Analysis of Incomplete Programs

a.k.a Programs with *closed-box* components

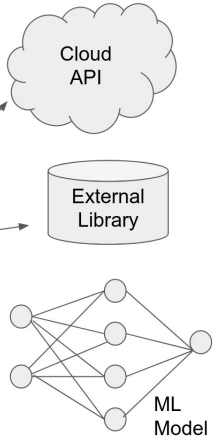
Subhajit Roy

Indian Institute of Technology Kanpur

Incomplete programs

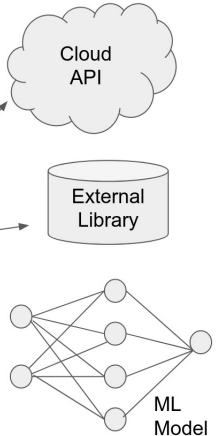
Incomplete programs

```
main() {
  input (a,b);
  …
  If (...)
     x = foo(a, b);
  …
  assert(...);
}
```

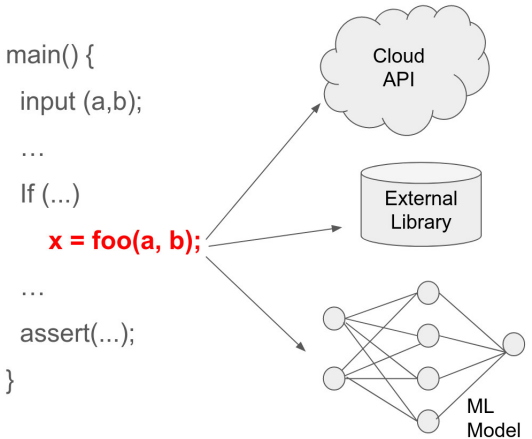


Incomplete programs

```
main() {
  input (a,b);
  …
  If (...)
     x = foo(a, b);
  …
  assert(...);
}
```

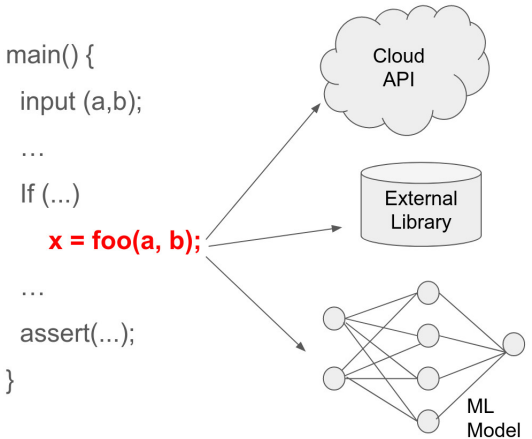


Incomplete programs



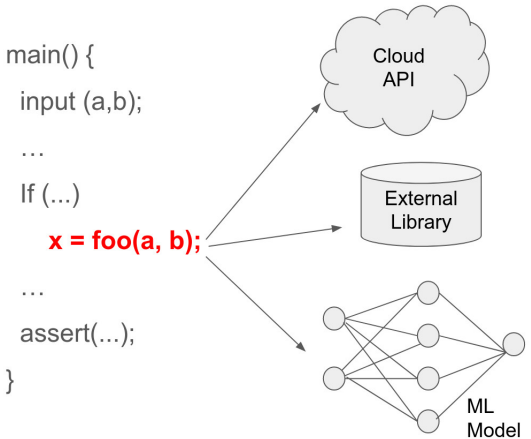
- *Unknown* (ongoing)

Incomplete programs



- *Unknown* (ongoing)
- *Closed-box* or Oracle access (ISSTA'19, ISSTA'22, OOPSLA'22a, **OOPSLA'22b**)

Incomplete programs



- *Unknown* (ongoing)
- *Closed-box* or Oracle access (ISSTA'19, ISSTA'22, OOPSLA'22a, **OOPSLA'22b**)
- Knowledge of probabilistic distribution (S&P(Oakland)'21, CAV'22, OOPSLA'22c)

Solution

Tests + Proofs

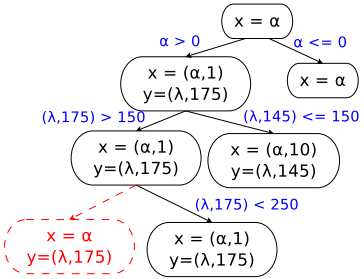
Solution

Tests + Proofs

(fuzzing, random sampling) (SMT solving, Computer Algebra Systems)

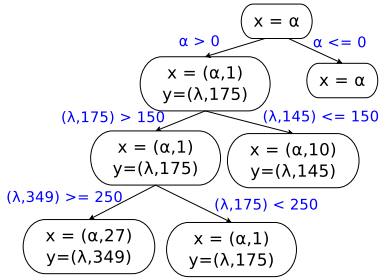
COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



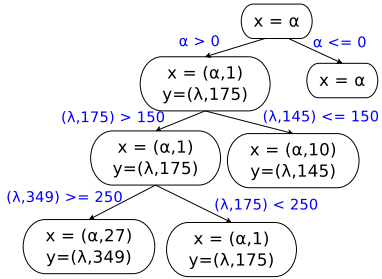
COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

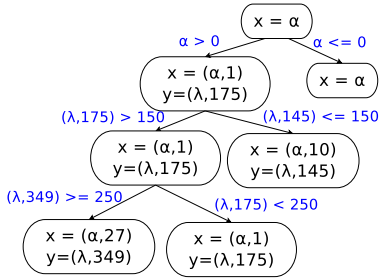
```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



$$\langle\alpha_1,0\rangle \geqslant -10 \wedge \langle\alpha_2,0\rangle = 3*\langle\alpha_1,0\rangle \wedge \langle\alpha_3,0\rangle = \text{ceil}(\langle\alpha_2,0\rangle) \wedge \langle\alpha_3,0\rangle \geqslant 0 \wedge$$
$$\langle\alpha_4,3\rangle = 3+\langle\alpha_3,0\rangle \wedge \langle\alpha_5,27\rangle = \text{pow}(\langle\alpha_4,3\rangle,3) \wedge \langle\alpha_5,27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```

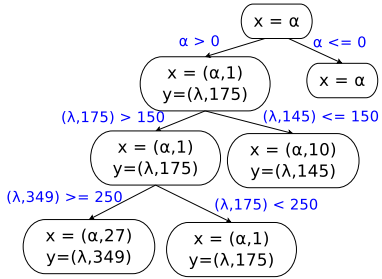


$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$
$$\Uparrow \text{Logic Solver}$$

$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



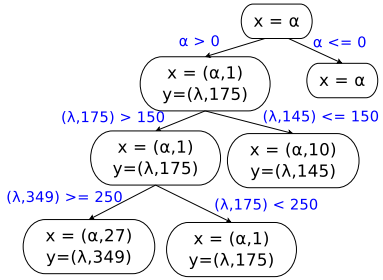
(Unsatisfiable!)

$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$
$$\Uparrow \text{Logic Solver}$$

$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

Colossus: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



(Unsatisfiable!)

$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Uparrow$ Logic Solver

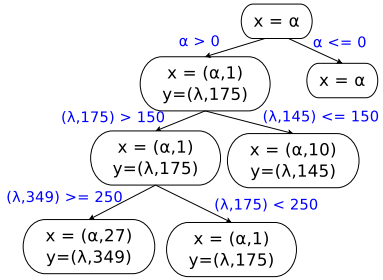
$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Downarrow$ Fuzz Solver

$$\alpha_1 \geqslant -10 \wedge \alpha_2 = 3*\alpha_1 \wedge \alpha_3 = \text{ceil}(\alpha_2) \wedge \alpha_3 \geqslant 0 \wedge$$
$$\alpha_4 = 3 + \alpha_3 \wedge \alpha_5 = \text{pow}(\alpha_4, 3) \wedge \alpha_5 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

Colossus: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



(Unsatisfiable!)

$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Uparrow$ Logic Solver

$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

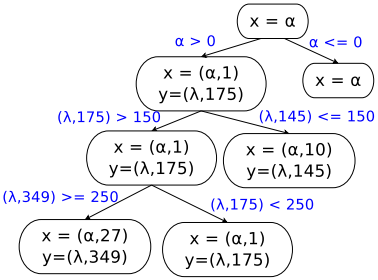
$\Downarrow$ Fuzz Solver

$$\alpha_1 \geqslant -10 \wedge \alpha_2 = 3*\alpha_1 \wedge \alpha_3 = \text{ceil}(\alpha_2) \wedge \alpha_3 \geqslant 0 \wedge$$
$$\alpha_4 = 3 + \alpha_3 \wedge \alpha_5 = \text{pow}(\alpha_4, 3) \wedge \alpha_5 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

(Solution : $\alpha_1 = 1, \alpha_2 = 3, \alpha_3 = 3, \alpha_4 = 6, \alpha_5 = 216$)

COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



(Unsatisfiable!)

$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Uparrow$ Logic Solver

$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Downarrow$ Fuzz Solver

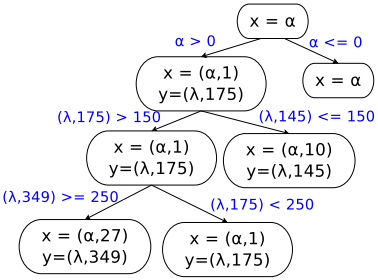
$$\alpha_1 \geqslant -10 \wedge \alpha_2 = 3*\alpha_1 \wedge \alpha_3 = \text{ceil}(\alpha_2) \wedge \alpha_3 \geqslant 0 \wedge$$
$$\alpha_4 = 3 + \alpha_3 \wedge \alpha_5 = \text{pow}(\alpha_4, 3) \wedge \alpha_5 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

(Solution : $\alpha_1 = 1, \alpha_2 = 3, \alpha_3 = 3, \alpha_4 = 6, \alpha_5 = 216$)

COLOSSUS improves branch coverage by 66.94% over concretization and 115% over full symbolic return.

COLOSSUS: Deferred Concretization (*Awanish Pandey et al. ISSTA'19*)

