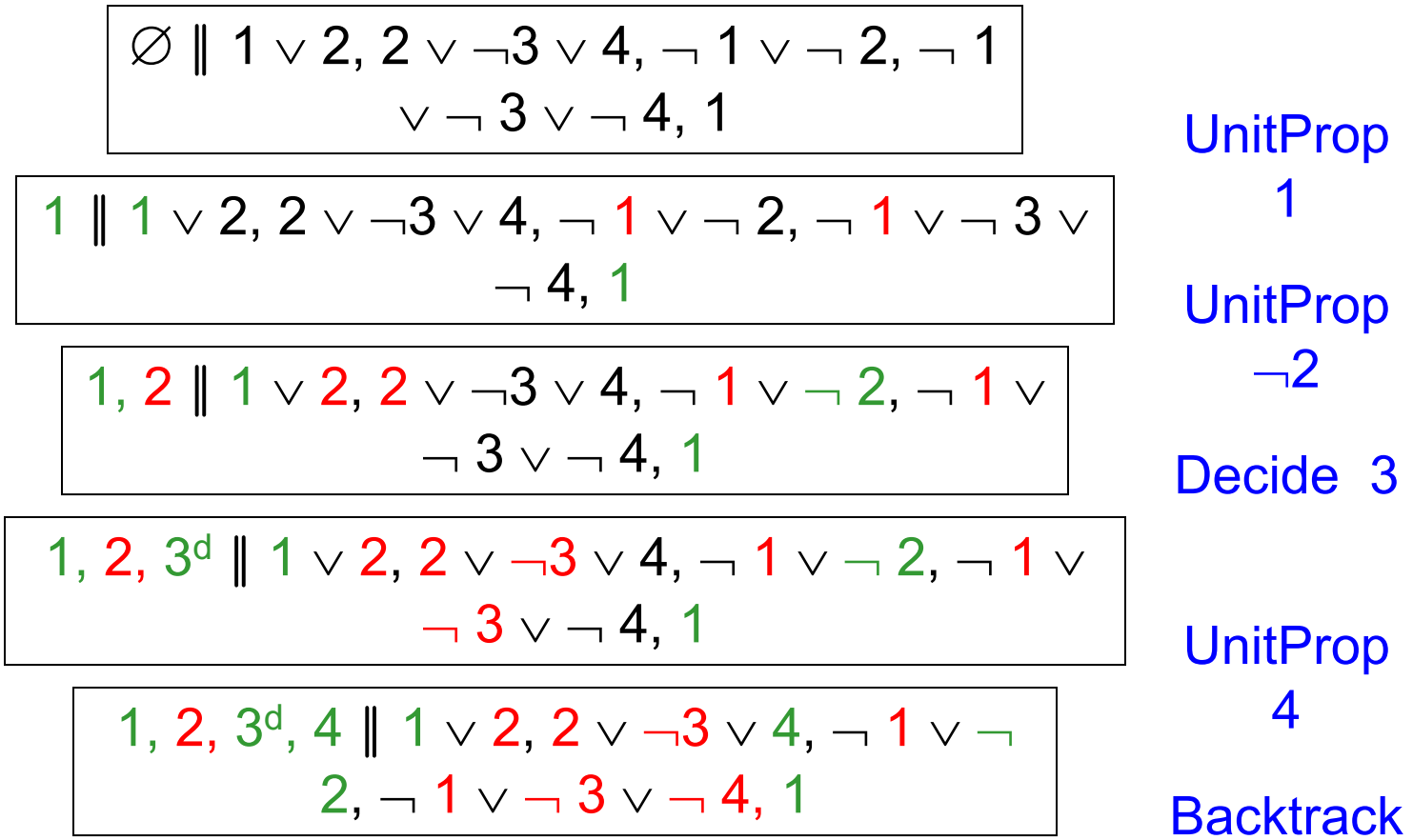
Transition Rules for the Original DPLL

Repairing the assignment:

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

Backtrack $\quad M\ l^d\ N \parallel F, C \rightarrow M\ \neg l \parallel F, C$ $\quad$ $M\ l^d\ N \models \neg C$; l is the last decision literal

Transition Rules DPLL – Example



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

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

Backtrack 3

Transition Rules for the Original DPLL

UnitProp

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

Decide

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

Fail

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

Backtrack

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

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 $\varnothing \parallel F$.
- 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 $\varnothing \parallel F$ and ending in $M \parallel F$, $M \models F$

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

Maintained in more general rules + theories

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

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



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



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



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

Learned Clause by Resolution

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

Modern CDCL

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

Model

Proof

Conflict Resolution

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

FIRST ORDER LOGIC

The language of First Order Logic

Functions , Variables, Predicates

- *f, g,…* *x, y, z, …* P, Q, =, <, …

Atomic formulas, Literals

- P(x,f(y)), ¬Q(y,z)

Quantifier free formulas

- P(f(a), b) ∧ c = g(d)

Formulas, sentences

- ∀x . ∀y . [P(x, f(x)) ∨ g(y,x) = h(y)]

Language: Signatures

A *signature* Σ is a finite set of:

- Function symbols:

$$\Sigma_F = \{ f, g, +, \ldots \}$$

- Predicate symbols:

$$\Sigma_P = \{ P, Q, =, \text{true}, \text{false}, \ldots \}$$

- And an *arity* function:

$$\Sigma \rightarrow \mathbb{N}$$

Function symbols with arity 0 are *constants*

- notation: $f_{/2}$ means a symbol with arity 2

A countable set *V* of *variables*

- disjoint from Σ

Language: Terms

The set of *terms* $T(\Sigma_F, V)$ is the smallest set formed by the syntax rules:

- $t \in T$ ::= v $\quad v \in V$
  | $f(t_1, \ldots, t_n)$ $\quad f \in \Sigma_F, t_1, \ldots, t_n \in T$

Ground terms are given by $T(\Sigma_F, \varnothing)$

Language: Atomic Formulas

$a \in Atoms \quad ::= P(t_1, \ldots, t_n)$

$P \in \Sigma_P \; t_1, \ldots, t_n \in T$

An atom is *ground* if $t_1, \ldots, t_n \in T(\Sigma_F, \varnothing)$

Literals are (negated) atoms:

$l \in Literals \quad ::= a \mid \neg a \qquad\qquad a \in Atoms$

Language: Quantifier free formulas

The set QFF(Σ,V) of *quantifier free formulas* is the smallest set such that:

$\varphi \in$ QFF	::= $a \in$ *Atoms*	*atoms*
	$\mid \neg\, \varphi$	*negations*
	$\mid \varphi \leftrightarrow \varphi'$	*bi-implications*
	$\mid \varphi \wedge \varphi'$	*conjunction*
	$\mid \varphi \vee \varphi'$	*disjunction*
	$\mid \varphi \rightarrow \varphi'$	*implication*

Language: Formulas

The set of *first-order formulas* are obtained by adding the formation rules:

$$\varphi ::= \ldots$$

$$\mid \quad \forall x \,.\, \varphi \qquad \textit{universal quant.}$$

$$\mid \quad \exists x \,.\, \varphi \qquad \textit{existential quant.}$$

Free (occurrences) of *variables* in a formula are theose not bound by a quantifier.

