ETH zürich

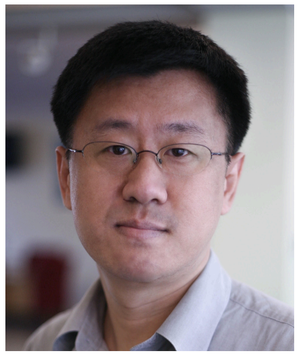
AST
ADVANCED SOFTWARE TECHNOLOGIES

Finding and Understanding Incompleteness Bugs in SMT Solvers

Mauro Bringolf
ETH Zurich, Switzerland

Dominik Winterer
ETH Zurich, Switzerland

Zhendong Su
ETH Zurich, Switzerland



ASE ‘22, Oct 13, 2022 (virtual)

SMT Problem

SMT Problem

$$\varphi : x > 0 \wedge x < 0$$

SMT Problem

$$\varphi : x > 0 \wedge x < 0$$

UNSAT

SMT Problem

$$\varphi : x > 0 \wedge x < 1$$

SMT Problem

$$\varphi : x > 0 \wedge x < 1$$

SAT

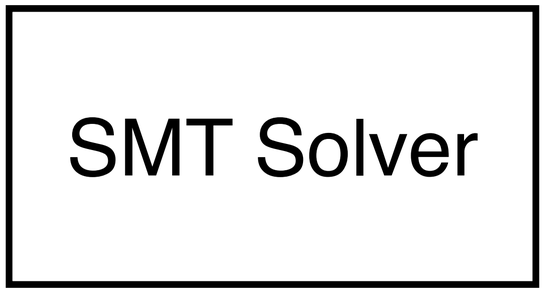
SMT Problem

$$x = 0.5$$

$$\varphi : x > 0 \wedge x < 1$$

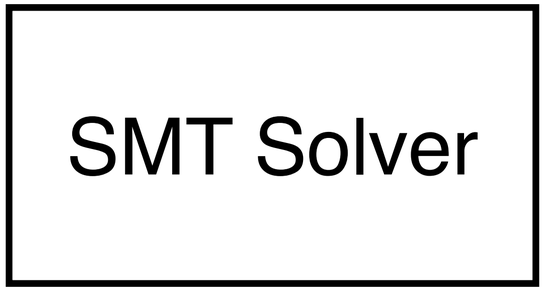
SAT

SMT Solver

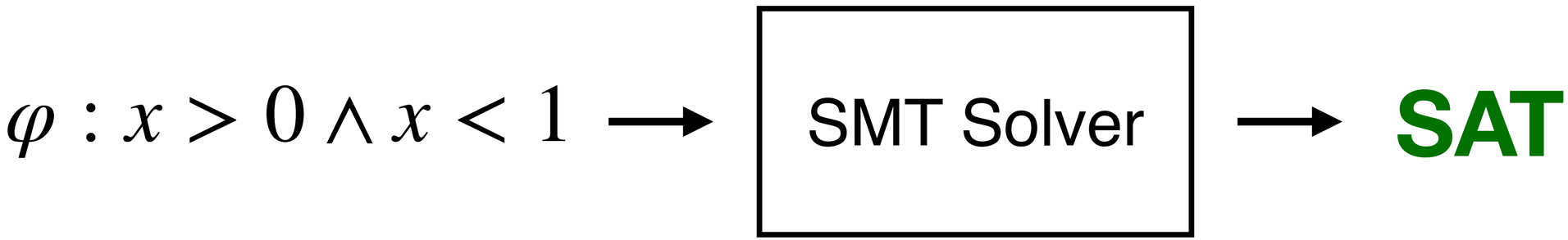


SMT Solver

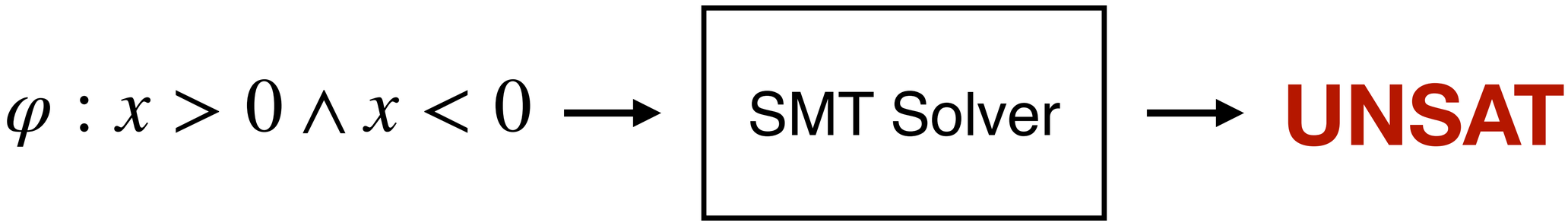
$$\varphi : x > 0 \wedge x < 1 \longrightarrow$$



