Software Verification

Satisfiability Modulo Theory and applications

Symbolic representations II

Ahmed Rezine

IDA, Linköpings Universitet

Spring 2024

Outline

SAT solving and verification

Theories and SMTLIB

Symbolic Execution

Further readings

Outline

Basics

Sat solvers usually take propositional formulas written in conjunctive normal (cnf):

- variables: propositional variables, e.g., $x, y, z, \ldots$
- literals: variables or their negations, e.g., $x, \overline{x}, y, \ldots$
- clause: disjunction of literals, e.g., $(x \vee \overline{y})$
- cnf formula: conjunction of clauses, e.g., $(x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{z}) \wedge (\overline{x})$
- a (partial) variable assignment associates (some of the) variables to Boolean values in $\{0, 1\}$

Basics: a well known NP-problem

Given a propositional formula (usually in cnf), does there exist a satisfying assignment (i.e., a variable assignment for which the formula evaluates to true)?

- $x \mapsto 1, y \mapsto 1, z \mapsto 0$ is a satisfying assignment for $(x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{z})$. The formula is **satisfiable**
- $(x \vee \overline{y}) \wedge (y \vee z) \wedge (\overline{z}) \wedge (\overline{x})$ has no satisfying assignment, it is **unsatisfiable**
- Success story with applications in hardware/software model checking, planning, combinatorial design, test pattern generation, protocol design, bioinformatics, etc.
- From 100 variables and 200 constraints in early 90s to more than a 1,000,000 variables and 5,000,000 constraints

Unit propagation

- A unit is a clause where there is one unassigned literal while all other literals are assigned 0.
- The only chance for the current assignment to satisfy the formula is to assign 1 to unassigned literals in unit clauses
- $(x) \wedge (\overline{x} \vee y) \wedge (\overline{y} \vee z \vee s) \wedge (z \vee \overline{y} \vee \overline{x}) \wedge \ldots$
  - implied assignment: $x = 1$, antecedent (x)
  - implied assignment: $y = 1$, antecedent $(\overline{x} \vee y)$
  - implied assignment: $z = 1$, antecedent $(z \vee \overline{y} \vee \overline{x})$
  - implied assignment: $s = 1$, antecedent $(\overline{y} \vee z \vee s)$

Davis-Putnam-Logemann-Loveland (DPLL)

- decides satisfiability for sat-cnf problems
- introduced in early 60s
- basis of modern sat solvers
- idea: alternate unit propagation, choosing a value for some variable, and recursively checking the result, if does not give satisfying assignment then backtrack (remove) with opposit value

Davis-Putnam-Logemann-Loveland (DPLL)

Algorithm 1: DPLL-recursive(φ, ρ)

Input: φ: cnf formula, ρ: partial assignment

Output: UNSAT or satisfying assignment

$(\varphi, \rho) := \mathit{UnitPropagate}(\varphi, \rho)$;

if φ *contains an empty clause* **then**

  return *UNSAT*;

if φ *has no clauses left* **then**

  Output ρ;

  return SAT;

$l :=$ a literal not assigned by ρ;

if *(DPLL-recursive*$(\varphi[l], \rho \cup \{l\}))$ **then**

  return *SAT*;

return *DPLL-recursive*$(\varphi[\neg l], \rho \cup \{\neg l\})$;

DPLL, Example

$$\varphi = (x \vee y) \wedge (y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee z)$$