A sentence is a first-order formula with no free variables.

Dreadbury Mansion Mystery

Someone who lived in Dreadbury Mansion kill Aunt Agatha. Agatha, the Butler and Charles were the only people who lived in Dreadbury Mansion. A killer always hates his victim, and is never richer than his victim. Charles hates no one that aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler also hates everyone Agatha hates. No one hates everyone. Agatha is not the butler.

Who killed Aunt Agatha?



Dreadbury Mansion Mystery

$killed/_2, hates/_2, richer/_2, a/_0, b/_0, c/_0$

$$\exists x \cdot killed(x, a) \tag{1}$$

$$\forall x \cdot \forall y \cdot killed(x, y) \implies (hates(x, y) \wedge \neg richer(x, y)) \tag{2}$$

$$\forall x \cdot hates(a, x) \implies \neg hates(c, x) \tag{3}$$

$$hates(a, a) \wedge hates(a, c) \tag{4}$$

$$\forall x \cdot \neg richer(x, a) \implies hates(b, x) \tag{5}$$

$$\forall x \cdot hates(a, x) \implies hates(b, x) \tag{6}$$

$$\forall x \cdot \exists y \cdot \neg hates(x, y) \tag{7}$$

$$a \neq b \tag{8}$$



Models (Semantics)

A model *M* is defined as:

- Domain *S;* set of elements.
- Interpretation, $f^M : S^n \rightarrow S$ for each $f \in \Sigma_F$ with arity(f) = n
- Interpretation $P^M \subseteq S^n$ for each $P \in \Sigma_P$ with arity(P) = n
- Assignment $x^M \in S$ for every variable $x \in V$

A formula φ is true in a model *M* if it evaluates to true under the given interpretations over the domain *S.*

M is a *model for the theory T* if all sentences of *T* are true in *M.*

Models (Semantics)

A term t in a model M is interpreted as:

- Variable $x \in V$ is interpreted as x^M
- $f(t_1, \ldots, t_n)$ is interpreted as $f^M(a_1, \ldots, a_n)$,
  - where t_i is interpreted as a_i

An $P(t_1, \ldots, t_n)$ atom in a model M is interpreted as b, where

- $b \leftrightarrow (a_1, \ldots, a_n) \in P^M$
- t_i is interpreted as a_i

Models (Semantics)

A formula φ in a model M is interpreted as:

- $M \models \neg \varphi$ iff $M \not\models \varphi$ (M is not a model for φ)
- $M \models \varphi \leftrightarrow \varphi'$ iff $M \models \varphi$ is equivalent to $M \models \varphi'$
- $M \models \varphi \wedge \varphi'$ iff $M \models \varphi$ and $M \models \varphi'$
- $M \models \varphi \vee \varphi'$ iff $M \models \varphi$ or $M \models \varphi'$
- $M \models \varphi \rightarrow \varphi'$ iff $M \models \varphi$ implies $M \models \varphi'$
- $M \models \forall x.\varphi$ iff for all $s \in S$, $M[x:=s] \models \varphi$
- $M \models \exists x.\varphi$ iff exists $s \in S$, $M[x:=s] \models \varphi$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

If $M(x) = a, M(y) = b, M(z) = c$, then

$M[\![+(+(x, y), z)]\!] =$

$M(+)(M(+)(M(x), M(y)), M(z)) = M(+)(M(+)(a, b), c) =$

$M(+)(b, c) = a$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

$$M \models (\forall x : (\exists y : +(x, y) = 0))$$

$$M \not\models (\forall x : (\exists y : x < y))$$

$$M \models (\forall x : (\exists y : +(x, y) = x))$$

Semantics: Exercise

Drinker's paradox:

There is someone in the pub such that, if he is drinking, everyone in the pub is drinking.

- $\exists x.\ (D(x) \rightarrow \forall y.\ D(y))$

Is this logical formula valid?

Or unsatisfiable?

Or satisfiable but not valid?



Theories

A (first-order) *theory T (over signature Σ) is a set of (deductively closed) sentences (over Σ and V) - axioms*

Let *DC*(Γ) be the deductive closure of a set of sentences Γ.

- For every theory T, *DC*(T) = T

A theory T is *constistent* if *false* $\notin$ *T*

We can view a (first-order) theory *T* as the class of all *models* of *T* (due to completeness of first-order logic).

Theory of Equality T_E

Signature: $\Sigma_E = \{ =, a, b, c, \ldots, f, g, h, \ldots, P, Q, R, \ldots. \}$

=, a binary predicate, interpreted by axioms

all constant, function, and predicate symbols.

Axioms:

1. $\forall x \,.\, x = x$ (reflexivity)
2. $\forall x, y \,.\, x = y \rightarrow y = x$ (symmetry)
3. $\forall x, y, z \,.\, x = y \wedge y = z \rightarrow x = z$ (transitivity)

Theory of Equality T_E

Signature: $\Sigma_E = \{ =, a, b, c, \ldots, f, g, h, \ldots, P, Q, R, \ldots. \}$
=, a binary predicate, interpreted by axioms
all constant, function, and predicate symbols.
Axioms:

4. for each positive integer *n* and *n*-ary function symbol *f*,

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$ (congruence)

5. for each positive integer *n* and *n*-ary predicate symbol *P*

$\forall x_1, \ldots, x_n, y_1, \ldots, y_n \,.\, \bigwedge_i x_i = y_i \rightarrow (P(x_1, \ldots, x_n) \leftrightarrow P(y_1, \ldots, y_n))$ (equivalence)

Peano Arithmetic (Natural Number) – An Example for a Theory

Signature: $\Sigma_{PA} = \{ 0, 1, +, *, = \}$

Axioms of T_{PA} : axioms for theory of equality, T_E , plus:

1. $\forall x.\ \neg (x+ 1 = 0)$ (zero)
2. $\forall x, y.\ x + 1 = y + 1 \rightarrow x = y$ (successor)
3. $F[0] \ \wedge\ (\forall x.F[x] \rightarrow F[x+ 1]) \rightarrow \forall x.F[x]$ (induction)
4. $\forall x.\ x + 0 = x$ (plus zero)
5. $\forall x, y.\ x + (y + 1) = (x + y) + 1$ (plus successor)
6. $\forall x.\ x * 0 = 0$ (times zero)
7. $\forall x, y.\ x * (y + 1) = x * y + x$ (times successor)

Line 3 is an axiom schema.