```
int main( ) {
  read (x) ;
  if (x > 0){
    y = foo(x);
    if (y > 150)
      print("less");
⟶  if (y > 250)
      assert(0);
    if (y != x)
      assert(0);
    if (y < 0)
      assert(0);
  }
}
```



(Unsatisfiable!)

$$0 \geqslant -10 \wedge 0 = 3*0 \wedge 0 \geqslant 0 \wedge 3 = 3+0 \wedge 27 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

$\Uparrow$ Logic Solver

$$\langle \alpha_1, 0\rangle \geqslant -10 \wedge \langle \alpha_2, 0\rangle = 3*\langle \alpha_1, 0\rangle \wedge \langle \alpha_3, 0\rangle = \text{ceil}(\langle \alpha_2, 0\rangle) \wedge \langle \alpha_3, 0\rangle \geqslant 0 \wedge$$
$$\langle \alpha_4, 3\rangle = 3+\langle \alpha_3, 0\rangle \wedge \langle \alpha_5, 27\rangle = \text{pow}(\langle \alpha_4, 3\rangle, 3) \wedge \langle \alpha_5, 27\rangle = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

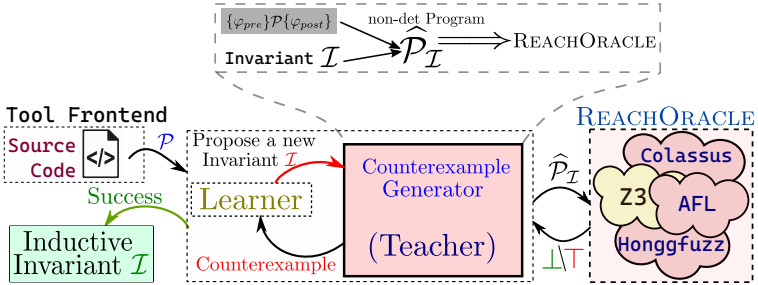
$\Downarrow$ Fuzz Solver

$$\alpha_1 \geqslant -10 \wedge \alpha_2 = 3*\alpha_1 \wedge \alpha_3 = \text{ceil}(\alpha_2) \wedge \alpha_3 \geqslant 0 \wedge$$
$$\alpha_4 = 3 + \alpha_3 \wedge \alpha_5 = \text{pow}(\alpha_4, 3) \wedge \alpha_5 = 216 \wedge$$
$$\alpha_6 + 259 > \alpha_7 \wedge \alpha_7 > 42$$

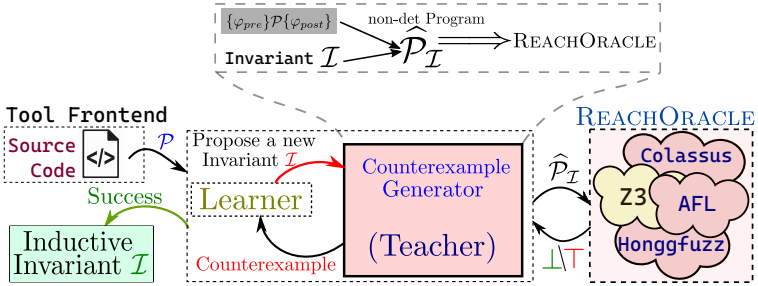
(Solution : $\alpha_1 = 1, \alpha_2 = 3, \alpha_3 = 3, \alpha_4 = 6, \alpha_5 = 216$)

COLOSSUS improves branch coverage by 66.94% over concretization and 115% over full symbolic return. COLOSSUS explores 38.6% extra execution tree missed due to concretization.

ACHAR: Almost Correct Invariants (*Sumit Lahiri et al. ISSTA'22*)

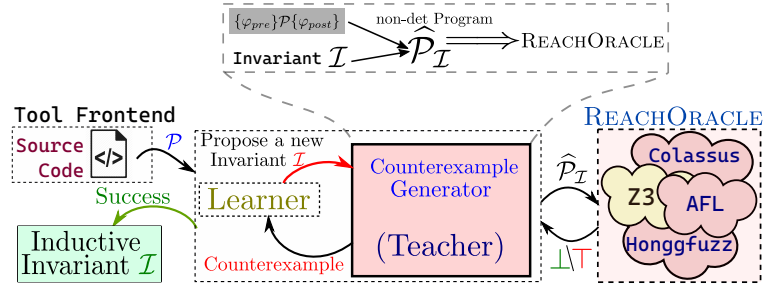


ACHAR: Almost Correct Invariants (*Sumit Lahiri et al. ISSTA'22*)



Proof Fuzzing

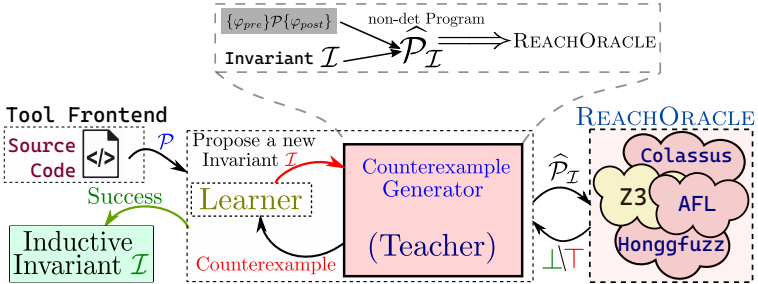
ACHAR: Almost Correct Invariants (*Sumit Lahiri et al. ISSTA'22*)



Proof Fuzzing

- **Theorem**: If a "goal" location in $\widehat{\mathrm{P}}_{\mathcal{I}}$ is *unreachable*, $\mathcal{I}$ is a *valid invariant* for $\{\varphi_{\mathrm{pre}}\}\mathcal{P}\{\varphi_{\mathrm{post}}\}$.

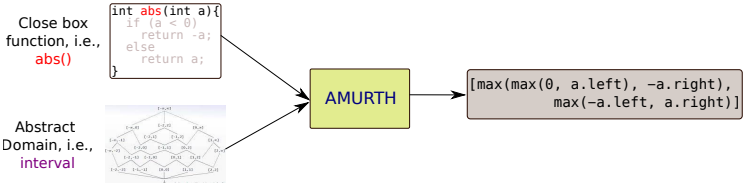
ACHAR: Almost Correct Invariants (*Sumit Lahiri et al. ISSTA'22*)



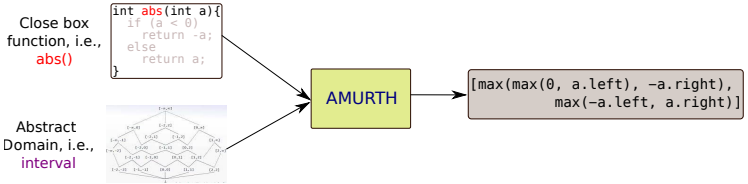
Proof Fuzzing

- **Theorem**: If a "goal" location in $\widehat{\mathrm{P}}_{\mathcal{I}}$ is *unreachable*, $\mathcal{I}$ is a *valid invariant* for $\{\varphi_{\mathrm{pre}}\}\mathcal{P}\{\varphi_{\mathrm{post}}\}$.
