ddSMT: A Delta Debugger for the SMT-LIB v2 Format

Aina Niemetz and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

SMT Workshop 2013
July 8 - 9, 2013
Helsinki, Finland

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (! (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

Motivation

Why delta debugging?

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (! (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

Motivation

What delta debugging?

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (! (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

```
(set-logic UFNIA)
(get-value (false))
(exit)
```

Motivation

What delta debugging?

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (! (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

```
(set-logic UFNIA)
(get-value (false))
(exit)
```

⟶ easier to debug

⟶ in a time efficient manner

Delta Debugging = input minimization

⟶ originally introduced by R. Hildebrandt and A. Zeller in [HZ00]
⟶ related work: **shrinking** in QuickCheck [CH00]

Basic Idea:

Given executable **Ex**, failure-inducing input **I**:

- Minimize (simplify) $I \longrightarrow I_{simp}$
- I_{simp} still triggers the original faulty behavior

Original minimization strategy:

- divide-and-conquer (binary)
- remove parts irrelevant to the original faulty behavior

⟶ highly customizable, wide range of applications

deltaSMT

- introduced by our group in [BB09]
- tailored to the **SMT-LIB v1** format
- does **not** support quantifiers
- implements hierarchical delta debugging strategy
- nodes are substituted one-by-one in a breadth-first-search (BFS) manner by simpler nodes or their children

  ⟶ bottleneck in the worst case

  ⟶ cases, where deltaSMT struggled or unable to minimize input

deltaSMT2

- recent and independent update of deltaSMT for SMT-LIB v2
- by P. F. Dobal and P. Fontaine at INRIA
- syntactically extends deltaSMT for SMT-LIB v2 compliance
- **no full** support for SMT-LIB v2
- still work-in-progress

deltaSMT

- introduced by our group in [BB09]
- tailored to the **SMT-LIB v1** format
- does **not** support quantifiers
- implements hierarchical delta debugging strategy
- nodes are substituted one-by-one in a breadth-first-search (BFS) manner by simpler nodes or their children

  $\longrightarrow$ bottleneck in the worst case

  $\longrightarrow$ cases, where deltaSMT struggled or unable to minimize input

deltaSMT2

- recent and independent update of deltaSMT for SMT-LIB v2
- by P. F. Dobal and P. Fontaine at INRIA
- syntactically extends deltaSMT for SMT-LIB v2 compliance
- **no full** support for SMT-LIB v2
- still work-in-progress

ddSMT

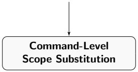
- input minimizer for the **SMT-LIB v2** format
- based on the exit code of a given executable
- supports **all** SMT-LIB v2 logics
- **not** based on deltaSMT
- implements divide-and-conquer delta debugging strategy
- employs simplification strategies for
  - macros (command **define-fun**)
  - command-level scopes (commands **push** and **pop**)
  - named annotations (attribute **:named**)
- especially effective in combination with fuzz testing

Technical Side Notes

- implemented in Python 3
- provides a dedicated, modular, standalone SMT-LIB v2 parser