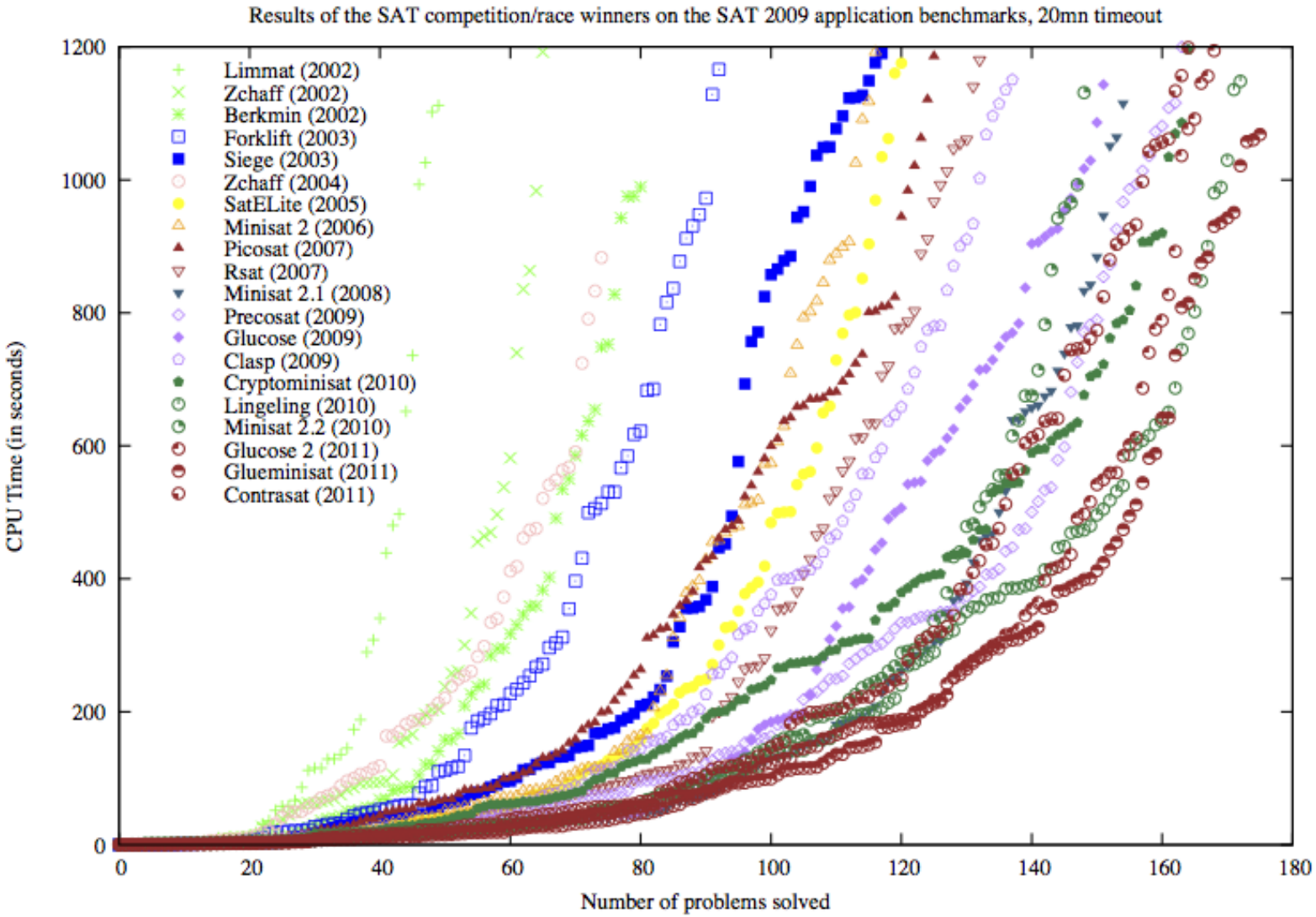
level	decision	formula	unit propagation
0	$x = 1$	$(1 \vee y) \wedge (y \vee z) \wedge (\overline{1} \vee \overline{y} \vee \overline{z}) \wedge (\overline{1} \vee 1)$	$z = 1@0$
1	$y = 1$	$(1 \vee 1) \wedge (1 \vee 1) \wedge (\overline{1} \vee \overline{1} \vee \overline{1}) \wedge (\overline{1} \vee 1)$	
1	$y = 0$	$(1 \vee 0) \wedge (0 \vee 1) \wedge (\overline{1} \vee \overline{0} \vee \overline{1}) \wedge (\overline{1} \vee 1)$	

$$x = 1@0, z = 1@0, y = 0@1$$

DPLL, Example 2

$$\varphi = (a \vee \overline{b} \vee d) \wedge (a \vee \overline{b} \vee e) \wedge (\overline{b} \vee \overline{d} \vee \overline{e}) \wedge (a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \overline{d}) \wedge (a \vee b \vee \overline{c} \vee e) \wedge (a \vee b \vee \overline{c} \vee \overline{e})$$

Modern solvers



What made it possible?

- Clause learning with non chronological backtracking
- search restarts
- lazy data-structures
- ...

