Advanced Test Coverage Criteria: Specify and Measure, Cover and Unmask

Sébastien Bardin & Nikolai Kosmatov

joint work with

Omar Chebaro, Robin David, Mickaël Delahaye, Michaël Marcozzi, Mike Papadakis, Virgile Prevosto, etc.

CEA LIST
Software Safety & Security Lab
(Paris-Saclay, France)

Overview

Dynamic Symbolic Execution (DSE) is great ! [Klee also !]

- ✓ robust, no false alarm, scale
- ✗ **But ...**

Overview

Dynamic Symbolic Execution (DSE) is great ! [Klee also !]

- ✓ robust, no false alarm, scale
- ✗ **But ... no native support for coverage criteria**

Overview

Dynamic Symbolic Execution (DSE) is great ! [Klee also !]

- ✓ robust, no false alarm, scale
- ✗ **But ... no native support for coverage criteria**

DSE can be efficiently lifted to coverage-oriented testing

- unified view of coverage criteria [ICST 14, ICST 17]
- a dedicated variant DSE* [ICST 14]
- moreover : infeasibility detection is feasible [ICST 15, ICSE 18]

Prototype LTest (Frama-C plugin) [TAP 14]

- all-in-one toolkit for testing C programs
- combination of Frama-C and PathCrawler

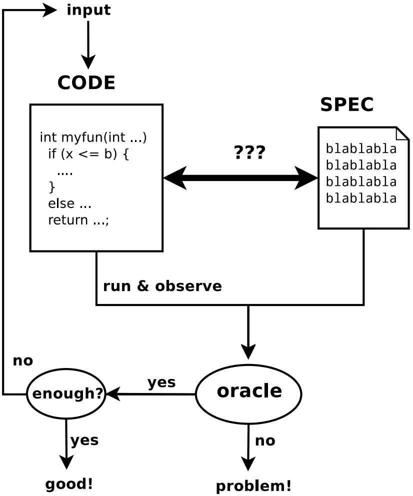
frama C Software Analyzers

PathCrawler

White-box software testing

Testing process

- Generate a test input
- Run it and check for errors
- Estimate coverage :
  if enough stop, else loop



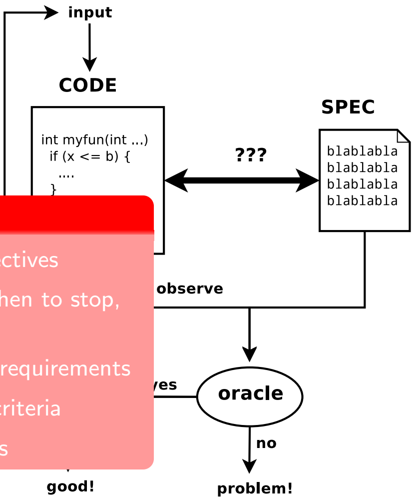
White-box software testing

Testing process

- Generate a test input
- Run it and check for errors
- Estimate coverage :
  if enough stop, else loop

Coverage criteria [decision, mcdc, etc.]

- systematic way of deriving test objectives
- major role : guide testing, decide when to stop, assess quality
- can be part of industrial normative requirements
- beware : lots of different coverage criteria
- beware : infeasible test requirements



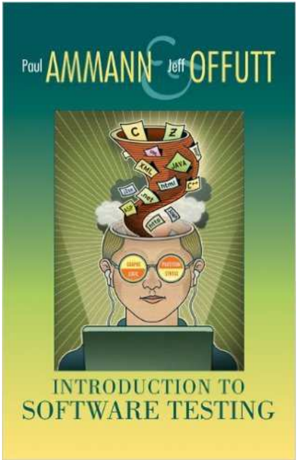
Variety and sophistication gap between literature and testing tools

Literature :

- 28 various white-box criteria in the Ammann & Offutt book

Tools :

- restricted to small subsets of criteria
- extension is complex and costly



Coverage criteria in white-box testing (2)

Another enemy : uncoverable test objectives

- waste generation effort, imprecise coverage ratios
- reason : structural coverage criteria are ... structural
- detecting uncoverable test objectives is undecidable

Recognized as a hard and important issue in testing

- no practical solution
- not so much work (compared to test gen.)
- **real pain** (e.g. aeronautics, mutation testing)

Goals and Challenges

Extend DSE to advanced coverage criteria

- in an efficient way
- in a unified way

Goals and Challenges

Extend DSE to advanced coverage criteria

- in an efficient way
- in a unified way

Not easy ! [Active Testing, Augmented DSE, Mutation DSE]

