Software Verification

Satisfiability Modulo Theory and applications

Symbolic representations II

Ahmed Rezine

IDA, Linköpings Universitet

Spring 2021

Outline

Lazy SMT solvers

Theories and SMTLIB

Nelson-Oppen Approach

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?

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

- 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?

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

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?

$$\varphi \triangleq (x_1 \geq 0) \wedge (x_1 < 1)$$
$$\wedge((f(x_1) = f(0)) \Rightarrow (rd(wr(P, x_2, x_3), x_2 + x_1) = x_3 + 1)$$

- 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?

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

- 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?

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

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

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 $a, b, \ldots$
- 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

Theories and SMTLIB

Nelson-Oppen Approach

Symbolic Execution

Further readings

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

```
ψ = to_cnf(φ);
while(true){
  res, M = check_SAT(ψ);
  if( res){
    M_T = to_theory(M);
    res = check_theory(M_T);
    if(res)
      return SAT;
    else
      ψ∧ = ¬M;
  }else
    return UNSAT;
}
```

Integrating SMT and SAT

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

- $M = \{1, \overline{2}, \overline{4}\}$
- $N = \{(1 : g(a) = c), (\overline{2} : f(g(a)) \neq f(c)), (\overline{4} : c \neq d)\}$ is unsat
- add $\{\overline{1} \vee 2 \vee 4\}$
- $M = \{1, 2, 3, \overline{4}\}$
- $N = \{(1 : g(a) = c), (2 : f(g(a)) = f(c)), (3 : g(a) = d), (\overline{4} : c \neq d)\}$
- add $\{\overline{1} \vee \overline{2} \vee \overline{3}, \overline{4}\}$
- SAT solver declares unsat

Integrating SMT and SAT

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

- $M = \{A_{12}, A_{21}, \neg A_{31}, \neg A_{41}, A_{61}, A_{72}\}$
- $M_T = \{(x_1 + x_3 \leq 5), (x_1 - x_5 \leq 1), \neg(3x_1 - 2x_2 \leq 3), \neg(3x_1 - x_3 \leq 6), (x_2 - x_4 \leq 6), (x_3 = 3x_5 + 4)\}$
- Theory solver: M_T is UNSAT. Add $\neg M$ to ψ

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

Combining Decision Procedures

$$x = y+1 \wedge a = write(b, x+1, 0) \wedge (read(a, y+z) = 1 \vee f(x+1) \neq f(z))$$

Such formulas can naturally arise in software verification. Need to reason over:

- Linear arithmetic
- Arrays
- uninterpreted functions
- Under some restrictions, Nelson-Oppen allows to combine individual theories in order to answer combinations like above.
- We can consider conjunctions of literals (put in dnf)

Example

$$1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2)$$

- In $T_{\mathbb{Z}}$ $1 \leq x \land x \leq 2$ implies $x \in \{1, 2\}$
- So $f(x) = f(1)$ or $f(x) = f(2)$

Non-deterministic Nelson-Oppen

- Given T_1, T_2 such that $\Sigma_1 \cap \Sigma_2 = \{=\}$
- Where each satisfiable formula in T_1 or in T_2 is also satisfiable over an interpretation with an infinite domain (stably infinite)
- Then we can combine two decision procedures $P1, P2$ for $T_1 \cup T2$ as follows.

Non-deterministic Nelson-Oppen

Phase 1: idea

- First transform any $(T_1 \cup T_2)$-conjunction F into the conjunction of a T_1-formulas and a T_2-formula
- For this, purify the formula by introducing new variables and conjunctions each time a function or a predicate mixes terms from different theories

Non-deterministic Nelson-Oppen

Phase 1: example1

- $1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- introduce w_1, w_2 to obtain $(1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2)$ and $(f(x) \neq f(w_1) \wedge f(x) \neq f(w_2))$
- $(1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2)$ is in $T_{\mathbb{Z}}$, and
- $(f(x) \neq f(w_1) \wedge f(x) \neq f(w_2))$ is in T_{UF}