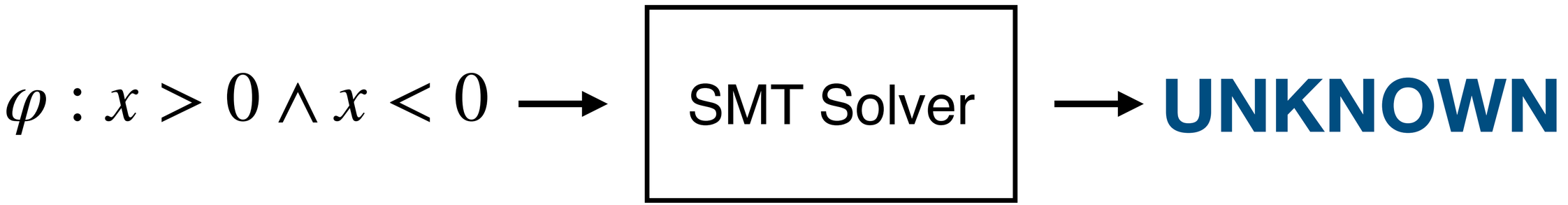
SMT Solver



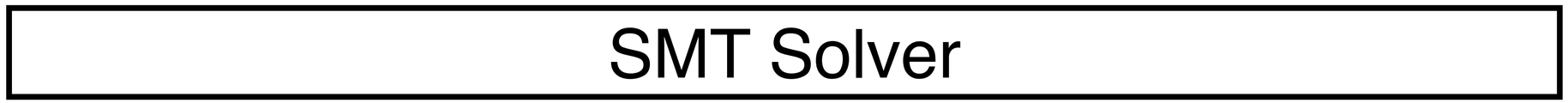
SMT Solver



SMT Solver

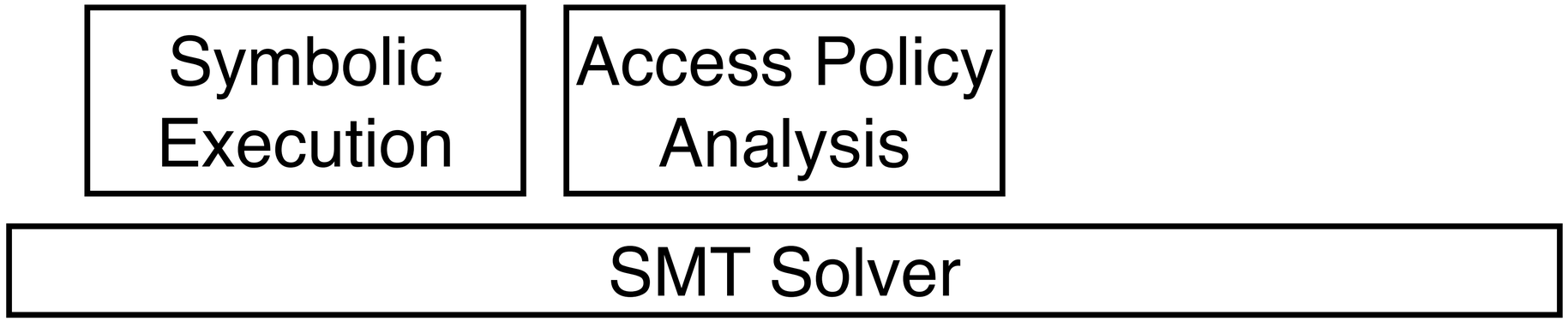


SMT Solver

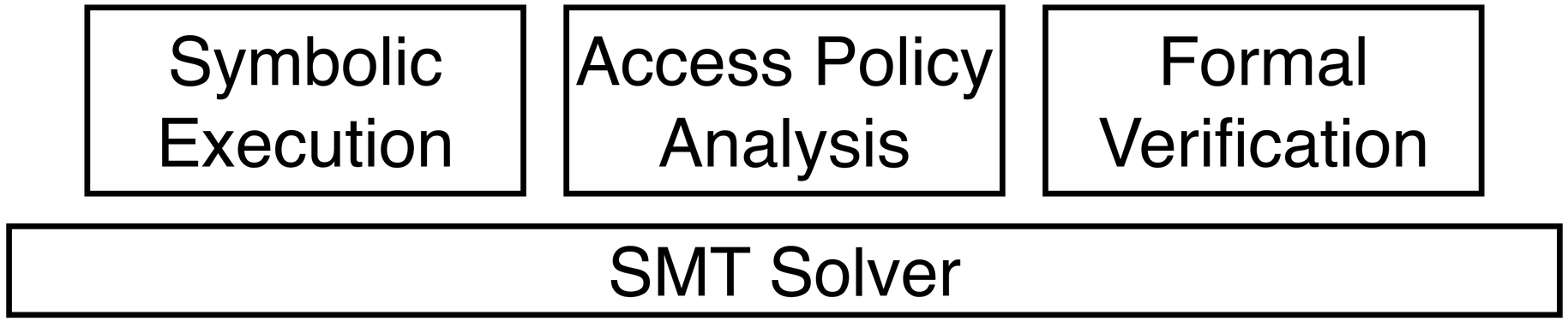


SMT Solver

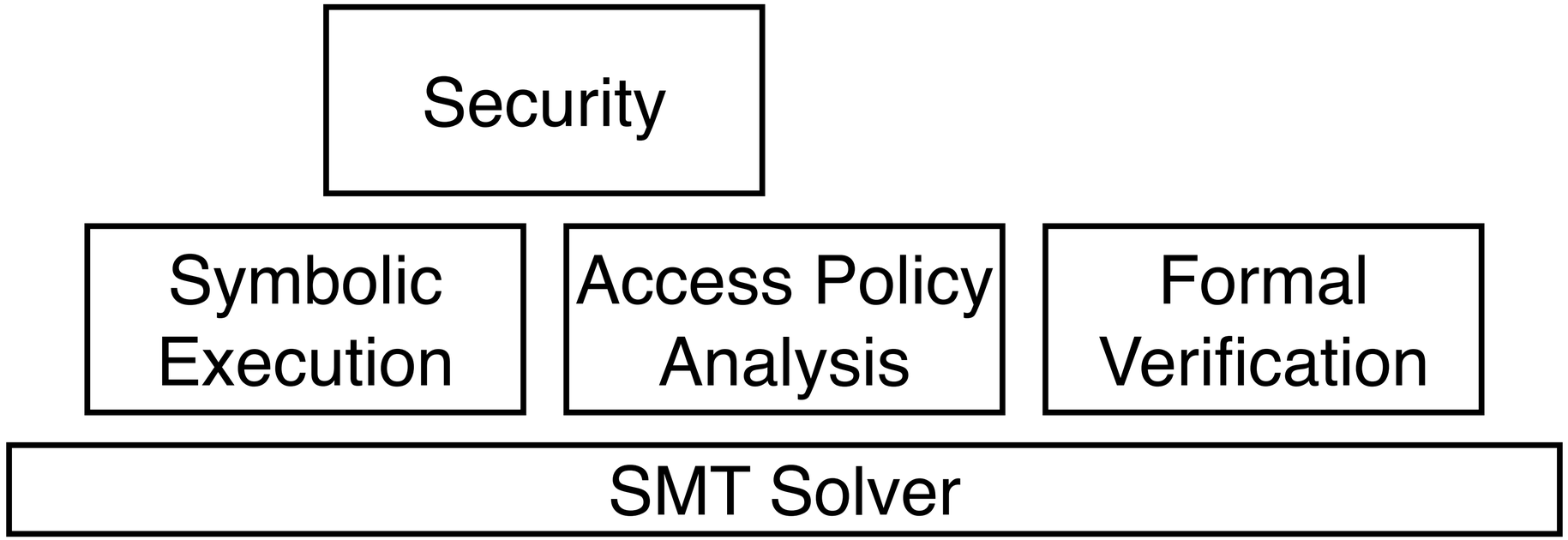
SMT Solver



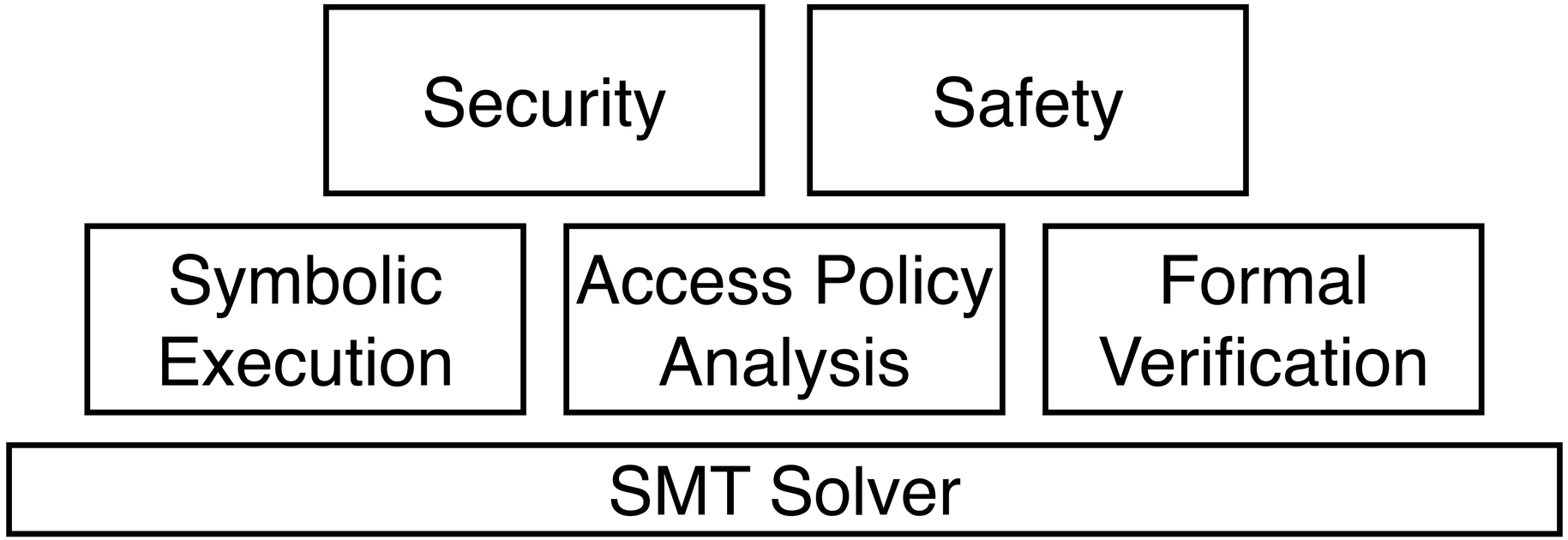
SMT Solver


Symbolic Execution
Access Policy Analysis
Formal Verification
SMT Solver

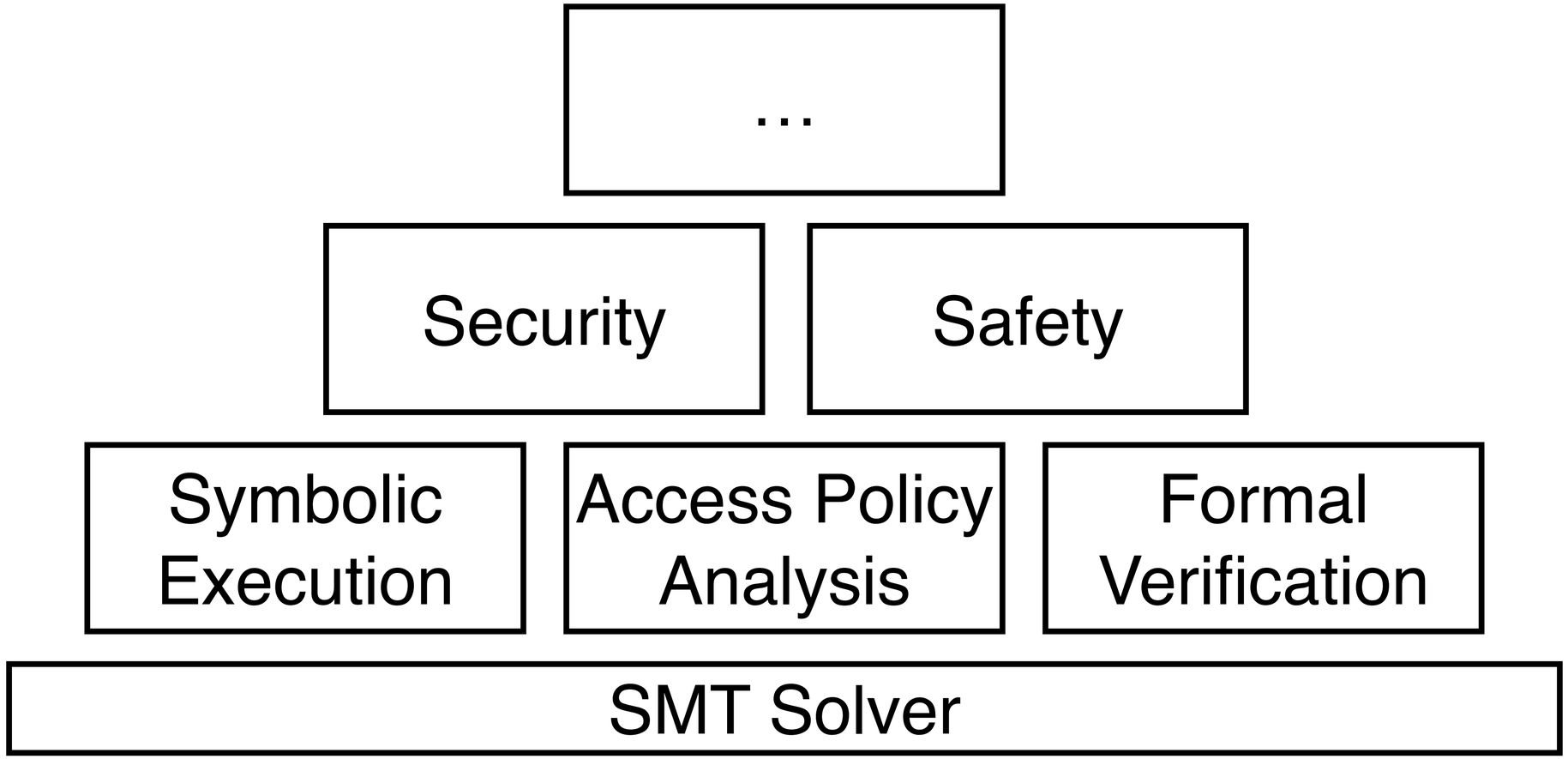
SMT Solver


Security
Symbolic Execution
Access Policy Analysis
Formal Verification
