Software Verification: Testing vs. Model Checking

A Comparative Evaluation of the State of the Art

Thomas Lemberger
Joint work with Dirk Beyer

LMU Munich, Germany

Null Hypothesis:

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Where's the numbers?

Overview

Terminology

- Testing:
  - Execute finite set of test cases on program
  - Observe compliance/violation of specification
  - Focus: Test-case generation

Terminology

- Testing:
  - Execute finite set of test cases on program
  - Observe compliance/violation of specification
  - Focus: Test-case generation
- Model checking:
  - Formally describe possible program states
  - Prove compliance/violation of specification
  - Abstraction important

Terminology

- Testing:
  - Execute finite set of test cases on program
  - Observe compliance/violation of specification
  - Focus: Test-case generation
- Model checking:
  - Formally describe possible program states
  - Prove compliance/violation of specification
  - Abstraction important
- Automated!

Scope

- Single, sequential programs
- Whitebox programs
- Task: bug finding

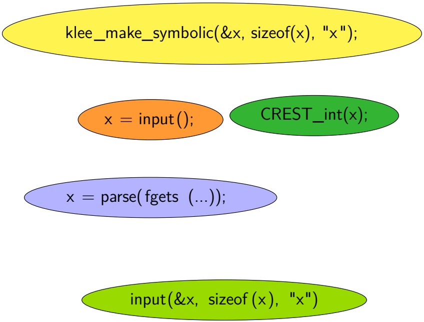
Comparability

Test-case generators

Comparability

Test-case generators

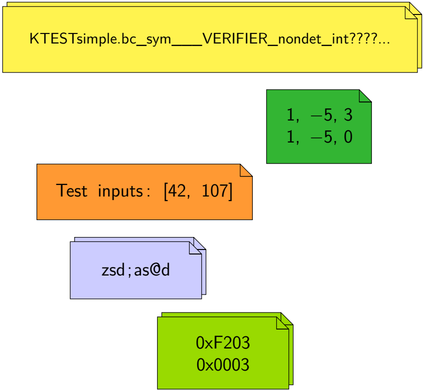
- Different conventions for program input



Comparability

Test-case generators

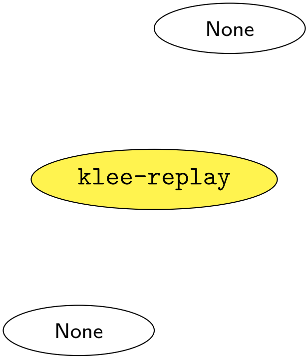
- Different conventions for program input
- Different output formats for test cases



Comparability

Test-case generators

- Different conventions for program input
- Different output formats for test cases
- Different/no test executors



Comparability

Model checkers

- Established standard for input programs

Comparability

Model checkers

- Established standard for input programs
- Established standard for output format of result

- 🔴 `FALSE`
- 🟠 `UNKNOWN`
- 🟢 `TRUE`

Comparability

Model checkers

- Established standard for input programs
- Established standard for output format of result

$\Rightarrow$ Adjust test-case generators to standards of model checkers

Framework

Framework: TBF

TBF: Test-based falsifier

- Apply test-case generators to model checker standards

Framework: TBF

TBF: Test-based falsifier

- Apply test-case generators to model checker standards
- Create, execute + observe tests

Framework: TBF

TBF: Test-based falsifier

- Apply test-case generators to model checker standards
- Create, execute + observe tests
- Only variable: Test-case generation tool

Framework: TBF

TBF: Test-based falsifier

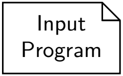
- Apply test-case generators to model checker standards
- Create, execute + observe tests
- Only variable: Test-case generation tool
- Specification: Never call __VERIFIER_error

Framework: TBF

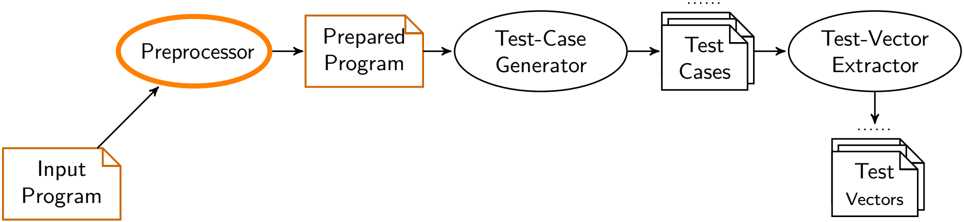
TBF: Test-based falsifier

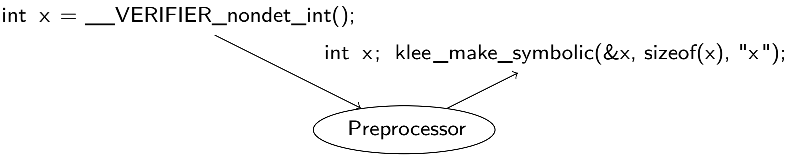
- Apply test-case generators to model checker standards
- Create, execute + observe tests
- Only variable: Test-case generation tool
- Specification: Never call __VERIFIER_error
- Disclaimer: Comparison of **tools**, not techniques