Theory of Arrays T_A

Signature: Σ_A = { read, write, = }

read (a, i) is a binary function:

- reads an array a at the index I
- in SMT-LIB: *(select a i)* is used instead read(a,i)

write (a, i, v) is a ternary function:

- writes a value v to the index i of array a
- in SMT-LIB *(store a i v)* is used instead write(a, i, v)

Axioms of T_A

Array congruence

- $\forall a, i, j . i = j \rightarrow read(a, i) = read(a, j)$

Read-Over-Write 1

- $\forall a, v, i, j. i = j \rightarrow read(write(a, i, v), j) = v$

Read-Over-Write 2

- $\forall a, v, i, j. i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$

Extensionality

- $a = b \leftrightarrow \forall i . read(a, i) = read(b, i)$

T-Satisfiability

A formula $\varphi(x)$ is T-satisfiable in a theory T if there is a model of $DC^*(T \cup \exists\, x\, \varphi(x))$. That is, there is a model M for T in which $\varphi(x)$ evaluates to true.

Notation:

$$M \models_T \varphi(x)$$

**DC = deductive closure*

T-Validity

A formula $\varphi(x)$ is T-*valid* in a theory T if $\forall\, x\ \varphi(x) \in T$.

That is, $\varphi(x)$ evaluates to *true* in every model M of T.

T-validity:

$$\models_T \varphi(x)$$

Fragment of a Theory

Fragment of a theory *T* is a syntactically restricted subset of formulae of the theory

Example:

- Quantifier-free fragment of theory T is the set of formulae without quantifiers that are valid in T

Often decidable fragments for undecidable theories

Theory *T* is *decidable* if *T*-validity is decidable for every formula *F* of *T*

- There is an algorithm that always terminates with "yes" if *F* is *T*-valid, and "no" if *F* is *T*-unsatisfiable

Exercises (1/2)

Find a model for $P(f(x,y)) \Rightarrow P(g(x,y,x))$

Write an axiom that will restrict that every model has to have exactly three different elements.

Write a FOL formula stating that *i* is the position of the minimal element of an integer array *A*

Write a FOL formula stating that *v* is the minimal element of an integer array *A*

Exercises (2/2)

Show whether the following sentence is valid or not

$$(\exists x \cdot P(x) \vee Q(x)) \iff (\exists x \cdot P(x)) \vee (\exists x \cdot Q(x))$$

Show whether the following FOL sentence is valid or not

$$(\exists x \cdot P(x) \wedge Q(x)) \iff (\exists x \cdot P(x)) \wedge (\exists x \cdot Q(x))$$

Completeness, Compactness, Incompleteness

Gödel Completeness Theorem of FOL

- any (first-order) formula which is true in *all* models of a theory, must be logically deducible from that theory, and vice versa

Corollary: Compactness Theorem

- A FOL theory G is SAT iff every finite subset G' of G is SAT
- A set G of FOL sentences is UNSAT iff exists a finite subset G' of G that is UNSAT

Incompleteness of FOL Theories

- A theory is *consistent* if it is impossible to prove both *p* and ~*p* for any sentence *p* in the signature of the theory
- A theory is *complete* if for every sentence *p* it includes either *p* or ~*p*
- There are FOL theories that are consistent but incomplete

https://terrytao.wordpress.com/2009/04/10/the-completeness-and-compactness-theorems-of-first-order-logic/

https://plato.stanford.edu/entries/goedel-incompleteness/

SMT SOLVERS

Satisfiability Modulo Theory (SMT)

Satisfiability is the problem of determining wither a formula F has a model

- if F is ***propositional***, a model is a truth assignment to Boolean variables
- if F is ***first-order formula***, a model assigns values to variables and interpretation to all the function and predicate symbols

SAT Solvers

- check satisfiability of propositional formulas

SMT Solvers

- check satisfiability of formulas in a ***decidable*** first-order theory (e.g., linear arithmetic, uninterpreted functions, array theory, bit-vectors)

Background Reading: SMT



September 2011

UNIVERSITY OF WATERLOO

isfiability Modulo Theories: Introduction & Applications

Leonardo de Moura
Microsoft Research
One Microsoft Way
Redmond, WA 98052
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
One Microsoft Way
Redmond, WA 98052
nbjorner@microsoft.com

RACT

int satisfaction problems arise in many diverse ar-
uding software and hardware verification, type infer-
atic program analysis, test-case generation, schedul-
nning and graph problems. These areas share a
n trait, they include a core component using logical
s for describing states and transformations between
The most well-known constraint satisfaction problem
sitional satisfiability, SAT, where the goal is to de-
ether a formula over Boolean variables, formed using
connectives can be made *true* by choosing *true/false*
or its variables. Some problems are more naturally
d using richer languages, such as arithmetic. A sup-
theory (of arithmetic) is then required to capture
ning of these formulas. Solvers for such formulations
monly called *Satisfiability Modulo Theories* (SMT)

SMT solvers have been the focus of increased recent attention thanks to technological advances and industrial applications. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean Satisfiability with domains, such as, those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, completeness and incompleteness of logical theories, and finally complexity theory. In this article, we present an overview of the field of Satisfiability Modulo Theories, and some of its applications.

key driving factor [4]. An important ingredient is a common interchange format for benchmarks, called SMT-LIB [33], and the classification of benchmarks into various categories depending on which theories are required. Conversely, a growing number of applications are able to generate benchmarks in the SMT-LIB format to further inspire improving SMT solvers.

There is a relatively long tradition of using SMT solvers in select and specialized contexts. One prolific case is theorem proving systems such as ACL2 [26] and PVS [32]. These use decision procedures to discharge lemmas encountered during interactive proofs. SMT solvers have also been used for a long time in the context of program verification and *extended static checking* [21], where verification is focused on assertion checking. Recent progress in SMT solvers, however, has enabled their use in a set of diverse applications, including interactive theorem provers and extended static checkers, but also in the context of scheduling, planning, test-case generation, model-based testing and program development, static program analysis, program synthesis, and run-time analysis, among several others.

We begin by introducing a motivating application and a simple instance of it that we will use as a running example.