SMT Solver

SMT Solver


Security
Safety
Symbolic Execution
Access Policy Analysis
Formal Verification
SMT Solver

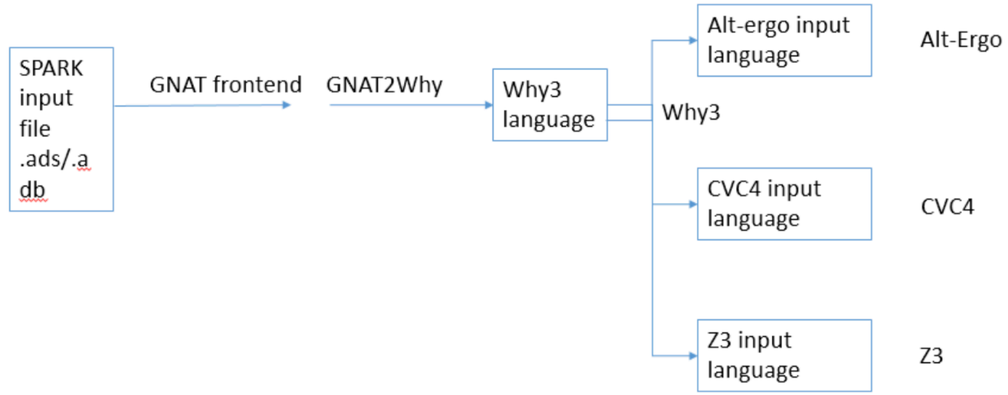
SMT Solver


...
Security
Safety
Symbolic Execution
Access Policy Analysis
Formal Verification
SMT Solver

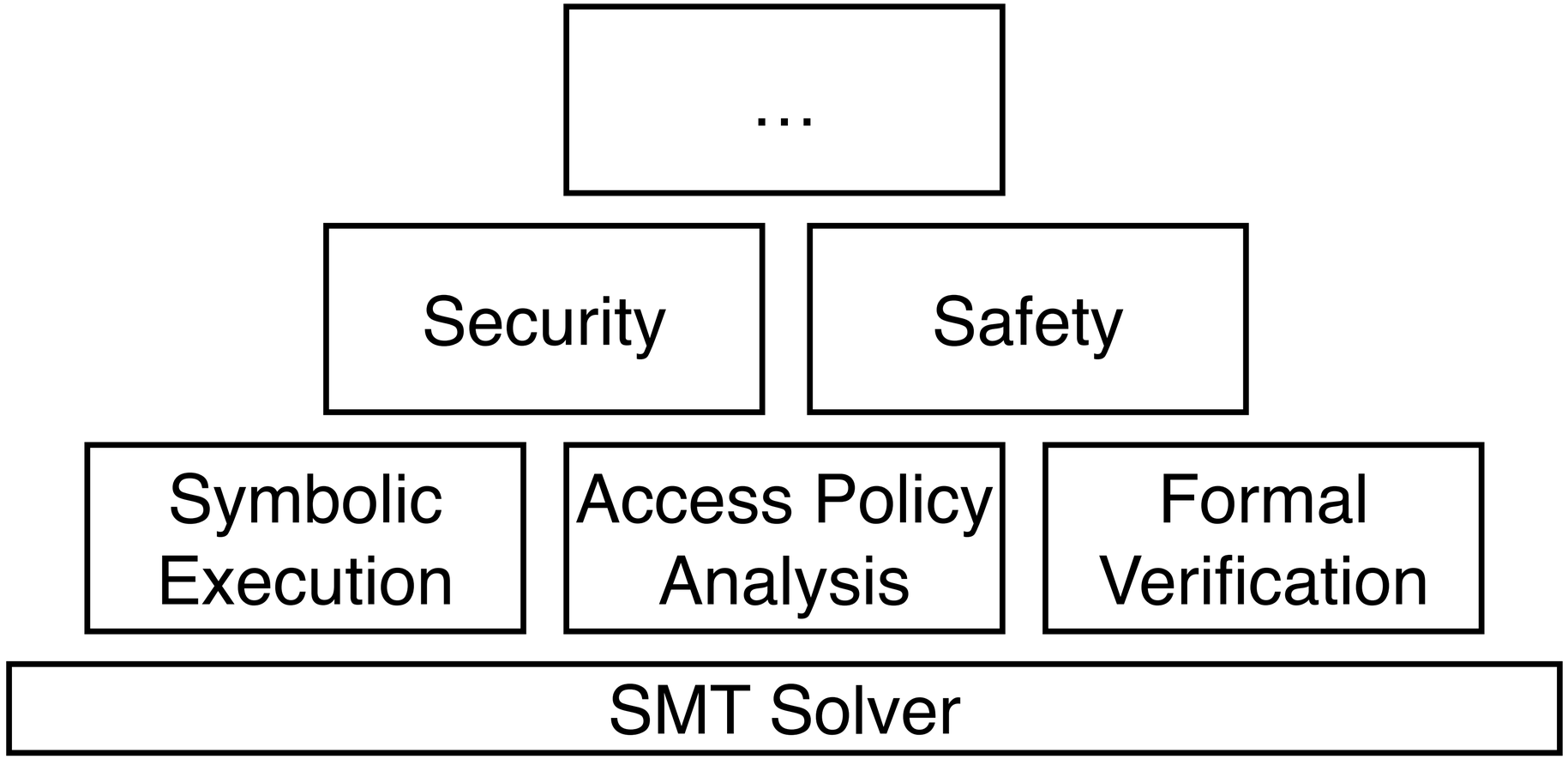
Industrial Applications


```
{
    "Resource": “bucket1”
    "Prohibited": { "AWS": “3333"},
}
```

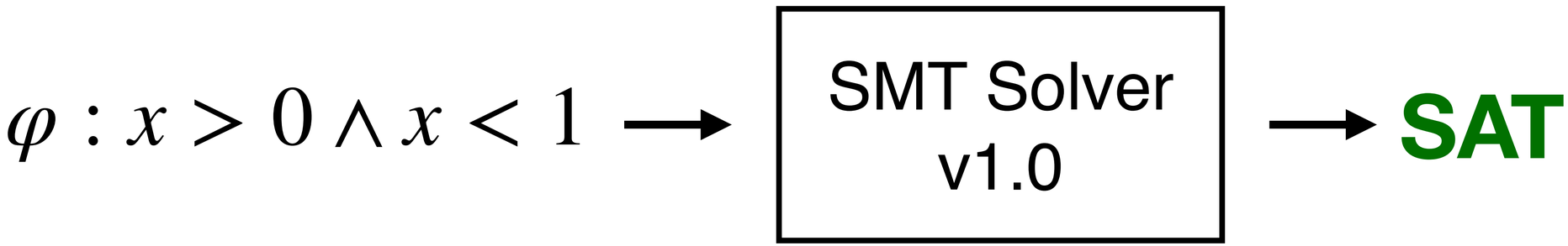
$(Resource = “bucket1”) \wedge (Prohibited \neq 3333)$



SMT Solvers should be reliable and fast!