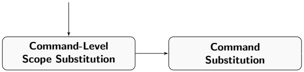
ddSMT

General Workflow



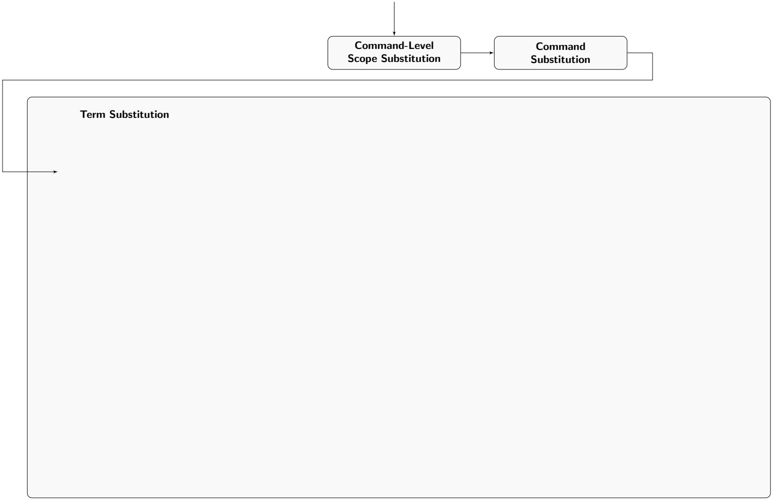
ddSMT

General Workflow



ddSMT

General Workflow



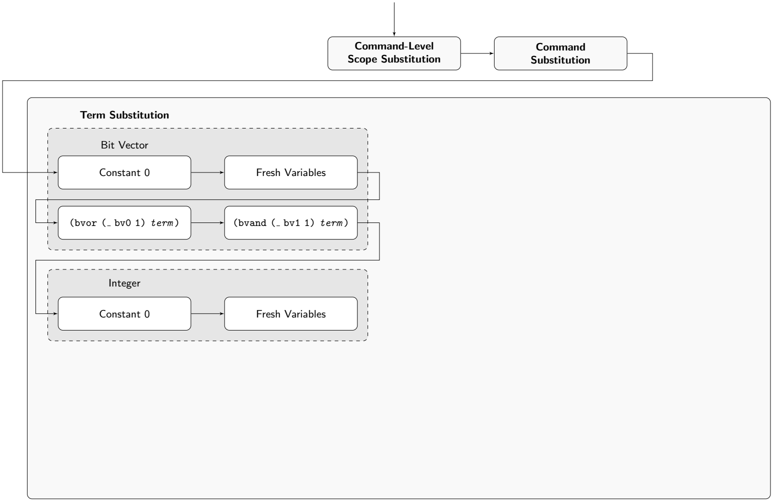

Command-Level
Scope Substitution
Command
Substitution
Term Substitution
for each set of define-fun,
assert and get-value commands

