Lecture 3

On the power/limits of dynamic symbolic execution
(from “Higher-order test generation”, Patrice Godefroid PLDI 2011)

Powering up DSE with Satisfiability Modulo Theories

Thanks to Patrice Godefroid, Leonardo de Moura

Review

DSE

```
Procedure executeSymbolic(P,I) =
  initialize M_0 and S_0
  path constraint pc = true
  C = getNextCommand()
  while (C ≠ stop)
    match (C):
      case (v := e):
        M = M + [&v ↦ evalConcrete(e)]
        S = S + [&v ↦ evalSymbolic(e)]
      case (if e then C' else C''):
        b = evalConcrete(e)
        c = evalSymbolic(e)
        if b then pc = pc ∧ c
        else pc = pc ∧ ¬c
    C = getNextCommand() // end of while loop
```

```
evalSymbolic(e) =
  match (e):
    case v: // Program variable v
      return S(&v)
    case +(e1, e2): // Addition
      f1 = evalSymbolic(e1)
      f2 = evalSymbolic(e2)
      if f1 and f2 are constants
          return evalConcrete(e)
      else
          return createExpression('+',f1,f2)
    etc.
    default: // default for unhandled expression

      return evalConcrete(e)
```

Soundness / Completeness

Input i

Path w

Path Constraint pc_w

Path constraint pc_w is <u>sound</u>

if $\forall i \; i \models pc_w \Rightarrow \text{Path}(i) = w$

Unsound Path Constraint → Divergence

complex(42)
= 567

```
foo(int x, int y) {
    if(x == complex(y)) {
        if(y==10) {
            ...
```

```
evalSymbolic(e) =
  match (e):
    case v: // Program variable v
      return S(&v)
    case +(e1, e2): // Addition
      f1 = evalSymbolic(e1)
      f2 = evalSymbolic(e2)
      if f1 and f2 are constants
          return evalConcrete(e)
      else
          return createExpression('+',f1,f2)
    etc.
    default: // default for unhandled expression
        pc = pc ∧ ⋀_{x_i ∈ e}(x_i = I_i)
      return evalConcrete(e)
```

Sound Path Constraint

```
foo(int x, int y) {
  if(x == complex(y)) {
    if(y==10) {
      ...
```

Sound Constraint Generation Lowers Coverage

```
foo(int x, int y) {
    if(x != complex(y)) {
        if(y==10) {
            ...
```

Satisfiability Modulo Theories (SMT)

Is formula *F* satisfiable modulo theory *T* ?

SMT solvers have specialized algorithms for *T*

Satisfiability Modulo Theories (SMT)

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

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), c-2)) \neq f(c-b+1)$

Substituting c by b+2

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), b+2-2)) \neq f(b+2-b+1)$

Simplifying

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), b)) \neq f(3)$

Satisfiability Modulo Theories (SMT)

b + 2 = c and f(read(write(a,b,3), b)) ≠ f(3)