Incompleteness Bug



Incompleteness Bug

Incompleteness Bug

Incompleteness Bugs frustrate users!

"I'm seeing a regression [...], where a lot of simple formulas that used to be unsat now give unknown."

https://github.com/Z3Prover/z3/issues/5516

"The following code will produce unsat in z3 version 4.8.10.0 but is unknown in later versions."

https://github.com/Z3Prover/z3/issues/5438

"This is pretty unexpected since the query is small and does not contain features where we would expect to see performance regressions when updating releases."

https://github.com/Z3Prover/z3/issues/5516

SMT-LIB Language

$$\varphi : x > 0 \wedge x < 1$$

SMT-LIB Language

$$\varphi : x > 0 \wedge x < 1 \longrightarrow$$

```
(declare-fun x () Real)
(assert (and (> x 0)(< x 1)))
(check-sat)
```

SMT-LIB Language

$$\varphi : x > 0 \wedge x < 1 \longrightarrow$$

```
(declare-fun x () Real)
(assert (and (> x 0)(< x 1)))
(check-sat)
```

SMT-LIB Language

$$\varphi : x > 0 \wedge x < 1 \longrightarrow$$

```
(declare-fun x () Real)
(assert (and (> x 0)(< x 1)))
(check-sat)
```

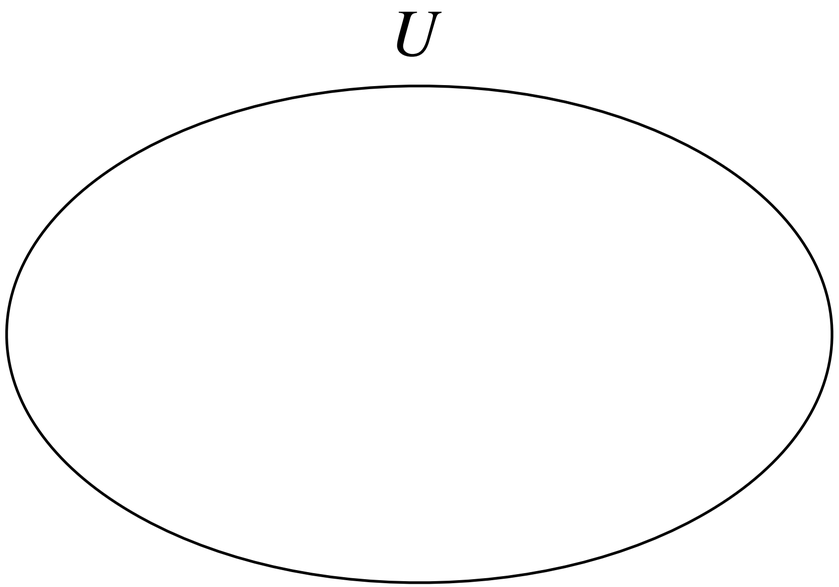
SMT-LIB Language

$$\varphi : x > 0 \wedge x < 1 \longrightarrow$$

```
(declare-fun x () Real)
(assert (and (> x 0)(< x 1)))
(check-sat)
```

