Murxla: A Modular and Highly Extensible API Fuzzer for SMT Solvers

Aina Niemetz

joint work with Mathias Preiner and Clark Barrett

Centaur Annual Meeting, July 13, 2022

SMT Solvers

- **Tools to solve the SMT Problem**
  - complex and large pieces of software
    - Bitwuzla: $\sim$ 90k LOC
    - cvc5: $\sim$ 300k LOC
    - z3: $\sim$ 500k LOC
  - back-ends in higher-level tool chains
  - **strong requirements:**
    - performance
    - robustness
    - correctness
  - **traditional testing:**
    - unit testing
    - maintaining a regression test suite
    - insufficient for achieving high levels of robustness
  - random stress testing (**fuzzing**)

Fuzz Testing SMT Solvers

SMT solvers provide **two** interfaces:

full knowledge of input structure

- textual interface (SMT-LIB)
  - input fuzzing
  - ▷ generate valid SMT-LIB input
  - \+ significantly less effort
  - − no solver-specific features
- application programming interface (API)
  - API fuzzing
  - ▷ generate valid sequences of solver API calls
  - ▷ link against solver library
  - \+ solver-specific features
  - \+ subsumes input fuzzing (except parser)
  - − more involved

Murxla

... a model-based API Fuzzer for SMT solvers

Murks ... a German (rather informal) word for

⇩ a *botch* or *screw-up*

Murxla ... a tool to find *Murks*es (bugs) in SMT solvers via API fuzzing

What do we consider a bug?

- soundness issues
  - solver answers *unsat* when input is *sat*
  - solver answers *sat* when input is *unsat*
- crashes (assertion failures, segmentation faults, ...)
- performance regressions

Murxla

... a model-based API Fuzzer for SMT solvers

lifts grammar-based input fuzzing to API level

- Semantic (data) model
  - defines constructs (theories, sorts, operators, commands)
  - based on SMT-LIBv2
- API model
  - defines the usage of the solver API itself
- Options model
  - defines solver configuration options and valid combinations

Murxla

... a model-based API Fuzzer for SMT solvers

- Model-based **API fuzzer**
  - ▷ generates valid sequences of solver API calls
  - ▷ general enough to support any SMT solver
  - ▷ highly extensible to support all solver-specific features
- **Tracer**
  - ▷ records API call sequences as an API trace
- **Untracer**
  - ▷ replays API traces to reproduce original behavior
- **Delta Debugger**
  - ▷ minimizes API traces while preserving the original behavior

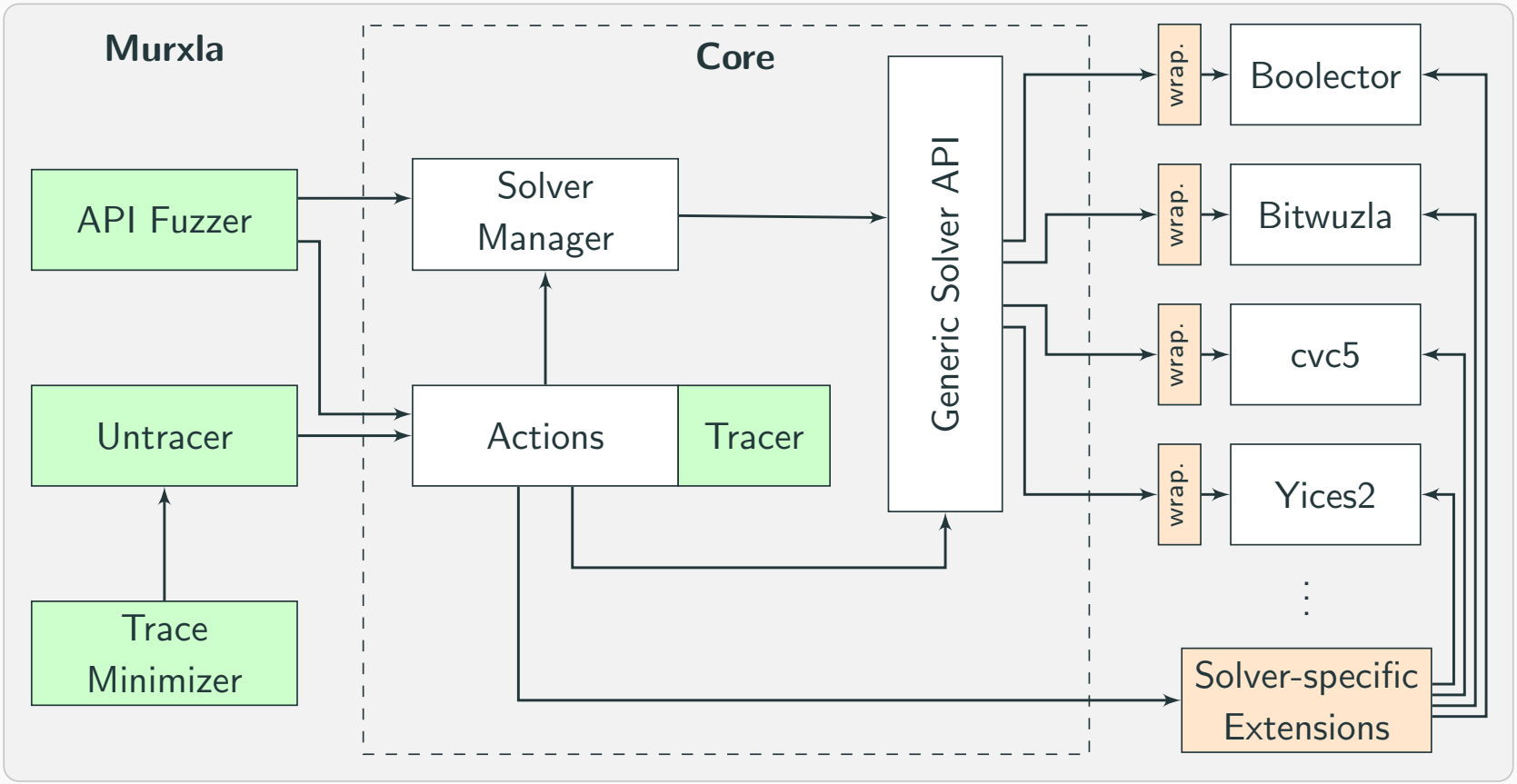
* Provided they allow being integrated into a C++ tool.

