Fuzzing and Delta-Debugging SMT Solvers

Robert Brummayer and Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

SMT 2009

Montreal, Canada

August 2nd, 2009

Piled Higher and Deeper ***by Jorge Cham*** **www.phdcomics.com**

title: "Debugging" - originally published 1/14/2006

SMT solvers often used as workhorses

- verification, symbolic execution, compiler optimization, scheduling, ...

SMT solver requirements

- correctness, robustness, efficiency, ...

Incorrect SMT solver may lead to incorrect overall results

Crashing SMT solver may lead to crash of the overall system

Non-terminating SMT solver may lead to non-terminating overall system

SMT solvers implement theoretically studied decision procedures

Demand for efficiency leads to error prone optimizations

Neglected topics in the SMT community

- how do we test/verify that our implementations are correct?
- how do we make sure that our solvers are robust?

Traditional approaches based on testing

- unit testing / regression testing
  - ➢ tedious and insufficient task of generating test cases manually

Grammar-Based Black-Box Fuzzing

- test SMT solver with random SMT formulas for specified theory

- randomness in input causes execution of untested code and corner cases

- impressingly effective black box approach

  ➢ no knowledge about implementation details needed

Formulas may be large which makes debugging hard/infeasible

Solution: Delta-debugging [Zeller] to minimize failure-inducing formulas

Divide formula into layers

- successful approach for generating random BTOR instances [Vida]

SMT formula can typically be divided into at least four layers

- input, main, predicate, and boolean

Start generating variables for input layer

For each non-input layer:
Generate random nodes by using previously created nodes of matching type

Finally, combine boolean nodes to one root

In contrast to other theories

- many operators, some use bit-vectors and naturals as operands
- many different types as bit-widths should be random within a range
- most bit-vector operators require operands with same bit-width
  - ➢ make bit-widths of randomly selected operands compatible
  - ➢ use `zero_extend`, `sign_extend` or `extract`

Encode boolean nodes as bit-vectors to find subtle defects

```
(ite $n24 bv1[1] bv0[1])
```

Add further layers

- array input, read, and write

Interleaving creation of bit-vector nodes, reads and writes

- reads are used as read/write indices, write values, and BV operands

Extensionality

- compare arrays for equality in boolean layer
- encode result of comparing arrays as bit-vector
  - ➢ may be used as (a part of a) BV op, read/write index, or write value

Delta-Debugging SMT formulas

Delta-debugger (DD) runs solver (or wrapper script) on original failure-inducing formula ϕ to obtain golden exit code

DD iteratively tries to simplify the failure-inducing formula

After each simplification, DD calls solver with a simplified formula ϕ'

- if exit code = golden exit code, success, continue simplifying ϕ'
- if exit code $\neq$ golden exit code, failure, backtrack, try other simplification

Instead of running the solver directly, a wrapper script can be used

- script determines whether the observable behavior is different or not
  - ➢ for example, grep for a specific error message

Perform search through boolean layer and try to replace root by sub-formula

- may immediately prune large parts of the whole formula

For each term/formula node n

- try to substitute n by constant 0/1 resp. false/true
- for each child c of n
  - ➢ if types of c and n are compatible, then try to substitute n by child c
- try to skip ”chains“ of unary operators and array writes

Finally, try to substitute root by remaining boolean nodes (e.g. inside `ite`)

Delta-Debugging with Timeouts

R. Brummayer, FMV, JKU Linz

Delta-debugger (DD) can use a time limit for each call to SMT solver

- if solver exceeds time limit, treat simplification as failure and backtrack
- necessary for non-terminating SMT solvers
  - ➢ DD may generate formula on which SMT solver does not terminate
  - ➢ Without time limit, DD would wait forever

Use wrapper script with timeout to delta-debug non-terminating solvers

- if solver exceeds time limit, treat it as non-terminating, return exit code
- if solver does not exceed time limit, return different exit code