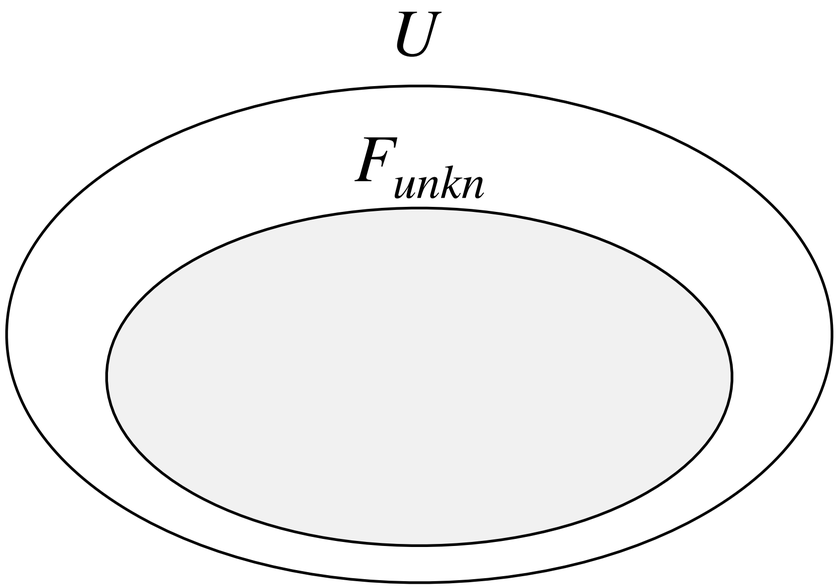
Problem Statement

Problem Statement



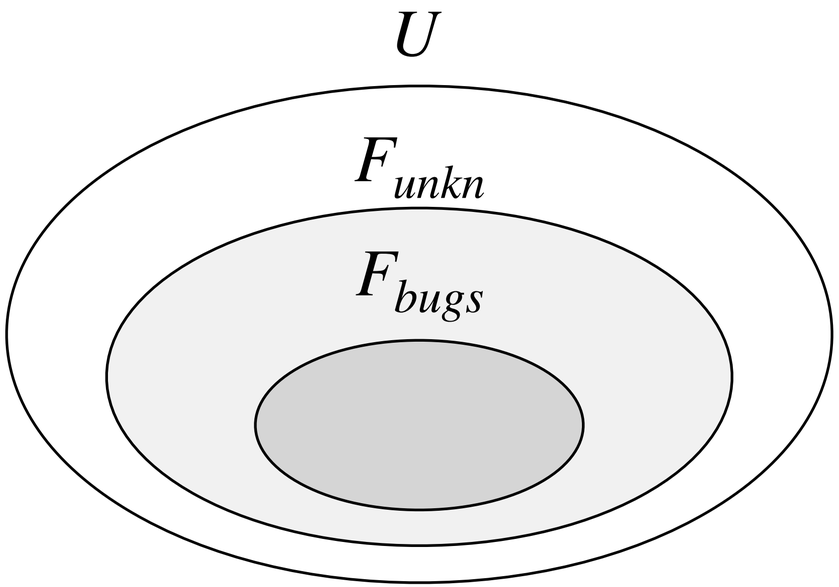
U: Universe of SMT formulas

Problem Statement



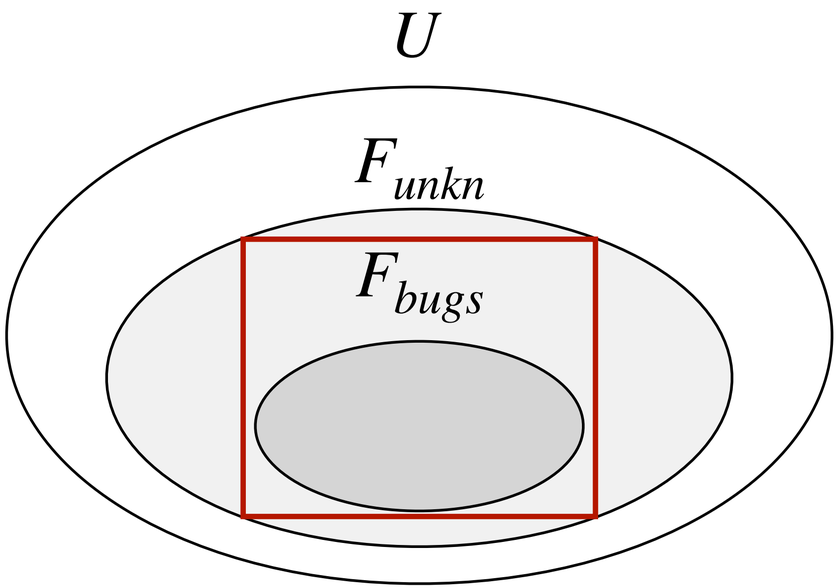
F_{unkn}: All unknown formulas

Problem Statement



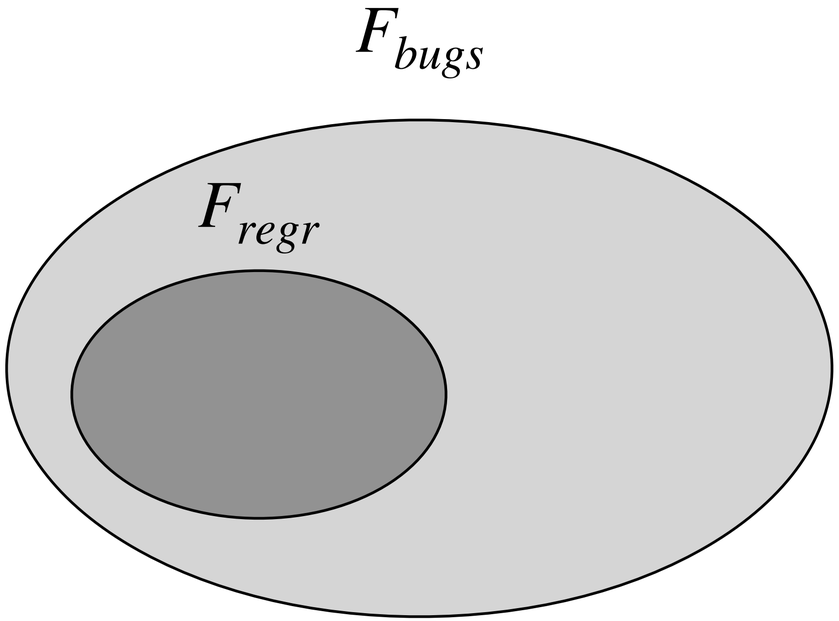
F_{bugs}: Incompleteness bugs

Problem Statement



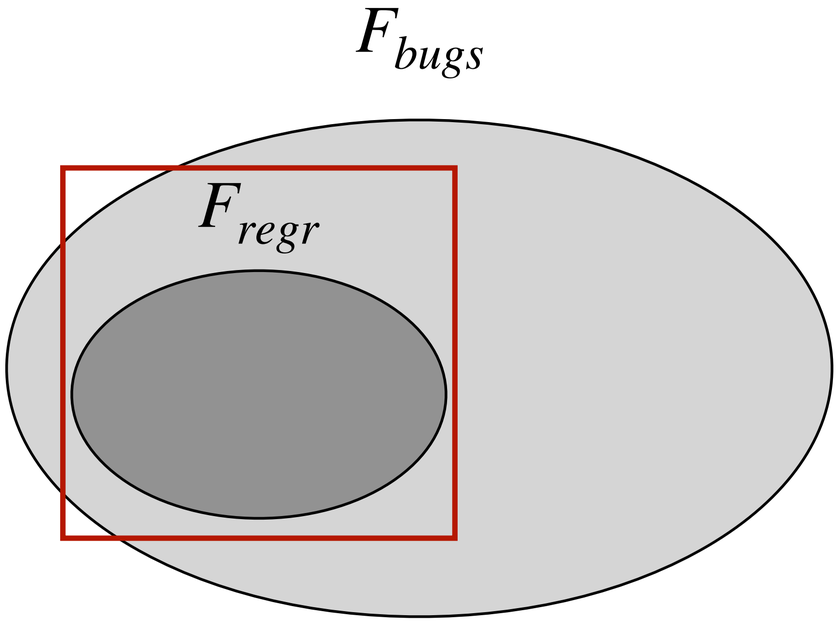
F_{bugs}: Incompleteness bugs

Problem Statement



F_{regr}: Regression incompleteness bugs

Problem Statement



F_{regr}: Regression incompleteness bugs

Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)
```

Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)
```

“All integers are equal to 0”

Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)

$ z3-4.8.10 formula.smt2
unsat
```

“All integers are equal to 0”

Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)

$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

“All integers are equal to 0”

Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)

$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

“All integers are equal to 0”

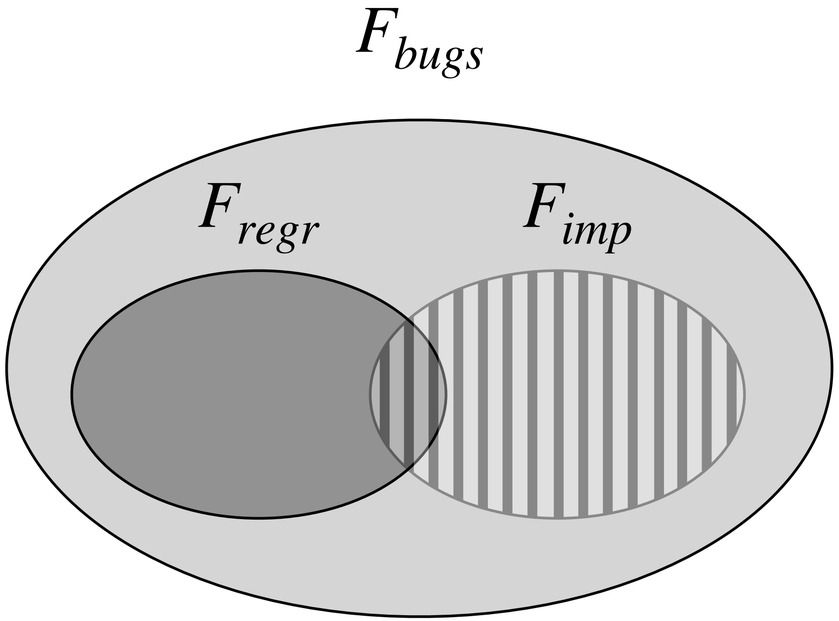
Regression Incompleteness

```
$ cat formula.smt2
(assert (forall ((v Int)) (= 0 v)))
(assert (= 0 (mod 0 0)))
(check-sat)

$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

“All integers are equal to 0”

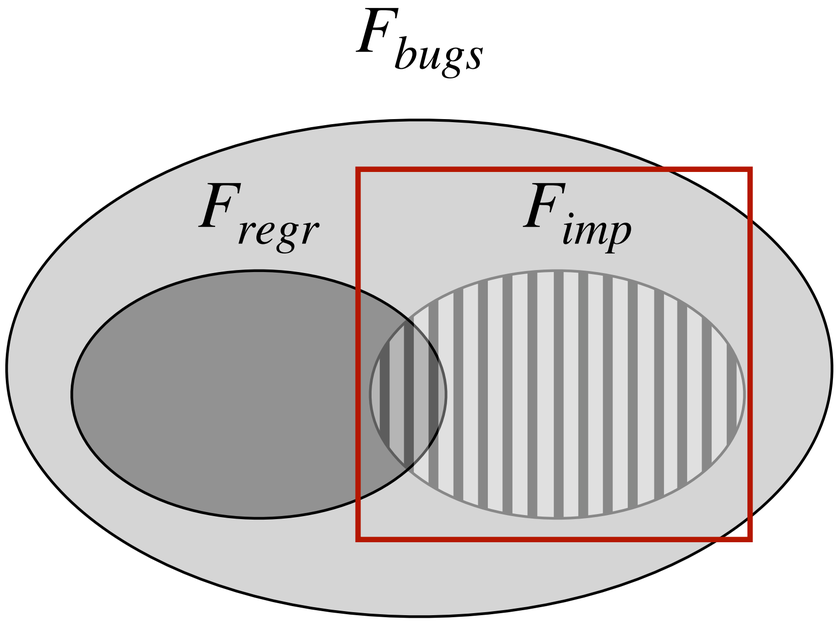
https://github.com/Z3Prover/z3/issues/5338

Problem Statement



F_{imp}: Implication incompleteness bugs

Problem Statement



F_{imp}: Implication incompleteness bugs

Implication Incompleteness

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)
```

Implication Incompleteness

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)
```

“Are there two reals multiplying to 1”

Implication Incompleteness

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

Implication Incompleteness

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

```
$ cat formula-new.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (>= (* s k) 1))
(check-sat)
```

Implication Incompleteness

“Are there two reals multiplying to greater/eq. 1”

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

```
$ cat formula-new.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (>= (* s k) 1))
(check-sat)
```

Implication Incompleteness

“Are there two reals multiplying to greater/eq. 1”

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

```
$ cat formula-new.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (>= (* s k) 1))
(check-sat)

$ cvc5 -q formula-new.smt2
unknown
```

Implication Incompleteness

“Are there two reals multiplying to greater/eq. 1”

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

```
$ cat formula-new.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (>= (* s k) 1))
(check-sat)

$ cvc5 -q formula-new.smt2
unknown
```

