# Making SMT-Based Verification Less Frustrating

DSLab
Systems and Formalisms Lab



Can Cebeci


George Candea


Clément Pit-Claudel

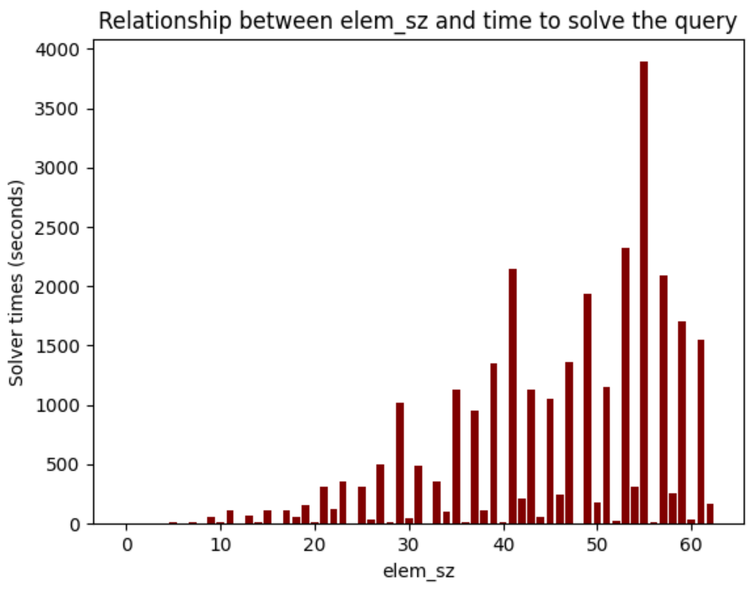

**Problem: Solver instability limits the design space and usability of automated verifiers**

- Program verifiers rely on SMT solvers for automation

Z3 CVC5

- **Instability:** unpredictable behavior (e.g. changes in performance or outcome due to small mutations on input)

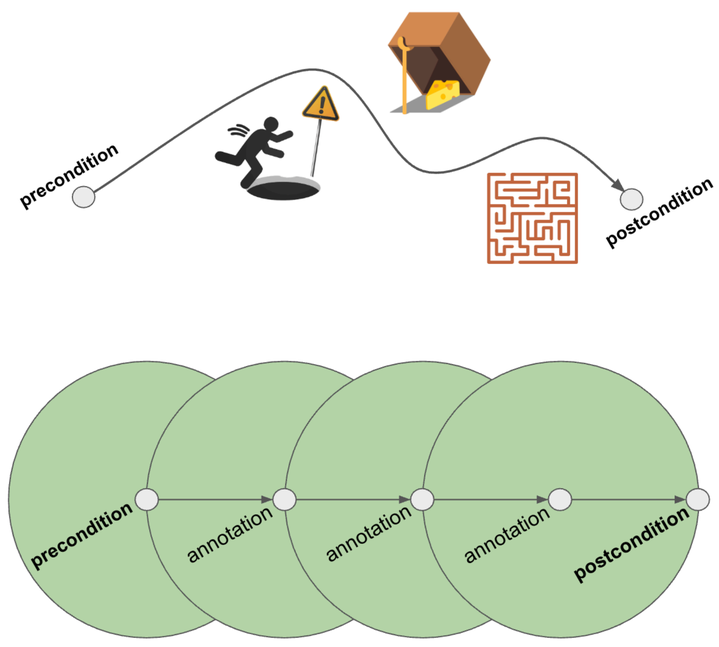
```
; Can a+100 point to array element b given that
; a has size 200 and is allocated below b?
(assume b = arr + <elem_sz> * i)
(assume a + 200 <= b)
(prove a + 100 < b)
```




SOTA: ”Instability is a fundamental theoretical limitation. Design verifiers that avoid it.”

Approaches:

- Modularize verification
- Expose the SMT encoding
- Export hard lemmas for manual proofs




**Observation:** Verifiers encounter instability mostly due to fixable <u>engineering problems</u>

```
(declare-fun H (Int) Bool)
(declare-fun C (Int) Bool)
(assert (forall ((x Int)) (!
 (and (not (H x)) (not (C x)))
  :pattern ((H x))
)))
(assert (or (H 0) (C 0)))
(check-sat)
```

UNCLEAR INTERFACE

SMT-LIB

```
lemma lemma_2toX()
 ensures power2(64) == 18446744073709551616;
 ensures power2(60) == 1152921504606846976;
 ensures power2(32) == 4294967296;
 ensures power2(24) == 16777216;
 ensures power2(19) == 524288;
 ensures power2(16) == 65536;
 ensures power2(8) ==  256;
{
   reveal_power2();
}
```

