# Towards Performance Interfaces for SMT Solvers



Can Cebeci


George Candea


Clément Pit-Claudel

EPFL DSLab Systems and Formalisms Lab


In order to combat solver explosion, we need abstractions that describe solver performance.

SMT solvers are black boxes with unstable performance

Performance interface=ideal white box

- Program verifiers rely on SMT solvers for automation

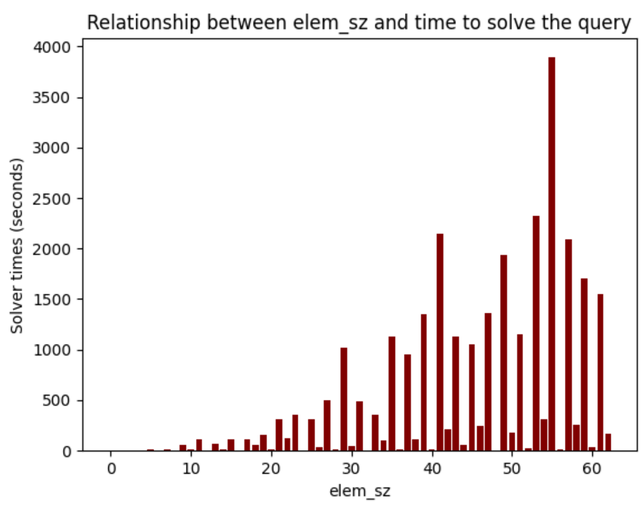
Z3 CVC5

- Solver explosion = timeouts on seemingly easy queries depending on user-transparent factors (e.g., reordering assertions, renaming variables, arithmetic rewrites)
- Outcome: more annotations, less automation

```
struct hyp_page *node_to_page(struct list_head *node)
/*@ accesses __hyp_vmemmap; hyp_physvirt_offset @*/
/*@ requires let phys=((integer)node)+hyp_physvirt_offset@*/
/*@ requires phys < power(2, 64) @*/
   . . .
/*@ ensures return == page @*/
/*@ ensures {__hyp_vmemmap} unchanged; {hyp_physvirt_offset}
unchanged @*/
{ return hyp_virt_to_page(node); }
```

*Pulte et al., POPL '23*

```
; 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)
```




```
def z3_unstable(query, config):
"""
# Input: an SMT query, a solver configuration
# Return true if the query would explode, false otherwise.
"""

    if query.logic == QF_LIA:
        # Linear integer arithmetic is stable.
        return False
    # . . .
    if query.logic == QF_AUFBV:
        num_bits_compared = 0
        for a in query.assertions:
            if a.is_bitvec_comparison():
                num_bits_compared += a.rhs.len() + a.lhs.len()
                if num_bits_compared > 120 and \
                    not config.incremental_solving:
                    # Too many bitvector comparisons
                    # leads to explosion in some configs.
                    return True
            if a.is_select() and a.child(0).is_store():
                # If the solution depends on a RAW simplification
                # and relevancy is enabled, Z3 will explode.
                if a.simplify() in solution(query):
                    if config.relevancy > 0:
                        return True
            # . . .
```

