Automated White-Box Testing Beyond Branch Coverage

Sébastien Bardin, Nikolai Kosmatov, Mickaël Delahaye

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

The problem

DSE is GREAT for automating structural testing

- ✓ very powerful approach to (white box) test generation
- ✓ many tools and many successful case-studies since mid 2000's

The problem

DSE is GREAT for automating structural testing

✓ very powerful approach to (white box) test generation
✓ many tools and many successful case-studies since mid 2000's

Yet, no real support for many structural coverage criteria
[except branch coverage]

Would be useful :

- when required to produce tests achieving some criterion
- for producing "good" tests for an external oracle
  [functional correctness, security, performance, etc.]

Our goals

- extend DSE to a large class of structural coverage criteria
  - recent efforts in this direction through instrumentation
  - but exponential explosion of the search space
- support these criteria in a unified way
- bonus : what about infeasible requirement detection ?

Outline

- Introduction
- Labels
- A label-based automated testing framework [TAP 14]
- Efficient DSE for Labels [ICST 14]
- Infeasible label detection [ongoing work]
- Conclusion

Labels

A well-defined specification mechanism for coverage criteria

- based on predicates, can easily encode a large class of criteria
- in the scope of standard program analysis techniques

Given a program P, a label l is a pair (loc, φ), where :

- φ is well-defined in P at location loc
- φ contains no side-effect expression

Basic definitions

- an annotated program is a pair $\langle P, L \rangle$, with L set of labels
- a test datum t covers l if $P(t)$ reaches loc and satisfies φ

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

Decision Coverage (**DC**)

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**)

Simulation of standard coverage criteria

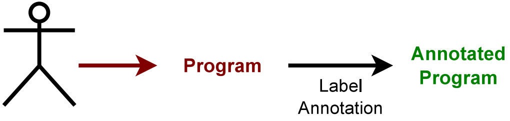
OBJ : generic specification mechanism for coverage criteria ✓

- **IC**, **DC**, **FC**, **CC**, **MCC**, **GACC**
- large part of Weak Mutations
- Input Domain Partition
- Run-Time Error

Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion

LTEST overview [TAP 14]



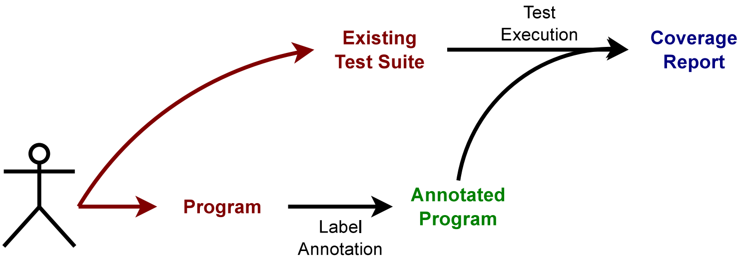
Supported criteria

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

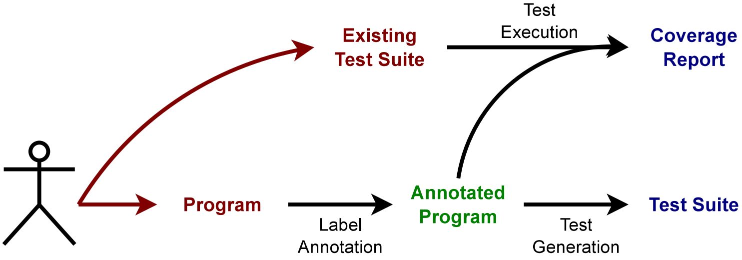
Criteria encoded with labels

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

LTEST overview [TAP 14]



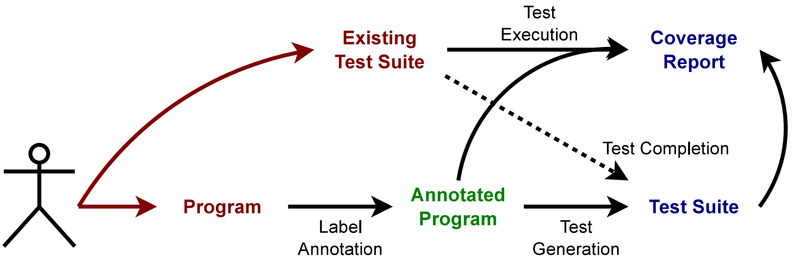
LTEST overview [TAP 14]



