Software Verification

Satisfiability Modulo Theory and applications

Symbolic representations II

Ahmed Rezine

IDA, Linköpings Universitet

Spring 2025

Outline

Lazy SMT solvers

Theories and SMTLIB

Symbolic Execution

Further readings

Introduction

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

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

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

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

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

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

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

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

Introduction

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

- 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, ...

Introduction: from SAT to SMT

- Eager approach with "bit-blasting" (UCLID):
  - Encode SMT formula in propositional logic
  - Use off-the-shelf SAT solver
  - Still dominant for bit-vector arithmetic
- Lazy-approach (CVC4, MathSat, Yices, Z3, ...)
  - Combine SAT (CDCL) and theory solvers
  - Sat-solver enumerates models for the boolean part
  - Theory solvers check satisfiability in the theory

Eager approach e.g.: EUF

- remove terms $f(a), f(b), f(c)$ by replacing with fresh constants A, B, C.
- add $a = b \Rightarrow A = B$, $a = c \Rightarrow A = C$ and $b = c \Rightarrow B = C$
- for n constants use $logn$ bits to encode value of each constant $a, b, ...$
- each $a = b$ is replaced by $P_{a,b}$
- add $P_{a,b} \wedge P_{b,c} \Rightarrow P_{a,c}$

Outline

Lazy SMT solvers

- Restrict theory solver to conjunctions of constraints
- Convert to disjunctive normal form and check one conjunction at a time
- Or use Sat to enumerate conjuncts

Basic lazy SMT

```
1  ψ = to_cnf(φ);
2  while(true){
3     res, M = check_SAT(ψ);
4     if( res){
5        M_T = to_theory(M);
6        res = check_theory(M_T);
7        if(res)
8           return SAT;
9        else
10          ψ∧ = ¬M;
11    }else
12       return UNSAT;
13 }
```

Integrating SMT and SAT

$$(1 : g(a) = c) \wedge ((\overline{2} : f(g(a)) \neq f(c)) \vee (3 : g(a) = d)) \wedge (\overline{4} : c \neq d)$$

- SAT-solver gives $\{1, \overline{2}, \overline{4}\}$
- But $\{(g(a) = c), (f(g(a)) \neq f(c)), (c \neq d)\}$ unsat by the Theory-solver
- Add $\{\overline{1} \vee 2 \vee 4\}$ to sat formula
- SAT-solver gives $\{1, 2, 3, \overline{4}\}$
- But $\{(g(a) = c), (f(g(a)) = f(c)), (g(a) = d), (c \neq d)\}$ unsat by the Theory solver
- Add $\{\overline{1} \vee \overline{2} \vee \overline{3}, \overline{4}\}$ to sat formula
- SAT-solver declares unsat, hence the original formula is unsat

Integrating SMT and SAT

ψ	$\triangleq$		$\psi_{\mathbb{B}}$	$\triangleq$
c_1	:	$\neg(2x_2 - x_3 > 2) \vee (x_1 + x_3 \leq 5)$	$\overline{A_{11}} \vee A_{12}$	
c_2	:	$\neg(x_1 - x_3 \leq 5) \vee (x_1 - x_5 \leq 1)$	$\overline{A_{21}} \vee A_{22}$	
c_3	:	$\neg(3x_1 - 2x_2 \leq 3) \vee \neg(x_1 - x_3 \leq 5)$	$\overline{A_{31}} \vee \overline{A_{32}}$	
c_4	:	$\neg(3x_1 - x_3 \leq 6) \vee \neg(x_1 + x_3 \leq 5)$	$\overline{A_{41}} \vee \overline{A_{42}}$	
c_5	:	$(x_1 + x_3 \leq 5) \vee (3x_1 - 2x_2 \leq 3)$	$A_{51} \vee A_{31}$	
c_6	:	$(x_2 - x_4 \leq 6) \vee \neg(x_1 + x_3 \leq 5)$	$A_{61} \vee \overline{A_{62}}$	
c_7	:	$(x_1 + x_3 \leq 5) \vee (x_3 = 3x_5 + 4) \vee \neg(x_1 - x_3 \leq 5)$	$A_{71} \vee A_{72} \vee \overline{A_{73}}$	

- Sat-solver gives $\{A_{12}, A_{22}, \overline{A_{31}}, \overline{A_{41}}, A_{51}, A_{61}, A_{72}\}$
- But $\left\{ \begin{array}{l} (x_1 + x_3 \leq 5), (x_1 - x_5 \leq 1), \neg(3x_1 - 2x_2 \leq 3), \\ \neg(3x_1 - x_3 \leq 6), (x_2 - x_4 \leq 6), (x_3 = 3x_5 + 4) \end{array} \right\}$ unsat by the Theory-solver
- Add $(\overline{A_{12}} \vee \overline{A_{22}} \vee A_{31} \vee A_{41} \vee \overline{A_{51}} \vee \overline{A_{61}} \vee \overline{A_{72}})$ to $\psi_{\mathbb{B}}$

Outline

SMT competition and SMTLIB

- Drive development, since 2005
- 15^{th} instance at `https://smt-comp.github.io/2020`
- 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.