FROM RESEARCH TO INDUSTRY

Sound and Quasi-Complete Detection of Infeasible Test Requirements

Robin David
Sébastien Bardin
Mickaël Delahaye
Nickolaï Kosmatov

— 30 juillet 2015

Outline

Context

Testing process

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

Coverage criteria [decision, mcdc, mutants, etc.] play a major role

- generate tests, decide when to stop, assess quality of testing
- definition : systematic way of deriving test requirements

Context

Testing process

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

Coverage cr

- generat
- definiti

The enemy : Infeasible test requirements

- waste generation effort, imprecise coverage ratios
- cause : structural coverage criteria are ... structural
- detecting infeasible test requirements is undecidable

→ Recognized as a hard and important issue in testing

Context

Testing process

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

Coverage criteria [decision, mcdc, mutants, etc.] play a major role

- generate tests, decide when to stop, assess quality of testing
- definition : systematic way of deriving test requirements

Testing oriented *but* scope beyond that :
$\rightarrow$ **original combination of two formal methods**

Our goals and results

→ Focus on white-box *(structural)* coverage criteria

Goals : automatic detection of infeasible test requirements

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

Our goals and results

→ Focus on white-box *(structural)* coverage criteria

Goals : automatic detection of infeasible test requirements

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

Results

- automatic, sound and generic method ✓
- **new combination of existing verification technologies** ✓
- experimental results : strong detection power [95%], reasonable detection speed [$\leq$ 1s/obj.], improve test generation ✓
- yet to be proved : scalability on large programs ?

  [promising results..]

Our goals and results

→ Focus on white-box *(structural)* coverage criteria

Goals : automatic detection of infeasible test requirements

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

Results

- automatic, sound and generic method ✓
- **new combination of existing verification technologies** ✓
- experimental results : strong detect
  detection speed [$\leq$ 1s/obj.], improve
- yet to be proved : scalability on lar
  [promising results..]

Take away

- VA $\oplus$ WP
- better than VA, WP
- plug-in Frama-C

Outline

Background : Labels

- Annotate programs with **labels** [ICST 2014]
  - 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 [see after]
  - in the scope of standard program analysis techniques

Background : Labels

- Annotate programs with **labels** [ICST 2014]
  - 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 [see after]
  - in the scope of standard program analysis techniques
  - infeasible label $(loc, \varphi) \Leftrightarrow$ valid assertion $(loc, \texttt{assert}\neg\varphi)$

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

Infeasible labels, valid assertions

```
int g(int x, int a) {
  int res;
  if(x+a >= x)
    res = 1;
  else
    res = 0;
  //@assert res ≠ 0      // valid
}
```

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)**

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

Condition Coverage **(CC)**

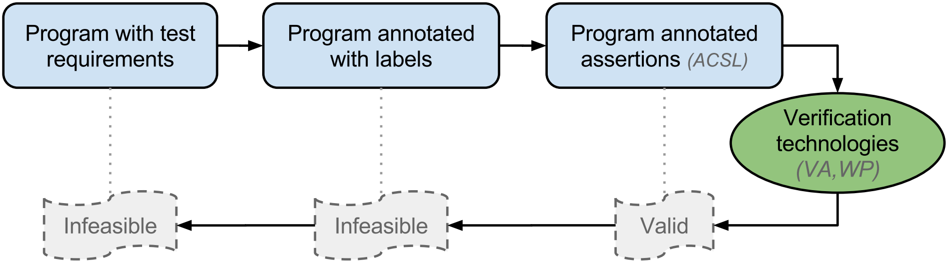
```
statement_1;
//l1: x==y && a<b
//l2: x==y && a>=b
//l3: x!=y && a<b
//l4: x!=y && a>=b
if (x==y && a<b)
    {...};
statement_3;
```

Multiple-Condition Coverage **(MCC)**

Also Weak Mutation, GACC *(weak MCDC)* etc.