TBF Architecture

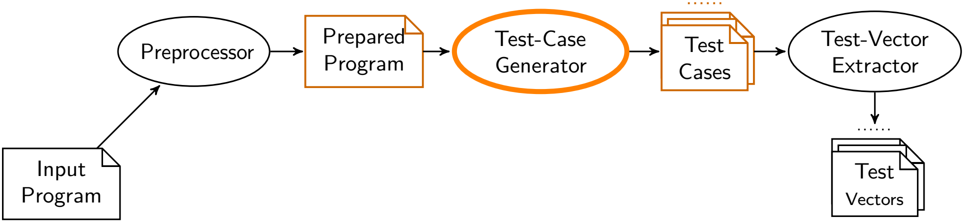


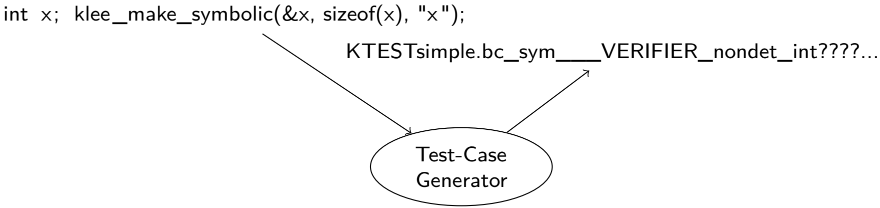
TBF Architecture



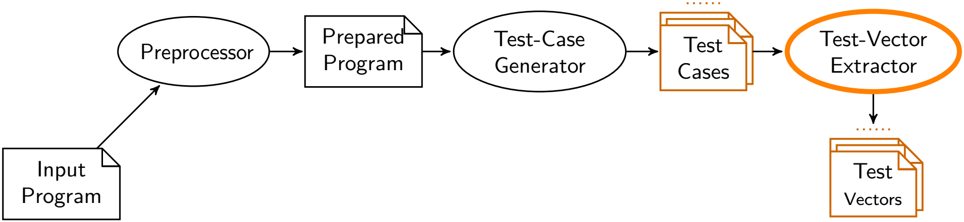


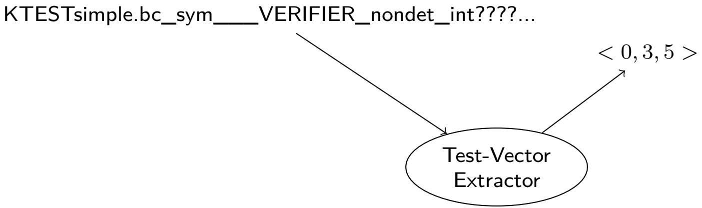
TBF Architecture



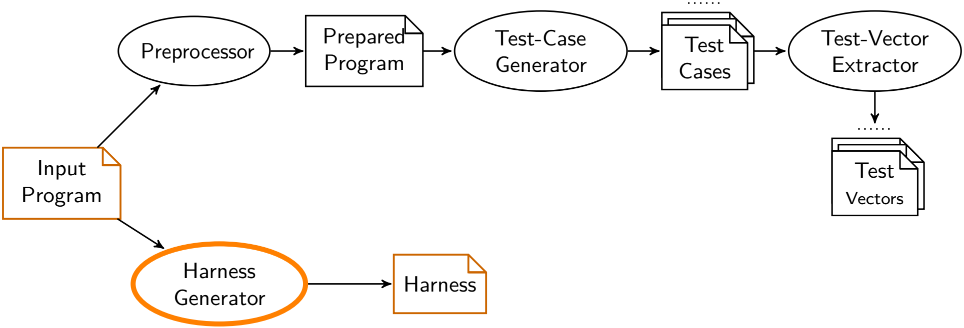


TBF Architecture

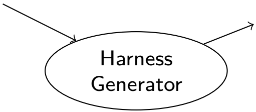




TBF Architecture

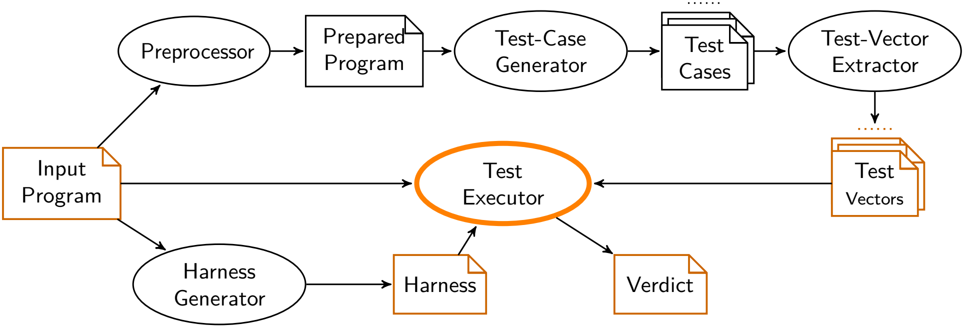



```
...
int x = __VERIFIER_nondet_int();
...
```




```
...
int __VERIFIER_nondet_int() {
  return (int) parse(input());
}
void __VERIFIER_error() {
  fprintf(stderr, "Err\n");
  exit(1);
}
```

TBF Architecture




```
for vec in test_vectors:
  stderr = run(prog, harness, vec)
  if "Err" in stderr:
    return FALSE
return UNKNOWN
```

Evaluation

Considered Tools

Tool	Technique
AFL-fuzz	Greybox fuzzing
Crest-ppc	Concolic execution, search-based
CPATiger	Model checking-based testing, based on CPAchecker
FShell	Model checking-based testing, based on Cbmc
Klee	Symbolic execution, search-based
PRtest	Random testing
Cbmc	Bounded model checking
CPA-Seq	Explicit-state, predicate abstraction, k-induction
Esbmc-incr	Bounded model checking, incremental loop bound
Esbmc-kInd	Bounded model checking, k-induction

Experiment Setup

- Benchmark tool: BenchExec
- Limits:
  - 2 CPUs
  - 15 GB of memory
  - 15 min CPU time
- Benchmark set
  - Openly available:
    `https://github.com/sosy-lab/sv-benchmarks`
  - Largest available benchmark set
  - C programs
  - 1490 tasks with known bug