Implication Incompleteness

“Are there two reals multiplying to greater/eq. 1”

```
$ cat formula.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (= (* s k) 1))
(check-sat)

$ cvc5 -q formula.smt2
sat
```

```
$ cat formula-new.smt2
(declare-fun s () Real)
(declare-fun k () Real)
(assert (>= (* s k) 1))
(check-sat)

$ cvc5 -q formula-new.smt2
unknown
```

https://github.com/cvc5/cvc5/issues/7009

Approach

Janus

- Tool: Janus

Janus

- Tool: Janus
- Can both bug types: regression and implication

Janus

- Tool: Janus
- Can both bug types: regression and implication
- Technique: Weakening and strengthening

Janus

- Tool: Janus
- Can both bug types: regression and implication
- Technique: Weakening and strengthening
  - Idea: $\varphi_1 \implies \varphi_2 \cdots \implies \varphi_{n-1} \implies \varphi_n$

Janus

- Tool: Janus
- Can both bug types: regression and implication
- Technique: Weakening and strengthening
  - Idea: $\varphi_1 \implies \varphi_2 \cdots \implies \varphi_{n-1} \implies \varphi_n$
  - Label AST with polarities +,-

Janus

- Tool: Janus
- Can both bug types: regression and implication
- Technique: Weakening and strengthening
  - Idea: $\varphi_1 \implies \varphi_2 \cdots \implies \varphi_{n-1} \implies \varphi_n$
  - Label AST with polarities +,-
  - Randomly choose an expression

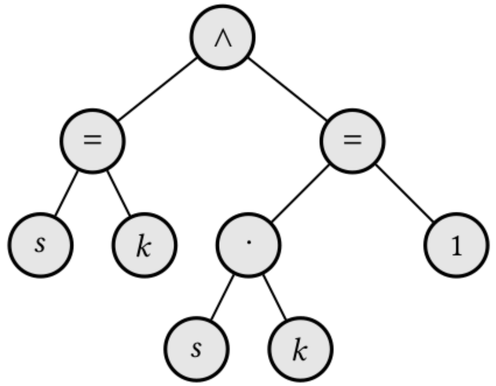
Janus

- Tool: Janus
- Can both bug types: regression and implication
- Technique: Weakening and strengthening
  - Idea: $\varphi_1 \implies \varphi_2 \cdots \implies \varphi_{n-1} \implies \varphi_n$
  - Label AST with polarities +,-
  - Randomly choose an expression
  - Apply weakening/strengthening rule to transform expression

Weakening & Strengthening

1

```
(declare-const s Real)
(declare-const k Real)
(assert (and (= s k)
             (= (* s k) 1)))
(check-sat)
```



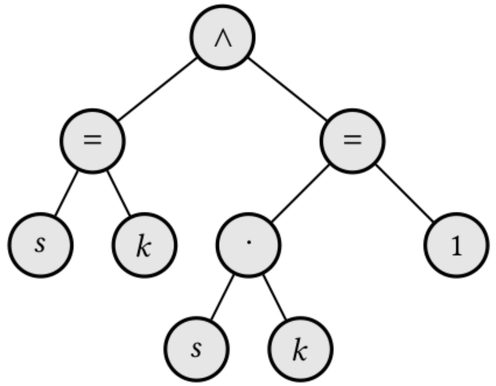
CVC5 sat

✓

Weakening & Strengthening

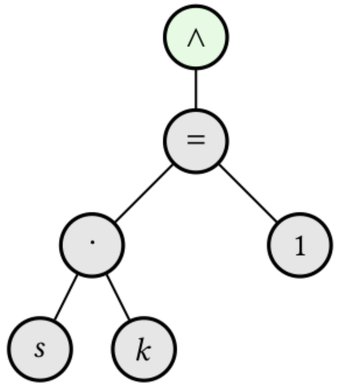
1

```
(declare-const s Real)
(declare-const k Real)
(assert (and (= s k)
             (= (* s k) 1)))
(check-sat)
```



2

```
(declare-const s Real)
(declare-const k Real)
(assert (and (= (* s k) 1)))
(check-sat)
```



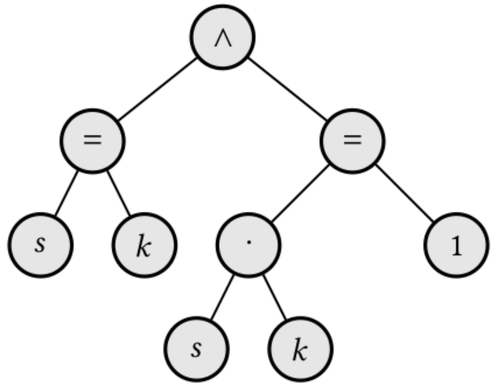
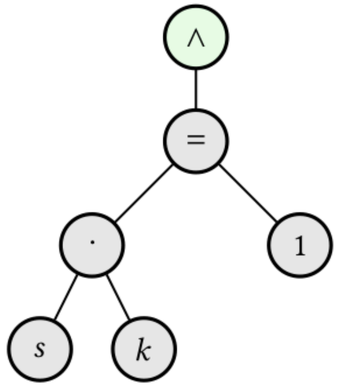
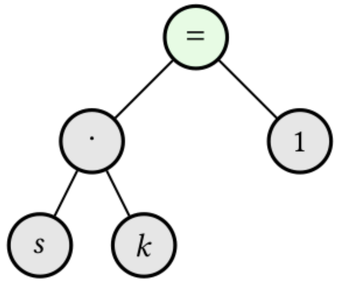
CVC5	sat	sat
	✓	✓

Weakening rule: Drop Conjunct

Weakening & Strengthening

	1	**2**	...	**n − 1**

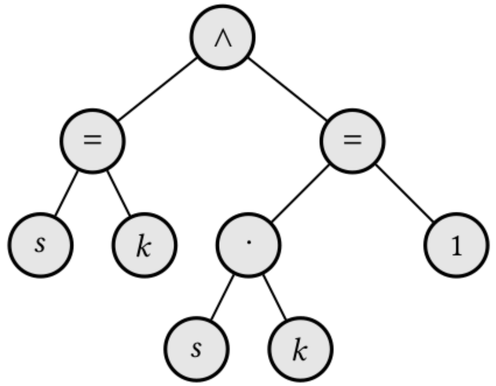
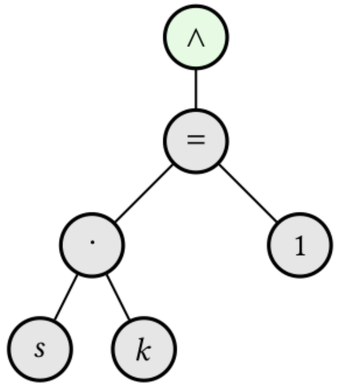
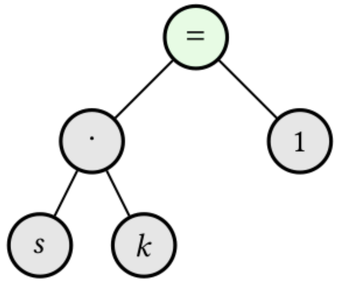
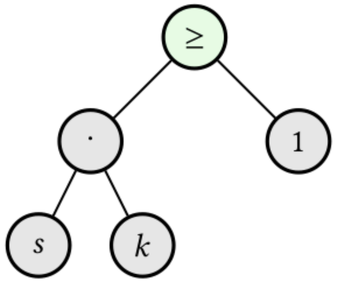
	`(declare-const s Real)`
	`(declare-const k Real)`
	`(assert (and (= s k)`
	`(= (* s k) 1)))`
	`(check-sat)`
		`(declare-const s Real)`
		`(declare-const k Real)`
		`(assert (and (= (* s k) 1)))`
		`(check-sat)`
			...	`(declare-const k Real)`
				`(declare-const s Real)`
				`(assert (= (* s k) 1))`
				`(check-sat)`

			...	
CVC5	sat	sat	...	sat
	✓	✓	...	✓

Weakening & Strengthening

	1	**2**	...	**n − 1**	**n**

	`(declare-const s Real)`
	`(declare-const k Real)`
	`(assert (and (= s k)`
	`(= (* s k) 1)))`
	`(check-sat)`
		`(declare-const s Real)`
		`(declare-const k Real)`
		`(assert (and (= (* s k) 1)))`
		`(check-sat)`
			...	`(declare-const k Real)`
				`(declare-const s Real)`
				`(assert (= (* s k) 1))`
				`(check-sat)`
					`(declare-const k Real)`
					`(declare-const s Real)`
					`(assert (>= (* s k) 1))`
					`(check-sat)`