- limited or unclear expressiveness
- explosion of the search space [APEX : 272x avg, up to 2,000x]

Goals and Challenges

Let's raise the bar : full automation for advanced coverage criteria

- specify the coverage objective (+ unified treatment)
- measure coverage of test suites
- cover the objectives in an efficient manner (DSE)
- unmask the infeasible or redundant objectives

Goals and Challenges

Let's raise the bar : full automation for advanced coverage criteria

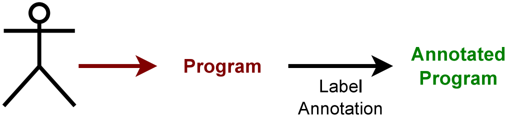
- specify the coverage objective (+ unified treatment)
  - labels, a simple specification mechanism
- measure coverage of test suites
  - thx to labels
- cover the objectives in an efficient manner (DSE)
  - DSE*, a variation of DSE
- unmask the infeasible or redundant objectives
  - an original combination of existing static analyses

The LTest plugin

LTest : All-in-one automated testing toolkit for C

- plugin of the Frama-C verification platform (open-source)
- based on PathCrawler for test generation
- the plugin itself is open-source except test generation

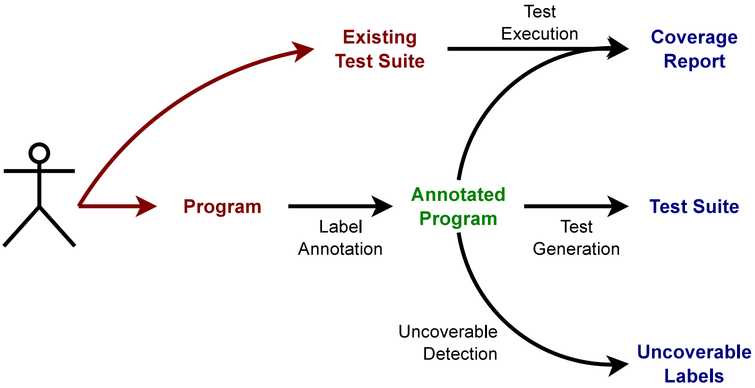
Software Analyzers



Supported criteria

- **DC, CC, MCC, GACC**
- **FC, IDC, WM**

- managed in a unified way
- rather easy to add new ones



Supported criteria

- **DC, CC, MCC, GACC**
- **FC, IDC, WM**

- managed in a unified way
- rather easy to add new ones

Specify : Labels

- Annotate programs with **labels**
  - predicate attached to a specific program instruction
- Label (loc, φ) is covered if a test execution
  - reaches the instruction at *loc*
  - satisfies the predicate φ
- **Good for us**
  - can easily encode a large class of coverage criteria
  - in the scope of standard program analysis techniques

Simulation of standard coverage criteria

```
statement_1;
if (x==y && a<b)
   {...};
statement_3;
```

→

```
statement_1;
// l1: x==y
// l2: !(x==y)
// l3: a<b
// l4: !(a<b)
if (x==y && a<b)
   {...};
statement_3;
```

Condition Coverage (**CC**)

Simulation of standard coverage criteria

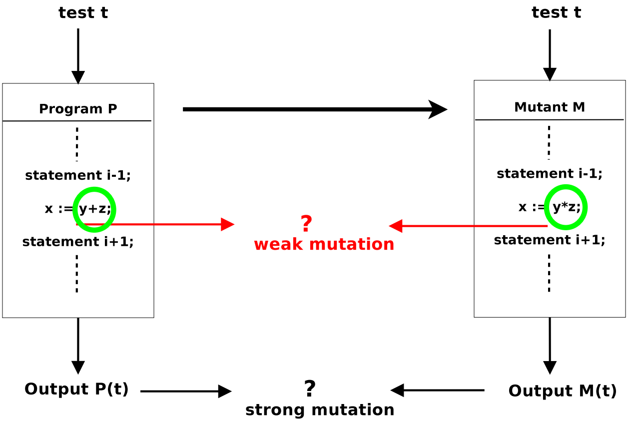
```
statement_1;
if (x==y && a<b)
   {...};
statement_3;
```

→

```
statement_1;
// l1: x==y && a<b
// l2: x==y && a>=b
// l3: x!=y && a<b
// l4: x!=y && a>=b
if (x==y && a<b)
   {...};
statement_3;
```