  - 4203 tasks without bug

Experiments

1. Bug-finding capabilities: Consider 1490 tasks with bug
2. Precision: Consider 4203 tasks without bug
3. Validity: Comparison with existing Klee-replay

1. Bug-Finding Capabilities I

	No. Programs	AFL-fuzz^{T}	CPATiger^{T}	Crest-ppc^{T}	FShell^{T}	Klee^{T}	PRtest^{T}	CBMC^{M}	CPA-seq^{M}	ESBMC-incr^{M}	ESBMC-kInd^{M}	Union Testers	Union MC	Union All
Total Found	1490	605	57	376	236	**826**	292	830	889	**949**	844	887	1092	1176
Compilable	1115	605	57	376	236	**826**	292	779	819	**830**	761	887	930	1014
Median CPU Time (s)		11	4.5	**3.4**	6.2	3.6	3.6	**1.4**	15	1.9	2.3			

1. Bug-Finding Capabilities I

	No. Programs	AFL-fuzz^T	CPATiger^T	Crest-ppc^T	FShell^T	Klee^T	PRtest^T	CBMC^M	CPA-seq^M	ESBMC-incr^M	ESBMC-kInd^M	Union Testers	Union MC	Union All
Total Found	1490	605	57	376	236	**826**	292	830	889	**949**	844	887	1092	1176
Compilable	1115	605	57	376	236	**826**	292	779	819	**830**	761	887	930	1014
Median CPU Time (s)		11	4.5	**3.4**	6.2	3.6	3.6	**1.4**	15	1.9	2.3			

- Model checkers find more bugs

1. Bug-Finding Capabilities I

	No. Programs	AFL-fuzz^{T}	CPATiger^{T}	Crest-ppc^{T}	FShell^{T}	Klee^{T}	PRtest^{T}	CBMC^{M}	CPA-seq^{M}	ESBMC-incr^{M}	ESBMC-kInd^{M}	Union Testers	Union MC	Union All
Total Found	1490	605	57	376	236	**826**	292	830	889	**949**	844	887	1092	1176
Compilable	1115	605	57	376	236	**826**	292	779	819	**830**	761	887	930	1014
Median CPU Time (s)		11	4.5	**3.4**	6.2	3.6	3.6	**1.4**	15	1.9	2.3			