1.1 An SMT Application - Scheduling

Consider the classical *job shop scheduling* decision problem. In this problem, there are n jobs, each composed of m tasks of varying duration that have to be performed consecutively on m machines. The start of a new task can be delayed as long as needed in order to wait for a machine

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Arithmetic

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Array theory

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Uninterpreted function

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

Example

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), c - 2)) \neq f(c - b + 1)$$

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\texttt{read}(\texttt{write}(a, b, 3), b)) \neq f(3)$$

then, by the array theory axiom: $\texttt{read}(\texttt{write}(v, i, x), i) = x$

$$b + 2 = c \wedge f(3) \neq f(3)$$

then, the formula is unsatisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge y \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable:

Example model:

$$x \rightarrow 1$$

$$y \rightarrow 2$$

$$f(1) \rightarrow 0$$

$$f(2) \rightarrow 1$$

$$f(\ldots) \rightarrow 0$$

SMT - Milestones

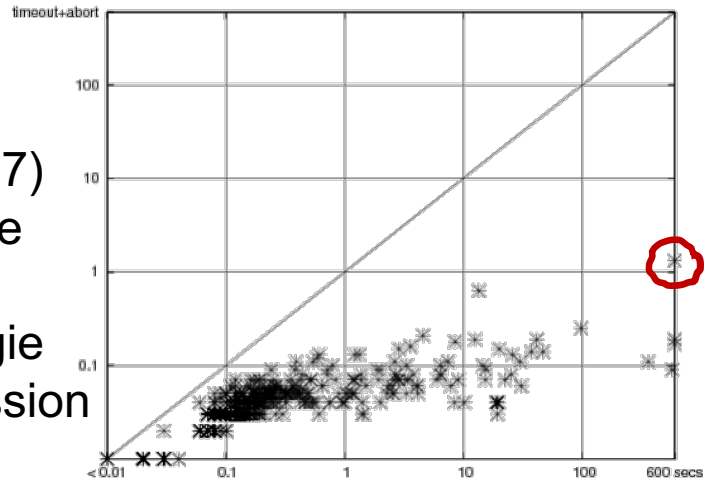
year	Milestone
1977	Efficient Equality Reasoning
1979	Theory Combination Foundations
1979	Arithmetic + Functions
1982	Combining Canonizing Solvers
1992-8	Systems: PVS, Simplify, STeP, SVC
2002	Theory Clause Learning
2005	SMT competition
2006	Efficient SAT + Simplex
2007	Efficient Equality Matching
2009	Combinatory Array Logic, …

Includes progress from SAT:



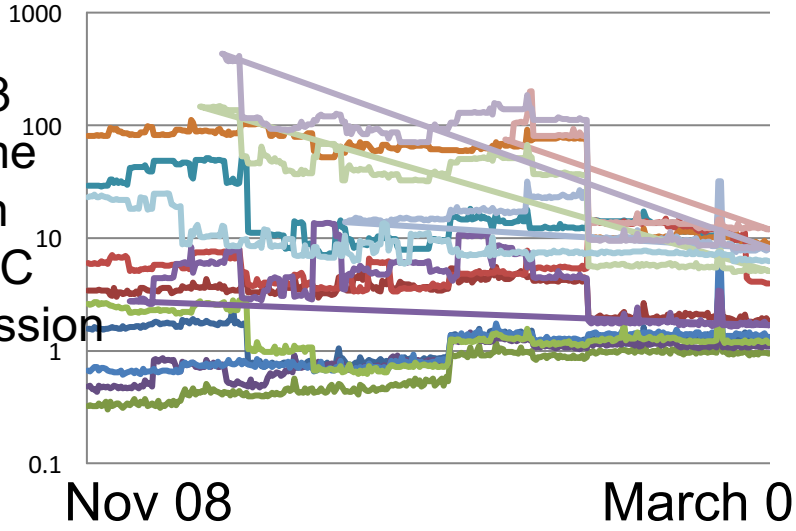
15KLOC + 285KLOC = Z3

Z3 (of '07) Time On Boogie Regression



Simplify (of '01) time

Z3 Time On VCC Regression



SMT-LIB: http://smt-lib.org

International initiative for facilitating research and development in SMT

Provides rigorous definition of syntax and semantics for theories

SMT-LIB syntax

- based on s-expressions (LISP-like)
- common syntax for interpreted functions of different theories
  - e.g. (and (= x y) (<= (* 2 x) z))
- commands to interact with the solver
  - (declare-fun …) declares a constant/function symbol
  - (assert p) conjoins formula p to the curent context
  - (check-sat) checks satisfiability of the current context
  - (get-model) prints current model (if the context is satisfiable)

