GLOBAL INITIATIVE OF ACADEMIC NETWORKS

University of Illinois at Urbana-Champaign

Tutorial 1: Modern SMT Solvers and Verification

Sayan Mitra

Electrical & Computer Engineering

Coordinated Science Laboratory

University of Illinois at Urbana Champaign

Tutorial 1: Modern SMT Solvers and Verification

Sayan Mitra

slides adapted from a lecture by Clark Barrett

Plan

- SAT problem
  - Logic and circuit representation
  - Conversion to CNF
  - DPLL
  - Modeling and BMC using SAT (Z3)
- SMT
  - Architecture
  - Theories
  - Examples

The satisfiability problem

SAT Problem: Given a well-formed formula α in propositional logic, decide whether there exists a satisfying solution for α.

Example: $\alpha(x_1, x_2, \ldots, x_n) \coloneqq (x_1 \wedge x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3 \vee x_2)$

Satisfying solution: $(x_1 = 1; x_2 = 1; x_3 = 0)$

Complexity: 2^n

First problem shown to be NP-complete [Cook'71]

Though exponential, makes sense to build SAT-solvers and 30+ years of engineering has led to solvers that can solve practical problems

SAT in Verification

Reachability and invariance questions automata can be encoded as SAT questions

Q. U is (not) reachable from Q_0 in n steps:

$F_{Q_0}(X_0) \wedge F_T(X_0, X_1) \wedge F_T(X_1, X_2) \wedge F_T(X_2, X_3) \wedge \cdots \wedge F_T(X_{n-1}, X_n) \wedge F_U(X_n)$
SAT iff U is reachable (UNSAT iff not reachable)

Q. I is (not) an inductive invariant:

$F_{Q_0}(X) \rightarrow F_I(X) \wedge F_I(X) \wedge F_T(X, X') \rightarrow F_I(X')$

Terminology

variables: x_1, x_2

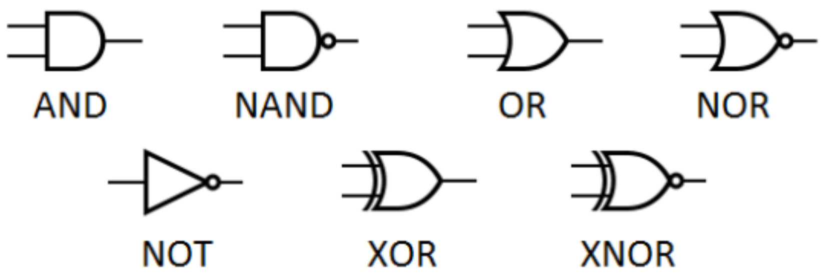
literals: positive or negative appearance of variables in a formula, e.g., $x_1, \neg x_2,$

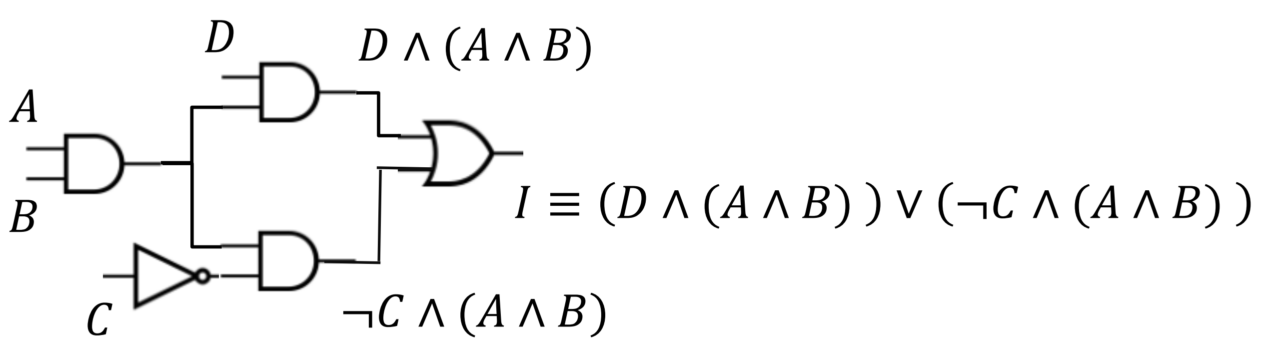
clause: disjunction of literals, e.g. $(x_1 \vee \neg x_2 \vee x_3)$

conjunctive normal form (CNF) formula: E.g., $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_1)$