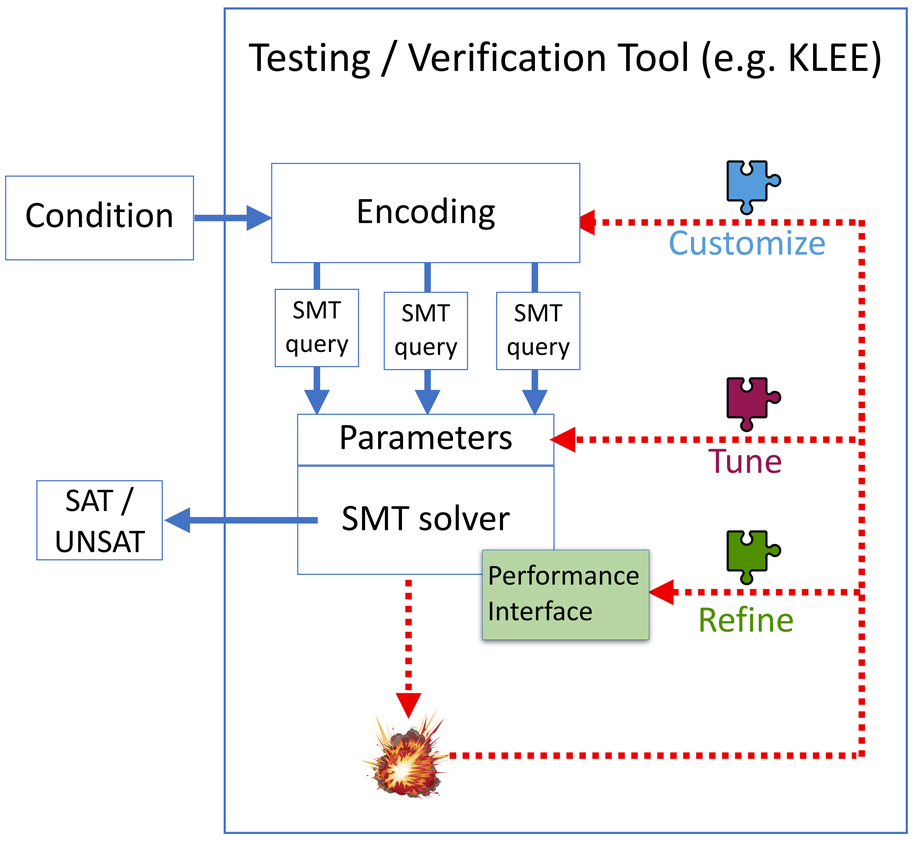
**Verifier developers need to stabilize SMT performance over the set of queries generated by their tool, but they lack the tools to do so.**

Open questions: succinctness, readability, actionability

## We propose a practical gray box to clarify **relevant** performance characteristics iteratively

Key Insights:

- SMT solvers are generic but unstable; domain-specific solvers are stable but hard to build (e.g. integer programming)
- Verifiers use SMT solvers in domain-specific ways
- Most of the time, SMT solvers do what is expected
- When they don't, it is hard to pinpoint why and take action
- Often, explosions can be fixed by changing solver parameters, or by slightly changing how SMT queries are constructed




### Insights about solver instability should be made explicit in an interface

```
def z3_unstable(query, config):
    return False
```

Pinpoint root cause, refine interface

```
def z3_unstable(query, config):
    for a in query.assertions:
        if a.is_select() and a.child(0).is_store():
            # If the solution depends on a RAW simplification
            # and relevancy is enabled, Z3 will explode.
            if a.simplify() in solution(query):
                if config.relevancy > 0:
                    return True
    return False
```

- The interface guides encoding choices (e.g., explicitly perform RAW simplifications)
- The interface informs parameter tuning in response to future explosions
- Automated tools can use the interface to identify subtle issues with the encoding

### Parameter fuzzing can *automatically* find *concise* configurations that fix solver explosion and help pinpoint the root cause

- 5 queries generated by Dafny (Mariposa benchmark) solved in less than a seconds with 1-3 parameters. Default solver configuration times out after 100 seconds.
- 5 queries generated by a KLEE-based tool fixed with a single parameter
- Parameters that avoid explosion also speed up non-exploding queries

| Mariposa benchmark set | # Queries | # Timeouts (at 20 seconds) | | Avg. time (seconds) | |
|---|---|---|---|---|---|
| | | Default | Fuzzed | Default | Fuzzed |
| stable-ext | 280 | 59 | 39 | 4.91 | 4.42 |
| unstable-ext | 378 | 179 | 107 | 4.44 | 4.45 |

Impact of using `rewriter.sort_disjunctions=false` with Dafny queries

### Instrumentation can help pinpoint reasoning steps that give the solver trouble

Insight: verifier developers often know whether an exploding query is satisfiable or unsatisfiable, and *why* it is so.

Idea: instrument the solver to bridge the gap between expected reasoning (deductive steps) and actual behavior.

```
a1: (assert array[i] == 0)
a2: (assert j < i || i < j)
a3: (assert (array[j:=v])[i] != 0)
...
```

SMT solver

```
d1: (deduce i != j from a2)
d2: (deduce array[j=v][i] == array[i]
      from d1)
d3: (deduce array[j=v][i] == 0
      from a1, d2)
d4: (conflict d3, a3)
```

```
# no case splits
d1: (deduce i != j from a2)
# 10000 splits, at depth 100
d2: (deduce array[j=v][i] == array[i]
      from d1)
# timeout
d3: (deduce array[j=v][i] == 0
      from a1, d2)
d4: (conflict d3, a3)
```