- see examples at http://rise4fun.com/z3

SMT-LIB Syntax

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
```

SMT Example

z3 Microsoft Research

Is this formula satisfiable?

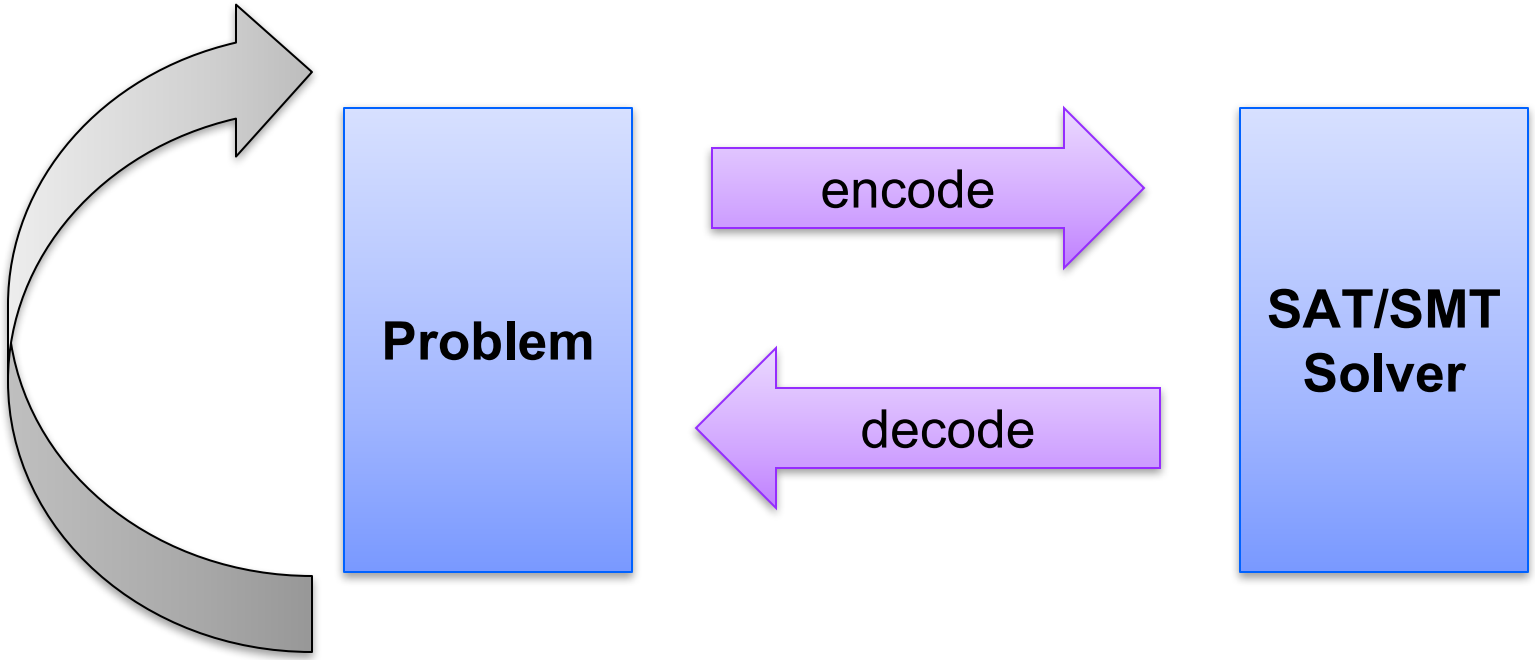
```
1  ; This example illustrates basic arithmetic and
2  ; uninterpreted functions
3
4  (declare-fun x () Int)
5  (declare-fun y () Int)
6  (declare-fun z () Int)
7  (assert (>= (* 2 x) (+ y z)))
8  (declare-fun f (Int) Int)
9  (declare-fun g (Int Int) Int)
10 (assert (< (f x) (g x x)))
11 (assert (> (f y) (g x x)))
12 (check-sat)
13 (get-model)
14 (push)
15 (assert (= x y))
16 (check-sat)
17 (pop)
18 (exit)
19
```

http://rise4fun.com/z3

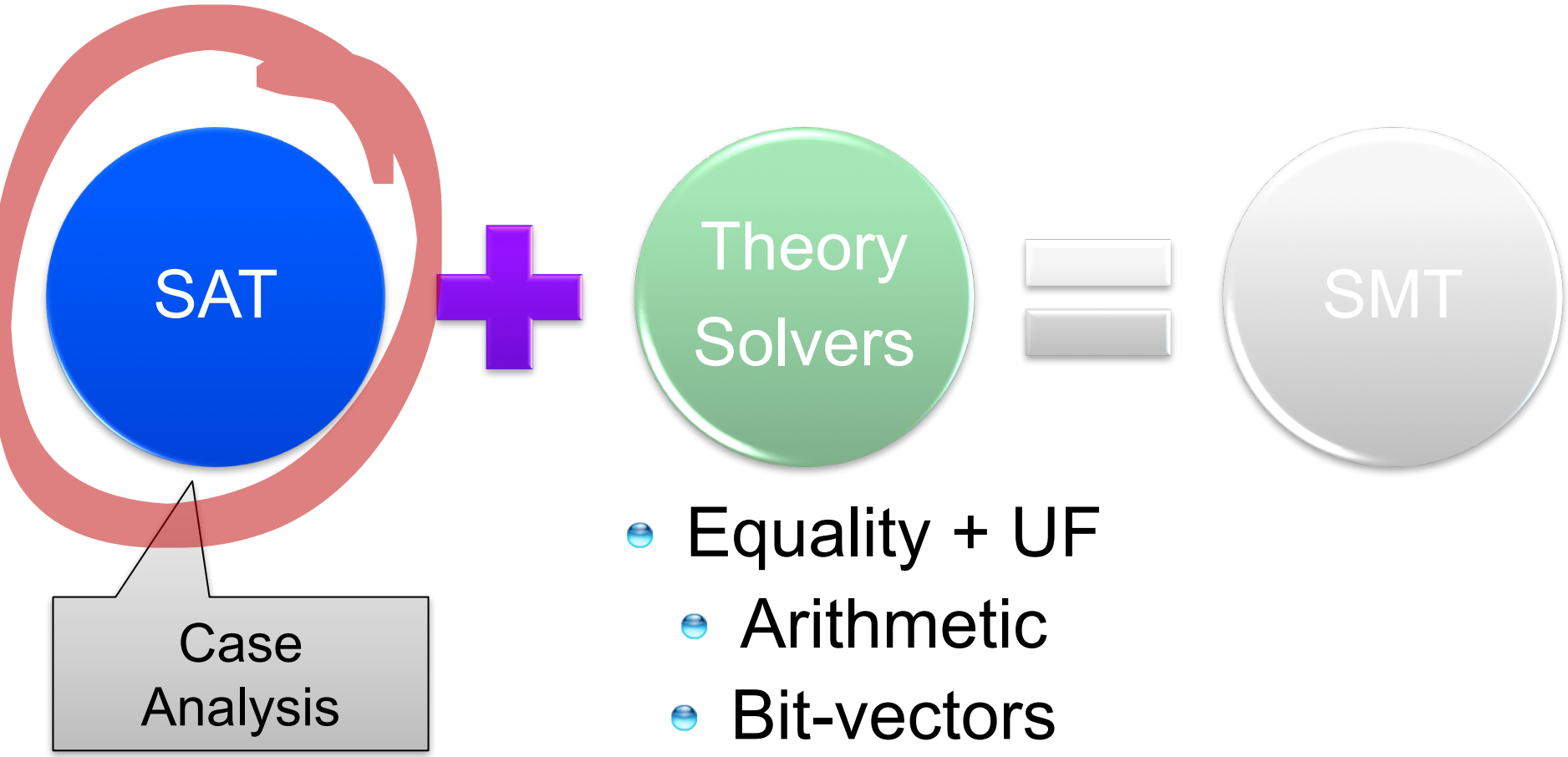
SAT/SMT Revolution

Solve any computational problem by effective reduction to SAT/SMT

- iterate as necessary



SMT : Basic Architecture



- Equality + UF
  - Arithmetic
  - Bit-vectors
    - …