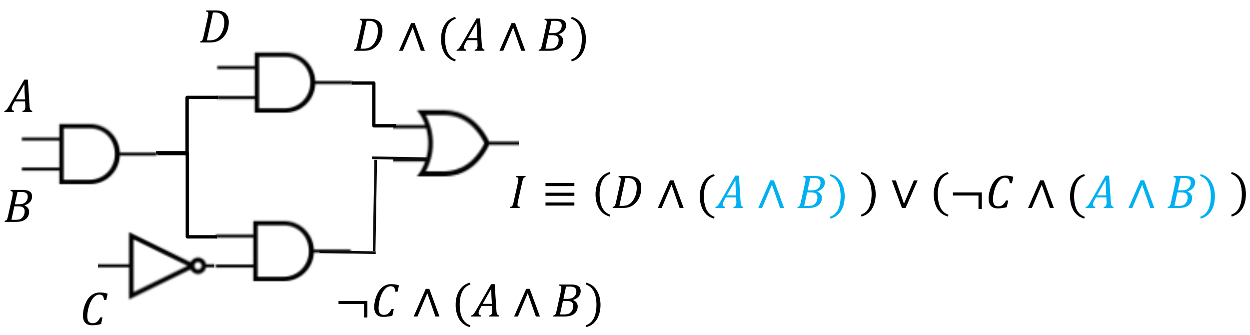
we will assume α to be in CNF

Propositional logic and circuits





Propositional logic and circuits



Overcome inefficiency by renaming subexpressions

Tautologically equivalent: Every satisfying solution of I is a satisfying solution of I'

Equisatisfiable: I is satisfiable iff I' is satisfiable

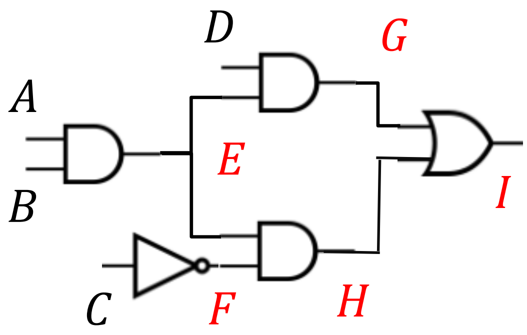
$(A \wedge B) \leftrightarrow E$

$I' \equiv (D \wedge E) \vee (\neg C \wedge E) \wedge ((A \wedge B) \leftrightarrow E)$

I' and I are not tautologically equivalent, but are equisatisfiable (e.g., $C = 0; A = B = 1; E = 0$ satisfies I)

Slides by Sayan Mitra (mitras@illinois.edu)

Converting to CNF



View formula as a circuit

1. Give new names to non-leaf nodes
2. For each non-leaf add conjunction of I/O clauses
3. Take conjunction of everything

$E \leftrightarrow (A \wedge B)$

$\equiv (\neg(A \wedge B) \vee E) \wedge \left(\neg E \vee (A \wedge B)\right)$

$\equiv (\neg A \vee \neg B \vee E) \wedge \left((\neg E \vee A) \wedge (\neg E \vee B)\right)$

$G \leftrightarrow (D \wedge E)$

$\neg F \leftrightarrow C$

$H \leftrightarrow (F \wedge E)$

$I \leftrightarrow (H \vee G)$

SMT formats

Alternative notations

- $(\neg A \vee \neg B \vee E) \wedge (\neg E \vee A) \wedge (\neg E \vee B) \wedge (\neg D \vee E)$
- $(A' \vee B' \vee E)(E' \vee A)(E' \vee B)(D' \vee E)$
- $(-1\ \ -2\ \ 5)(-5\ 1) \wedge (-5\ 2) \wedge (-4\ 5)$ [DIMACS]