MISCONFIGURATION

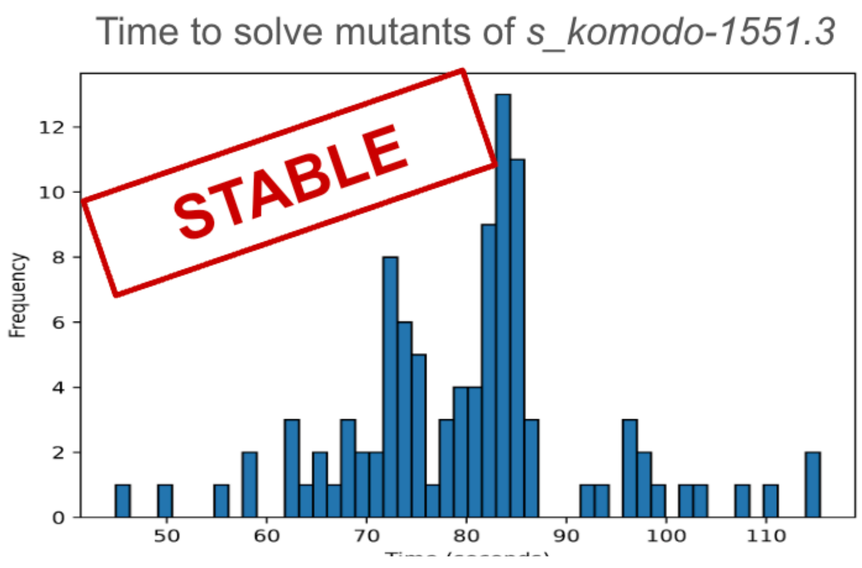
Dafny

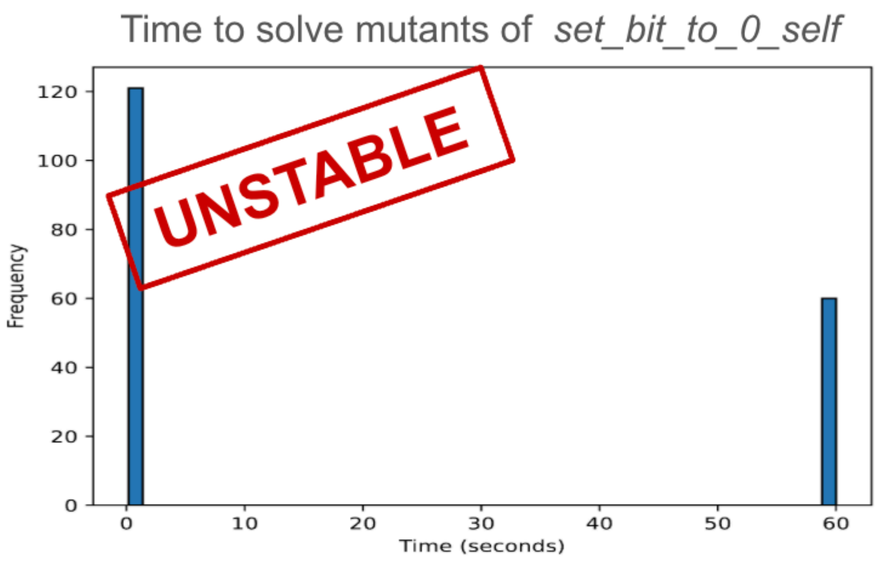
```
char arr[];
int f(char);
...
assume(mdata.p == &arr[0]);
assume(f(arr[2]) < f(arr[3]));
assume((uintptr_t)&mdata % 4096 == 0);
metadata *mdata_aligned =
   ((uintptr_t)&mdata/4096)*4096;
*(mdata_aligned->p) = 0;
assert(f(arr[2]) < f(arr[3]));
```

SOLVER BUG

**For better automation, enable diagnosing and fixing solver instability rather than expecting verifiers to be designed around it!**

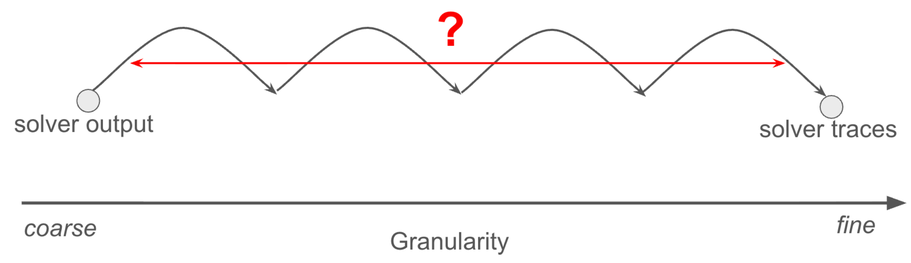
## Make instability easier to detect (better metrics, explicit randomness)




## Provide abstractions and for debugging instability (proof-sketch debugging, branch-local reasoning)

How do describe and observe solver behavior for debugging purposes?




### Unstable query

```
declare-fun loc (...) Int
declare-fun addr_mdata () ...
declare-fun addr_arr () ...
declare-fun mdata () (Array ...)
declare-fun arr () (Array ...)
declare-fun f (...) Int)
; mdata is page aligned
assert loc(addr_mdata) % 4096 == 0
; mdata.p points to the array's base
assert mdata[0+:8] ==
addr_arr
; the assumption
assert f(arr[2]) < f(arr[3])
; negation of the assertion
let loc_mdata_aligned :=
      (loc(addr_mdata)/4096)*4096
let p_offset := loc(addr_mdata) -
                    loc_mdata_aligned
let p := mdata[p_offset+:8];
let arr' := store(arr,
  loc(p) - loc(addr_arr), #x00);
assert not f(arr'[2]) < f(arr'[3])
```

### Proof sketch

```
(1): f(arr[2]) < f(arr[3])
(2): f(arr'[2]) >= f(arr'[3])
; by 1, 2, array axioms
(3): arr'[2] != arr[2] ∨ arr'[3] != arr[3]
; by 3, array axioms
(4): loc(p)-loc(addr_arr)==2 ∨
     loc(p)-loc(addr_arr)==3
; by Ackermann axiom
(5): p == addr_arr ==>
     loc(p) == loc(addr_arr)
; by 4,5
(6): p != addr_arr
; by 6
(7): mdata[p_offset+:8] != mdata[0+:8]
; by 7, array axioms
(8): p_offset != 0
; by modular artihmetic
(9): loc_mdata_aligned == loc(addr_mdata)
; by 9
(10): p_offset == 0
; contradiction between 8 and 10
```