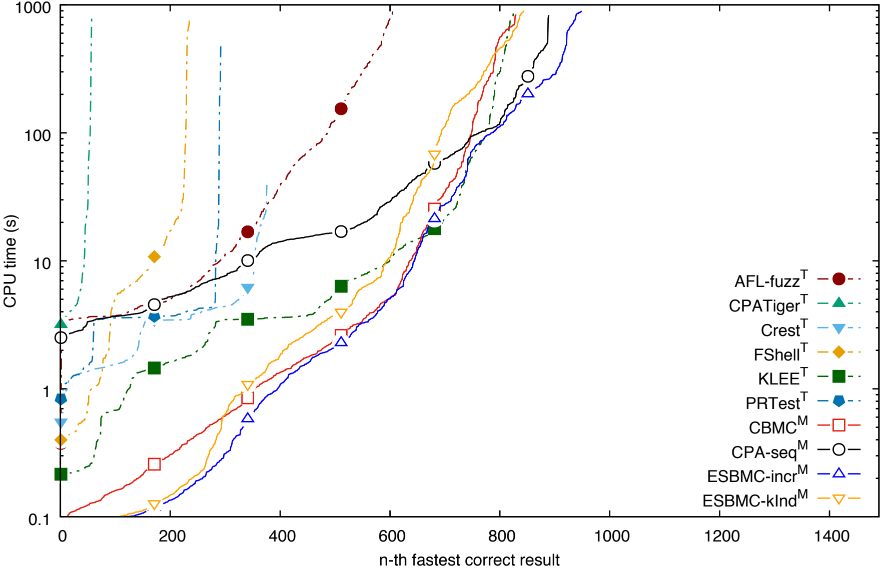
- Model checkers find more bugs
- Model checkers don't need stubs

1. Bug-Finding Capabilities I

	No. Programs	AFL-fuzz^T	CPATiger^T	Crest-ppc^T	FShell^T	Klee^T	PRtest^T	CBMC^M	CPA-seq^M	ESBMC-incr^M	ESBMC-kInd^M	Union Testers	Union MC	Union All
Total Found	1490	605	57	376	236	**826**	292	830	889	**949**	844	887	1092	1176
Compilable	1115	605	57	376	236	**826**	292	779	819	**830**	761	887	930	1014
Median CPU Time (s)		11	4.5	**3.4**	6.2	3.6	3.6	**1.4**	15	1.9	2.3			

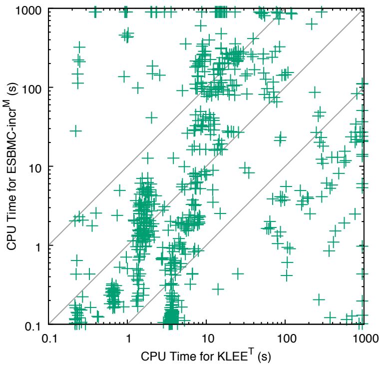
- Model checkers find more bugs
- Model checkers don't need stubs
- Model checkers are comparable in speed

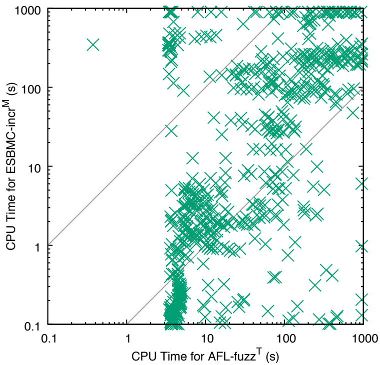
1. Bug-Finding Capabilities II



Time Performance

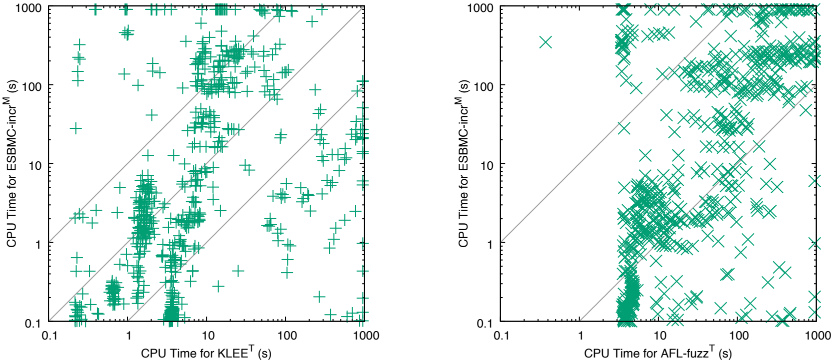
- CPU time of KLEE^T/AFL-FUZZ^T vs. ESBMC-INCR^M on solvable tasks