Basic Majority Voting Framework for SMT

R. Brummayer, FMV, JKU Linz

Assume we have an SMT theory T, a set of solvers S for T, and a time limit l

- repeatedly use fuzzer to generate random formula ϕ of theory T

- for each solver s in S
  - ➢ result := execute s with ϕ under time limit l
  - ➢ if result = sat or result = unsat, then remember result for s
  - ➢ else mark ϕ as failure-inducing input for s

- if solvers disagree on satisfiability status of ϕ
  - ➢ assume majority is correct; mark ϕ as failure-inducing for minority

	no-div		guard-div	
solver	crash	incorrect	crash	incorrect
Beaver 1.1 rc1	0	0	12430	1
Boolector 1.0	0	0	0	0
Boolector 1.1	0	0	0	0
CVC3 1.5 April 29th	902	8	-	-
MathSAT 4.2.3	0	113	2097	83
OpenSMT internal	19871	8	-	-
Spear 2.7	0	6	3577	71
Sword smt-comp08	0	1	0	0
Z3 1.2	0	0	2264	0
Z3 smt-comp08	0	0	0	0

- formulas in `no-div` do not contain any division operators
- formulas in `guard-div` use restricted form of division

R. Brummayer, FMV, JKU Linz

	no-div					guard-div				
solver	f	c	t	s	r	f	c	t	s	r
Beaver 1.1 rc1	-	-	-	-	-	469	12	5	319	98%
CVC3 1.5 April 29th	139	9	172	2429	98%	-	-	-	-	-
MathSAT 4.2.3	50	1	10	611	97%	190	5	58	3709	76%
OpenSMT internal	154	4	5	492	96%	-	-	-	-	-
Spear 2.7	6	1	5	401	96%	100	2	4	228	99%
Sword smt-comp08	1	1	4	135	99%	-	-	-	-	-
Z3 1.2	-	-	-	-	-	50	1	734	254	99%

f = # formulas
c = # bug classes
t = average delta-debugging time (seconds)
s = average reduced file size (bytes)
r = average file size reduction

statistical outliers: median values better, see details in paper

solver	error	incorrect
Barcelogic smt-comp08	20	0
CVC3 1.5 June 24th	0	0
MathSAT 4.2.5	72	19
Sateen smt-comp08	190	229
Yices 1.0.21	0	0
Z3 smt-comp08	0	19

solver	f	c	t	s	r
Barcelogic smt-comp08	20	1	29	979	55%
MathSAT 4.2.5	69	2	1	191	92%
Sateen smt-comp08	103	4	3	1231	46%
Z3 smt-comp08	17	1	3	614	71%

- errors for Barcelogic are cases where it does not seem to terminate
- statistical outliers: medians better (73% for Bareclogic, 92% for Z3)

FuzzSMTBV

- first prototype for bit-vector and array formulas
- written in Python 2

FuzzSMT

- highly configurable fuzzer for a large number of SMT logics:

  QF_A, QF_AUFBV, QF_AUFLIA, QF_AX, QF_BV, QF_IDL, QF_LIA, QF_LRA, QF_NIA, QF_NRA, QF_RDL, QF_UF, QF_UFBV, QF_UFIDL, QF_UFLIA, QF_UFLRA,QF_UFNIA, QF_UFNRA, QF_UFRDL, AUFLIA, AUFLIRA and AUFNIRA.

- written in Java 5

DeltaSMT

- SMT delta-debugger supporting timeouts, written in Java 5
- supports most of the logics that are supported by FuzzSMT

VoteSMT

- majority voting framework for SMT implemented by Bash scripts
- automatically classifies results as correct and incorrect
- can be used to find failure-inducing formulas
- runs one fuzz testing process on each processor core

Conclusion and Future Work

R. Brummayer, FMV, JKU Linz

Fuzz testing

- automatic approach that uses random formulas to find defects

Delta-debugging

- automatic approach to minimize failure-inducing formulas

Majority voting

- automatic approach to classify results as correct or incorrect

Future work

- we work on a paper to apply our techniques to SAT/QBF solvers