			...		
CVC5	sat	sat	...	sat	unknown
	✓	✓	...	✓	✗

Weakening rule: = to >=

Rules and Proofs

Finding and Understanding Incompleteness Bugs in SMT Solvers

Mauro Bringolf
Department of Computer Science
ETH Zurich, Switzerland
mauro@bringolf.com

Dominik Winterer
Department of Computer Science
ETH Zurich, Switzerland
dominik.winterer@inf.ethz.ch

Zhendong Su
Department of Computer Science
ETH Zurich, Switzerland
zhendong.su@inf.ethz.ch

ABSTRACT

We propose *Janus*, an approach for finding incompleteness bugs in SMT solvers. The key insight is to mutate SMT formulas with local weakening and strengthening rules that preserve the satisfiability of the seed formula. The generated mutants are used to test SMT solvers for incompleteness bugs, *i.e.*, inputs on which SMT solvers unexpectedly return unknown. We realized *Janus* on top of the SMT solver fuzzing framework YinYang. From June to August 2021, we stress-tested the two state-of-the-art SMT solvers Z3 and CVC5 with Janus and totally reported 31 incompleteness bugs. Out of these, 26 have been confirmed as unique bugs and 19 are already fixed by the developers. Our diverse bug findings uncovered functional, regression, and performance bugs—several triggered discussions among the developers sharing their in-depth analysis.

ACM Reference Format:
Mauro Bringolf, Dominik Winterer, and Zhendong Su. 2022. Finding and Understanding Incompleteness Bugs in SMT Solvers. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3551349.3560435

1 INTRODUCTION

Satisfiability Modulo Theories (SMT) solvers are fundamental tools for software engineering and programming language advances *e.g.*, symbolic execution [7, 12], program synthesis [23], solver-aided programming [24], and program verification [9, 10]. An SMT solver returns sat on an input formula φ if there is an assignment to φ's variables that evaluates the formula to true, unsat if there is no such assignment and unknown if the SMT solver cannot decide the formula. Incompleteness bugs, *i.e.* unexpected unknown-results, impact the performance of SMT solvers' client applications frustrating their developers—especially since SMT solvers are usually at the very core of their client software solving NP-hard problems. Formula φ may realize a path constraint in a symbolic execution engine (*e.g.* KLEE [7], Microsoft's SAGE [13]), an access policy of a web service (*e.g.* AWS's Zelkova [2]), or a model of a safety-critical system (*e.g.* AdaCores's Spark [1]). Potential consequences of incompleteness bugs include missed bugs in the software under test, slow (or even non-terminating) verification of safety-critical or security-critical properties and other undesirable effects. Fig. 1 shows a incompleteness bug in Z3 [8] which returns unknown on a simple SMT formula. The first statement of the script declares an integer variable, the second specifies a constraint, and the third queries the SMT solver. Since we cannot choose an admissible value for x to satisfy the constraint (as there is no square number equal to all integers), the formula is unsatisfiable, and Z3 should return unsat. Why does Z3 fail at solving such a simple formula? As it turns out, it is caused by a bug in the implementation of model-based quantifier instantiation (MBQI). MBQI guesses values for the universal quantifiers (v in our case) to check whether the formula is unsatisfiable. On inspecting the issue deeper, we noticed that even if we run MBQI a million iterations, Z3 could not decide the formula. [1] However, fixing a random integer v which is not a square number would have been enough to determine unsatisfiability. We reported this bug to the issue tracker of Z3. Z3's main developer promptly fixed it.

```
$ z3 bug.smt2
unknown

$ cat bug.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
```

Fig. 1: Command-line trace showing an incompleteness bug in SMT solver Z3 on a simple SMT formula.
https://github.com/Z3Prover/z3/issues/5376

Incompletenesses in SMT solvers. Not every unknown-result indicates a bug. As SMT solvers support undecidable logics, they are necessarily incomplete. An SMT solver returns unknown on a formula if it has no decision procedure to solve the formula or to avoid a timeout. In practice, SMT solver can solve most problem instances from undecidable logics relevant to users. Similar to decidable logics, SMT solver developers enhance their solvers by rewriter rules, pre-processors *etc.* However, distinguishing expected from unexpected incompletenesses is difficult and confuses users:

> *"I'm seeing a regression [...], where a lot of simple formulas that used to be unsat now give unknown."*
> https://github.com/Z3Prover/z3/issues/5516

> *"The following code will produce unsat in z3 version 4.8.10.0 but is unknown in later versions."*
> https://github.com/Z3Prover/z3/issues/5438

[1] z3 smt.mbqi.max_iterations=1000000 bug.smt2

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3560435

Table 1: Weakening and strengthening rules for core logic, reals and integers, strings, and regexes. Symmetric cases are omitted for brevity. The legend column describes newly introduced symbols per group.

Type	Strong	Weak	Legend
Real/Int			$n_1, n_2 \in \mathbb{R}$ or $n_1, n_2 \in \mathbb{N}$
	$n_1 = n_2$	$n_1 \geq n_2 \mid n_1 \leq n_2$	
	$n_1 > n_2$	$n_1 \geq n_2 \mid n_1 \neq n_2$	
	$n_1 < n_2$	$n_1 \leq n_2 \mid n_1 \neq n_2$	
	$n_1 \odot n_2$	$(n_1 + c) \odot (n_2 + c) \mid n_1 \odot (n_2 + c) \mid n_1 \odot (n_2 + c)$	$c \in \mathbb{N}$ or $c \in \mathbb{R}$ $\odot \in \{=, >, <, \leq, \geq\}$
Bool			$\varphi, \varphi_1, \varphi_2$ are boolean formulas
	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2 \mid \varphi_1$	
	$\varphi_1 \oplus \varphi_2$	$\varphi_1 \vee \varphi_2$	$\oplus$ is the logical xor
	$\forall x \colon \varphi \mid \varphi[x \mapsto B]$	$\exists x \colon \varphi$	B is an expression of same type as x
	$x_1 = \ldots = x_n$	$f(x_1) = \ldots = f(x_2)$	$x_1, \cdots, x_n$ are terms of arbitrary type f: SMT-LIB built-in function
	$\varphi_1 \vee \varphi_2$	$\exists b \colon ite(b, \varphi_1, \varphi_2)$	*ite* is the if-than-else operator b is a boolean variable
	$ite(B, \varphi_1, \varphi_2)$	$B \rightarrow \varphi_1 \mid \neg B \rightarrow \varphi_2$	
	$\varphi_1 \rightarrow \varphi_2$	$ite(\varphi_1, \varphi_2, \top) \mid ite(\neg\varphi_1, \top, \varphi_2) \mid \forall b \colon (\varphi_1 \wedge b \rightarrow \varphi_2 \wedge b)$	
	$\varphi_1 \vee \varphi_2$	$\neg\varphi_1 \rightarrow \varphi_2$	
	$\forall x.\varphi$	$\varphi[x \mapsto B]$	
	φ_1	$\varphi_1 \vee \varphi_2$	
String			s_1, s_2, s_3 are strings
	$s_1 = s_2$	$s_1 \neq (s_1 \mathbin{++} s_3) \mid s_1 \leq_s s_2 \mid prefixof(s_1, s_2) \wedge suffixof(s_1, s_2) \mid prefixof(s_1, s_2) \wedge prefixof(s_2, s_1) \mid contains(s_1, s_2) \mid suffixof(s_1, s_2) \wedge suffixof(s_2, s_1) \mid prefixof(s_1, s_2) \mid suffixof(s_1, s_2)$	$\mathbin{++}$: string concatenation
	$s_1 <_s s_2$	$s_1 \neq s_2 \mid s_1 \leq_s s_2$	$\leq_s$: lexicographical ordering
	$s_1 \leq_s s_2$	$substr(s_1, 0, ite(0 \leq i \leq len(s_1) - 1, i, len(s_1))) \leq_s s_2 \mid s_1 \leq_s (s_2 \mathbin{++} s_3)$	
	$contains(s_1, s_2)$	$len(s_1) \geq len(s_2)$	
Regex			$r, r_1, \cdots, r_n$ are regexes, $n \in \mathbb{N}$
	r	r^+	+: Kleene plus
	r	$loop(1, n, r)$	$loop(i, n, r) = L(r)^i \cup \cdots \cup L(r)^n$
	r	$opt(r)$	$opt(r) := union(r, (\text{str.to\_re } ""))$
	$r_1 +\!\!+_r \ldots +\!\!+_r r_n$	$union(r_1, \ldots, r_n)^n$	$+\!\!+_r$: regex concat
	r	$\forall x \colon union(r, s)$	for an arbitrary string s
	r^+	r^*	$*$: Kleene star
	$range(s_1, s_2)$	$range(s_3, s_4)$	for fixed strings s_1, s_2 choose strings s_3, s_4 s.t. $range(s_1, s_2) range(s_3, s_4)$