- Use fuzzing to test for reachability in $\hat{\mathrm{P}}_{\mathcal{I}}$.

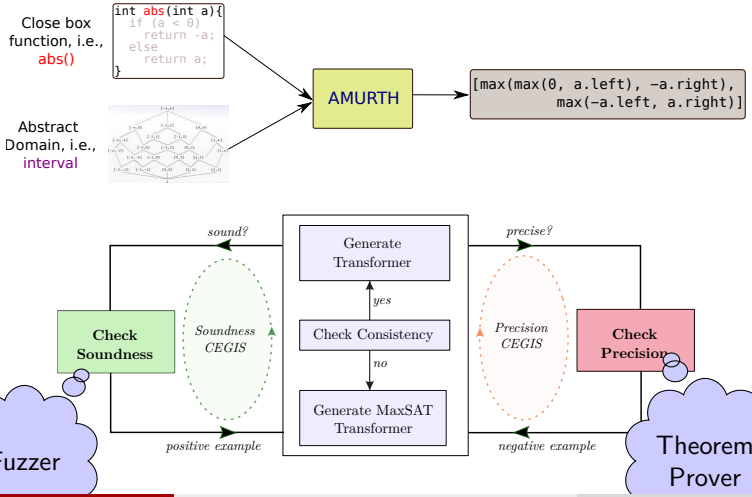
AMURTH: Synthesizing Abstract Transformers (*Pankaj Kalita et al. OOPSLA'22*)



AMURTH: Synthesizing Abstract Transformers (*Pankaj Kalita et al. OOPSLA'22*)

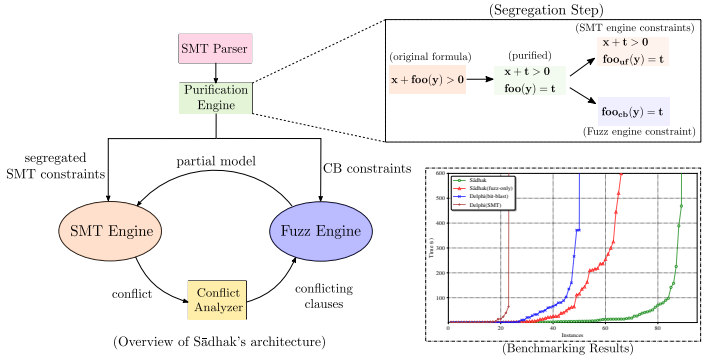


AMURTH: Synthesizing Abstract Transformers (*Pankaj Kalita et al. OOPSLA'22*)



Today's Focus: SĀDHAK

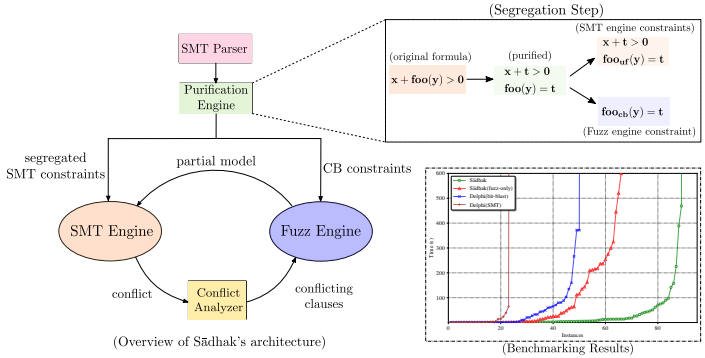
SĀDHAK: Satisfiability Modulo Fuzzing (*Sujit Muduli et al. OOPSLA'22*)



(Overview of Sādhak's architecture)

(Benchmarking Results)

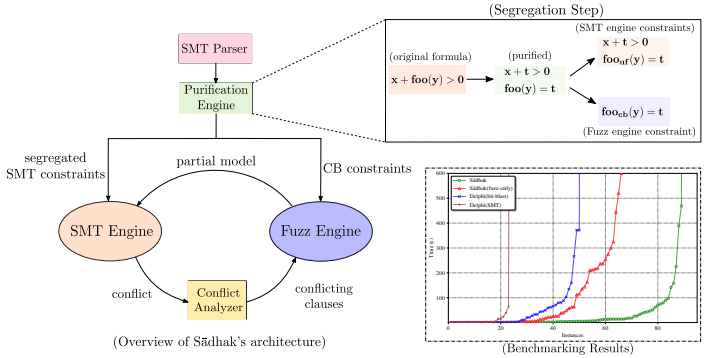
SĀDHAK: Satisfiability Modulo Fuzzing (*Sujit Muduli et al. OOPSLA'22*)



Conflict Driven Fuzz Loop (CDFL)

- We introduce the theory of **Closed-box (CB) functions**;

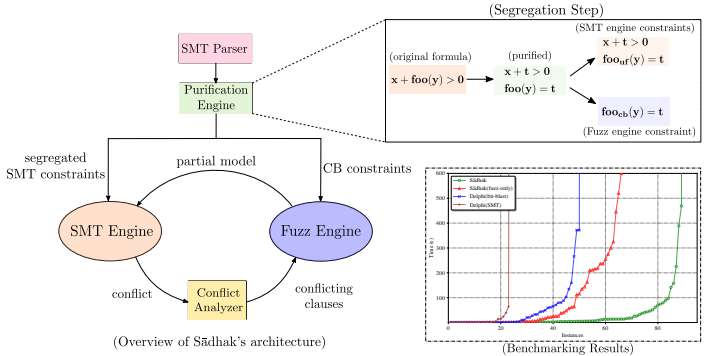
SĀDHAK: Satisfiability Modulo Fuzzing (*Sujit Muduli et al. OOPSLA'22*)



Conflict Driven Fuzz Loop (CDFL)

- We introduce the theory of **Closed-box (CB) functions**;

SĀDHAK: Satisfiability Modulo Fuzzing (*Sujit Muduli et al. OOPSLA'22*)



Conflict Driven Fuzz Loop (CDFL)

- We introduce the theory of **Closed-box (CB) functions**;
- The procedure is sound but not complete.

Introduction

$$(\mathfrak{u} = 6) \wedge (\mathfrak{bar}(\mathfrak{u}, \mathfrak{v}) = 42)$$

Introduction

$$(u = 6) \wedge (\mathtt{bar}(u, v) = 42)$$

- External library call
- Web API call

Introduction

$$(\mathfrak{u} = 6) \wedge (\mathtt{bar}(\mathfrak{u}, \mathfrak{v}) = 42)$$

Closed-box (CB) Function

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Closed-Box (CB) Function

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Closed-Box (CB) Function

1. It must be functional

Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

Closed-Box (CB) Function

1. It must be functional
2. An input-output oracle interface is available

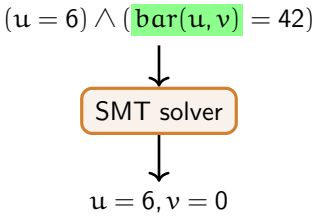
Problem Statement

Test satisfiability of first-order logic constraints containing closed-box (CB) functions

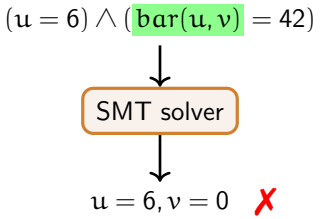
Closed-Box (CB) Function

1. It must be functional
2. An input-output oracle interface is available

Constraint Solving with UF Theory



Constraint Solving with UF Theory




