SMT Solvers in Application to Static and Dynamic Symbolic Execution: a Case Study

Nikita Malyshev, Irina Dudina, Daniil Kutz, Alexander Novikov, Sergey Vartanov

December 5, 2019

Ivannikov Institute for System Programming of the RAS

Lomonosov Moscow State University

Static and Dynamic Symbolic Execution

Svace

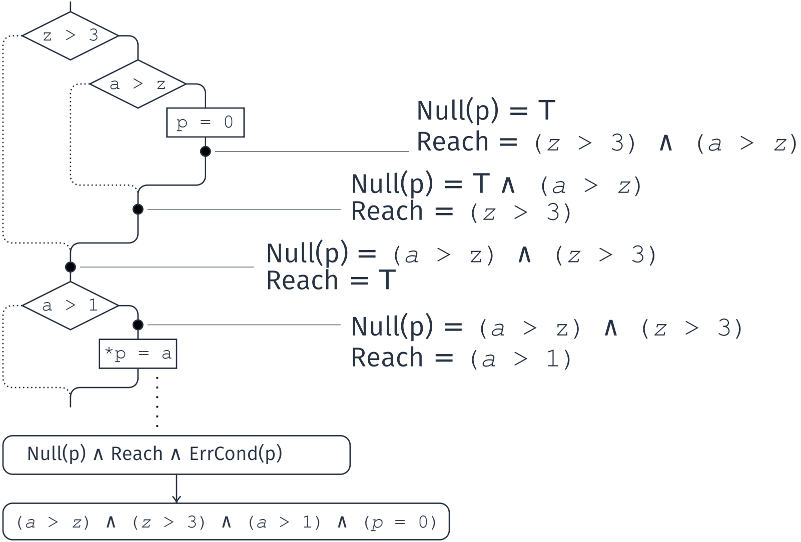
- Static analyzer for C, C++, Java code
- Performs many types of analyses including path-sensitive analysis
- Utilizes symbolic execution for solving path conditions

Anxiety

- Dynamic symbolic execution framework
- Finds new feasible execution paths given several concrete ones
- Generates input values for such paths to be reached

Solvers in Symbolic Execution

Path-Sensitive Static Analysis



Null(p) = T
Reach = $(z > 3) \land (a > z)$

Null(p) = T ∧ $(a > z)$
Reach = $(z > 3)$

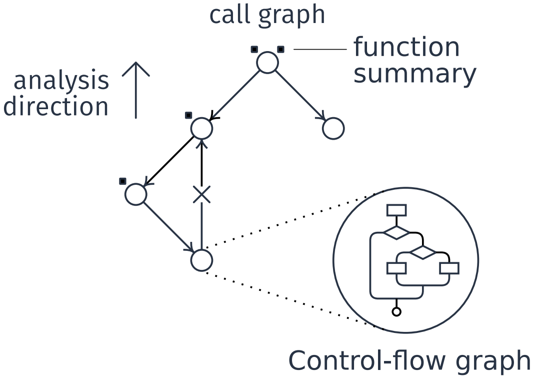
Null(p) = $(a > z) \land (z > 3)$
Reach = T

Null(p) = $(a > z) \land (z > 3)$
Reach = $(a > 1)$

Null(p) ∧ Reach ∧ ErrCond(p)

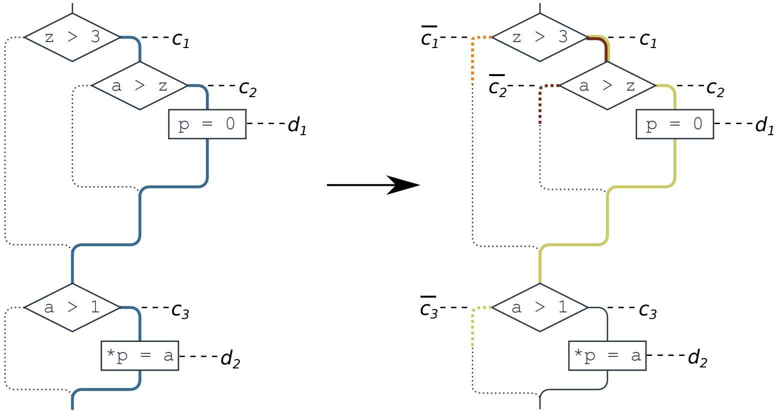
$(a > z) \land (z > 3) \land (a > 1) \land (p = 0)$