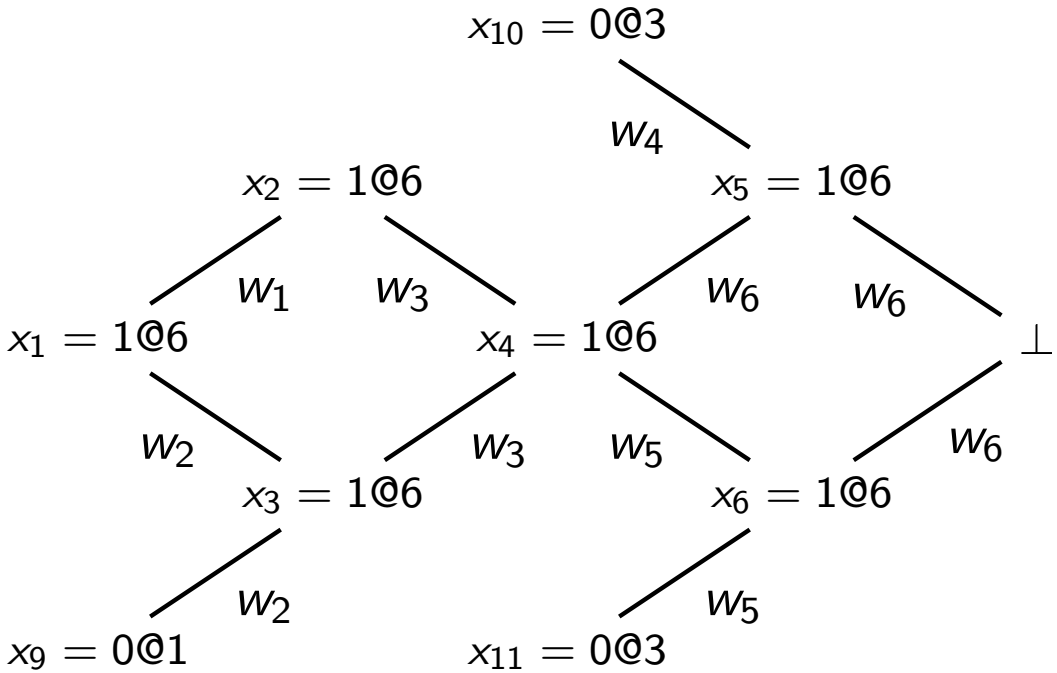
Clause learning

- capture "cause" of encountered conflict as a clause
- the conflict clause is added to the formula
- if deciding $x = 1$ and $y = 0$ led to unsat then remember this by adding $(\overline{x} \vee y)$ to the formula
- learn "reasons" of discovered inconsistencies in order to avoid them in the future

Conflict clauses

- Current assignment: $x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, \ldots$
- Current decision: $x_1 = 1@6$

$$
\begin{aligned}
w_1 &= (\overline{x_1} \vee x_2) \\
w_2 &= (\overline{x_1} \vee x_3 \vee x_9) \\
w_3 &= (\overline{x_2} \vee \overline{x_3} \vee x_4) \\
w_4 &= (\overline{x_4} \vee x_5 \vee x_{10}) \\
w_5 &= (\overline{x_4} \vee x_6 \vee x_{11}) \\
w_6 &= (\overline{x_5} \vee \overline{x_6}) \\
w_7 &= (x_1 \vee x_7 \vee \overline{x_{12}}) \\
w_8 &= (x_1 \vee x_8) \\
w_9 &= (\overline{x_7} \vee \overline{x_8} \vee \overline{x_{13}}) \\
&\ldots.
\end{aligned}
$$



Implication graph

- source nodes of implication graph can be used as a conflict clause
- here, $(\overline{x_1} \vee x_9 \vee x_{10} \vee x_{11})$ can be added as a conflict clause
- better clauses with "unique implication point"
- here, add $(\overline{x_4} \vee x_{10} \vee x_{11})$

SMTs as generalizations of SAT

Originates from automating proof-search for first order logic.

- Variables: $x, y, z, \ldots$
- Constants: $a, b, c, \ldots$
- N-ary functions: $f, g, h, \ldots$
- N-ary predicates: $p, q, r, \ldots$
- Atoms: $\bot, \top, p(t_1, \ldots, t_n)$
- Literals: atoms or their negation
- A FOL formula is a literal, boolean combinations of formulas, or quantified ($\exists$, $\forall$) formulas.