SAT solving algorithms

- Davis Putnam Logemann Loveland (DPLL) 1962
- Davis Putnam algorithm (DP) 1960

Basic idea: Given α perform a sequence of satisfiability preserving transformations; if this ends with empty clause then UNSAT and if this ends with no clauses then SAT

The DP rules

1. Unit propagation: If a clause has a single literal p then
   - remove all instances of $\neg p$ from all clauses
   - remove all clauses with p
2. Pure literal: If a variable appears only positively or negatively in all clauses then delete all clauses containing that literal
3. Resolution: Choose literal p (appears both positively and negatively)
   - Let P be the set of clauses in which p is +ve
   - Let N be the set of clauses in which p is –ve
   - Replace P, N with clauses obtained by resolving p in all pairs
   - For a single pair $(p \vee \ell_1 \vee \ell_2 \ldots \ell_m)$; $(\neg p \vee k_1 \vee k_2 \ldots k_n)$ resolved clause is $(\ell_1 \vee \ell_2 \ldots \ell_m \vee k_1 \vee k_2 \ldots k_n)$
   - Quadratic blow-up in size of formula

Some experimental results

Problem	tautology	dptaut	dplltaut
prime 3	0.00	0.00	0.00
prime 4	0.02	0.06	0.04
prime 9	18.94	2.98	0.51
prime 10	11.40	3.03	0.96
prime 11	28.11	2.98	0.51
prime 16	>1 hour	out of memory	9.15
prime 17	>1 hour	out of memory	3.87
ramsey 3 3 5	0.03	0.06	0.02
ramsey 3 3 6	5.13	8.28	0.31
mk_adder_test 3 2	>>1 hour	6.50	7.34
mk_adder_test 4 2	>>1 hour	22.95	46.86
mk_adder_test 5 2	>>1 hour	44.83	170.98
mk_adder_test 5 3	>>1 hour	38.27	250.16
mk_adder_test 6 3	>>1 hour	out of memory	1186.4
mk_adder_test 7 3	>>1 hour	out of memory	3759.9

From talk by Clark Barrett

Incomplete SAT: GSAT [SLM92]

```
Input: a set of clauses F, MAX-FLIPS, MAX-TRIES
Output: a satisfying truth assignment of F
        or ∅, if none found
for i := 1 to MAX-TRIES
  v := a randomly generated truth assignment
  for j := 1 to MAX-FLIPS
    if v satisfies F then return v
    p := a propositional variable such that a
         change in its truth assignment gives the
         largest increase in the total number of
         clauses of F that are satisfied by v
    v := v with the assignment to p reversed
  end for
end for
return ∅
```

Stålmarck's Method [SS98]

Breadth-first approach instead of depth-first.

Dilemma Rule

Given a set of formulas Δ and any basic deduction algorithm, R, the dilemma rule performs a case split on some literal p by considering the new sets of formulas $\Delta \cup \{(\neg p)\}$ and $\Delta \cup \{(p)\}$.

To each of these sets, the algorithm R is applied to yield Δ_0 and Δ_1 respectively.

The original set Δ is then augmented with $\Delta_0 \cap \Delta_1$.

Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called *Abstract DPLL* [NOT06].

- Abstract DPLL uses *states* and *transitions* to model the progress of the algorithm.
- Most states are of the form $M \parallel F$, where
  - M is a *sequence of* annotated *literals* denoting a partial truth assignment, and
  - F is the CNF formula being checked, represented as a *set of clauses*.
- The *initial state* is $\emptyset \parallel F$, where F is to be checked for satisfiability.
- Transitions between states are defined by a set of *conditional transition rules*.