```
u32 bar(u32 u, u32 v) {
   return u * v;
}
```

Core Idea

1. **Reduce satisfiability of a formula to reachability in a program**

Core Idea

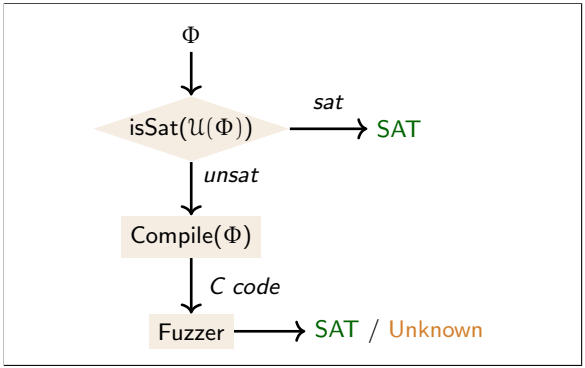
1. **Reduce satisfiability of a formula to reachability in a program**
2. **Use fuzzing to solve the reachability problem**

Prior Works

Colossus

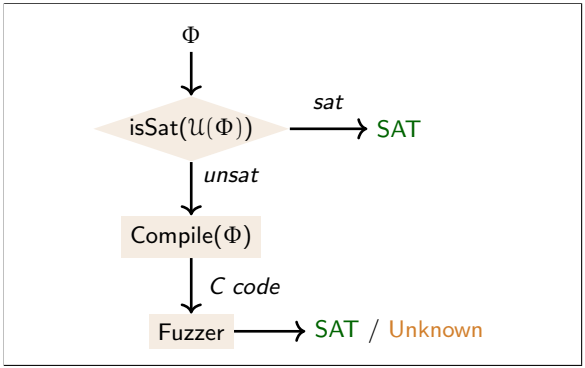
Prior Works

COLOSSUS



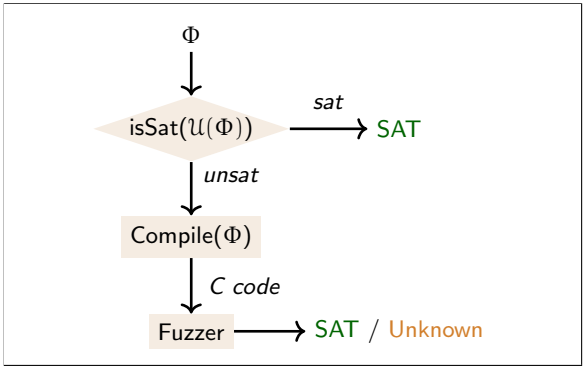
Prior Works

COLOSSUS



Prior Works

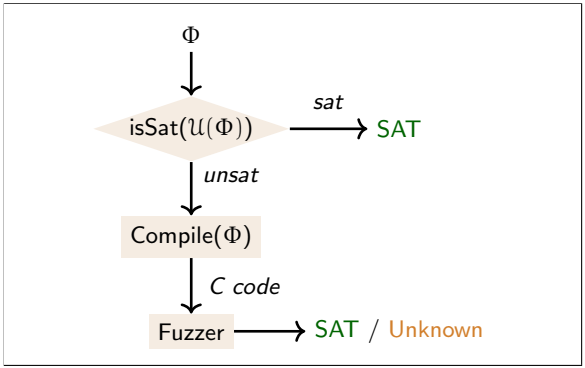
Colossus



- Underapproximate the input formula and check for satisfiability

Prior Works

Colossus



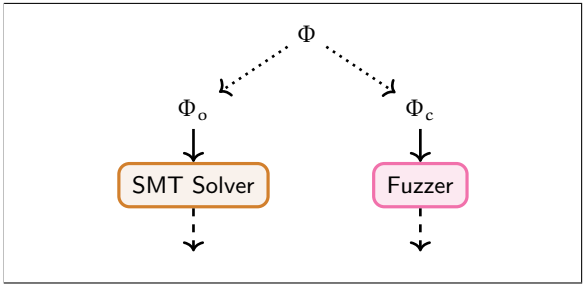
- Underapproximate the input formula and check for satisfiability
- If the underapproximated formula is UNSAT, Colossus uses fuzzing

Prior Works

ACHAR

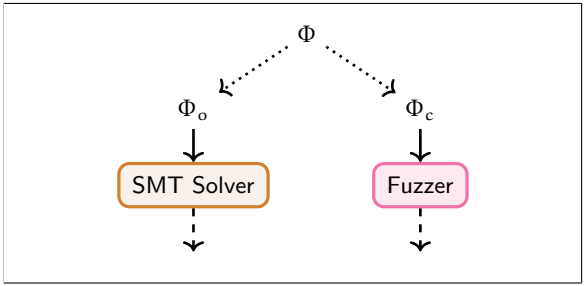
Prior Works

ACHAR



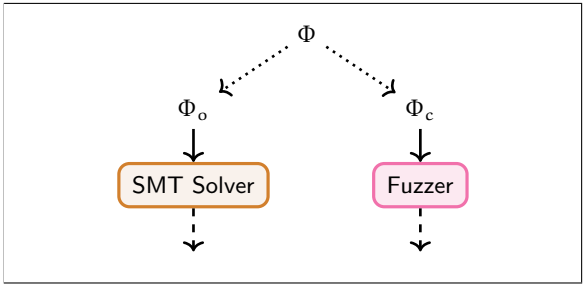
Prior Works

ACHAR



Prior Works

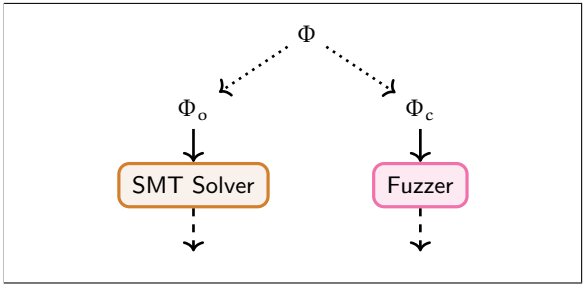
ACHAR



- Create a disjunctive partition of open-box and closed-box components

Prior Works

ACHAR



- Create a disjunctive partition of open-box and closed-box components
- Use SMT for open-box and fuzzing for closed-box

Contributions

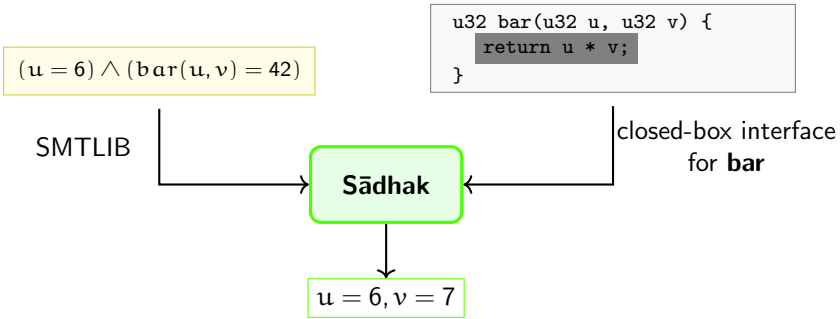
1. Introduced CB theory to support closed-box functions in SMT solvers

Contributions

1. Introduced CB theory to support closed-box functions in SMT solvers
2. A conflict-driven fuzz loop (CDFL) algorithm for solving CB constraints

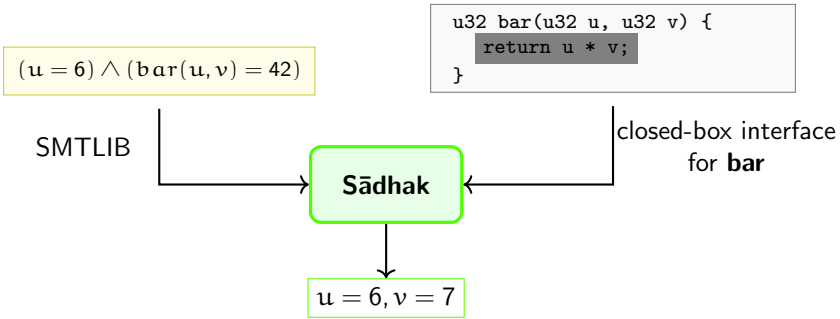
Synergistic Combination of SMT Solving and Fuzzing

In SĀDHAK SMT solver and fuzzer communicate with each other, exchanging information for efficient solving of CB constraints.



Synergistic Combination of SMT Solving and Fuzzing

In SĀDHAK SMT solver and fuzzer communicate with each other, exchanging information for efficient solving of CB constraints.



SĀDHAK is sound but not complete

SMTLIB Encoding

$$(u = 6) \wedge (bar(u, v) = 42)$$

(SMTLIB encoding of above constraint with closed-box function)