Evaluation of formula φ, with respect to interpretation I over non-empty (possibly infinite) domains for variables and constants gives true or false (resp. $I \models \varphi$ or $I \not\models \varphi$)

Satisfiability and Validity

A formula φ is:

- satisfiable if $I \models \varphi$ for **some** interpretation I
- valid if $I \models \varphi$ for **all** interpretations I

Satisfiability of FOL is undecidable. Instead, target decidable or domain-specific fragments.

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\varphi \triangleq g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

- EUF: Equality over Uninterpreted functions
- Satisfiable?

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\begin{aligned}\varphi \quad \triangleq \quad & (x_1 \geq 0) \wedge (x_1 < 1) \\ & \wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(a, x_2, x_3), x_2 + x_1) = x_3 + 1)\end{aligned}$$

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\begin{aligned}\varphi \quad \triangleq \quad & (x_1 \geq 0) \wedge (x_1 < 1) \\ & \wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(a, x_2, x_3), x_2 + x_1) = x_3 + 1)\end{aligned}$$

- involves arrays with read (rd) and write (wr):
  - $\forall arr\ \forall i\ \forall val.\ (rd(wr(arr, i, val), i) = val)$
  - $\forall arr\ \forall i\ \forall j\ \forall val.\ (i \neq j \Rightarrow rd(wr(arr, i, val), j)) = rd(arr, j))$

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\begin{aligned}\varphi \quad \triangleq \quad & (x_1 \geq 0) \wedge (x_1 < 1) \\ & \wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(a, x_2, x_3), x_2 + x_1) = x_3 + 1)\end{aligned}$$

- Linear Integer Arithmetic (LIA)

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\begin{aligned}\varphi \triangleq\ & (x_1 \geq 0) \wedge (x_1 < 1) \\ & \wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(a, x_2, x_3), x_2 + x_1) = x_3 + 1)\end{aligned}$$

- Linear Integer Arithmetic (LIA)
- Equality over Uninterpreted functions (EUF)
- Arrays (A)

Introduction to SMT

Given a quantifier free FOL formula and a combination of theories, is there an interpretation to the free variables that makes the formula true?

$$\begin{aligned}\varphi \quad \triangleq \quad & (x_1 \geq 0) \wedge (x_1 < 1) \\ & \wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(a, x_2, x_3), x_2 + x_1) = x_3 + 1)\end{aligned}$$

- LIA: $x_1 = 0$
- EUF: $f(x_1) = f(0)$
- A: $rd(wr(a, x_2, x_3), x_2) = x_3$
- Bool: $rd(wr(P, x_2, x_3), x_2) = x_3 + 1$
- LIA: $\bot$

Introduction to SMT

- Sometimes more natural to express in logics other than propositional logic
- SMT decide satisfiablity of ground FO formulas wrt. background theory
- Many applications: Model checking, predicate abstraction, symbolic execution, scheduling, test generation, ...

Outline

SMT competition and SMTLIB

- Drive development, since 2005
- 19^{th} instance at `https://smt-comp.github.io/2024/`
- Papers at SAT, CADE, CAV, FMCAD, TACAS, ...
- SMTLIB key initiative to promote common input and output for SMT solvers, benchmarks, tutorials, ...
- at `http://smtlib.cs.uiowa.edu/`

Equality with uninterpreted Functions (EUF)

- Consider $a * (f(b) + f(c)) = d \wedge b * (f(a) + f(c)) \neq d \wedge a = b$
- Formula is unsat, could be abstracted with
- $h(a, g(f(b), f(c))) = d \wedge h(b, g(f(b), f(c))) \neq d \wedge a = b$
- EUF used to abstracted non-supported theories such as non-linear multiplication or ALUs in circuits.

Arithmetic

Several restricted fragments, whether real or integer variables:

- Bounds $x \sim k$ with $\sim \in \{<, \leq, =, \geq, >\}$
- Difference logic $x - y \sim k$ with $\sim \in \{<, \leq, =, \geq, >\}$
- UTVPI $\pm x \pm y \sim k$ with $\sim \in \{<, \leq, =, \geq, >\}$
- Linear Arithmetic $x + 2y - 3z \leq 2$
- Non-linear arithmetic $xy - 4xy^2 + 2z \leq 2$

Arrays

- Special functions *read* and *write*
- Axioms:
  - $\forall a \forall i \forall v (read(write(a, i, v), i) = v)$
  - $\forall a \forall i \forall j \forall v (i \neq j \Rightarrow read(write(a, i, v), j)) = read(a, j))$
- Used for software (arrays) and hardware (memories) verification