Abstract DPLL

The *final state* is either:

- a special fail state: $fail$, if F is unsatisfiable, or
- $M \parallel G$, where G is a CNF formula equisatisfiable with the original formula F, and M satisfies G

We write $M \models C$ to mean that for every truth assignment v, $v(M) = \mathit{True}$ implies $v(C) = \mathit{True}$.

Abstract DPLL Rules

UnitProp :

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

PureLiteral :

$$M \parallel F \quad \Longrightarrow \quad M\ l \parallel F \quad \text{if} \begin{cases} l \text{ occurs in some clause of } F \\ -l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Decide :

$$M \parallel F \quad \Longrightarrow \quad M\ l^{\mathsf{d}} \parallel F \quad \text{if} \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

Backtrack :

$$M\ l^{\mathsf{d}}\ N \parallel F, C \quad \Longrightarrow \quad M\ \neg l \parallel F, C \quad \text{if} \begin{cases} M\ l^{\mathsf{d}}\ N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

Fail :

$$M \parallel F, C \quad \Longrightarrow \quad \mathit{fail} \quad \text{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Example

$\emptyset \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Pureliteral 4)

$4 \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Decide 1)

$4\ 1^d \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Unitprop $\bar{2}$)

$4\ 1^d\ \bar{2} \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Unitprop 3)

$4\ 1^d\ \bar{2}\ 3 \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Backtrack)

$4\ \bar{1} \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Unitprop)

$4\ \bar{1}\ \bar{2}\ \bar{3} \quad | \; 1 \vee \bar{2}, \bar{1} \vee \bar{2}, 2 \vee 3, \bar{3} \vee 2, 1 \vee 4 \Rightarrow$ (Fail)

fail

Result: Unsatisfiable

Modeling for SAT

Input:
$\mathcal{A} = \langle Q, Q_0, T \subseteq Q \times Q \rangle$, Invariant I or unsafe set U

Output:
I is (not) an invariant of $\mathcal{A}$
U is (not) reachable from Q_0
U is (not) reachable from Q_0 in n steps

Modeling for SAT (2)

$\mathcal{A} = \langle Q, Q_0, T \subseteq Q \times Q \rangle$, Invariant I or unsafe set U
Assume Q is finite
Select k such that $|Q| \leq 2^k$
Define state variables $\mathrm{X} = \{x_1, x_2, \ldots, x_k\}, type(x_i) = \{0,1\}$
Then, $Q = val(X)$
$Q_0 \longmapsto F_{Q_0}(X)$ a formula encoding initial set
$U \longmapsto F_U(X)$ a formula encoding unsafe set
Define additional vars $Y = \{y_1, y_2, \ldots, y_k\}, type(y_i) = \{0,1\}$
$T \longmapsto F_T(X, Y)$ a formula encoding transition relation

Bounded model checking

$Q_0 \;\mapsto F_{Q_0}(X)$ a formula encoding initial set
$U \;\mapsto F_U(X)$ a formula encoding unsafe set
$T \;\mapsto F_T(X, Y)$ a formula encoding transition relation
We need $n + 1$ copies of variables: $X_0 = \{x_{01}, x_{02}, \ldots x_{0k}\}, X_1 = \{x_{11}, x_{12}, \ldots x_{1k}\}, \ldots, X_n$

Q. U is (not) reachable from Q_0 in n steps:
$F_{Q_0}(X_0) \wedge F_T(X_0, X_1) \wedge F_T(X_1, X_2) \wedge F_T(X_2, X_3) \wedge \cdots \wedge F_T(X_{n-1}, X_n) \wedge F_U(X_n)$
SAT iff U is reachable (UNSAT iff not reachable)