Multiple-Condition Coverage (**MCC**)

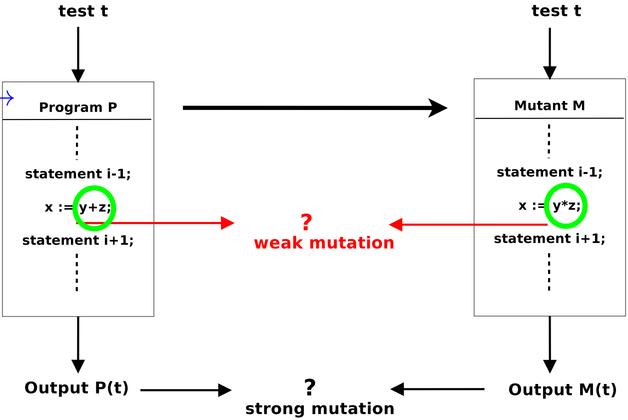
And also Weak Mutations (WM)



And also Weak Mutations (WM)

To simulate by labels, insert before the mutated instruction :

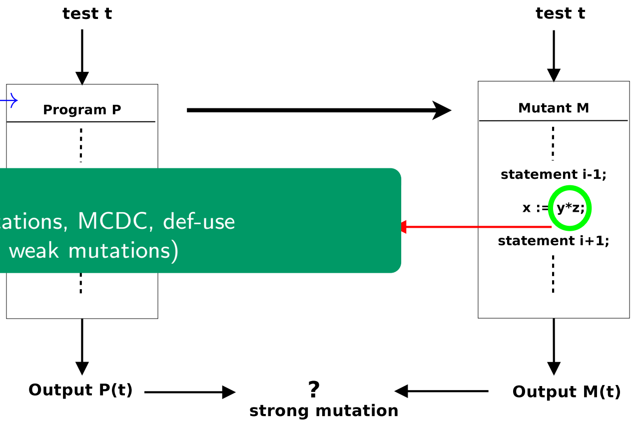
`// l1: y+z != y*z` →



And also Weak Mutations (WM)

To simulate by labels, insert before the mutated instruction :

`// l1: y+z != y*z` →

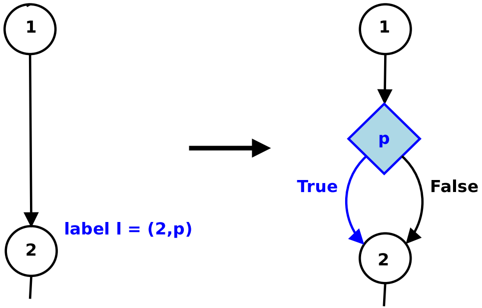


Out of scope :

. strong mutations, MCDC, def-use

. (side-effect weak mutations)

Cover : DSE & direct instrumentation



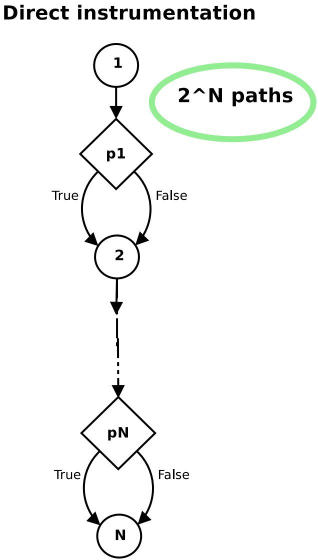
Covering label l ⇔ Covering branch `True`

- ✓ sound & complete instrumentation
- ✗ complexification of the search space [#paths, shape of paths]
- ✗ dramatic overhead [theory & practice] [Apex : avg 272x, max 2000x]

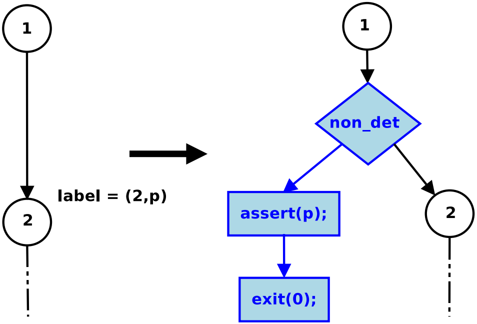
Direct instrumentation is not good enough