```
1 (declare-const u (_ BitVec 32))
2 (declare-const v (_ BitVec 32))
3 (declare-cb bar ((_ BitVec 32) (_ BitVec 32)) (_ BitVec 32))
4 (assert (= u (_ bv6 32)))
5 (assert (= (bar u v) (_ bv42 32)))
6 (check-sat)
7 (get-model)
```

CB Theory

CB Theory

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories

CB Theory

$$\sum_{CB}\{T_1,T_2,\dots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories

CB Theory

$$\sum_{CB}\{T_1,T_2,\dots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories
- F : set of all (sorted) function symbols

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories
- F : set of all (sorted) function symbols
- F^{CB} : set of (sorted) closed-box functions

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories
- F : set of all (sorted) function symbols
- F^{CB} : set of (sorted) closed-box functions
- B : predicates from theories

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories
- F : set of all (sorted) function symbols
- F^{CB} : set of (sorted) closed-box functions
- B : predicates from theories
- R : schema for translating sorts and expressions into a program

CB Theory

$$\sum_{CB}\{T_1,T_2,\ldots\} = \langle S, C, F, F^{CB}, B, R\rangle$$

- S : set of sorts in theories
- C : set of (sorted) constants in theories
- F : set of all (sorted) function symbols
- F^{CB} : set of (sorted) closed-box functions
- B : predicates from theories
- R : schema for translating sorts and expressions into a program

Example

closed-box function

$$\mathbf{f}(x, y) > 255 \wedge \mathbf{f}(x, y) < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ \mathrm{isPow2}(p) \wedge \mathrm{isPow2}(q) \wedge \mathrm{isPow2}(r) \wedge (p \times q \times r = 64)$$

Segregation

Purification

$$\mathbf{f}(x, y) > 255 \wedge \mathbf{f}(x, y) < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ \mathrm{isPow2}(p) \wedge \mathrm{isPow2}(q) \wedge \mathrm{isPow2}(r) \wedge (p \times q \times r = 64)$$

$$\mathbf{z} = \mathrm{f}(x, y) \wedge \mathbf{z} > 255 \wedge \mathbf{z} < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ \mathrm{isPow2}(p) \wedge \mathrm{isPow2}(q) \wedge \mathrm{isPow2}(r) \wedge (p \times q \times r = 64)$$

(Purification)

Segregation

Separation

$$z = f(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ \mathrm{isPow2}(p) \wedge \mathrm{isPow2}(q) \wedge \mathrm{isPow2}(r) \wedge (p \times q \times r = 64)$$

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ \mathrm{isPow2}(p) \wedge \mathrm{isPow2}(q) \wedge \mathrm{isPow2}(r)$$
$$\wedge\ (p \times q \times r = 64)$$

Fuzz Engine

$$z = f_{cb}(x, y)$$

(Separation)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$

Fuzz Engine

$$z = f_{cb}(x, y)$$

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$

Fuzz Engine

$$z = f_{cb}(x, y)$$

$x = 0, y = 0, z = 0$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$

Fuzz Engine

$$z = f_{cb}(x, y)$$

propagate

$x = 0, y = 0, z = 0$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1

SMT Engine

$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$
$\wedge \boxed{(x > y)} \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$
$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$
$\wedge\ (p \times q \times r = 64)$

Fuzz Engine

$z = f_{cb}(x, y)$

$x = 0, y = 0, z = 0$

(partial model)

(conflict)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1

SMT Engine

$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$

$\wedge$ $(x > y)$ $\wedge$ $(p \geq 1) \wedge (q > 1) \wedge (r > 1)$

$\wedge$ $isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$

$\wedge$ $(p \times q \times r = 64)$

lemma

Fuzz Engine

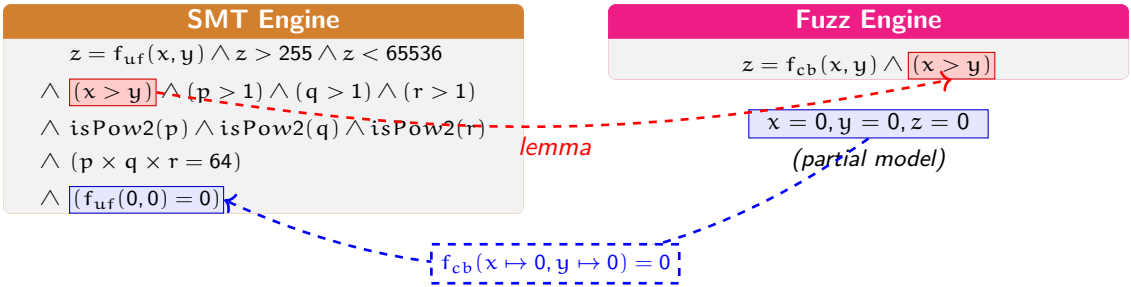
$z = f_{cb}(x, y) \wedge (x > y)$

$x = 0, y = 0, z = 0$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 1



Conflict-driven Fuzz Loop (CDFL)

Iteration - 2

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y)$$

Conflict-driven Fuzz Loop (CDFL)

Iteration - 2

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y)$$