Path-Sensitive Static Analysis: Svace



1. Formulas become larger towards the end of analysis
2. Similar formulas are produced by checkers on similar code
3. Maximum summary size impacts precision of the analysis

Dynamic Symbolic Execution



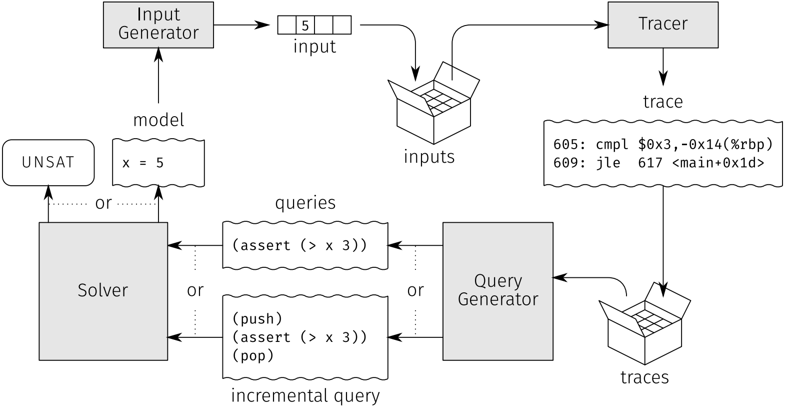
Path condition: $c_1 \wedge c_2 \wedge d_1 \wedge c_3 \wedge d_2$

New path condition: $\overline{c_1}$

New path condition: $c_1 \wedge \overline{c_2}$

New path condition: $c_1 \wedge c_2 \wedge d_1 \wedge \overline{c_3}$

Dynamic Symbolic Execution: Anxiety



```
605: cmpl $0x3,-0x14(%rbp)
609: jle  617 <main+0x1d>
```

```
x = 5
```

```
(assert (> x 3))
```

```
(push)
(assert (> x 3))
(pop)
```

Svace and Anxiety: Similarities and Differences

1. Logics
    - ”Core” for both — `QF_BV`
2. Models
    - Svace uses models to builds error trace
    - Anxiety generates new inputs from models
3. Formulas
    - Anxiety: single point in code — multiple paths through — `multiple` formulas for *every* path
    - Svace: single point in code — multiple paths through — `single` formula combining *all* paths

Technical Details and Pitfalls

1. Integrated vs. separated solvers.

Problem. If solver is integrated into analysis process by using API, its crashes can not be always handled properly.

Solution. Run solver as separate process instead.

Details. We discovered that solvers might crash on some inputs. While inter-process communication and conversion of requests to SMT-LIB 2 incur additional overhead, such design allows to handle crashes just by restarting the solver.

Technical Details and Pitfalls

2. Interactive solving.

Problem. Starting new process for every request costs pretty much.

Solution. Use `reset` command to clean solver's state after every request.

Details. Some solvers tend to crash when reset repeats many times. This way, it is necessary to keep track of the solver's state and restart it every so often. Also a few solvers may not support it (Boolector).

Technical Details and Pitfalls

3. Following standard.

Problem. Different solvers implement standard rather loosely.

Solution. Correct formulas by hand.

Details. For example, some do not parse *n*-ary `and`-s or `concat`-s while others do. This way, despite SMT-LIB 2 intended purpose, jumping from one solver to another still requires a fair effort to do.

Technical Details and Pitfalls: Svace

4. Parallel solving.

Problem. Most solvers do not support parallel solving.

Solution. We don’t need it anyway.

Details. Sometimes only a few functions are analyzed at a time, as everything else is higher on the call graph and requires their analysis result. Thus, parallelism in solver itself is not mandatory though it may be desired.

Technical Details and Pitfalls: Svace

5. Caching requests.

Problem. Many requests to solver are actually the same.