Time Performance

- CPU time of KLEE^{T}/AFL-FUZZ^{T} vs. ESBMC-INCR^{M} on solvable tasks



⇒ Time performance is task-specific

2. Precision

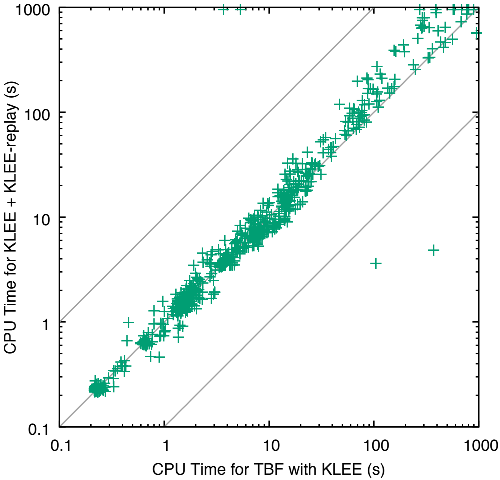
- 4203 tasks without bug
- Testers: No false alarms
- Model Checkers: Negligible

  Worst: ESBMC-INCR, 6 false alarms

3. Validity

Comparison of TBF with KLEE-REPLAY

- Specific to KLEE test case format
- Same concept as TBF
- Comparable performance



Conclusion I

- TBF:
  - makes 5 existing test-case generators **comparable**
  - allows **easy integration** of new generators
  - automatically transforms generated test cases to **executable tests**

Conclusion II

Can we confirm our null hypothesis?

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Conclusion II

Can we confirm our null hypothesis?

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Conclusion II

Can we confirm our null hypothesis?

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Conclusion II

Can we confirm our null hypothesis?

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Conclusion II

Can we confirm our null hypothesis?

- Testing is better at finding bugs than model checking.
- Testing is faster than model checking.
- Testing is more precise than model checking.
- Testing is easier to use than model checking.

Conclusion III

New null hypothesis:

- Model Checking
  - can **find more** bugs
  - in **less time**
  - requires **less adjustments** to input program

Consequence

$\Rightarrow$ Give us “better” benchmark tasks

$\Rightarrow$ Invest more time in development of testing tools

$\Rightarrow$ Use model checking (or symbolic execution)

Benchmark Resources

- Computing Resources:
  - Intel Xeon E3-1230 v5 CPU, 3.4 GHz, 8 CPUs each
  - 33 GB of memory
  - Ubuntu 16.04 with Linux 4.4

Benchmark Set: Programs with known bug

Category	Tasks	LOC					C features
		Sum	Min	Max	Avg	Median	
Arrays	40	1389	15	57	35	35	C arrays
BitVectors	14	2236	13	636	160	32	Bit vector arithmetics
ControlFlow	42	83 034	220	10 835	1977	1694	Complicated control flow
ECA	411	11 948 617	566	185 053	29 072	4827	Lots of (deep) branching
Floats	31	963	15	154	31	31	Floats (+ arithmetics)
Heap	66	50 430	19	4605	764	656	Heap structures
Loops	51	3989	14	1644	78	22	C loops
ProductLines	265	620 859	847	3789	2343	2951	Lots of branching
Recursive	45	1227	12	49	27	27	Use of recursion
Sequentialized	170	325 168	330	18 239	2126	1098	Sequentialized threading
LDV	355	6 116 255	1389	85 772	17 229	13 420	Linux device driver modules
Total	1490	19 154 167	12	185 053	12 855	2984	

Benchmark Set: Programs with no known bug

Category	Tasks	LOC					C features
		Sum	Min	Max	Avg	Median	
Arrays	95	4108	14	1161	43	30	C arrays
BitVectors	36	8275	15	696	320	47	Bit vector arithmetics
ControlFlow	52	100 841	94	22 300	1939	1057	Complicated control flow
ECA	738	17 737 301	344	185 053	24 034	2590	Lots of (deep) branching
Floats	142	46 536	9	1122	328	48	Floats (+ arithmetics)
Heap	107	86 519	11	4576	809	437	Heap structures
Loops	105	5781	14	476	55	25	C loops
ProductLines	332	539 446	838	3693	1625	979	Lots of branching
Recursive	53	1730	12	100	33	30	Use of recursion
Sequentialized	103	255 233	330	18 239	2478	1223	Sequentialized threading
LDV	2440	35 241 787	339	227 732	14 443	8664	Linux device driver modules
Total	4203	54 027 557	9	227 732	12 855	4055	

Discussion

- Use case for test-case generators:
  Create realiable test suite
- Use case for model checker:
  Prove program/entity safe
- “Does a test suite cover a bug?” directly correlates with test-suite quality
- 15 min should be enough time to cover bug in considered programs