SAT + Theory solvers

Basic Idea

$x \geq 0, y = x + 1, (y > 2 \vee y < 1)$



Abstract (aka “naming” atoms)

$p_1, p_2, (p_3 \vee p_4)$ $p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2), p_4 \equiv (y < 1)$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$



Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$ $\quad p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$



SAT Solver

SAT + Theory solvers

Basic Idea

$x \geq 0, y = x + 1, (y > 2 \vee y < 1)$



Abstract (aka "naming" atoms)

$p_1, p_2, (p_3 \vee p_4)$ $p_1 \equiv (x \geq 0), p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2), p_4 \equiv (y < 1)$



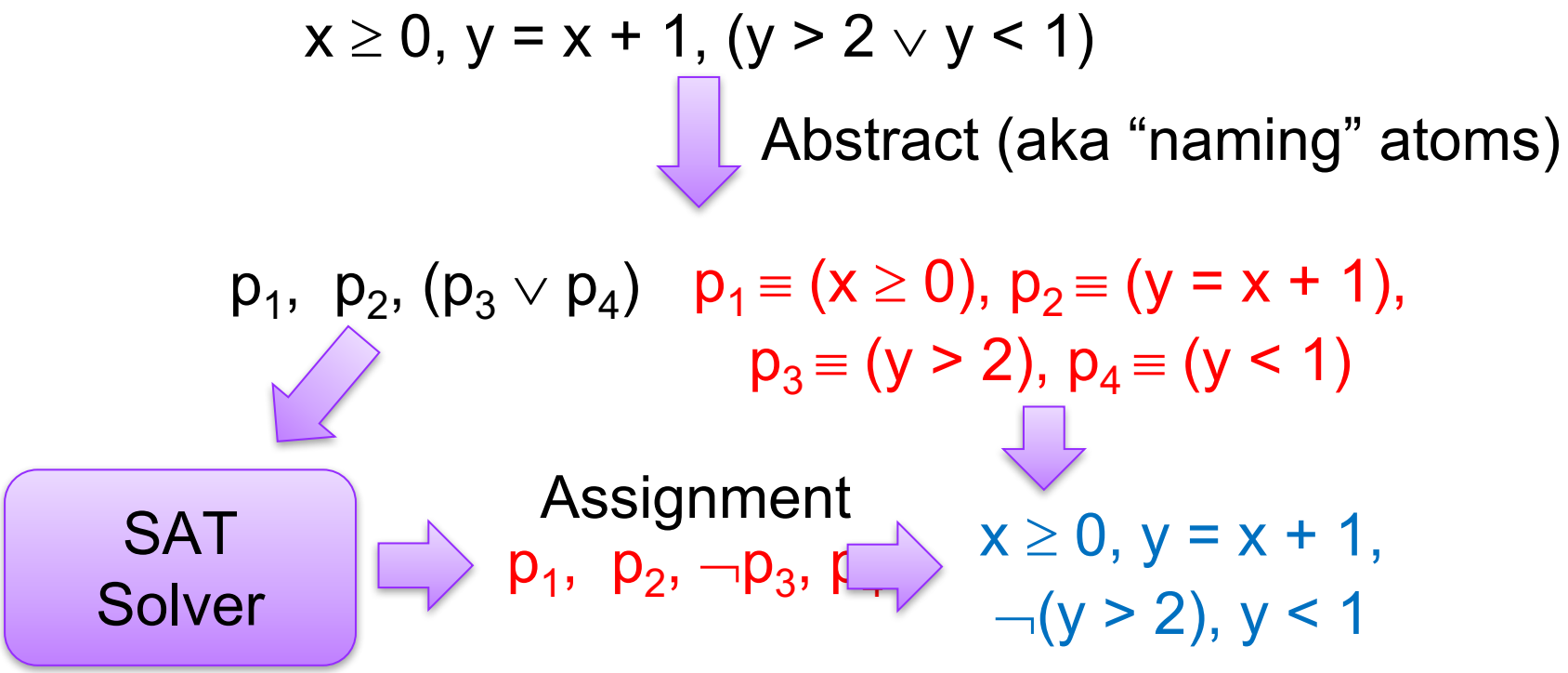
SAT Solver



Assignment
$p_1, p_2, \neg p_3, p_4$

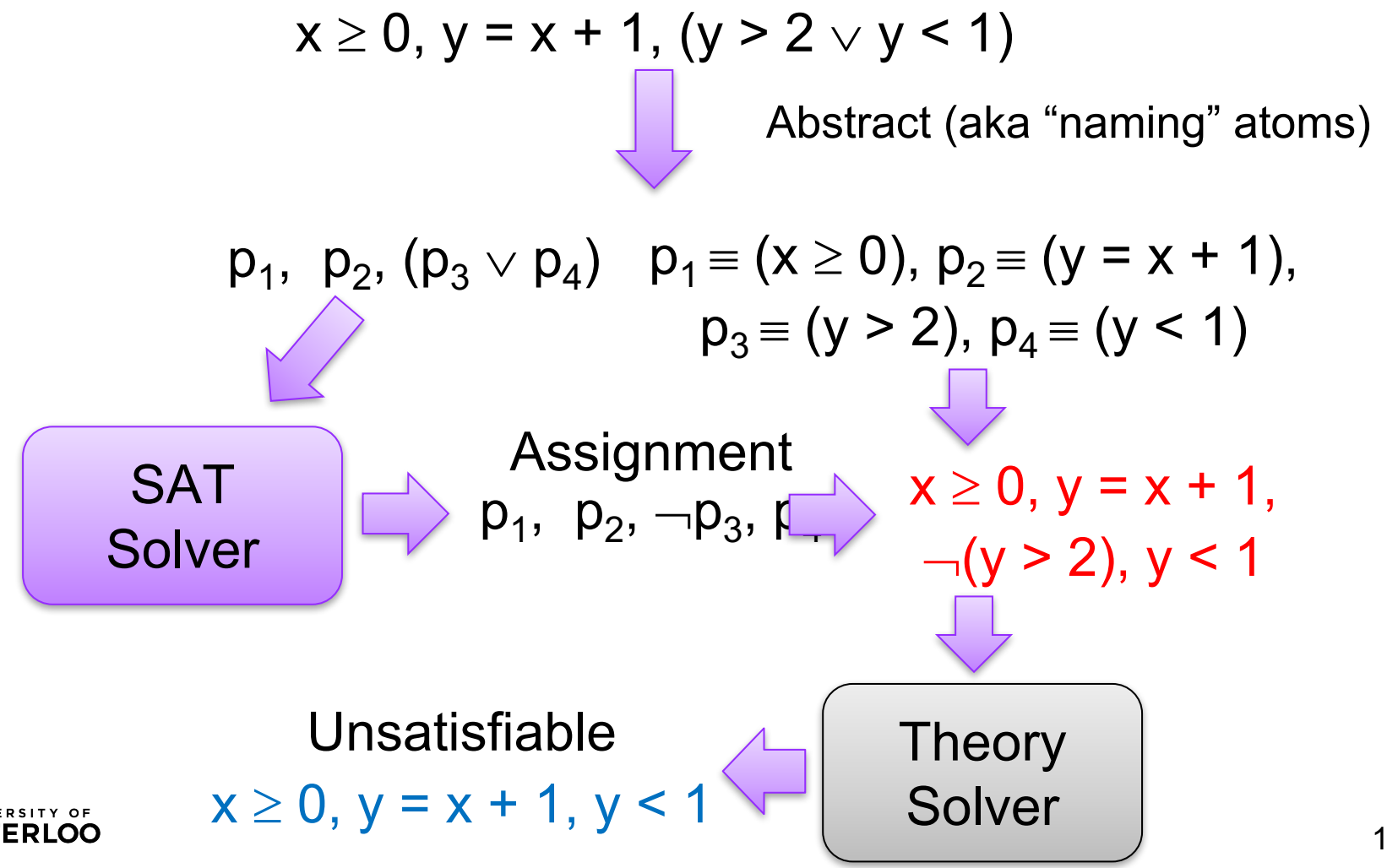
SAT + Theory solvers

Basic Idea



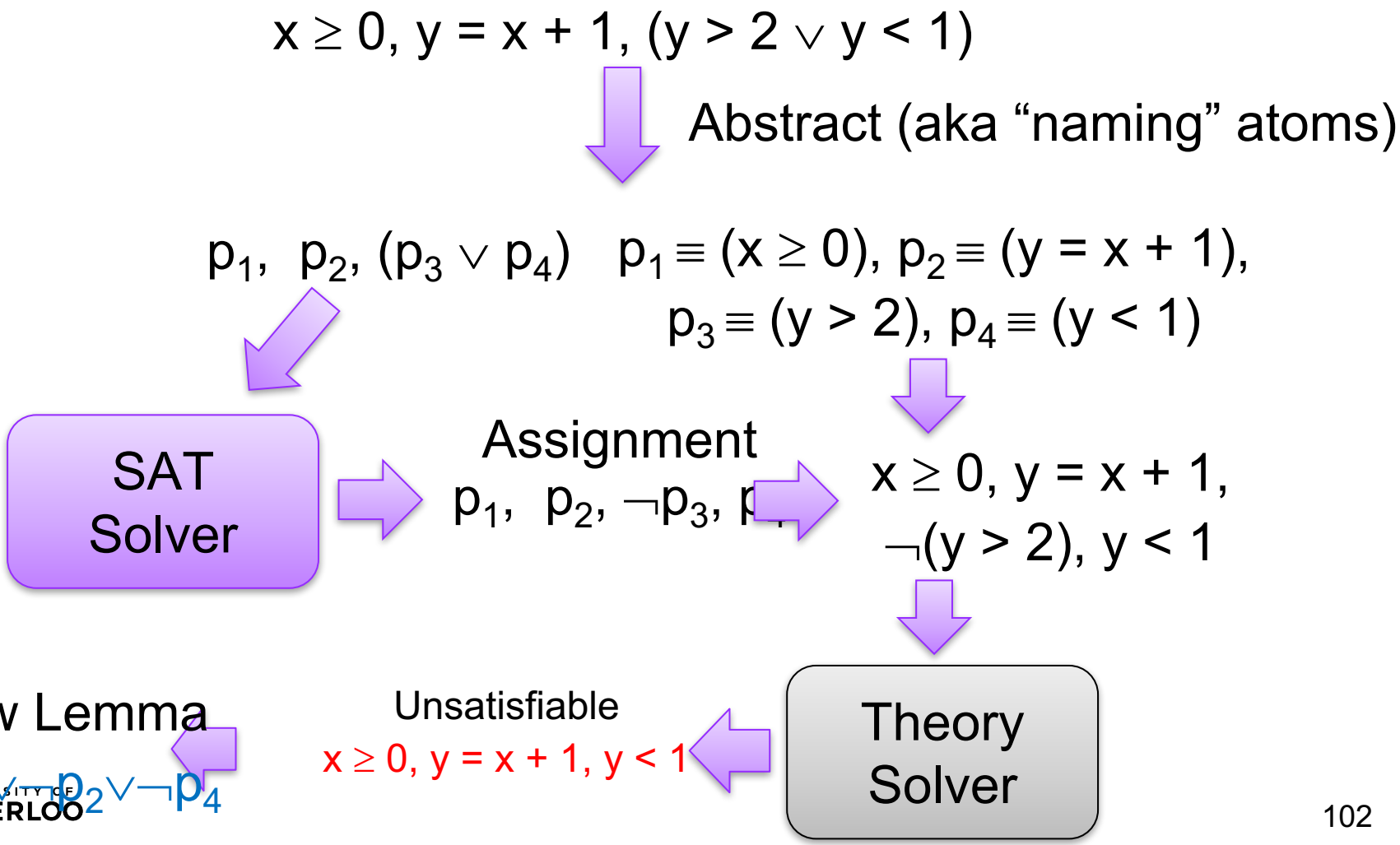
SAT + Theory solvers

Basic Idea