DSE⋆ procedure

- DSE with native support for labels
- extension of PATHCRAWLER

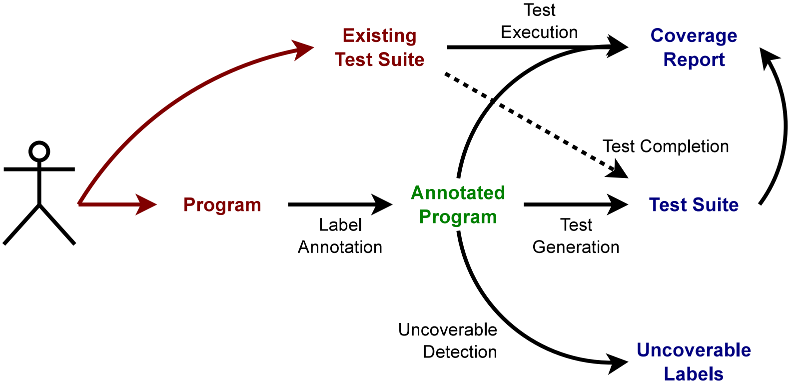
LTEST overview [TAP 14]



Service cooperation

- share label statuses
- `Covered`, `Infeasible`, ?

LTEST overview [TAP 14]



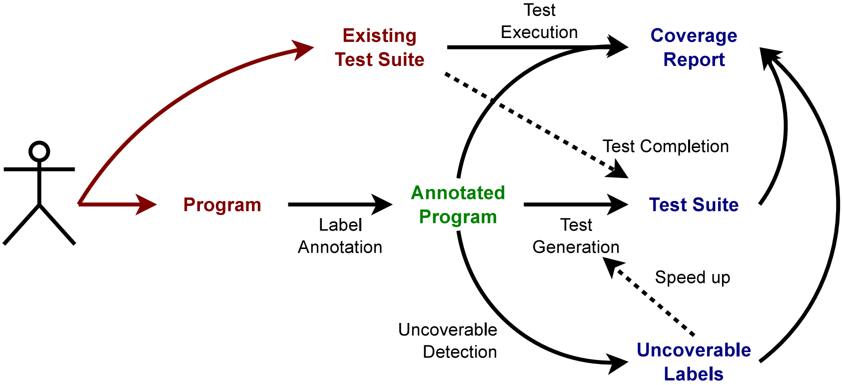
Reuse static analyzers from FRAMA-C

- sound detection !
- several modes : VA, WP, VA $\otimes$ WP

Service cooperation

- share label statuses
- `Covered`, `Infeasible`, ?

LTEST overview [TAP 14]

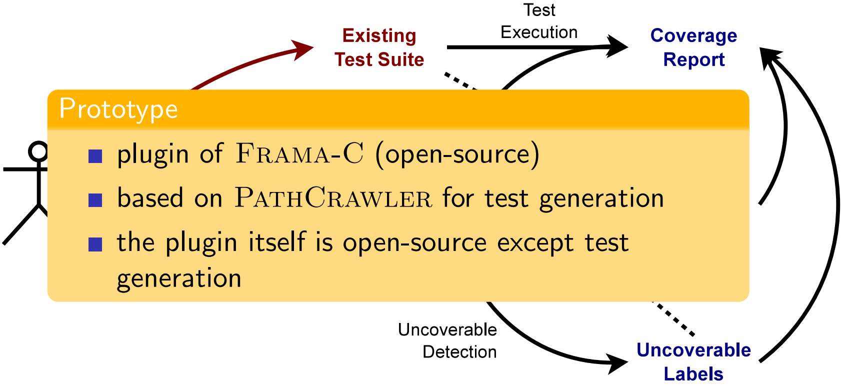


Reuse static analyzers from FRAMA-C

- sound detection !
- several modes : VA, WP, VA $\otimes$ WP

Service cooperation

- share label statuses
- `Covered`, `Infeasible`, ?