Solution. Cache requests.

Details. We have found two reasons: that different checkers verify the same code and that there are very similar parts of code. With caching we managed to save 22% of total time for SMT solving in Svace.

Technical Details and Pitfalls: Svace

6. Deterministic time limit.

Problem. For industrial purposes, analysis needs to be *deterministic* — the results must not differ for subsequent runs on the same code.

Solution. Use solver with the option to set some other limit than just CPU time.

Details. Not so many solvers support deterministic limits, and most of corresponding options are not very reliable. Currently, Z3 is maybe the only solver with good way to set such limit.

Technical Details and Pitfalls: Anxiety

7. Incremental solving.

Problem. Solving similar requests independently is a waste of time.

Solution. Use incremental solving with `push` and `pop` commands.

Details. Incremental solving may be used to invert a number of consecutive branch conditions of one path condition. This allows the solver to construct lemmas learned for the common parts just once.

Technical Details and Pitfalls: Anxiety

8. **Different models**.

Problem. If the formula is satisfiable, there can be more than one model.

Solution. Have a way to choose model generation algorithm.

Details. Execution for input data generated from different models may differ depending on unconstrained values. This may have a significant impact on the analysis process, number of paths executed and defects detected.

Solver Comparison

Choosing Solvers

Table 1: Solvers competed in SMTCOMP 2018

	Multi-platform	Free commercial use	Parallel solving	Deterministic limit	Incremental solving
MathSAT	✓				✓
Yices2	✓	✓			✓
Z3	✓	✓		✓	✓
Boolector		✓			✓
CVC4	✓	✓		✓	✓
Minkeyrink *			✓	✓	✓
STP *	✓	✓	✓	✓	✓

Experimental Setup

1. Projects to analyze: Android 5 and Tizen 5 (Svace); JasPer JPEG-2000, FAAD, Yodl and others (Anxiety).
2. Requests from analysis were captured and saved.
3. Simulate tools workflow: parallel instances, piping queries, interactivity (+ for Svace, - for Anxiety), operating time limit (90 seconds).
4. Crashes and TLE’s are UNKNOWNS.

Solving Statistics: Svace

Table 2: Statistics for Svace requests

	Z3-v4.4.1	Z3-v4.8.5	Yices2	STP	CVC4
SAT	98 758	98 758	98 758	98 758	98 735
UNSAT	63 130	63 130	63 130	63 130	63 130
UNKNOWN	8	8	8	8	31
Total	161 896	161 896	161 896	161 896	161 896

Solving Statistics: Anxiety

Table 3: Statistics for Anxiety requests

	Z3-v4.4.1	Z3-v4.8.5	Yices2	STP	CVC4	Boolector
SAT	14 072	14 072	14 072	14 072	14 072	14 072
UNSAT	31 280	31 280	31 280	31 280	31 280	31 280
UNKNOWN	0	0	0	0	0	0
Total	45 352	45 352	45 352	45 352	45 352	45 352