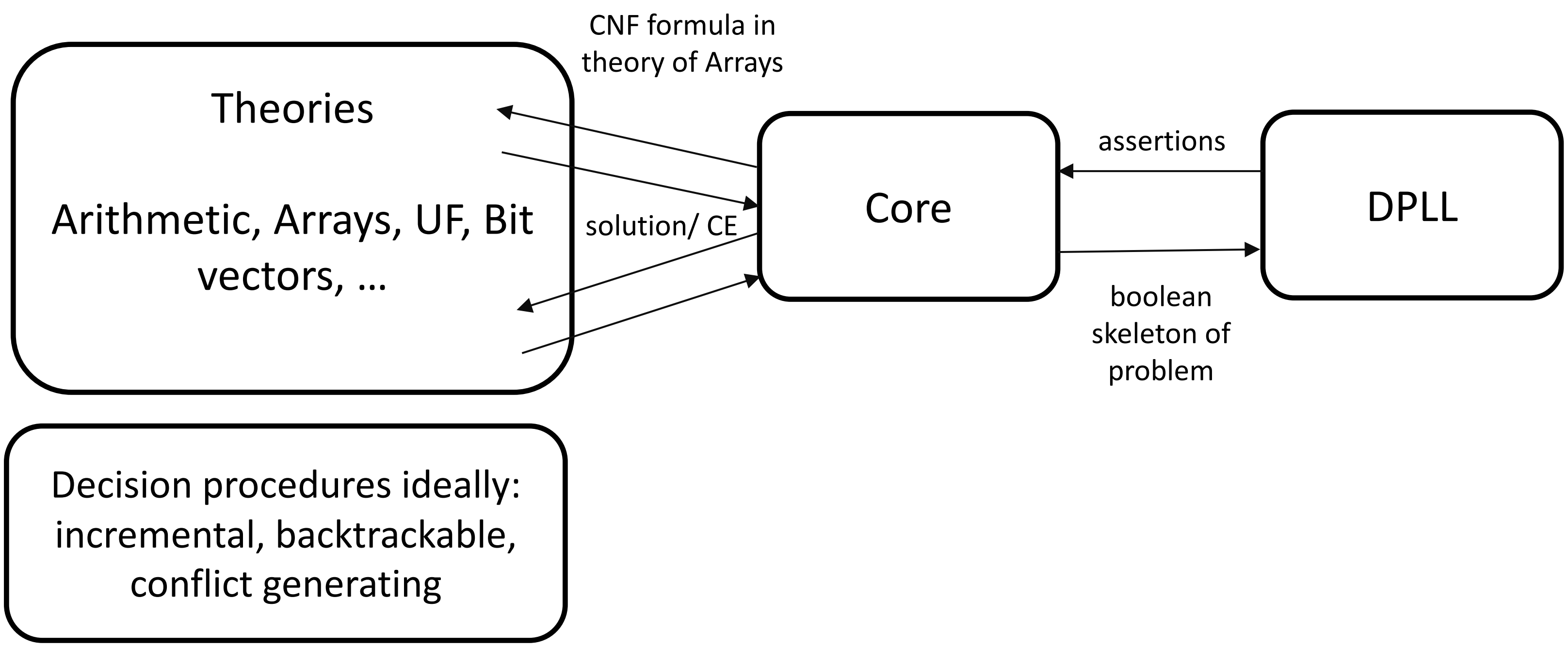
Tutorial 1

FROM SAT TO SMT

Slides by Sayan Mitra (mitras@illinois.edu)

Architecture of SMT Solvers

Question: Input $\alpha(x)$ formula in some set of logical theories, $\exists x, x \models \alpha$?



Theories and terminology

- Signature : function symbol, predicate symbol, arity, set of variables
- $Terms(\Sigma, V)$:
  - $v | f(t_{0,..}, t_k) |$
  - ground terms
- Atomic formula $AF(\Sigma, V)$:
  - $T, F, p(t_0, \ldots, t_k)$
  - literal: AF or its negation
- $QFF(\Sigma, V)$: $\phi, \neg\phi, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \phi_1 \rightarrow \phi_2$, where $\phi, \phi_1 \in AF$
- $FOF(\Sigma, V)$:
  - QFF under universal and existential quantifiers
  - Free and bound variables
- Sentence: FOF with no free variables
- $Theory(\Sigma, V)$: set of all sentences

