

# MODEL-BASED API TESTING FOR SMT SOLVERS

Aina Niemetz<sup>\*†</sup>, Mathias Preiner<sup>\*†</sup>, Armin Biere<sup>\*</sup>

<sup>\*</sup> Johannes Kepler University, Linz, Austria

<sup>†</sup> Stanford University, USA

SMT Workshop 2017, July 22 – 23  
Heidelberg, Germany



# SMT Solvers

- highly complex
- usually serve as back-end to some application

- **key requirements:**

- correctness
- robustness
- performance

→ **full verification** difficult and still an open question

→ solver development relies on **traditional testing** techniques

# Testing of SMT Solvers

## State-of-the-art:

- **unit** tests
- **regression** test suite
- grammar-based black-box **input fuzzing** with **FuzzSMT** [SMT'09]
  - generational input fuzzer for SMT-LIB v1
  - patched for SMT-LIB v2 compliance
  - generates random but valid SMT-LIB input
  - **especially effective in combination with delta debugging**
- ◇ **not possible** to test solver features **not supported by the input language**

**This work:** **model-based API fuzz testing**

→ generate **random valid API call sequences**

# Model-Based API fuzz testing

→ generate **random valid API call sequences**

## ■ Previously: **model-based API testing framework for SAT** [TAP'13]

- implemented for the SAT solver **Lingeling**
- allows to test random solver configurations (**option fuzzing**)
- allows to **replay** erroneous solver behavior

→ results **promising** for other solver back-ends

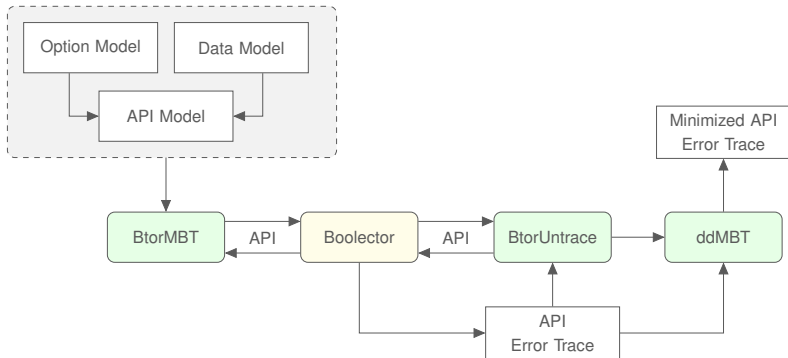
## ■ Here: **model-based API testing framework for SMT**

- lifts SAT approach to SMT
- implemented for the SMT solver **Boolector**
- ◇ tailored to Boolector
- ◇ for QF\_(AUF)BV with non-recursive first-order lambda terms

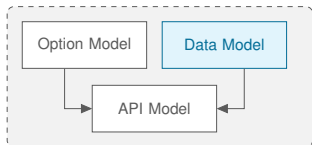
→ **effective** and promising for other SMT solvers

→ more **general** approach left to **future work**

# Workflow



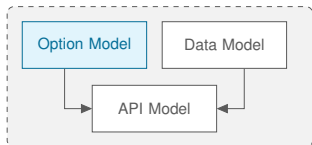
# Models



## Data Model

- SMT-LIB v2
- quantifier-free bit-vectors
- arrays
- uninterpreted functions
- lambda terms

# Models



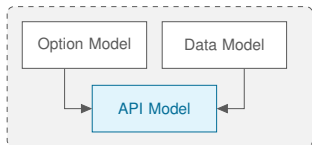
## Option Model

- default values
- min / max values
- (in)valid combinations
- solver-specific

## Boolector:

- multiple solver engines
- 70+ options (total)
- query all options (+ min, max and default values) via API

# Models



## API model

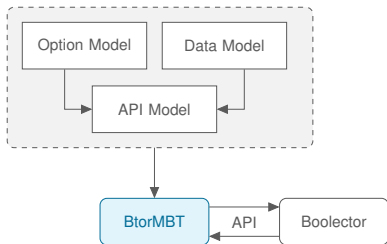
- full feature set available via API
- finite state machine

## Boolector:

- full access to complete solver feature set
- 150+ API functions



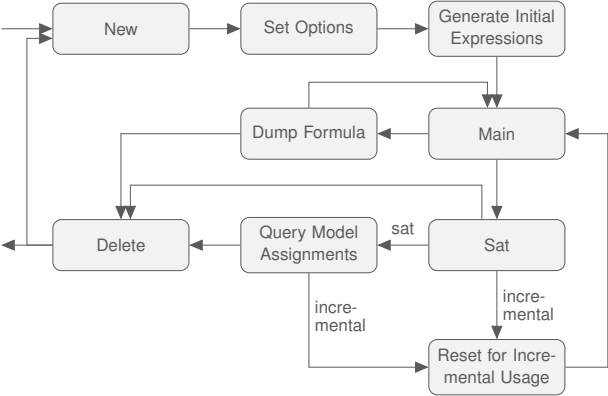
# BtorMBT



- test case generation engine
- API fuzz testing tool
- implements API model
- dedicated tool for testing random configurations of Boolector
- integrates Boolector via C API
- fully supports all functionality provided via API