Solving Time: Svace

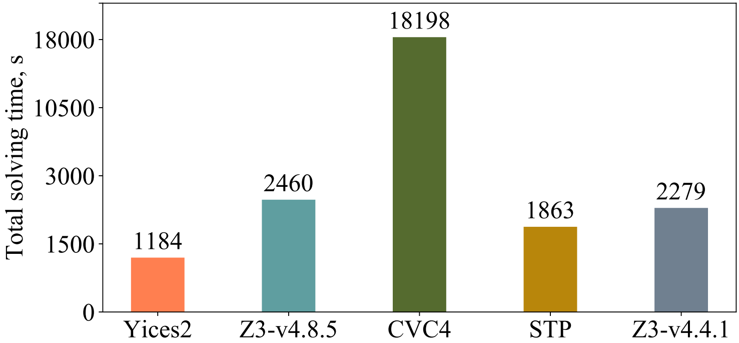


Figure 1: Total time needed to solve all requests

Cumulative Distribution: Svace

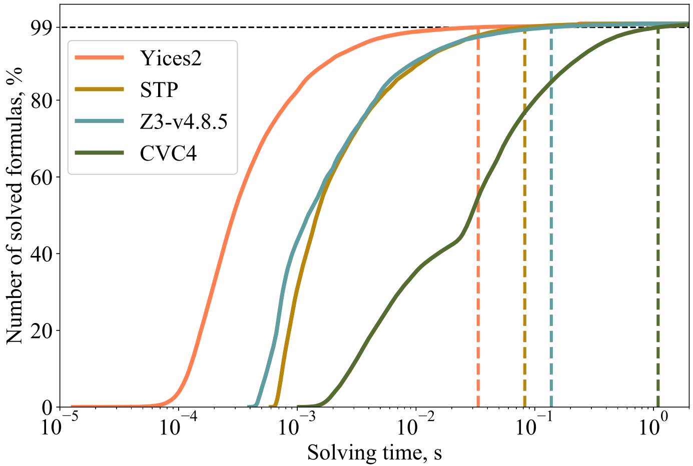


Figure 2: Percentage of requests taking to solve no more than given time

Solving Time: Anxiety

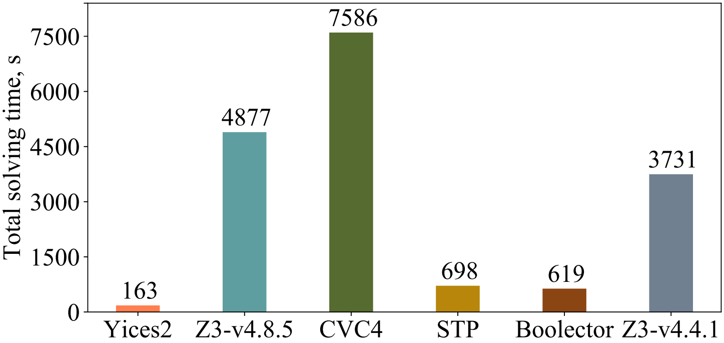


Figure 3: Total time needed to solve all requests

Cumulative Distribution: Anxiety

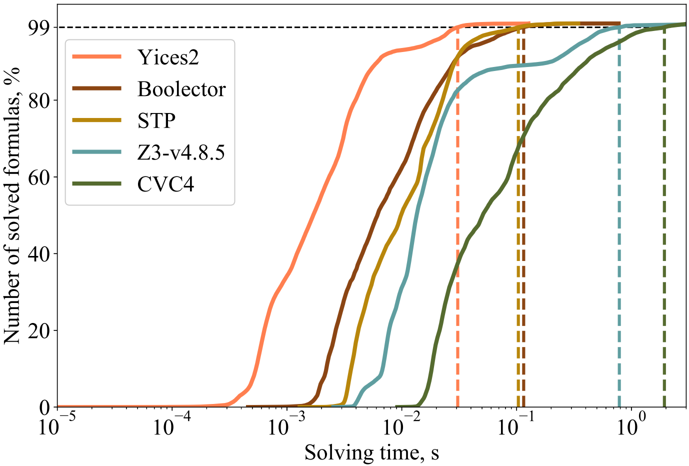


Figure 4: Percentage of requests taking to solve no more than given time

Advanced Experiments

Incremental Solving (Anxiety): Motivation

To get all possible paths from a single path condition $d_1 \wedge c_1 \wedge \cdots \wedge d_n \wedge c_n$, we can either solve one incremental query:

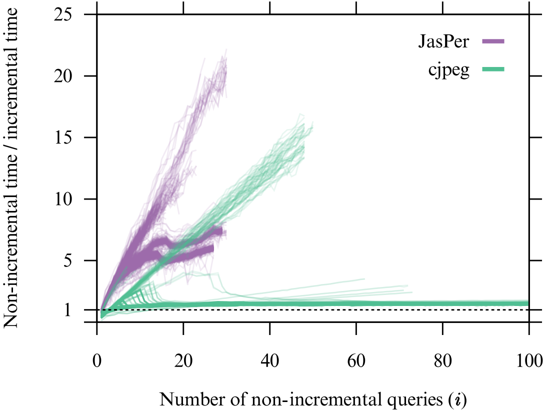
$$A_n = d_1\langle\wedge\overline{c_1}\rangle \wedge c_1 \wedge d_2\langle\wedge\overline{c_2}\rangle \wedge c_2 \wedge \ldots \wedge d_n \wedge \overline{c_n},$$