Prototype

- plugin of FRAMA-C (open-source)
- based on PATHCRAWLER for test generation
- the plugin itself is open-source except test generation

Supported criteria

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

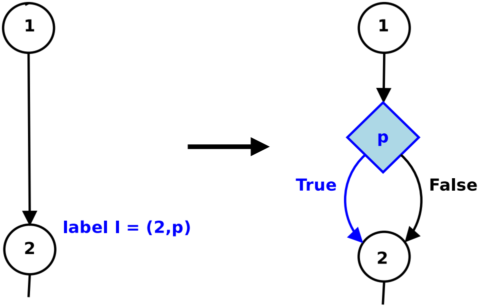
Service cooperation

- share label statuses
- `Covered`, `Infeasible`, `?`

Outline

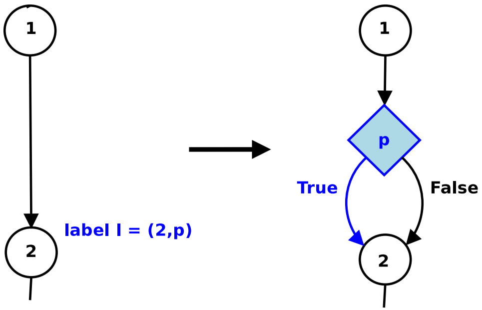
- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion

Direct instrumentation



Covering label l ⇔ Covering branch `True`

Direct instrumentation



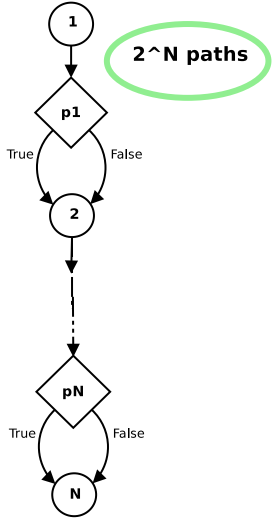
Covering label l ⇔ Covering branch `True`

✓ sound & complete instrumentation

✗ dramatic overhead [theory & practice]

Direct instrumentation is not good enough

Direct instrumentation



Non-tightness 1

- × P' has exponentially more paths than P

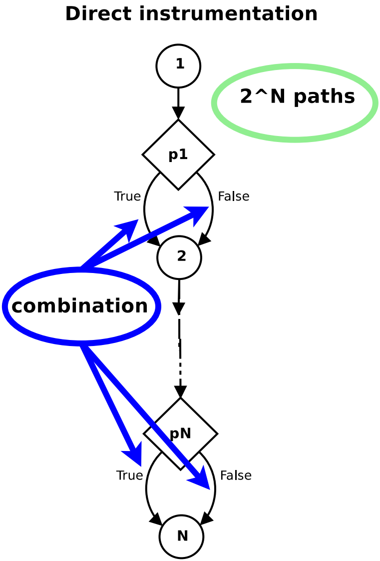
Direct instrumentation is not good enough

Non-tightness 1

- ✗ P' has exponentially more paths than P

Non-tightness 2

- ✗ Paths in P' too complex
  - at each label, require to cover p or to cover $\neg p$
  - π' covers up to N labels

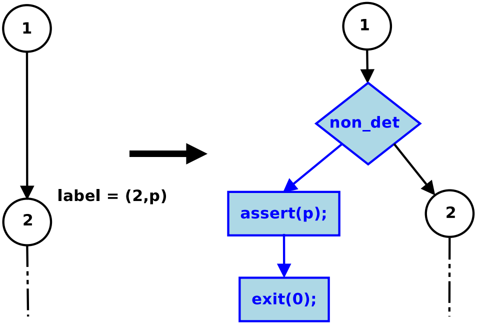


Our approach

The DSE* algorithm [ICST 14]