Non-tightness 1

- ✗ P' has exponentially more paths than P



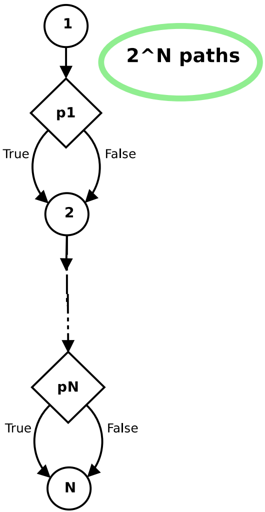
DSE⋆ : Tight Instrumentation



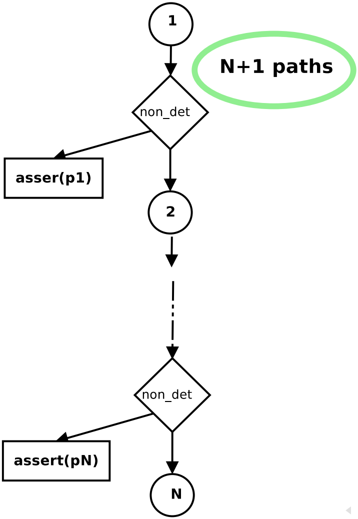
Covering label l ⇔ Covering `exit(0)`

- ✓ sound & complete instrumentation
- ✓ no complexification of the search space

Direct instrumentation



Tight Instrumentation



Experiments

Benchmark : Standard benchmarks [Siemens, Verisec, Mediabench]

- 12 programs (50-300 loc), 3 criteria (**CC**, **MCC**, **WM**)
- 26 pairs (program, coverage criterion)
- 1,270 test requirements

Performance overhead

	DSE	DSE'	DSE*
Min	×1	×1.02	×0.49
Median	×1	×1.79	×1.37
Max	×1	×**122.50**	×**7.15**
Mean	×1	×**20.29**	×**2.15**
Timeouts	0	**5** *	**0**

* : TO are discarded for overhead computation
cherry picking : 94s vs TO [1h30]

Experiments

Benchmark : Standard benchmarks [Siemens, Verisec, Mediabench]

- 12 programs (50-300 loc), 3 criteria (**CC**, **MCC**, **WM**)
- 26 pairs (program, coverage criterion)
- 1,270 test requirements

Coverage

	Random	DSE	DSE*
Min	37%	61%	62%
Median	63%	90%	**95%**
Max	100%	100%	100%
Mean	70%	87%	**90%**

vs DSE : +39% coverage on some examples

Experiments

Benchmark : Standard benchmarks [Siemens, Verisec, Mediabench]

- 12 programs (50-300 loc), 3 criteria (**CC**, **MCC**, **WM**)
- 26 pairs (program, coverage criterion)
- 1,270 test requirements

DSE*

- DSE* significantly outperforms DSE'
- overhead kept reasonable
- better coverage than DSE

Unmask infeasibility

Automatic detection of uncoverable test objectives

- a *sound* method
- applicable to a large class of coverage criteria
- strong detection power, reasonable speed
- rely as much as possible on existing verification methods :

Observation :

Label (loc, p) is uncoverable $\Leftrightarrow$ Assertion `assert (`$\neg p$`);` at location *loc* is valid

Unmask infeasibility

Automatic detection of uncoverable test objectives

- a *sound* method
- applicable to a large class of coverage criteria
- strong detection power, reasonable speed
- rely as much as possible on existing verification methods :

Observation :

Label (loc, p) is uncoverable $\Leftrightarrow$ Assertion `assert (`$\neg p$`);` at location *loc* is valid

Rely on a combination of

- abstract interpretation (infer context, not precise)
- weakest precondition (context-blind, locally precise)

VA and WP may fail

```
int main() {
  int a = nondet(0 .. 20);
  int x = nondet(0 .. 1000);
  return g(x,a);
}

int g(int x, int a) {


  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
 //l1: res == 0
}
```

VA and WP may fail

```
int main() {
  int a = nondet(0 .. 20);
  int x = nondet(0 .. 1000);
  return g(x,a);
}

int g(int x, int a) {


  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
 //@assert res != 0
}
```

VA and WP may fail

```
int main() {
  int a = nondet(0 .. 20);
  int x = nondet(0 .. 1000);
  return g(x,a);
}

int g(int x, int a) {


  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
 //@assert res != 0      // both VA and WP fail
}
```

Combine them! VA⊕WP succeeds!

```
int main() {
  int a = nondet(0 .. 20);
  int x = nondet(0 .. 1000);
  return g(x,a);
}

int g(int x, int a) {
 //@assume 0 <= a <= 20
 //@assume 0 <= x <= 1000
  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
 //@assert res != 0
}
```