# BtorMBT

## API Model



# BtorMBT

## Option Fuzzing

- multiple **solver engines**
  - configurable with **70+ options** (total)
  - several **SAT solvers** as back-end
1. choose logic (QF\_BV, QF\_ABV, QF\_UFBV, QF\_AUFBV)
  2. choose solver engine (depends on logic)
  3. choose configuration options and their values
    - within their predefined value ranges
    - based on option model
    - exclude **invalid** combinations
    - choose more **relevant** options with **higher** probability (e.g. incrementality)

# BtorMBT

## Expression Generation

- generate **initial set** of expressions
  1. randomly sized shares of **inputs**
    - Boolean variables
    - bit-vector constants and variables
    - uninterpreted function symbols
    - array variables
  2. **non-input** expressions
    - combine inputs and already generated non-input expressions
    - with operators

→ until a max number of initial expressions is reached
  
- randomly generate **new** expressions after initialization
  - choose expressions from the initial set with lower probability
  - to increase expression depth

# BtorMBT

## Dump Formula

- output format: **BTOR**, **SMT-LIB v2** and **AIGER**
  
- BTOR and SMT-LIB v2:
  1. dump to temp file
  2. parse temp file (into temp Booletor instances)
  3. check for parse errors
  
- AIGER
  - ◇ QF\_BV only
  - currently no AIGER parser
  - dump to stdout without error checking

# BtorMBT

## Solver-Internal Checks

- **model validation** for **satisfiable** instances
  - after each SAT call that concludes with **satisfiable**
  
- check **failed assumptions** for **unsatisfiable** instances
  - in case of **incremental** solving
  - determine the set of inconsistent (failed) assumptions
  - check if failed assumptions are indeed inconsistent
  
- check internal state of **cloned instances**
  - data structures
  - allocated memory
  
- automatically enabled in debug mode

# BtorMBT

## Shadow Clone Testing

### ■ full clone

- exact **disjunct** copy of solver instance
- exact **same behavior**
- deep** copy

→ includes (bit-blasted) **AIG** layer and **SAT** layer

→ requires SAT solver to support cloning

### ■ term layer clone

- term layer** copy of solver instance
- does not guarantee exact same behavior

→ **shadow clone testing** to test **full** clones

# BtorMBT

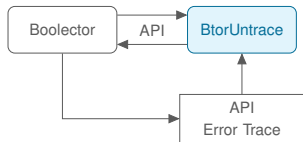
## Shadow Clone Testing

1. generate shadow clone (**initialization**)
  - may be initialized **anytime** prior to the first SAT call
  - is randomly **released** and **regenerated** multiple times
  - solver checks **internal state** of the freshly generated clone
2. shadow clone **mirrors** every API call
  - solver checks state of shadow clone after each call
3. **return values** must correspond to results of original instance

→ **enabled at random**



# BtorUntrace

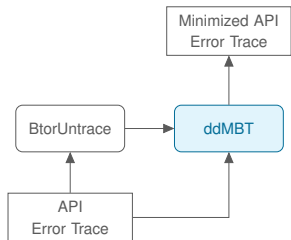


- replay API traces
- reproduce solver behavior
- ◇ failed test cases
- ◇ faulty behavior outside of API testing framework
- without the need for the original (complex) setup of the tool chain
- for traces generated by Boolector
- integrates Boolector via C API

## Example API Trace

```
1 | new
2 | return b1
3 | set_opt b1 1 incremental 1
4 | set_opt b1 14 rewrite-level 0
5 | bitvec_sort b1 1
6 | return s1@b1
7 | array_sort b1 s1@b1 s1@b1
8 | return s3
9 | array b1 s3@b1 array1
10 | return e2@b1
11 | var b1 s1@b1 index1
12 | return e3@b1
13 | var b1 s1@b1 index2
14 | return e4@b1
15 | read b1 e2@b1 e3@b1
16 | return e6@b1
17 | read b1 e2@b1 e4@b1
18 | return e8@b1
19 | eq b1 e3@b1 e4@b1
20 | return e9@b1
21 | ne b1 e6@b1 e8@b1
22 | return e-10@b1
23 | assert b1 e9@b1
24 | assume b1 e-10@b1
25 | sat b1
26 | return 20
27 | failed b1 e-10@b1
28 | return true
29 | sat b1
30 | return 10
31 | release b1 e2@b1
32 | release b1 e3@b1
33 | release b1 e4@b1
34 | release b1 e6@b1
35 | release b1 e8@b1
36 | release b1 e9@b1
37 | release b1 e-10@b1
38 | release_sort b1 s1@b1
39 | release_sort b1 s3@b1
40 | delete b1
```