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

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 2

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y)$$

propagate

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

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 2

SMT Engine

$z = f_{uf}(x, y) \wedge \boxed{z > 255} \wedge z < 65536$
$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$
$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$
$\wedge\ (p \times q \times r = 64)$
$\wedge\ (f_{uf}(0, 0) = 0)$

(conflict)

Fuzz Engine

$z = f_{cb}(x, y) \wedge (x > y)$

$\boxed{x = 1, y = 0, z = 0}$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 2

SMT Engine

$z = f_{uf}(x, y) \wedge \boxed{z > 255} \wedge z < 65536$
$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$
$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$
$\wedge\ (p \times q \times r = 64)$
$\wedge\ (f_{uf}(0, 0) = 0)$

Fuzz Engine

$z = f_{cb}(x, y) \wedge (x > y) \wedge \boxed{(z > 255)}$

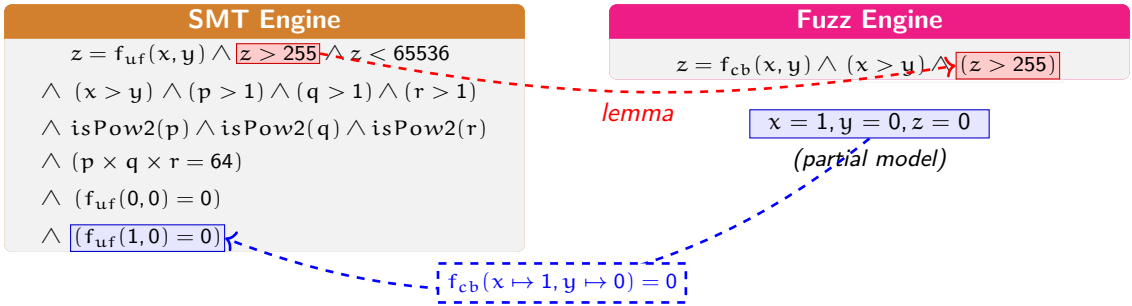
lemma

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

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 2



Conflict-driven Fuzz Loop (CDFL)

Iteration - 3

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$

Conflict-driven Fuzz Loop (CDFL)

Iteration - 3

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$

$x = 65536, y = 1, z = 65536$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 3

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$

propagate

$x = 65536, y = 1, z = 65536$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 3

SMT Engine

$z = f_{uf}(x, y) \wedge z > 255 \wedge \boxed{z < 65536}$

$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$

$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2$

$\wedge\ (p \times q \times r = 64)$

$\wedge\ (f_{uf}(0, 0) = 0)$

$\wedge\ (f_{uf}(1, 0) = 0)$

(conflict)

Fuzz Engine

$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$

$x = 65536, y = 1, z = 65536$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 3

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge \boxed{z < 65536}$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$

lemma

Fuzz Engine

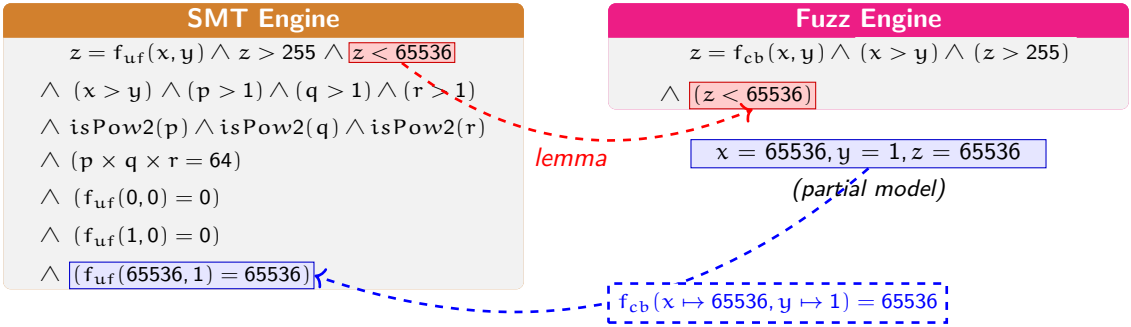
$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ \boxed{(z < 65536)}$$

$$x = 65536, y = 1, z = 65536$$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 3



Conflict-driven Fuzz Loop (CDFL)

Iteration - 4

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$
$$\wedge\ (f_{uf}(65536, 1) = 65536)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ (z < 65536)$$

Conflict-driven Fuzz Loop (CDFL)

Iteration - 4

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$
$$\wedge\ (f_{uf}(65536, 1) = 65536)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ (z < 65536)$$

$x = 256, y = 1, z = 256$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 4

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$
$$\wedge\ (f_{uf}(65536, 1) = 65536)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ (z < 65536)$$

propagate

$x = 256, y = 1, z = 256$

(partial model)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 4

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$
$$\wedge\ (f_{uf}(65536, 1) = 65536)$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ (z < 65536)$$

$x = 256, y = 1, z = 256$

(partial model)

(consistent)

Conflict-driven Fuzz Loop (CDFL)

Iteration - 4

SMT Engine

$$z = f_{uf}(x, y) \wedge z > 255 \wedge z < 65536$$
$$\wedge\ (x > y) \wedge (p > 1) \wedge (q > 1) \wedge (r > 1)$$
$$\wedge\ isPow2(p) \wedge isPow2(q) \wedge isPow2(r)$$
$$\wedge\ (p \times q \times r = 64)$$
$$\wedge\ (f_{uf}(0, 0) = 0)$$
$$\wedge\ (f_{uf}(1, 0) = 0)$$
$$\wedge\ (f_{uf}(65536, 1) = 65536)$$

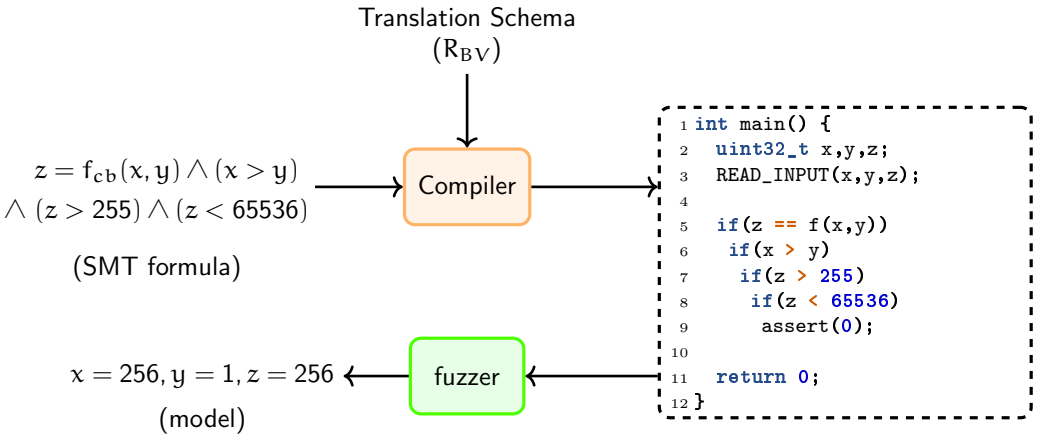
$$x = 256, y = 1, z = 256,$$
$$p = 2, q = 8, r = 4$$

Fuzz Engine

$$z = f_{cb}(x, y) \wedge (x > y) \wedge (z > 255)$$
$$\wedge\ (z < 65536)$$