Murxla

... a model-based API Fuzzer for SMT solvers

- **translate** API traces to **SMT-LIBv2**
  - if trace doesn't contain solver-specific extensions
  - especially useful for minimized traces
  - can then be further reduced with ddSMT [Kremer et al., CAV 2021]
- **generate** SMT-LIBv2 input
  - can be used as SMT-LIB input fuzzer with any solver binary
- **cross-check** two solver instances
  - two integrated solvers under test
  - one integrated solvers vs. a solver via the SMT-LIBv2 interface

Murxla Architecture



Demo

Evaluation

- **Input Fuzzers**
  - **Storm** [Mansur et al., ESEC/FSE'20]
    - mutates Boolean structure
  - **TypeFuzz** [Park et al., OOPSLA'21]
    - hybrid approach (mutational with generative elements)
    - for integers, reals strings
- **Model-Based API Fuzzers**
  - **BtorMBT** [Niemetz et al., SMT 2017]
    - tailored to (exclusively) Boolector
    - covers all features of Boolector except quantifiers
    - rigorously applied during development and for every release
    - recent fuzzing campaigns found no issues in covered code

Evaluation

Murxla vs. BtorMBT (Boolector)

Murxla			BtorMBT		
L [%]	F [%]	I [#]	L [%]	F [%]	I [#]
81.1	87.5	18	72.3	80.6	0

Murxla vs. Input Fuzzers (cvc5, QF_SLIA)

Murxla			Storm			Murxla-cc			TypeFuzz		
L [%]	F [%]	I [#]	L [%]	F [%]	I [#]	L [%]	F [%]	I [#]	L [%]	F [%]	I [#]
37.8	52.5	7	20.2	34.3	0	21.5	36.3	1	17.4	30.8	0

I ... Number of issues
F ... Function coverage
L ... Line coverag

Murxla-cc ... cross-checking configuration (Z3 vs cvc5)

1 hour, with 1 second time limit per round

Evaluation

Murxla with/without Option Fuzzing

Option	Bitwuzla			Boolector			cvc5			Yices2		
Fuzzing	L [%]	F [%]	I [#]	L [%]	F [%]	I [#]	L [%]	F [%]	I [#]	L [%]	F [%]	I [#]
no	47.4	63.9	7	68.5	79.2	6	38.9	56.8	11	37.0	42.4	1
yes	62.9	75.8	23	81.1	87.7	13	49.1	66.8	21	-	-	

I ... Number of issues
F ... Function coverage
L ... Line coverage

1 hour, with 1 second time limit per round

Conclusion

- Murxla is a tool for fuzzing and debugging SMT solvers
- quick and effective in finding issues
  - even for logics subjected to month-long fuzzing campaigns
- found many issues while finalizing the tool
  - more than 100 issues in cvc5 alone
- being integrated into development workflow of Bitwuzla and cvc5