- Tight instrumentation $P^\star$: totally prevents “complexification”
- Iterative Label Deletion : discards some redundant paths
- Both techniques can be implemented in black-box

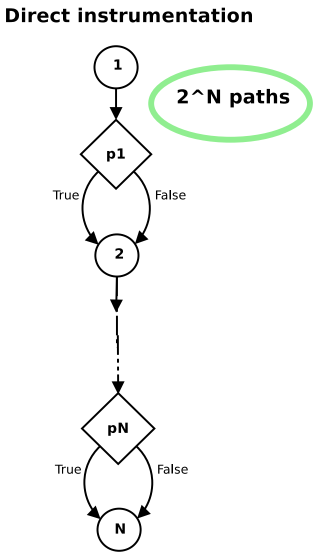
DSE* : Tight Instrumentation

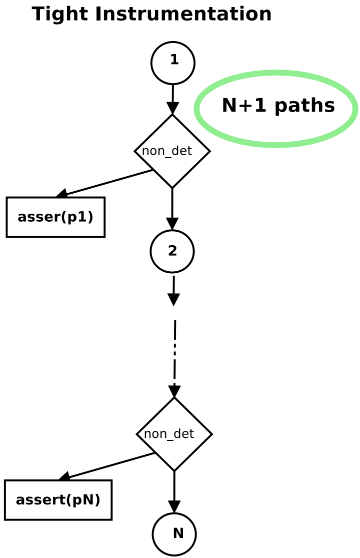


Covering label l $\Leftrightarrow$ Covering `exit(0)`

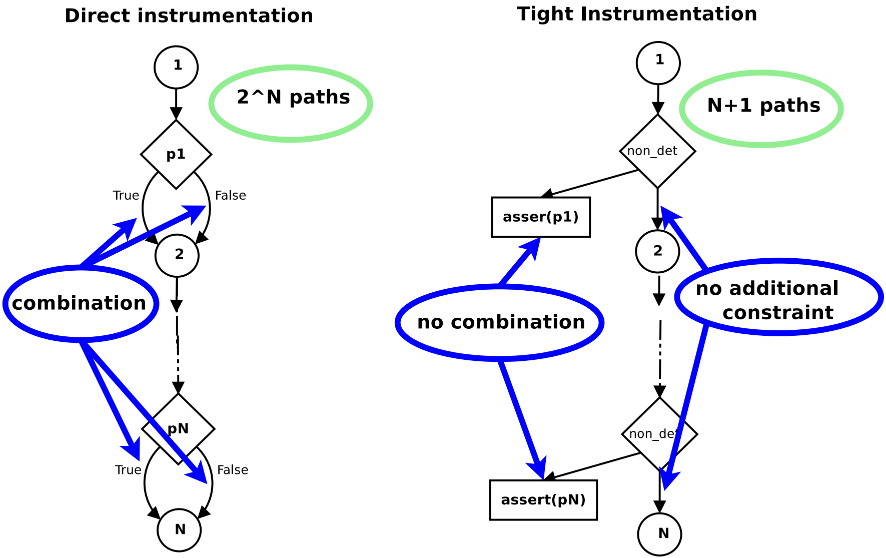
- ✓ sound & complete instrumentation
- ✓ no complexification of the search space

DSE* : Tight Instrumentation (2)





DSE⋆ : Tight Instrumentation (2)



DSE* : Iterative Label Deletion

Observations

- we need to cover each label only once
- yet, DSE explores paths of $P^\star$ ending in already-covered labels
- burden DSE with "useless" paths w.r.t. label coverage

Solution : Iterative Label Deletion

- keep a cover status for each label
- symbolic execution ignores paths ending in covered labels
- dynamic execution updates cover status [truly requires DSE]

Iterative Label Deletion is relatively complete w.r.t. label coverage

Experiments (1)

Goal of experiments

- evaluate DSE* versus DSE'
- evaluate overhead of handling labels

Benchmark programs

- 12 programs taken from standard benchmarks (Siemens, Verisec, MediaBench) [beware : small programs]
- 3 coverage criteria : **CC**, **MCC**, **WM**
  [uncoverable labels not discarded]