```
u32 f(u32 x, u32 y) {
  return x * y;
}
```

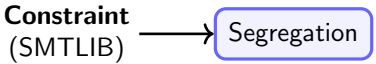
Fuzz Engine


Translation Schema
(R_{BV})
$z = f_{cb}(x, y) \wedge (x > y)$
$\wedge\ (z > 255) \wedge (z < 65536)$
(SMT formula)
Compiler
1 int main() {
2 uint32_t x,y,z;
3 READ_INPUT(x,y,z);
4
5 if(z == f(x,y))
6 if(x > y)
7 if(z > 255)
8 if(z < 65536)
9 assert(0);
10
11 return 0;
12 }
fuzzer
$x = 256, y = 1, z = 256$
(model)

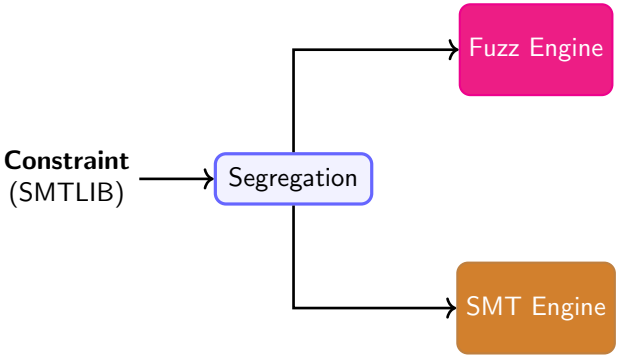
SĀDHAK: An SMT Solver for CB Constraints

Constraint
(SMTLIB)

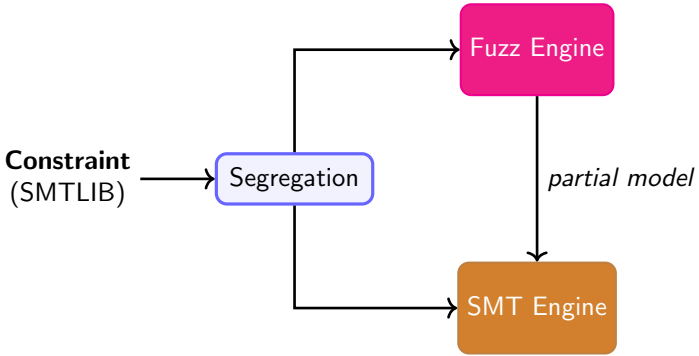
SĀDHAK: An SMT Solver for CB Constraints



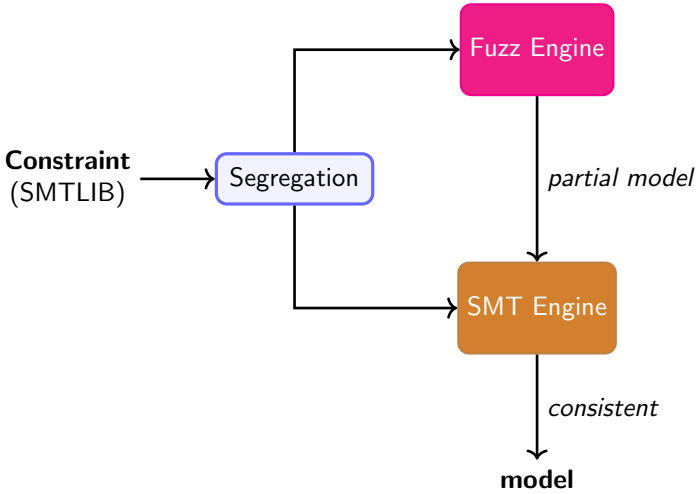
SĀDHAK: An SMT Solver for CB Constraints



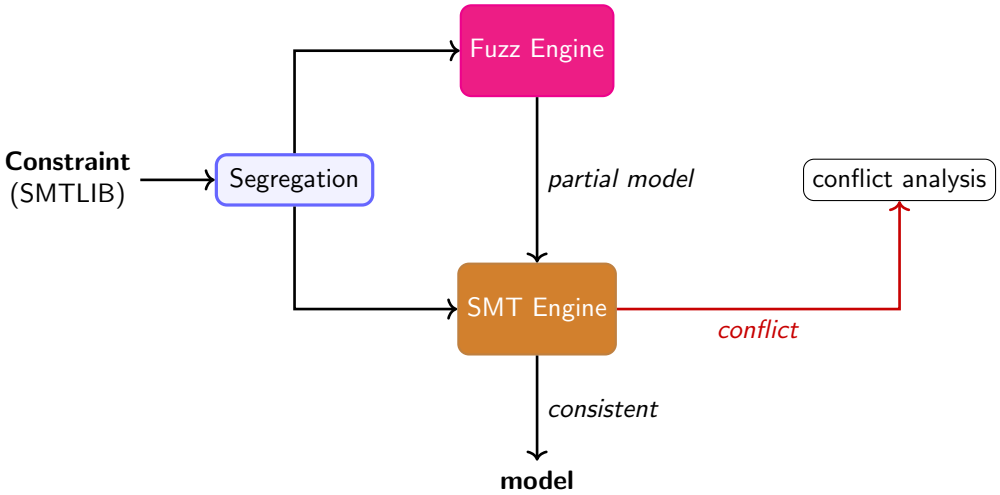
SĀDHAK: An SMT Solver for CB Constraints



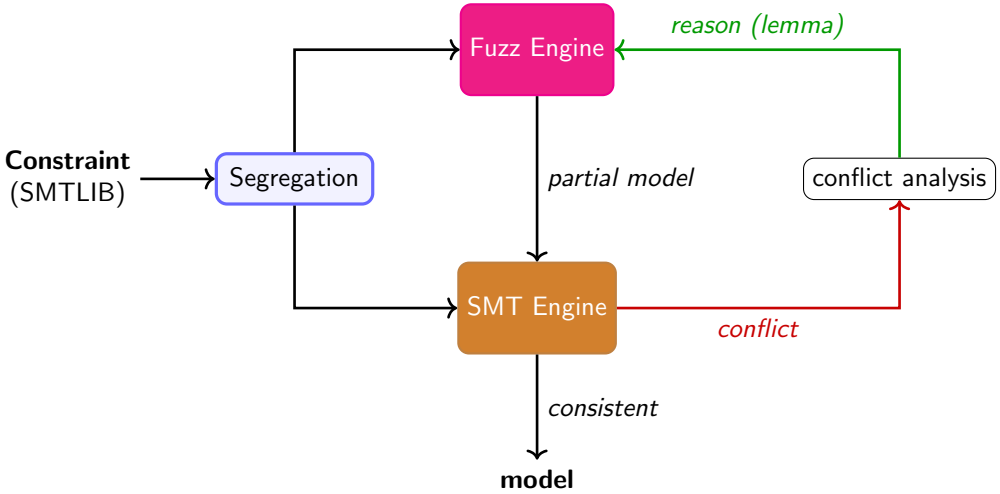
SĀDHAK: An SMT Solver for CB Constraints



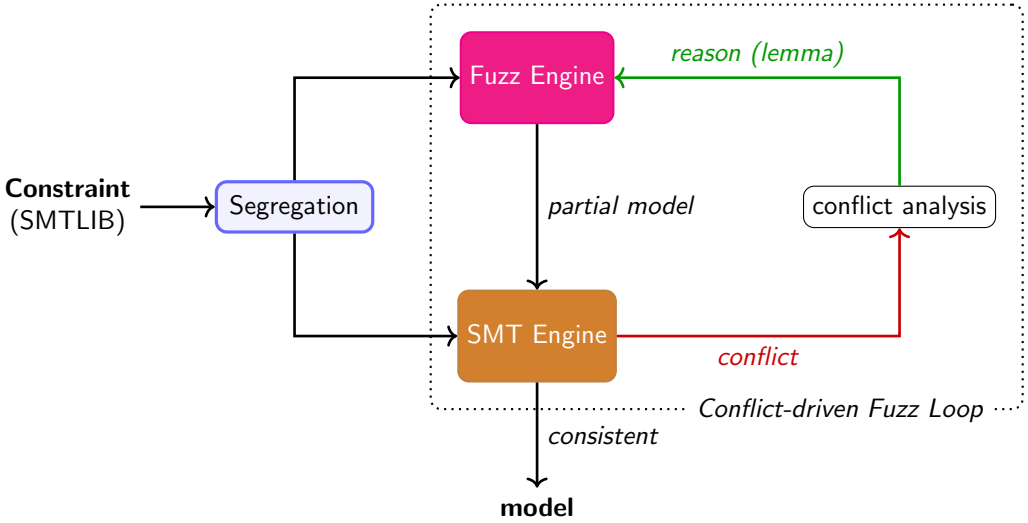
SĀDHAK: An SMT Solver for CB Constraints



SĀDHAK: An SMT Solver for CB Constraints



SĀDHAK: An SMT Solver for CB Constraints

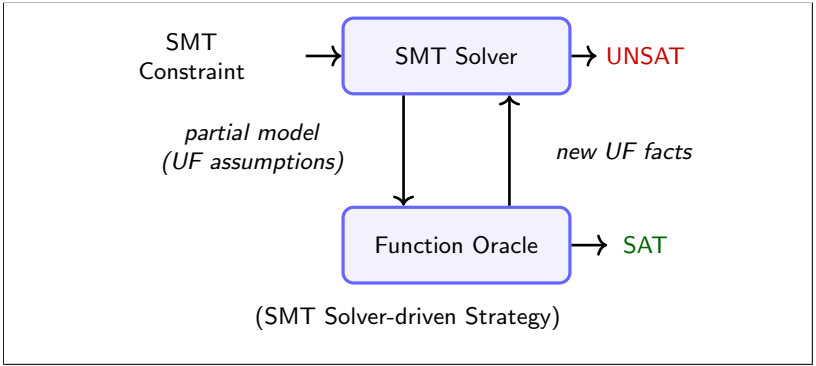


Related Work

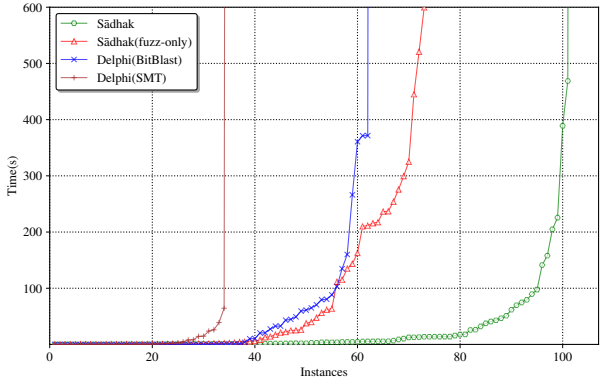
DELPHI [*Polgreen et al. VMCAI'22*]

Related Work

DELPHI [*Polgreen et al. VMCAI'22*]



Evaluation



	Sādhak		Delphi	
	Fuzz only	CDFL	SMT	BitBlast
# solved	73	**101**	34	62

Conclusion

- Introduced closed-box function theory (CB theory)

(paper and artifact)

(Thanks to Sujit for letting me steal his slides :))

Conclusion

- Introduced closed-box function theory (CB theory)
- Conflict-driven fuzz loop (CDFL) for using fuzzing in synergy with the SMT solving

(paper and artifact)

(Thanks to Sujit for letting me steal his slides :))

Conclusion

- Introduced closed-box function theory (CB theory)
- Conflict-driven fuzz loop (CDFL) for using fuzzing in synergy with the SMT solving
- Sādhak is built on top of CVC4 SMT solver.

(paper and artifact)

(Thanks to Sujit for letting me steal his slides :))

Conclusion

- Introduced closed-box function theory (CB theory)