or n non-incremental queries:

$$\begin{aligned} B_1 &= d_1 \wedge \overline{c_1}, \\ B_2 &= d_1 \wedge c_1 \wedge d_2 \wedge \overline{c_2}, \\ &\ldots \\ B_n &= d_1 \wedge c_1 \wedge d_2 \wedge c_2 \wedge \ldots \wedge d_n \wedge \overline{c_n}, \end{aligned}$$

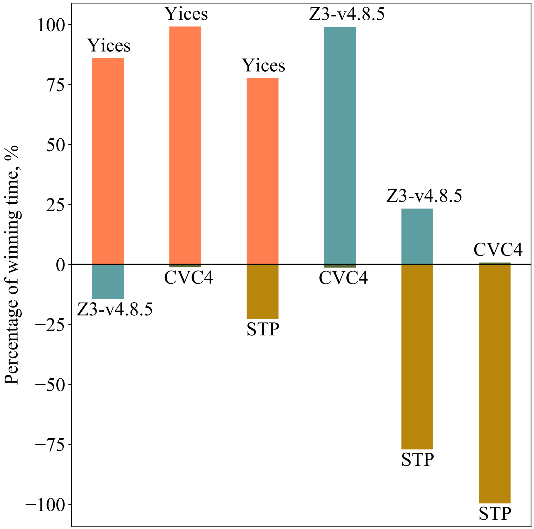
Incremental Solving (Anxiety): Results

Every line on the graph corresponds to a single base path. Every point on a line corresponds to an incremental query A_i, consisting of conditions up to i index and first i non-incremental queries.



Solvers Portfolio (Svace): Motivation

Some Svace requests still take more time to solve with Yices2 than with Z3 or STP.



Solvers Portfolio (Svace): Special Aspects

Classic approach — *SATZilla*, a SAT solver portfolio. Was then adapted for bit-vector solving. Has:

- pre- and backup solvers;
- many expensive features;
- huge time limits.

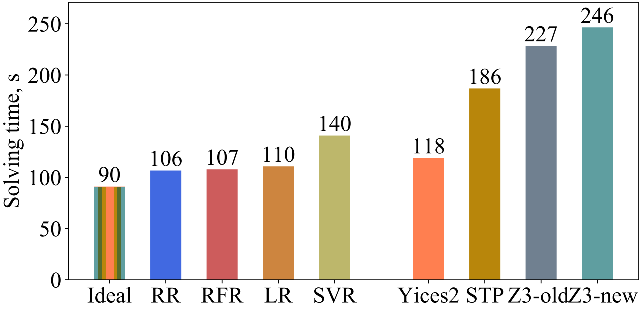
Problem: very little average solving time for Svace.

Consequences:

- no pre- or post-processing;
- only most lightweight features of formula, *BUT* some unique analysis properties are also used as features.

Solvers Portfolio (Svace): Results

We tried different machine-learning regression models: ridge, random forest, linear, support vector. We were summing for every request the solving time of whatever solver predicted to be the best.



Ideal solution is an abstract algorithm that always knows which solver is best.

Conclusion

Symbolic Execution

- Hundreds of thousands formulas may be produced in one instrument run.
- The majority of the formulas are solved in tenths and hundredths of seconds, while a few requests may be incredibly complex.
- For both static and dynamic symbolic execution many formulas are built on the points of a single path. This induces high affinity and common subformulas.
- Dynamic symbolic execution workflow highly depends on models generated. Ability to choose from several models may increase number of paths generated.

Solvers

- In terms of produced answers all solvers are nearly identical.
- For both of our tools, Yices2 proved to be the best at solving.
- Deterministic time limit is required for analyses relying on reproducibility of results.
- Well-implemented `reset` is important for huge number of symbolic execution requests.
- Incremental solving perfectly suits symbolic execution problem.