Combine them! VA⊕WP succeeds!

```
int main() {
  int a = nondet(0 .. 20);
  int x = nondet(0 .. 1000);
  return g(x,a);
}

int g(int x, int a) {
 //@assume 0 <= a <= 20
 //@assume 0 <= x <= 1000
  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
 //@assert res != 0     // VA ⊕ WP succeeds
}
```

Detection power

Reuse the same benchmarks [Siemens, Verisec, Mediabench]

- 1,270 test requirements, 121 infeasible ones

	#Lab	#Inf	VA		WP		VA ⊕ WP	
			#d	%d	#d	%d	#d	%d
Total	1,270	121	84	69%	73	**60%**	118	**98%**
Min		0	0	0%	0	0%	2	**67%**
Max		29	29	100%	15	100%	29	100%
Mean		4.7	3.2	63%	2.8	**82%**	4.5	**95%**

#d : number of detected infeasible labels

%d : ratio of detected infeasible labels

Detection power

Reuse the same benchmarks [Siemens, Verisec, Mediabench]

- 1,270 test requirements, 121 infeasible ones

	#Lab	#Inf	VA		WP		VA ⊕ WP	
			#d	%d	#d	%d	#d	%d
Total	1,270	121	84	69%	73	**60%**	118	**98%**
Min		0	0	0%	0	0%	2	**67%**
Max		29	29	100%	15	100%	29	100%
Mean		4.7	3.2	63%	2.8	**82%**	4.5	**95%**

#d : number of detected infeasible labels

%d : ratio of detected infeasible labels

- VA ⊕ WP achieves almost perfect detection
- detection speed is reasonable [$\leq$ 1s/obj.]

Impact on test generation

report more accurate coverage ratio

	Coverage ratio reported by DSE*		
Detection method	None	VA ⊕WP	Perfect*
Total	90.5%	99.2%	100.0%
Min	61.54%	91.7%	100.0%
Max	100.00%	100.0%	100.0%
Mean	91.10%	99.2%	100.0%

* preliminary, manual detection of infeasible labels

[new] Detect polluting objectives @scale

More recent work [Marcozzi et al. ICSE 2018]

- other sources of "pollution" :
  - duplicate and/or subsumed test objectives
  - harmful effect
- detection technique :
  - WP-based dedicated algorithms
  - enhanced with multi-core and fine tuning
- achievements :
  - detecting a large number of polluting test objectives (up to 27% of the total number of objectives)
  - scales : SQLite (200 kloc, 90k objectives, 9h, 15% identified as polluting)

[new] Limitations of labels

- Labels encode only criteria whose objectives are reachability constraints
- Typical examples of criteria above labels:

Call Coverage

```
int f() {
if (...) { /* loc_1 */ g(); }
if (...) { /* loc_2 */ g(); }}
```

→ cover loc_1 or loc_2

All-defs

```
/* loc_1 */ a := x;
if (...) /* loc_2 */ res := x+1;
else /* loc_3 */ res := x-1;
```

→ Cover path loc_1 to loc_2
or path loc_1 to loc_3

MCDC

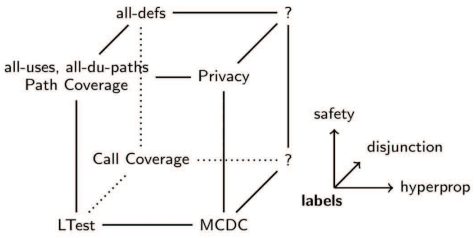
```
statement_0;
// loc_1
if (x==y && a<b) {...};
statement_2;
```

→ Cover if condition twice in a correlated way:
- a<b stays identical
- x==y and (x==y && a<b) change

DISJUNCTION **SAFETY** **HYPERPROPERTIES**

HTOL

- extend labels along the three axes (hyperlabels)
    - $l \triangleright \{v \mapsto ...\}$, $< h|\phi >$, $h \cdot h'$, $h + h'$, $h \rightarrow h'$
- give a formal semantic
- start extending LTest
    - generic coverage measurement technique
    - cover and unmask need update



Conclusion

Dynamic Symbolic Execution is great !

- ✓ robust, no false alarm, scale
- ✓ **can be efficiently lifted to coverage-oriented testing**

Advanced test criteria can be fruitfully automated !

- ✓ specify
- ✓ measure
- ✓ cover
- ✓ unmask