Bit vectors

- Operations on vectors of bits
  - String like: concatenation, extraction, ...
  - Logical: bit-wise or, not, and...
  - Arithmetic: add, substract, multiply, ...
- $a[0:1] \neq b[0:1] \wedge (a|b) = c \wedge c[0] = 0 \wedge a[1] + b[1] = 0$

Outline

Testing

- Most common form of software validation
- Explores only one possible execution at a time
- For each new value, run a new test.
- On a 32 bit machine, `if(i==2023) bug()` would require 2^{32} different values to make sure there is no bug.
- The idea in symbolic testing is to associate **symbolic values** to the variables

Symbolic Testing

- Main idea by JC. King in "Symbolic Execution and Program Testing" in the 70s
- Use symbolic values instead of concrete ones
- Along the path, maintain a *Path Constraint* (PC) and a symbolic state (σ)
- PC collects constraints on variables' values along a path,
- σ associates variables to symbolic expressions,
- We get concrete values if PC is satisfiable
- The program can be run on these values
- Negate a condition in the path constraint to get another path

Symbolic Execution: a simple example

- Can we get to the ERROR? explore using SSA forms.
- Useful to check array out of bounds, assertion violations, etc.

```
1 foo(int x,y,z){
2  x = y - z;
3  if(x==z){
4     z = z - 3;
5     if(4*z < x + y){
6        if(25 > x + y) {
7           ...
8        }
9        else{
10          ERROR;
11       }
12    }
13 }
14 ...
```

$PC_1 = true$	
$PC_2 = PC_1$	$x \mapsto x_0, y \mapsto y_0, z \mapsto z_0$
$PC_3 = PC_2 \wedge x_1 = y_0 - z_0$	$x \mapsto (y_0 - z_0), y \mapsto y_0, z \mapsto z_0$
$PC_4 = PC_3 \wedge x_1 = z_0$	$x \mapsto (y_0 - z_0), y \mapsto y_0, z \mapsto z_0$
$PC_5 = PC_4 \wedge z_1 = z_0 - 3$	$x \mapsto (y_0 - z_0), y \mapsto y_0, z \mapsto (z_0 - 3)$
$PC_6 = PC_5 \wedge 4 * z_1 < x_1 + y_0$	$x \mapsto (y_0 - z_0), y \mapsto y_0, z \mapsto (z_0 - 3)$
$PC_{10} = PC_6 \wedge 25 \leq x_1 + y_0$	$x \mapsto (y_0 - z_0), y \mapsto y_0, z \mapsto (z_0 - 3)$

$PC = (x_1 = y_0 - z_0 \wedge x_1 = z_0 \wedge z_1 = z_0 - 3 \wedge 4 * z_1 < x_1 + y_0 \wedge 25 \leq x_1 + y_0)$

Check satisfiability with a solver (e.g., z3, cvc, yices, boolector,stp,...)

Symbolic execution today

- Leverages on the impressive advancements of SMT solvers
- Modern symbolic execution frameworks are not purely symbolic and are often dynamic: Sage, Klee (open source), Pex:
  - They can follow a concrete execution while collecting constraints along the way, or
  - They can treat some of the variables concretely, and some other symbolically
- This allows them to scale, to handle closed code or complex queries

Symbolic execution today

- C (actullay llvm) `http://klee.github.io/`
- Java (more than a symbolic executer)
  `http://babelfish.arc.nasa.gov/trac/jpf`
- C# (actually .net)
  `http://research.microsoft.com/en-us/projects/pex/`
- ...

Outline

Further readings


A. R. Bradley and Z. Manna.
(chap 10) The calculus of computation: decision procedures with applications to verification.
Springer Science & Business Media, 2007.


C. Cadar, D. Dunbar, D. R. Engler, et al.
Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.
In *OSDI*, volume 8, pages 209–224, 2008.


L. De Moura and N. Bjørner.
Satisfiability modulo theories: introduction and applications.
Communications of the ACM, 54(9):69–77, 2011.


P. Godefroid, M. Y. Levin, and D. Molnar.
Sage: whitebox fuzzing for security testing.
Queue, 10(1):20–27, 2012.


R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t).
J. ACM, 53(6):937–977, Nov. 2006.