			DSE	DSE'	DSE*
utf8-5 108 loc	wm 84 *l*	#paths time cover	680 2s	**11,111** 40s 82/84	**743** 8.1s 82/84
utf8-7 108 loc	wm 84 *l*	#paths time cover	3,069 5.8s	**81,133** **576s** 82/84	**3,265** **35s** 82/84
tcas 124 loc	wm 111 *l*	#paths time cover	4,420 5.6s	**300,213** **662s** 101/111	**6,014** **27s** 101/111
replace 100 loc	wm 79 *l*	#paths time cover	866 2s	**87,498** **245s** 70/79	**2,347** **14s** 70/79
get_tag-6 240 loc	cc 20 *l*	#paths time cover	76,456 3,011s	**TO**	76,468 **1,512s** 20/20
	wm 47 *l*	#paths time cover	76,456 3,011s	**TO**	76,481 **1,463s** 44/47
gd-5 319 loc	wm 63 *l*	#paths time cover	14,516 50s	**TO**	14,607 **94s** 62/63
gd-6 319 loc	wm 63 *l*	#paths time cover	107,410 3,740s	**TO**	107,521 **2,232s** 63/63

Experiments (3)

- DSE' : 4 TO, max overhead 122x [excluding TO]
- DSE* : no TO, max overhead 7x, average : 2.4x
- cherry picking : 94s vs TO [1h30]

Experiments (3)

- DSE' : 4 TO, max overhead 122x [excluding TO]
- DSE* : no TO, max overhead 7x, average : 2.4x
- cherry picking : 94s vs TO [1h30]

Conclusion

- DSE* performs significantly better than DSE'
- The overhead of handling labels is kept reasonable
- still room for improvement

Experiments (3)

- DSE' : 4 TO, max overhead 122x [excluding TO]
- DSE* : no TO, max overhead 7x, average : 2.4x
- cherry picking : 94s vs TO [1h30]

Conclusion

- DSE* performs significantly better than DSE'
- The overhead of handling labels is kept reasonable
- still room for improvement
- also : high coverage [min : 61%, max : 100%, mean : 91% - see after]
- also : DSE* covers more than DSE [0%-39%]

Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion

Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]

WP : Weakest Precondition [highly scalable wrt prog. size]

VA $\oplus$ WP : combination

Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA $\oplus$ WP : combination

Detection power

	#Lab	#Inf	VA		WP		VA $\oplus$ 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

(speed : $\leq 1s$ per label)

Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA ⊕ WP : combination

Improving DSE* : reported coverage ratio

	Coverage ratio reported by DSE*			
Detection Method	**None**	**VA**	**WP**	**VA ⊕ WP**
Total	**90%**	97%	96%	**99%**
Min **Max**	**61%** 100%	80% 100%	67% 100%	**91%** 100%
Mean	**91%**	96%	97%	**99%**

Infeasibility detection [ongoing work]

VA : Value-Analysis, abstract interpretation [beware of prog. size]
WP : Weakest Precondition [highly scalable wrt prog. size]
VA $\oplus$ WP : combination

Improving DSE* : speed-up

		VA	WP	VA $\oplus$ WP
		Speedup	**Speedup**	**Speedup**
RT(1s) +LUncov + DSE*	**Total**	**2.4x**	**2.2x**	**2.2x**
	Min	0.5x	0.1x	0.1x
	Max	**107.0x**	**74.1x**	**55.4x**
	Mean	**7.5x**	**5.1x**	**3.8x**

RT : random testing
Speedup wrt DSE* alone

Outline

- Introduction
- Labels
- A label-based automated testing framework
- Efficient DSE for Labels
- Infeasible label detection
- Conclusion

Conclusion

LTest : an all-in-one toolkit for whitebox testing of C programs

- ✓ a well-defined and expressive specification mechanism for coverage criteria
- ✓ an efficient integration into DSE
- ✓ a sound and efficient infeasibility detection