References

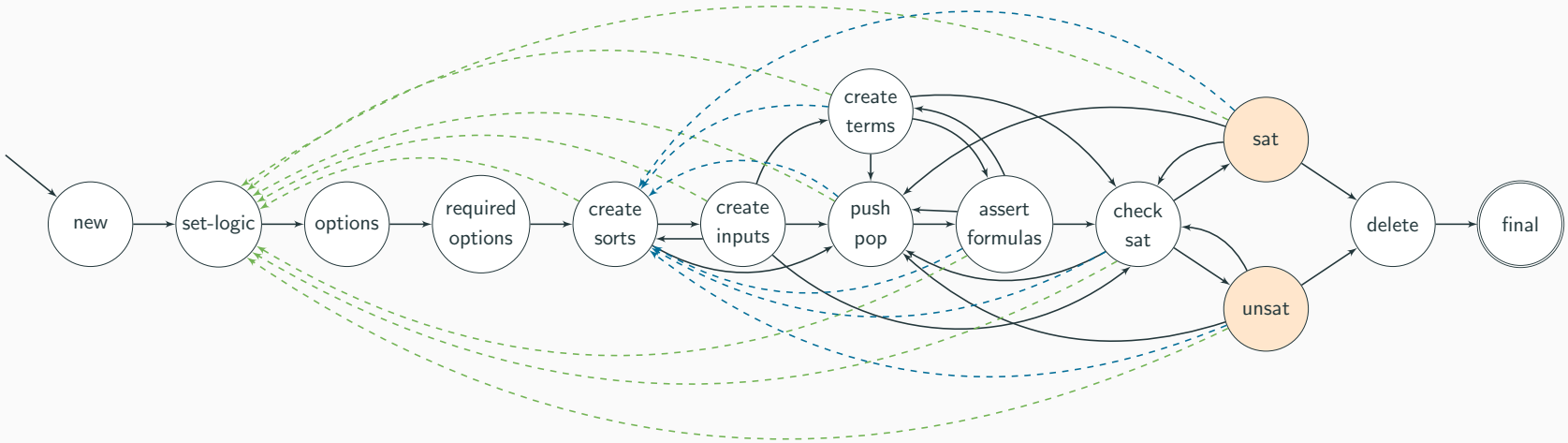
- A. Niemetz, M. Preiner and A. Biere. Model-Based API Testing for SMT Solvers. In Proc. of SMT'17, pages 3–14, 2017. http://ceur-ws.org/Vol-1889/paper1.pdf
- M. N. Mansur, M. Christakis, V. Wüstholz and F. Zhang. Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. In Proc. of ESEC/FSE'20, pages 701–712, ACM, 2020. https://doi.org/10.1145/3368089.3409763
- J. Park, D. Winterer, C. Zhang and Z. Su. Generative type-aware mutation for testing SMT solvers. In Proc. of OOPSLA'21, pages 1–19, ACM, 2021. https://doi.org/10.1145/3485529
- G. Kremer, A. Niemetz and M. Preiner. ddSMT 2.0: Better Delta Debugging for the SMT-LIBv2 Language and Friends. In Proc. of CAV'21, pages 231–242, Springer, 2021. http://dx.doi.org/10.1007/978-3-030-81688-9_11

Input vs. API Fuzzing

```
Solver slv;
Sort sort_int = slv.getIntegerSort(), "x");    // Int
Term x = slv.mkConst(sort_int);                // (declare-fun x () Int)
Term y = slv.mkConst(sort_int, "y");           // (declare-fun y () Int)
slv.assertFormula(slv.mkTerm(DISTINCT, x, y); // (assert (distinct x y))
slv.checkSat();                                // (check-sat)
Term p = slv.mkTerm(PLUS, x, y);               // (+ x y)
Term v = slv.getValue(p);                      // (get-value ((+ x y)))
Term e = slv.mkTerm(EQUAL, p, v);              // ???
slv.checkSatAssuming(e);                       // (check-sat-assuming (???))
```

API Fuzzer

- weighted finite state machine
- transition executes action
- actions are recorded/traced
- step through until solver crashes, time limit or final state is reached



Trace Minimizer

Original: 157 Lines

```
91764 new
45977 set-logic QF_AUFSNIA
  848 set-option produce-difficulty true
  848 set-option strings-exp true
43555 set-option incremental false
22267 set-option produce-unsat-cores true
50067 mk-sort SORT_STRING
      return s1
92374 mk-const s1 "_x0"
      return t1
29153 mk-value s1 ""
      return t2
89065 mk-const s1 "_x1"
      return t3
        .
        .
        .
94281 assert-formula t32
17138 assert-formula t46
77242 assert-formula t43
10259 assert-formula t24
37044 assert-formula t55
28759 check-sat
81533 cvc5-get-difficulty
10993 get-unsat-core
```

Minimized: 25 Lines (16%)

```
91764 new
848 set-option produce-difficulty true
50067 mk-sort SORT_STRING
return s1
92374 mk-const s1 "_x0"
return t1
29153 mk-value s1 ""
return t2
23432 mk-term OP_STR_SUFFIXOF SORT_BOOL 2 t1 t1
return t20 s2
532 mk-term str.tolower SORT_STRING 1 t1
return t23 s1
51711 mk-term OP_EQUAL SORT_BOOL 2 t20 t20
return t24 s2
63692 mk-term OP_STR_SUFFIXOF SORT_BOOL 2 t1 t2
return t25 s2
30349 mk-term OP_NOT SORT_BOOL 1 t20
return t27 s2
81085 mk-term OP_AND SORT_BOOL 2 t27 t24
return t28 s2
29866 mk-term OP_AND SORT_BOOL 2 t28 t20
return t32 s2
94281 assert-formula t32
28759 check-sat
10993 get-unsat-core
```

```
Fatal failure within cvc5::UnsatCore cvc5::SolverEngine::getUnsatCoreInternal() at cvc5/src/smt/solver_engine.cpp:1295
Check failure
    pepf != nullptr
Aborted (core dumped)
```