ddSMT

General Workflow

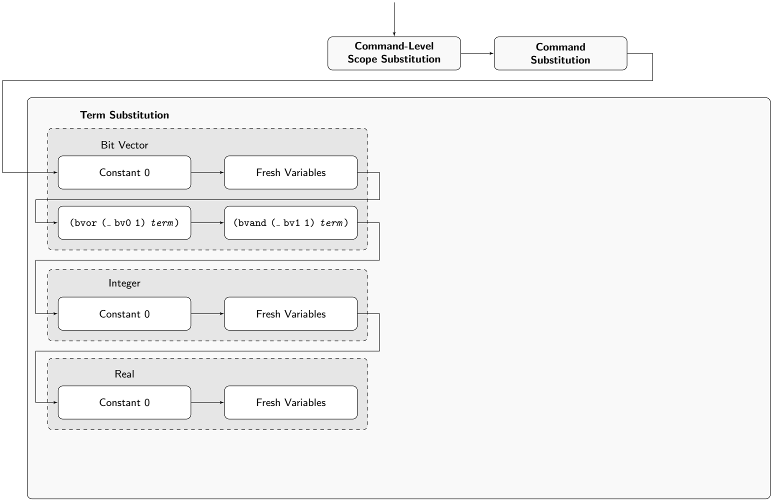
ddSMT

General Workflow



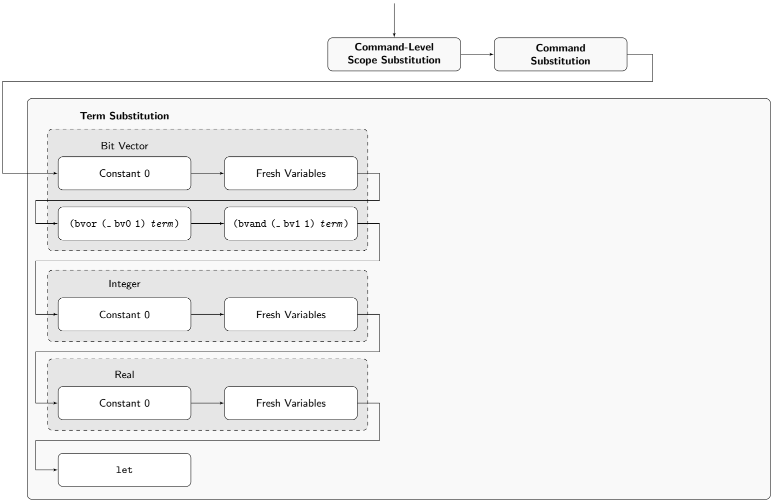
ddSMT

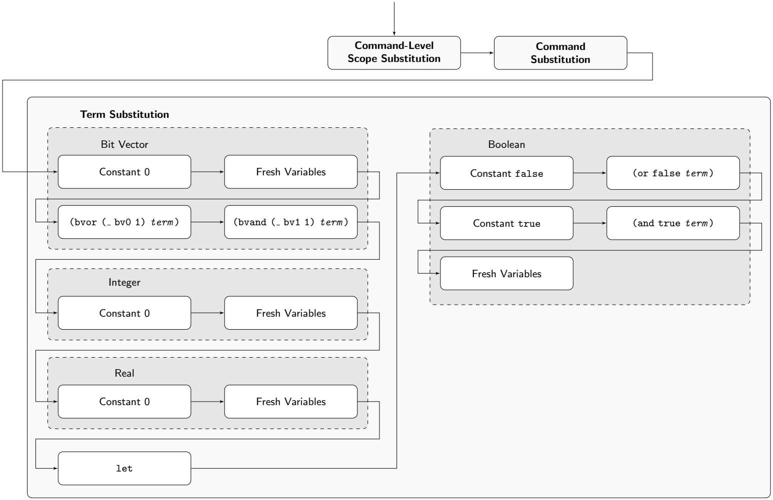
General Workflow



ddSMT

General Workflow

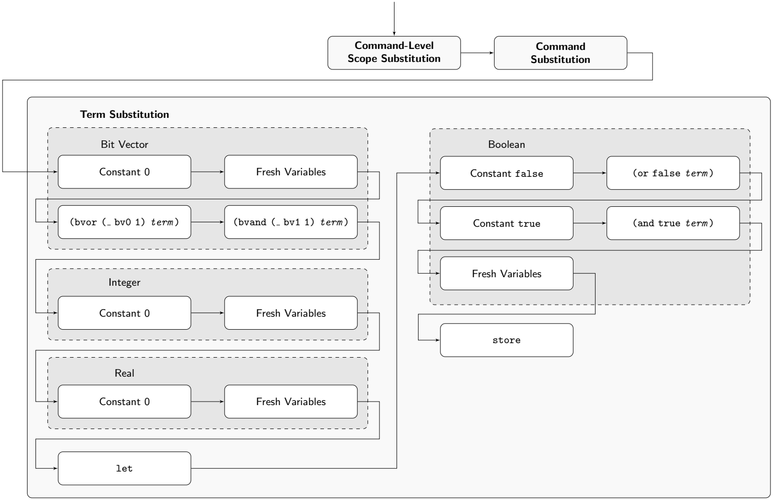


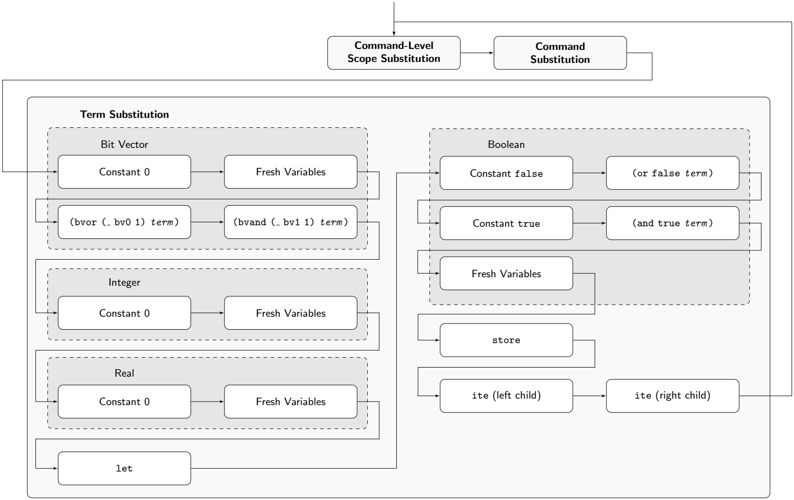


Command-Level Scope Substitution
Command Substitution
Term Substitution
Bit Vector
Constant 0
Fresh Variables
(bvor (_ bv0 1) term)
(bvand (_ bv1 1) term)
Integer
Constant 0
Fresh Variables
Real
Constant 0
Fresh Variables
let
Boolean
Constant false
(or false term)
Constant true
(and true term)
Fresh Variables

ddSMT

General Workflow




Command-Level
Scope Substitution
Command
Substitution
Term Substitution
Bit Vector
Constant 0
Fresh Variables
(bvor (_ bv0 1) term)
(bvand (_ bv1 1) term)
Integer
Constant 0
Fresh Variables
Real
Constant 0
Fresh Variables
let
Boolean
Constant false
(or false term)
Constant true
(and true term)
Fresh Variables
store
ite (left child)
ite (right child)

1. filter nodes by some criteria and collect into list **superset**
2. **substitute**:
   1. divide superset into $\mathbf{n := \lceil len(superset)/g \rceil}$ subsets,
      start with granularity $g := len(superset)$, $n = 1$
   2. for each **subset** in subsets:
      - substitute all **not substituted** items in subset with **simpler** expression or **None**
      - test current configuration
      - if successful keep substitution of subset, $\mathbf{subsets := subsets \setminus subset}$
      - else reset substitutions of current subset
   3. $g := g/2$
   4. $k := len(subsets)$, $\mathbf{superset := \bigcup_{i=1}^{k} subset_i}$