# ddMBT



- minimize trace file
  - while preserving behavior when replayed with BtorUntrace
  - based on solver exit code and error message
  - works in rounds
1. remove lines (divide and conquer)
  2. substitute terms with fresh variables
  3. substitute terms with expressions of same sort

# Experimental Evaluation

## Configurations

- **BtorMBT** as included with Boolector 2.4
  - Boolector compiled with support for Lingeling, PicoSAT, MiniSAT
- **FuzzSMT** patched for SMT-LIB v2 compliance
- **with** and **without** option fuzzing
  - randomly choosing solver engines and SAT solvers enabled even when option fuzzing disabled

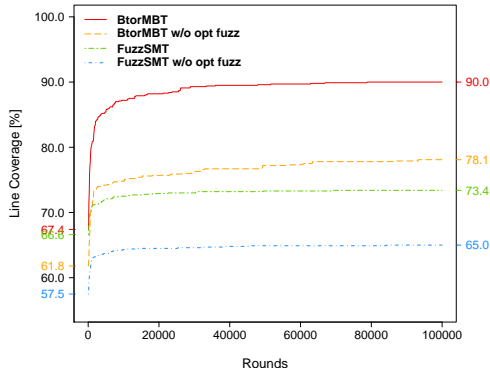
# Experimental Evaluation

## Throughput

- important measure of **efficiency** and **effectiveness**
  - high throughput: test cases **too trivial**
  - low throughput: test cases **too difficult**
- goal:** as many good test cases in as little time as possible
- 100k runs
- solver timeout: 2 seconds
- ◇ **BtorMBT:** 45 rounds / second
  - +20% throughput **without** shadow clone testing
  - 20% of SAT calls **incremental**
  - 25% of solved instances is **satisfiable**
- ◇ **FuzzSMT:** 7 rounds / second

# Experimental Evaluation

## Code Coverage (gcc gcov)



	BtorMBT	BtorMBT w/o opt fuzz
10k	87 %	75 %
100k	90 %	78 %

→ >98% API coverage

	FuzzSMT	FuzzSMT w/o opt fuzz
10k	73 %	62 %
100k	74 %	65 %

→ >52% API coverage

(incomplete SMT-LIB v2 support)

# Experimental Evaluation

## Defect Insertion

### Test configurations:

- 4626 **faulty** configurations (total)
- **TC<sub>A</sub>** randomly inserted **abort** statement (2305 configurations)
- **TC<sub>D</sub>** randomly **deleted** statement (2321 configurations)
  
- all configurations are **faulty** configurations
- 100k runs (BtorMBT) and 10k runs (FuzzSMT)
- solver timeout: 2 seconds

# Experimental Evaluation

## Defect Insertion

Rounds		BtorMBT		BtorMBT w/o opt fuzz		FuzzSMT		FuzzSMT w/o opt fuzz	
		Found	[%]	Found	[%]	Found	[%]	Found	[%]
100k	TC <sub>A</sub> (2305)	2088	90.6	1789	77.6				
	TC <sub>D</sub> (2321)	1629	70.2	1366	58.9				
	<b>TC (4626)</b>	<b>3717</b>	<b>80.4</b>	<b>3155</b>	<b>68.2</b>				
10k	TC <sub>A</sub> (2305)	2028	88.0	1719	74.6	1735	75.3	1523	66.1
	TC <sub>D</sub> (2321)	1510	65.1	1277	55.0	1304	56.2	1153	49.7
	<b>TC (4626)</b>	<b>3538</b>	<b>76.5</b>	<b>2996</b>	<b>64.8</b>	<b>3039</b>	<b>65.7</b>	<b>2676</b>	<b>57.8</b>

→ success rates for TC<sub>A</sub> roughly correspond to code coverage



## Conclusion

- model-based API testing tool set for Boolector
- generates random valid sequences of API calls
- allows to test random solver configurations on random input formulas

## Future Work:

- let BtorMBT take over **API tracing**
- more **balanced ratio** of sat to unsat instances
- maximize code coverage with **symbolic execution** techniques
- **solver-independent model-based api testing framework**

## References I

- [SMT'09] R. Brummayer and A. Biere. Fuzzing and Delta-Debugging SMT Solvers. In Proc. of the 7th International Workshop on Satisfiability Modulo Theories (SMT'09), 5 pages, ACM, 2009.
- [TAP'13] C. Artho, A. Biere and M. Seidl. Model-Based Testing for Verification Back-Ends. In Proc. of the 7th International Conference on Tests and Proofs (TAP 2013), LNCS volume 7942, pages 39–55, Springer, 2013.