Applying array theory axiom

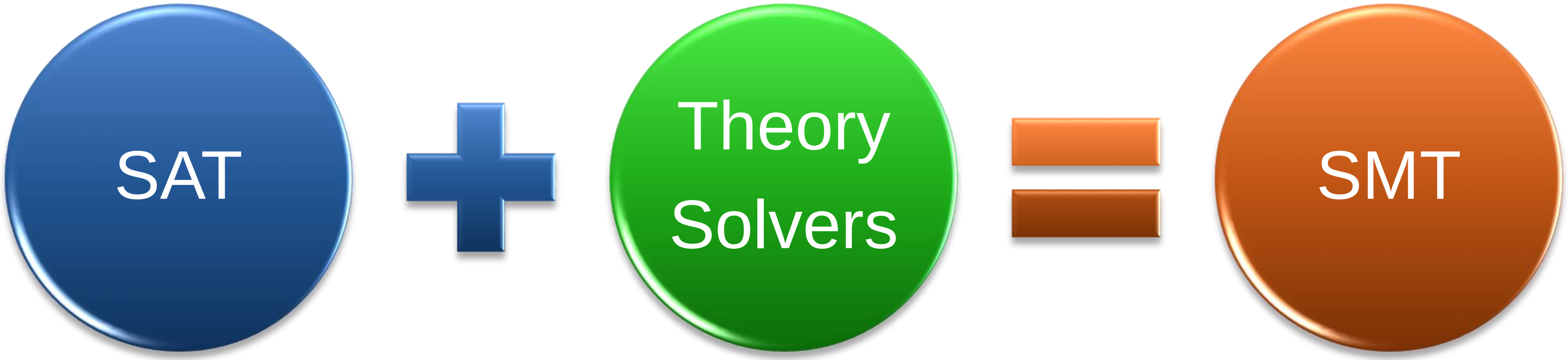
forall a,i,v: read(write(a,i,v), i) = v

Satisfiability Modulo Theories (SMT)

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

Inconsistent

SMT : Basic Architecture



Case Analysis

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

DPLL

The Davis–Putnam–Logemann–Loveland (DPLL) algorithm tries to find a satisfying assignment for logic formulae in conjunctive normal form (CNF).

DPLL is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form.

DPLL (abstract view)

evolving

M | F

Partial model

Set of clauses

DPLL (abstract view)

Guessing

$p \mid p \vee q, \neg q \vee r$

$p, \neg q \mid p \vee q, \neg q \vee r$

DPLL (abstract view)

Deducing

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

$$\Downarrow$$

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

DPLL (abstract view)

Backtracking

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

$\Downarrow$

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

SAT + Theory solvers

Basic Idea

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

$\Downarrow$ 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 + 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

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

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

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

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

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

SAT + Theory solvers

Basic Idea

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

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

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

SAT Solver

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

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

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

SAT + Theory solvers

Basic Idea

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

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

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

SAT Solver

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

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

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

Array Theory

$$\forall a, i, v.\ select(store(a, i, v), i) = v$$

$$\forall a, i, j, v:\ i = j \vee select(store(a, i, v), j) = \mathrm{select}(a, j)$$

Array Theory: a more familiar notation

$$\forall a, i, v.\ select(store(a, i, v), i) = v$$

$$\forall a, i, j, v:\ \ i = j \vee select(store(a, i, v), j) = select(a, j)$$

$$\Downarrow$$

$$\forall a, i, v.\ store(a, i, v)[i] = v$$

$$\forall a, i, j, v:\ \ i = j \vee store(a, i, v)[j] = a[i]$$

Extentional Array Theory

$$\forall a, b\colon (\forall i\colon a[i] = b[i]) \Rightarrow a = b$$

Arrays are actually “maps”

- We have arrays from D to R
- D does not need to be the Integers

Models for arrays are "finite graphs"

$$a = store(b, 0, 5),\ b = store(c, 1, 10),\ c[0] = 2$$

$$\mathrm{M}(a) = \{\ 0 \rightarrow 5,\ 1 \rightarrow 10,\ \text{else} \rightarrow 0\ \}$$
$$\mathrm{M}(b) = \{\ 0 \rightarrow 2,\ 1 \rightarrow 10,\ \text{else} \rightarrow 0\ \}$$
$$\mathrm{M}(c) = \{\ 0 \rightarrow 2,\ \text{else} \rightarrow 0\ \}$$

Z3 API for Arrays

- Array(name,dom,range)
- Select(a,i)
- Update(a,i,v)

Assignment: create a SymbolicDictionary

- Use Z3’s array theory to support Python’s dictionary (dict), modelling the following operations
  - Length: __length__(self)
  - Get: __getitem__(self,key)
  - Set: __setitem__(self,key,value)
  - Lookup: __contains__(self,key)

- Should support SymbolicInteger as a key and a value

- What about?
  - Delete: __delitem__(self,key)