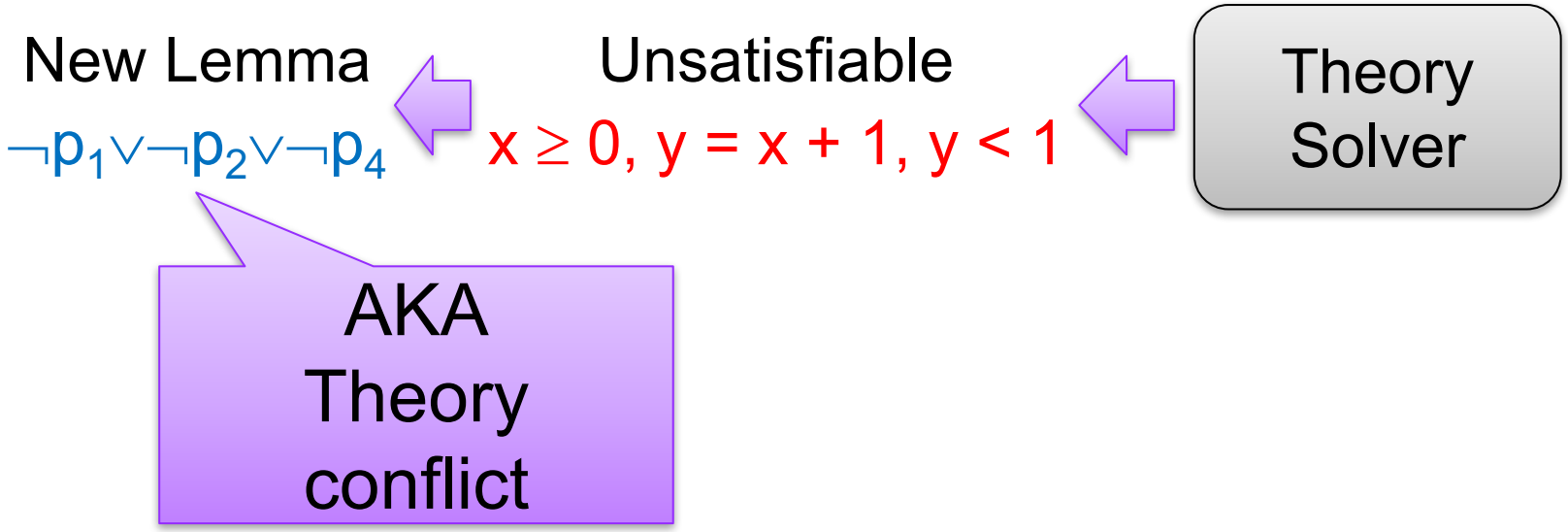


SAT + Theory solvers

Basic Idea



SAT + Theory solvers

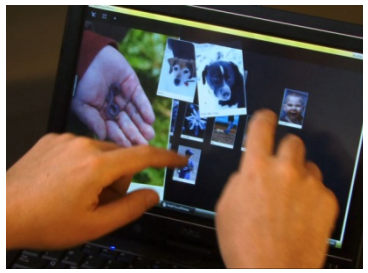


USING Z3 AND Z3PY

Job Shop Scheduling





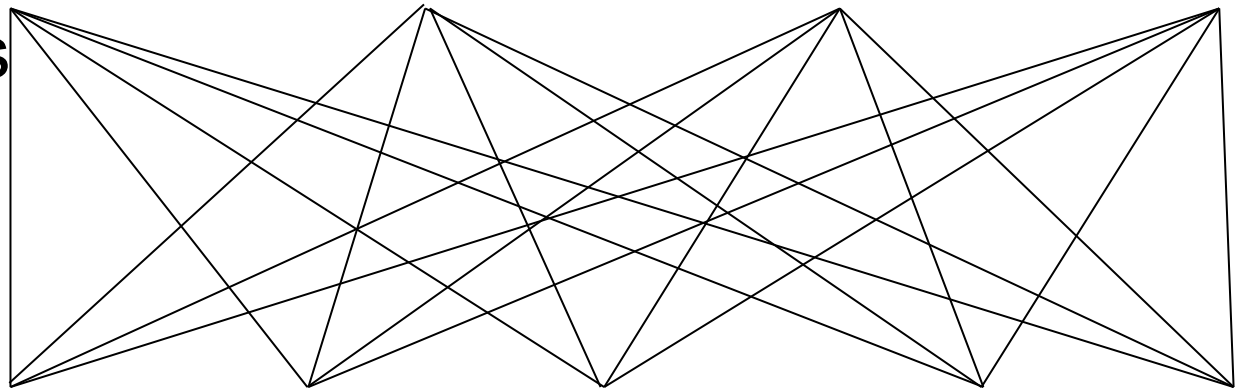




Machines

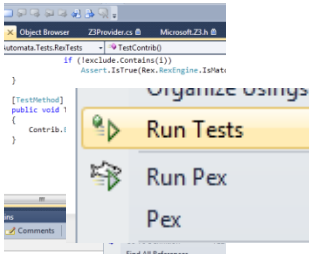
Tasks

Jobs



P = NP?





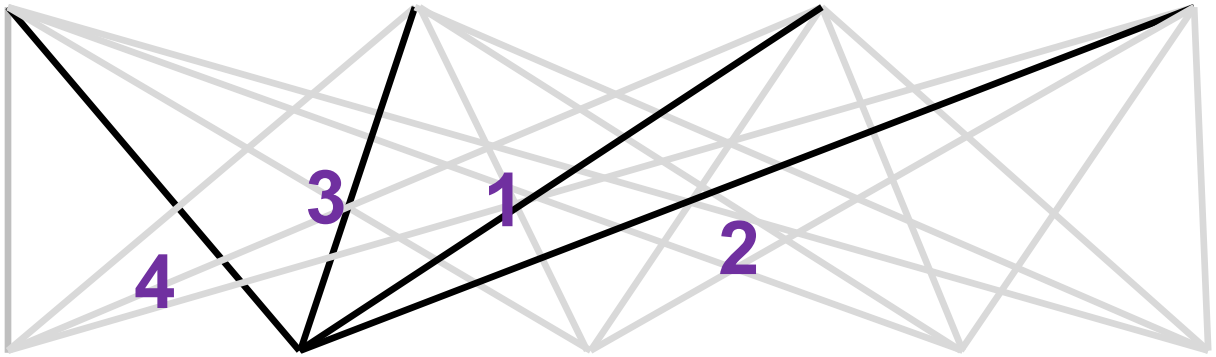


$$\zeta(s) = 0 \Rightarrow s = \frac{1}{2} + ir$$

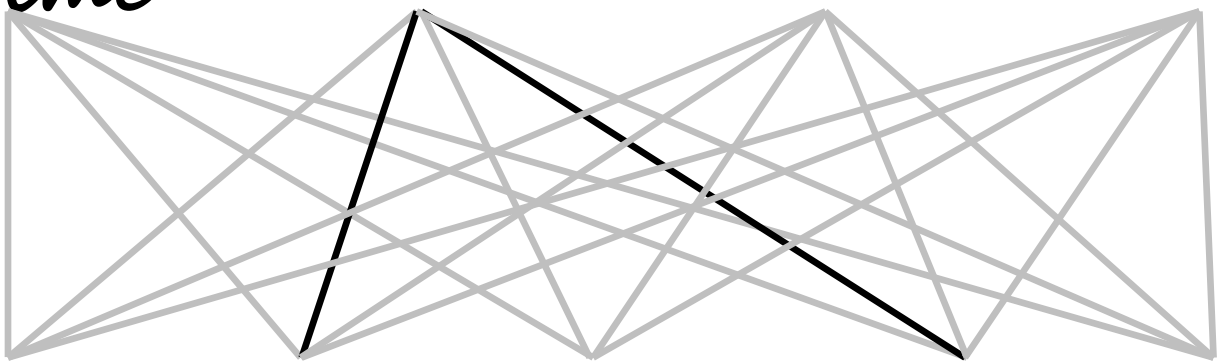
Job Shop Scheduling

Constraints:

Precedence: between two tasks of the same job



Resource: Machines execute at most one job at a time

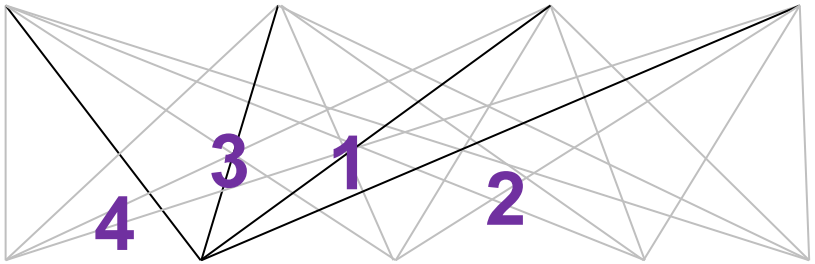


$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

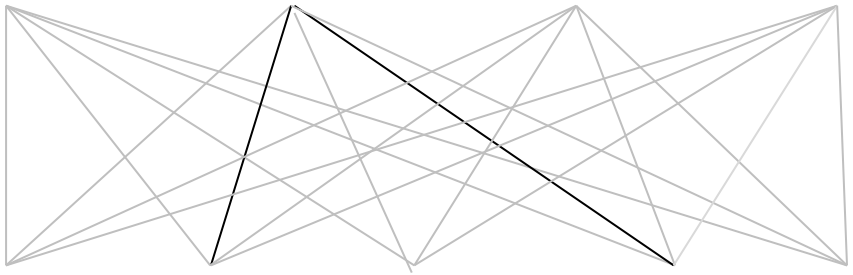
Job Shop Scheduling

Constraints:

Precedence:



Resource:



$$[start_{2,2}..end_{2,2}] \cap [start_{4,2}..end_{4,2}] = \emptyset$$

Encoding:

$t_{2,3}$ - start time of job 2 on mach 3

$d_{2,3}$ - duration of job 2 on mach 3

$$t_{2,3} + d_{2,3} \leq t_{2,4}$$

Not convex

$$t_{2,2} + d_{2,2} \leq t_{4,2}$$
$$\vee$$
$$t_{4,2} + \mathrm{d}_{4,2} \leq t_{2,2}$$

Job Shop Scheduling

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution

$t_{1,1} = 5,\ t_{1,2} = 7,\ t_{2,1} = 2,$
$t_{2,2} = 6,\ t_{3,1} = 0,\ t_{3,2} = 3$

Encoding

$$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge$$
$$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge$$
$$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge$$
$$((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge$$
$$((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge$$
$$((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge$$
$$((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge$$
$$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge$$
$$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$$