Non-deterministic Nelson-Oppen

Phase 1: example2

- $f(x) = x + y \wedge x \leq y + z \wedge x + z \leq y \wedge y = 1 \wedge f(x) \neq f(2)$
- replace $f(x) = x + y$ by $w_1 = x + y \wedge w_1 = f(x)$
- replace $f(x) \neq f(2)$ by $f(x) \neq f(w_2) \wedge w_2 = 2$
- This gives the equisatisfiable conjunction $(w_1 = x + y \wedge x \leq y + z \wedge x + z \leq y \wedge y = 1 \wedge w_2 = 2)$ in $T_{\mathbb{Z}}$ and $(w_1 = f(x) \wedge f(x) \neq f(w_2))$ in T_{UF}

Non-deterministic Nelson-Oppen

Phase 2: guess and check

- let $V = \mathit{free}(F_1) \cap \mathit{free}(F_2)$ where $F_1 \wedge F_2$ obtained after purification
- $F_1 \wedge F_2$ is satisfiable iff
  - there is an equivalence relation $\sim$ over V s.t
  - $\alpha = \bigwedge_{(u \sim v)} u = v \wedge \bigwedge_{(u \not\sim v)} u \neq v$, and
  - both $F_1 \wedge \alpha$ and $F_2 \wedge \alpha$ are satisfiable
- otherwise $F_1 \wedge F_2$ is unsatisfiable

Non-deterministic Nelson-Oppen

Phase 2: example 1

- Consider $F : 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- with $F_{\mathbb{Z}} : 1 \leq x \wedge x \leq 2 \wedge w_1 = 1 \wedge w_2 = 2$, and
- $F_{UF} : f(x) \neq f(w_1) \wedge f(x) \neq f(w_2)$

The shared variables are $\{x, w_1, w_2\}$ which gives the following possible equivalences

- $\{\{x, w_1, w_2\}\}$ unsat because $x = w_1$ and $f(x) \neq f(w_1)$
- $\{\{x, w_1\}, \{w_2\}\}$ unsat because $x = w_1$ and $f(x) \neq f(w_1)$
- $\{\{x, w_2\}, \{w_1\}\}$ unsat because $x = w_2$ and $f(x) \neq f(w_2)$
- $\{\{x\}, \{w_1, w_2\}\}$ unsat because $w_1 = w_2$ and $w_1 = 1 \wedge w_2 = 2$
- $\{\{x\}, \{w_1\}, \{w_2\}\}$ unsat because $x = 1 \vee x = 2$ and $w_1 = 1 \wedge w_2 = 2$

So F is $(T_{\mathbb{Z}} \cup T_{UF})$-unsatisfiable

Non-deterministic Nelson-Oppen

Incremental:

- Consider
  $F: f(x) = x + y \wedge x \leq y + z \wedge x + z \leq y \wedge y = 1 \wedge f(x) \neq f(2)$.
- $F_{\mathbb{Z}}: w_1 = x + y \wedge x \leq y + z \wedge x + z \leq y \wedge y = 1 \wedge w_2 = 2$
- $F_{UF}: w_1 = f(x) \wedge f(x) \neq f(w_2)$
- shared variables $\{x, w_1, w_2\}$.

1. attempt $x = w_1$, gives $y = 0$ contradicts $y = 1$, so $x \neq w_1$
2. $F_{\mathbb{Z}} \wedge x \neq w_1$ and $F_{UF} x \neq w_1$ are satisfiable
3. attempt $x = w_2$, but $f(x) \neq f(w_2)$ so $x \neq w_2$
4. $F_{\mathbb{Z}} \wedge x \neq w_1 \wedge x \neq w_2$ and $F_{UF} x \neq w_1 \wedge x \neq w_2$ are satisfiable
5. attempt $w_1 = w_2$, no contradiction

$\{\{x\}, \{w_1, w_2\}\}$ make F is $(T_{\mathbb{Z}} \cup T_{UF})$-satisfiable

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==2014) 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.

```
foo(int x,y,z){
x = y - z;
if(x==z){
  z = z - 3;
  if(4*z < x + y){
    if(25 > x + y) {
      ...
    }
    else{
      ERROR;
    }
  }
}
...
```

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

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

Check satisfiability with a solver (e.g., `http://rise4fun.com/Z3`)

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

Outline

Lazy SMT solvers

Theories and SMTLIB

Nelson-Oppen Approach

Symbolic Execution

Further readings

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.