Evaluation

Empirical Evaluation

- Tool: Janus 

Empirical Evaluation

- Tool: Janus 
- Bug hunting: Jun 2021 - Aug 2021

Empirical Evaluation

- Tool: Janus 
- Bug hunting: Jun 2021 - Aug 2021
- Testing targets: Z3 CVC5

Empirical Evaluation

- Tool: Janus 
- Bug hunting: Jun 2021 - Aug 2021
- Testing targets: Z3 CVC5
- Seeds: preprocessed SMT-LIB files (sat/unsat)

Bug Findings

Status	Z3	CVC5	Total
Reported	13	18	31
Confirmed	8	18	26
Fixed	8	11	19
Rejected	5	0	5

Bug Findings

Status	Z3	CVC5	Total
Reported	13	18	31
Confirmed	8	18	26
Fixed	8	11	19
Rejected	5	0	5

Bug Findings

Status	Z3	CVC5	Total
Reported	13	18	31
Confirmed	8	18	26
Fixed	8	11	19
Rejected	5	0	5

Bug Findings

Status	Z3	CVC5	Total
Reported	13	18	31
Confirmed	8	18	26
Fixed	8	11	19
Rejected	5	0	5

Bug Findings

Status	Z3	CVC5	Total
Regression	7	12	19
Implication	1	6	7

Bug Findings

Status	Z3	CVC5	Total
Regression	7	12	19
Implication	1	6	7

Bug Findings

Status	Z3	CVC5	Total
Regression	7	12	19
Implication	1	6	7

Bug Samples

Z3 #5376

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
```

Z3 #5376

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
```

“Every integer is a square number”

Z3 #5376

“Every integer is a square number”

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
$ z3-4.8.10 formula.smt2
unsat
```

Z3 #5376

“Every integer is a square number”

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

Z3 #5376

“Every integer is a square number”

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

Root Cause: Functional bug in MBQI hidden by “unknown"

https://github.com/Z3Prover/z3/issues/5376

CVC5 #6717

```
$ cat formula.smt2
(declare-const v String)
(assert true)
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

CVC5 #6717

```
$ cat formula.smt2
(declare-const v String)
(assert true)
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

```
$ cvc4-1.8 formula.smt2
sat
```

CVC5 #6717

```
$ cat formula.smt2
(declare-const v String)
(assert true)
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

```
$ cvc4-1.8 formula.smt2
sat
```

```
$ cat formula.smt2
(declare-const v String)
(assert (ite T T true))
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

CVC5 #6717

```
$ cat formula.smt2
(declare-const v String)
(assert true)
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

```
$ cvc4-1.8 formula.smt2
sat
```

```
$ cat formula.smt2
(declare-const v String)
(assert (ite T T true))
(assert (or T (and (str.prefixof v ")
(exists ((x Int)) (= "t"
(str.substr v 0 x))))))
(check-sat)
```

```
$ cvc5 formula.smt2
unknown
```

CVC5 #6717

"Commit 11c1fba added new rewrites for ITE. Due to the new rewrites taking precedence over existing rewrites, it could happen that some of the previous rewrites did not apply anymore even though they would have further simplified the ITE"

CVC5 developer

Janus 0.1.0 Release

github.com/testsmt/janus

SCAN ME

Key Takeaways

Key Takeaways

- Incompleteness bugs frustrate tool developers & users

Key Takeaways

- Incompleteness bugs frustrate tool developers & users
- Janus: pioneering approach to detect them

Key Takeaways

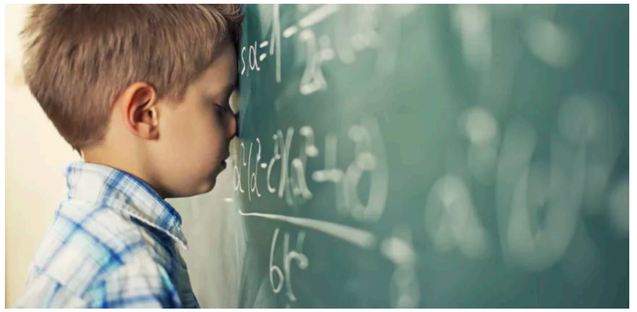
- Incompleteness bugs frustrate tool developers & users
- Janus: pioneering approach to detect them
- SMT solvers

Key Takeaways

- Incompleteness bugs frustrate tool developers & users
- Janus: pioneering approach to detect them
- SMT solvers
  - Can solve very hard problems

Key Takeaways

- Incompleteness bugs frustrate tool developers & users
- Janus: pioneering approach to detect them
- SMT solvers
  - Can solve very hard problems
  - But: can (sometimes) not solve simple classroom problems



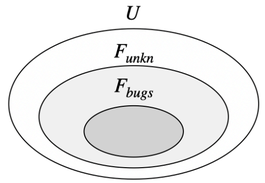
SMT Solver

$\varphi : x > 0 \wedge x < 0 \longrightarrow$ SMT Solver $\longrightarrow$ **UNKNOWN**

Incompleteness Bug

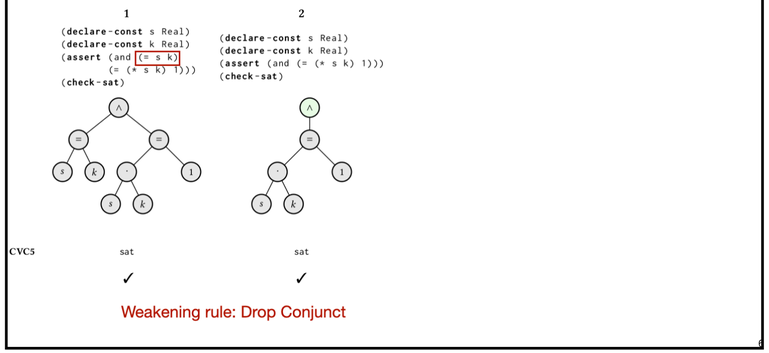
$\varphi : x > 0 \wedge x < 1 \longrightarrow$ SMT Solver v1.1 $\longrightarrow$ **UNKNOWN**

Problem Statement



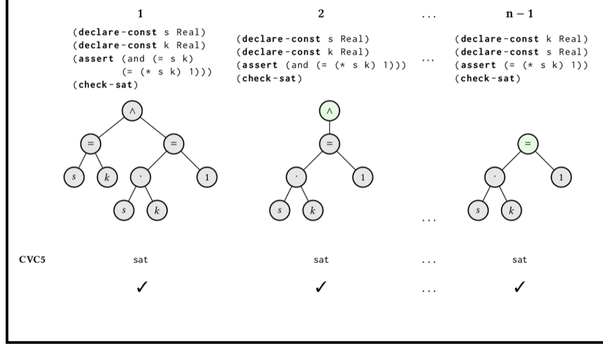
F_{bugs}: Incompleteness bugs

Weakening & Strengthening

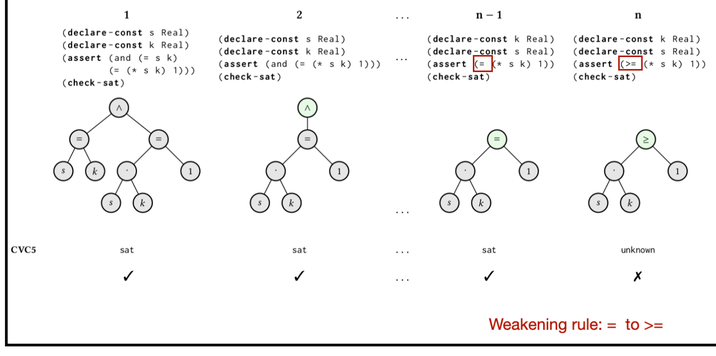


Weakening rule: Drop Conjunct

Weakening & Strengthening



Weakening & Strengthening



Weakening rule: = to >=

Bug Findings

Status	Z3	CVC5	Total
Reported	13	18	31
Confirmed	8	18	26
Fixed	8	11	19
Rejected	5	0	5

Z3 #5376

"Every integer is a square number"

```
$ cat formula.smt2
(declare-const x Int)
(assert (forall ((v Int)) (= v (* x x))))
(check-sat)
$ z3-4.8.10 formula.smt2
unsat

$z3-trunk formula.smt2
unknown
```

SCAN ME

97

Backup Slides

Bug Findings