Example

Original Input

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (!  (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

Example

Executable

```
#!/bin/sh

if [ `grep -c "\<get-value\>" $1` -ne 0 ];
  then exit 1
fi

exit 0
```

⟶ simulates: SMT Solver does not support **get-value** commands

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
(push 1)
  (define-sort sort2 () Bool)
  (declare-fun x () sort2)
  (declare-fun y () sort2)
  (assert (and (as x Bool) (as y Bool)))
  (assert (!  (not (as x Bool)) :named z))
  (assert z)
(pop 1)
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

redundant

Example

Command-Level Scope Substitution

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
```

```
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
(assert (= x y ))
```

redundant

```
(assert (forall ((z Int)) (exists ((zz Int)) (= z zz))))
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

```
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

Example

Term Substitution

Int with Constant 0

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

non-constant Int terms

```
(check-sat)
(get-value ((let ((x 1) (y 1)) (= x y))))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

```
(check-sat)
(get-value ((let ((x 1) (y 1)) (= 0 0))))
(exit)
```

Example

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

all variable bindings substituted

```
(check-sat)
(get-value ((let ((x 1) (y 1)) (= 0 0))))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)

(check-sat)
(get-value ((= 0 0)))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

non-constant Boolean term

```
(check-sat)
(get-value ((= 0 0)))
(exit)
```

Example

Term Substitution

Bool with Constant *false*

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

```
(check-sat)
(get-value (false))
(exit)
```

```
(set-logic UFNIA)
(declare-sort sort1 0)
(declare-fun x () sort1)
(declare-fun y () sort1)
```

redundant

```
(check-sat)
(get-value (false))
(exit)
```

Example
Final Result

```
(set-logic UFNIA)