- $\Sigma_f := \{0, +\}, \Sigma_p := \{<\}, arity(0) := 0, arity(+) := 2, arity(<) := 2, V := \{x, y, z\}$
- Terms: $x, y, z, 0, +(x, y), +(+(x, y), 0)$
- AF:
  $x < y, +(x, y) = +(y, x)$
- QFF:
  $+(x, y) = 0 \wedge x > y$
- FOF: $\forall x, \exists y:\ +(x, y) = 0 \wedge x > y$

Decision procedures

Models give meaning to symbols and formula

A model M for Σ, V defines a domain, gives interpretation to all symbols and assignment to all the variables

Given a theory T a theory solver (decision procedure) takes as input a set of literals Φ and determines whether Φ is T-satisfiable, i.e., does there exist a model M, such that $M \models \Phi$?

Example theories

Uninterpreted functions (UF)
$\Sigma_f = \{f, g, ..\}, \Sigma_p = \{=\}, V = x_i$
$x_1 \neq x_2 \wedge x_3 \neq x_2 \wedge f(x_3) = f(x_2)$
Arithmetic
$\Sigma_p = \{<, >, \leq, \geq, =\}$
Difference logic
$\Sigma_f = \{-\}, \Sigma_p = \{<, >, \leq, \geq, =\}$
$x_1 - x_2 > k$
Linear arithmetic: $7x_1 - 3x_2 + 6x_3 \leq 10$
Nonlinear arithmetic: $7x_1^2 - 3x_2x_1 + 6x_3^3 \leq 1$
Arrays
Bit vectors

A decision procedure for UF

$$\Phi \coloneqq x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3)$$

Rules:

1. Put all variables and function instances in their own classes
2. if $t_i = t_j$ is the predicate then merge the containing classes; repeat
3. If t_i and t_j are in the same class, then merge $F(t_i)$ and $F(t_j)$; repeat
4. If $t_i \neq t_j$ is in Φ such that t_i and t_j are in the same class then return UNSAT else return SAT

$\{x_1\}\{x_2\}\,\{x_3\}\,\{x_4\}\,\{x_5\}\{F(x_1)\}\{F(x_3)\}$
$\{x_1, x_2\}\,\{x_3\}\,\{x_4, x_5\}\{F(x_1)\}\{F(x_3)\}$
$\{x_1, x_2, x_3\}\,\{x_4, x_5\}\{F(x_1)\}\{F(x_3)\}$
$\{x_1, x_2, x_3\}\,\{x_4, x_5\}\{F(x_1), F(x_3)\}$
UNSAT

Back to SMT

Two approaches

- Eager: Translate to equisatisfiable propositional formula

- Lazy: Abstract to propositional form, feed to DPLL, refine

SMT solver example

$$\Phi \coloneqq \underbrace{g(a) = c}_{1} \land \underbrace{f\big(g(a)\big) \neq f(c)}_{\bar{2}} \lor \underbrace{g(a) = d}_{3} \land \underbrace{c \neq d}_{\bar{4}}$$

Send $\{1, \bar{2} \lor 3, \bar{4}\}$ to SAT

SAT solver returns model $\{1, \bar{2}, \bar{4}\}$

UF-solver finds concretization of $\{1, \bar{2}, \bar{4}\}$ UNSAT

Send $\{1, \bar{2} \lor 3, \bar{4}, \neg(1 \land \bar{2} \land \bar{4})\}$ to SAT

Send $\{1, \bar{2} \lor 3, \bar{4}, \bar{1} \lor 2 \lor 4)\}$ to SAT

SAT solver returns model $\{1, 3, \bar{4}\}$

UF-solver finds concretization of $\{1, 3, \bar{4}\}$ UNSAT

Send $\{1, \bar{2} \lor 3, \bar{4}, \bar{1} \lor 2 \lor 4, \bar{1} \lor \bar{3} \lor 4\}$ to SAT