- Conflict-driven fuzz loop (CDFL) for using fuzzing in synergy with the SMT solving
- Sādhak is built on top of CVC4 SMT solver.
- A set of 95 new benchmarks (SMTLIB queries with closed-box constraints)

(paper and artifact)

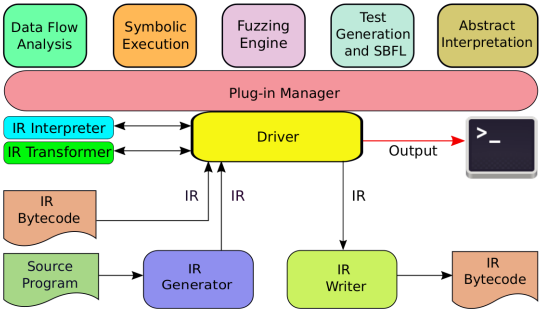
(Thanks to Sujit for letting me steal his slides :))

CHIRON: A framework to teach program analysis [ASE'23]

Fun Source Language, Simple IR, Plug-and-play Architecture, Small code-base

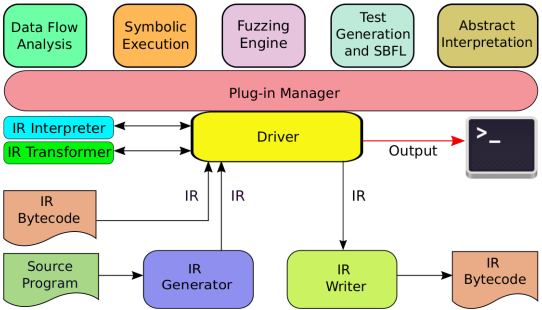
CHIRON: A framework to teach program analysis [ASE'23]

Fun Source Language, Simple IR, Plug-and-play Architecture, Small code-base



CHIRON: A framework to teach program analysis [ASE'23]

Fun Source Language, Simple IR, Plug-and-play Architecture, Small code-base



Module	Module Size[#]	Module Effort	Student Codesize[*]
CORE FRAMEWORK	0.77	8	-
DATA FLOW ANALYSIS	0.14	2	0.2–0.3
FUZZING	0.11	2	0.06–0.1
SYMBOLIC EXECUTION	0.31	3	0.15–0.17
ABSTRACT INTERPRETATION	0.15	2	0.1–0.3
SBFL FRAMEWORK	0.24	2	0.03–0.12

[*]Code written by students for the assignments. [#]Code written by CHIRON developers for framing the assignments.

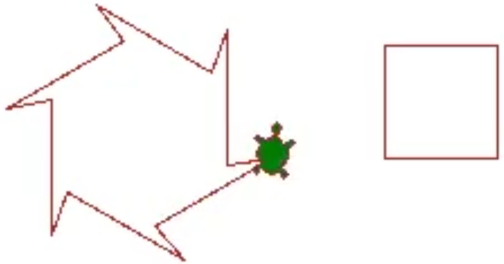
Example: Abstract Interpretation with CHIRON

Example: Abstract Interpretation with CHIRON

```
input(:vara, :varb, :varc)
goto (0, 0)
pendown
repeat 6 [
  if (:vara != :varb) [
    if ( :vara > :varb) [ right :vara ] else [ left :varb ]
  ]
  else [
    if ((:vara <= :varc) || (:varb <= :varc)) [
      :vara = :varc / :vara
      :varb = :varb / :varc
      :varc = :varb
    ]
  ]
  forward :vara
  right :varb
  forward :varc
  left :varc
]
penup
```

Example: Abstract Interpretation with CHIRON

```
input(:vara, :varb, :varc)
goto (0, 0)
pendown
repeat 6 [
   if (:vara != :varb) [
      if ( :vara > :varb) [ right :vara ] else [ left :varb ]
   ]
   else [
     if ((:vara <= :varc) || (:varb <= :varc)) [
        :vara = :varc / :vara
        :varb = :varb / :varc
        :varc = :varb
      ]
   ]
   forward :vara
   right :varb
   forward :varc
   left :varc
]
penup
```



Questions?