(get-value (false))
(exit)
```

Setup: 3.4 GHz Intel Core i7-2600, 16GB RAM, on a 64 Bit Arch Linux OS

	TS	Files	Red. [%]			Time [s]			Runs			Mem. [MB]			
			avg	min	max	avg	min	max	avg	min	max	avg	min	max	
deltaSMT	1	2	0	0	0	257	14	500	4051	655	7446	113	108	117	SMT-LIB v1
	2	95	94.0	0	99.9	49	0.1	1738	599	5	7296	111	33	153	
	3	5	66.6	0	93.8	12	3	34	608	262	1297	107	76	126	
	4	53	99.6	98.8	99.9	8	0.6	20	463	4	852	128	52	142	
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	
ddSMT	1	2	90.0	83.9	96.0	44	9	79	1412	782	2041	13	10	16	SMT-LIB v2
	2	95	94.7	68.2	99.9	92	0.1	1594	1499	2	3790	15	10	24	
	3	5	80.4	66.8	87.2	23	14	35	1533	1171	1764	11	10	12	
	4	53	99.8	99.3	99.9	57	1	246	431	13	1240	28	15	42	
	5	5	97.4	95.7	98.3	12	5	16	247	215	371	39	10	59	

Originally SMT-LIB v1 input (QF_AUFBV), no SMT-LIB v2-specific features, SMT Solver: Boolector

	TS	Files	Red. [%] avg	Red. [%] min	Red. [%] max	Time [s] avg	Time [s] min	Time [s] max	Runs avg	Runs min	Runs max	Mem. [MB] avg	Mem. [MB] min	Mem. [MB] max	
deltaSMT	1	2	0	0	0	257	14	500	4051	655	7446	113	108	117	SMT-LIB v1
	2	95	94.0	0	99.9	49	0.1	1738	599	5	7296	111	33	153	
	3	5	66.6	0	93.8	12	3	34	608	262	1297	107	76	126	
	4	53	99.6	98.8	99.9	8	0.6	20	463	4	852	128	52	142	
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	
ddSMT	1	2	90.0	83.9	96.0	44	9	79	1412	782	2041	13	10	16	SMT-LIB v2
	2	95	94.7	68.2	99.9	92	0.1	1594	1499	2	3790	15	10	24	
	3	5	80.4	66.8	87.2	23	14	35	1533	1171	1764	11	10	12	
	4	53	99.8	99.3	99.9	57	1	246	431	13	1240	28	15	42	
	5	5	97.4	95.7	98.3	12	5	16	247	215	371	39	10	59	

Originally SMT-LIB v1 input (QF_AUFBV),
no SMT-LIB v2-specific features,
SMT Solver: Boolector

	TS	Files	Red. [%]			Time [s]			Runs			Mem. [MB]			
			avg	min	max	avg	min	max	avg	min	max	avg	min	max	
deltaSMT	1	2	0	0	0	257	14	500	4051	655	7446	113	108	117	SMT-LIB v1
	2	95	94.0	0	99.9	49	0.1	1738	599	5	7296	111	33	153	
	3	5	66.6	0	93.8	12	3	34	608	262	1297	107	76	126	
	4	53	99.6	98.8	99.9	8	0.6	20	463	4	852	128	52	142	
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	
ddSMT	1	2	90.0	83.9	96.0	44	9	79	1412	782	2041	13	10	16	SMT-LIB v2
	2	95	94.7	68.2	99.9	92	0.1	1594	1499	2	3790	15	10	24	
	3	5	80.4	66.8	87.2	23	14	35	1533	1171	1764	11	10	12	
	4	53	99.8	99.3	99.9	57	1	246	431	13	1240	28	15	42	
	5	5	97.4	95.7	98.3	12	5	16	247	215	371	39	10	59	

Originally SMT-LIB v2 input (AUFLIRA),
no push and pop,
SMT Solver: CVC4

	TS	Files	Red. [%]			Time [s]			Runs			Mem. [MB]			
			avg	min	max	avg	min	max	avg	min	max	avg	min	max	
deltaSMT	1	2	0	0	0	257	14	500	4051	655	7446	113	108	117	SMT-LIB v1
	2	95	94.0	0	99.9	49	0.1	1738	599	5	7296	111	33	153	
	3	5	66.6	0	93.8	12	3	34	608	262	1297	107	76	126	
	4	53	99.6	98.8	99.9	8	0.6	20	463	4	852	128	52	142	
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	
ddSMT	1	2	90.0	83.9	96.0	44	9	79	1412	782	2041	13	10	16	SMT-LIB v2
	2	95	94.7	68.2	99.9	92	0.1	1594	1499	2	3790	15	10	24	
	3	5	80.4	66.8	87.2	23	14	35	1533	1171	1764	11	10	12	
	4	53	99.8	99.3	99.9	57	1	246	431	13	1240	28	15	42	
	5	5	97.4	95.7	98.3	12	5	16	247	215	371	39	10	59	

	TS	Files	Red. [%]			Time [s]			Runs			Mem. [MB]			
			avg	min	max	avg	min	max	avg	min	max	avg	min	max	
deltaSMT	1	2	0	0	0	257	14	500	4051	655	7446	113	108	117	**SMT-LIB v1**
	2	95	94.0	0	99.9	49	0.1	1738	599	5	7296	111	33	153	
	3	5	66.6	0	93.8	12	3	34	608	262	1297	107	76	126	
	4	53	99.6	98.8	99.9	8	0.6	20	463	4	852	128	52	142	
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	
ddSMT	1	2	90.0	83.9	96.0	44	9	79	1412	782	2041	13	10	16	**SMT-LIB v2**
	2	95	94.7	68.2	99.9	92	0.1	1594	1499	2	3790	15	10	24	
	3	5	80.4	66.8	87.2	23	14	35	1533	1171	1764	11	10	12	
	4	53	99.8	99.3	99.9	57	1	246	431	13	1240	28	15	42	
	5	5	97.4	95.7	98.3	12	5	16	247	215	371	39	10	59	

Conclusion

ddSMT

- an input minimizer for the **SMT-LIB v2** format
- with support for **all** SMT-LIB v2 logics
- simplification strategies for
  - macros
  - command-level scopes
  - named annotations
- based on a **divide-and-conquer** delta debugging strategy
- especially effective in combination with **fuzz** testing

Future Work

- further simplification strategies for **annotations** (other than :named)
- **hybrid** approach: selective hierarchical delta debugging strategies
- comparison with model-based delta debugging on the **API** level

References

Robert Brummayer and Armin Biere.
Fuzzing and Delta-Debugging SMT Solvers.
In *Proc. 7th Intl. Workshop on Satisfiability Modulo Theories (SMT'09)*, page 5. ACM, 2009.

Koen Claessen and John Hughes.
Quickcheck: a lightweight tool for random testing of haskell programs.
In Martin Odersky and Philip Wadler, editors, *ICFP*, pages 268–279. ACM, 2000.

Ralf Hildebrandt and Andreas Zeller.
Simplifying failure-inducing input.
In *ISSTA*, pages 135–145, 2000.