SAT solver returns UNSAT; Original formula is UNSAT in UF

Summary

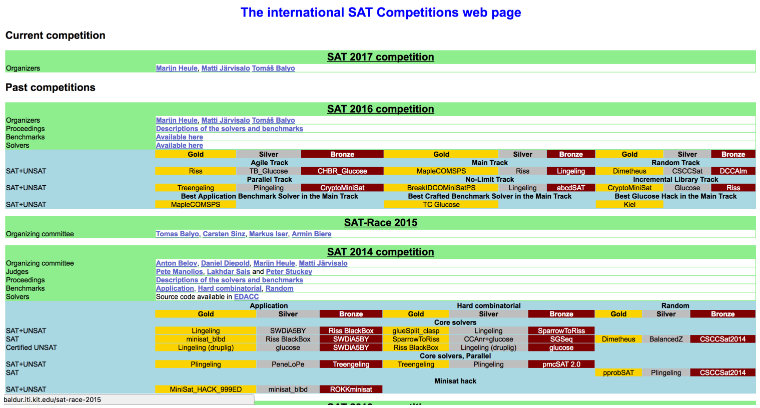
This was just an introduction to SMT solvers

Modern solvers Z3, CVC4, Chaff, have been used to solve practical verification problems

Many, many tools use SAT solvers for verification, synthesis, symbolic simulation, etc.

SMT competitions:

http://www.satcompetition.org/



The international SAT Competitions web page

Current competition

SAT 2017 competition	
Organizers	Marijn Heule, Matti Järvisalo Tomáš Balyo

Past competitions

SAT 2016 competition	
Organizers	Marijn Heule, Matti Järvisalo Tomáš Balyo
Proceedings	Descriptions of the solvers and benchmarks
Benchmarks	Available here
Solvers	Available here

	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track			Random Track		
SAT+UNSAT	Riss	TB_Glucose	CHBR_Glucose	MapleCOMSPS	Riss	Lingeling	Dimetheus	CSCCSat	DCCAlm
	Parallel Track			No-Limit Track			Incremental Library Track		
SAT+UNSAT	Treengeling	Plingeling	CryptoMiniSat	BreakIDCOMiniSatPS	Lingeling	abcdSAT	CryptoMiniSat	Glucose	Riss
	Best Application Benchmark Solver in the Main Track			Best Crafted Benchmark Solver in the Main Track			Best Glucose Hack in the Main Track		
SAT+UNSAT	MapleCOMSPS			TC Glucose			Kiel		

SAT-Race 2015	
Organizing committee	Tomas Balyo, Carsten Sinz, Markus Iser, Armin Biere

SAT 2014 competition	
Organizing committee	Anton Belov, Daniel Diepold, Marijn Heule, Matti Järvisalo
Judges	Pete Manolios, Lakhdar Sais and Peter Stuckey
Proceedings	Descriptions of the solvers and benchmarks
Benchmarks	Application, Hard combinatorial, Random
Solvers	Source code available in EDACC

	Application			Hard combinatorial			Random		
	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
	Core solvers								
SAT+UNSAT	Lingeling	SWDiA5BY	Riss BlackBox	glueSplit_clasp	Lingeling	SparrowToRiss			
SAT	minisat_blbd	Riss BlackBox	SWDiA5BY	SparrowToRiss	CCAnr+glucose	SGSeq	Dimetheus	BalancedZ	CSCCSat2014
Certified UNSAT	Lingeling (druplig)	glucose	SWDiA5BY	Riss BlackBox	Lingeling (druplig)	glucose			
	Core solvers, Parallel								
SAT+UNSAT	Plingeling	PeneLoPe	Treengeling	Treengeling	Plingeling	pmcSAT 2.0			
SAT							pprobSAT	Plingeling	CSCCSat2014
	Minisat hack								
SAT+UNSAT	MiniSat_HACK_999ED	minisat_blbd	ROKKminisat						

baldur.iti.kit.edu/sat-race-2015

Slides by Sayan Mitra (mitras@illinois.edu)