Overview of the approach

- labels as a unifying criteria
- label infeasibility ⇔ assertion validity
- s-o-t-a verification for assertion checking



Outline

Checking assertion validity

Two broad categories of sound assertion checkers

- **Value Analysis :** state-approximation
  - compute an invariant of the program
  - then, analyze all assertions (labels) in one run
- **Weakest-Precondition calculus :** Goal-oriented checking
  - perform a dedicated check for each assertion
  - a single check usually easier, but many of them

	VA	WP
sound for assert validity	✓	✓
blackbox reuse	✓	✓
local precision	✗	✓
calling context	✓	✗
calls / loop effects	✓	✗
global precision	✗	✗
scalability wrt. #labels	✓	✓ (orange)
scalability wrt. code size	✗	✓

hypothesis : VA is interprocedural

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
}
```

Proposal : VA $\oplus$ WP (1)

Goal = get the best of the two worlds

- idea : VA passes to WP the global info. it lacks

Which information, and how to transfer it ?

- VA computes (internally) some form of invariants
- WP naturally takes into account assumptions `//@ assume`

→ **Solution : VA exports its invariants on the form of WP-assumptions** *(Frama-C→ACSL)*

Proposal : VA $\oplus$ WP (1)

Goal = get the best of the two worlds

- idea : VA passes to WP the global info. it lacks

Which information, and how to transfer it ?

- VA computes (internally) some form of invariants
- WP naturally takes into account assumptions `//@ assume`

→ **Solution : VA exports its invariants on the form of WP-assumptions** *(Frama-C→ACSL)*

Notes : **No** manually-inserted WP-assumption

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) {


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

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!

```
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
}
```

Proposal : VA $\oplus$ WP (2)

Exported invariants

- only names appearing in program
  - independent from memory size
- non-relational information
  - linear in VA
- only numerical information
  - sets, intervals, congruence

Proposal : VA $\oplus$ WP (2)

Soundness ok as long as VA is sound

Exhaustivity of "export" only affect deductive power

- Finding the right trade-off
- in practice : exhaustive export has very low overhead

Invariant export strategies

```
int fun(int a, int b, int c) {
  //@assume a [...]
  //@assume b [...]
  //@assume c [...]
  int x=c;



  //@assert a < b
  if(a < b)
    {...}
  else
    {...}
}
```

Parameters annotations

Invariant export strategies


```
int fun(int a, int b, int c) {



  int x=c;

  //@assume a [...]
  //@assume b [...]
  //@assert a < b
  if(a < b)
    {...}
  else
    {...}
}
```

Label annotations

Invariant export strategies

```
int fun(int a, int b, int c) {
  //@assume a [...]
  //@assume b [...]
  //@assume c [...]
  int x=c;
  //@assume x [...]
  //@assume a [...]
  //@assume b [...]
  //@assert a < b
  if(a < b)
    {...}
  else
    {...}
}
```

Complete annotations

Invariant export strategies

```
int fun(int a, int b, int c) {
  //@assume a [...]
  //@assume b [...]
  //@assume c [...]
  int x=c;
  //@assume x [...]
  //@assume a [...]
  //@assume b [...]
  //@assert a < b
  if(a < b)
    {...}
  else
    {...}
}
```

Complete annotations

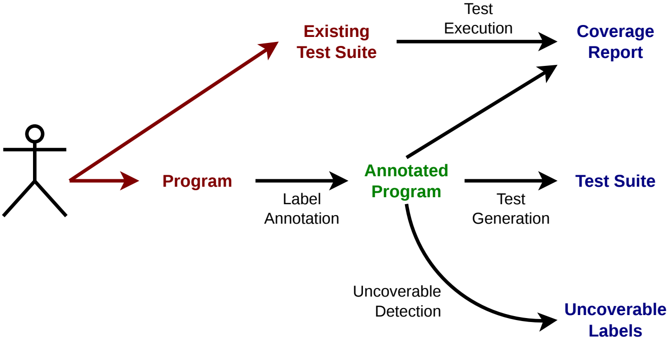
Conclusion: Complete annotation very slight overhead
(but label annotation experimentaly the best trade-off).

Summary

	VA	WP	VA $\oplus$ WP
sound for assert validity	✓	✓	✓
blackbox reuse	✓	✓	✓
local precision	✗	✓	✓
calling context	✓	✗	✓
calls / loop effects	✓	✗	✓
global precision	✗	✗	✗
scalability wrt. #labels	✓	✓	✓
scalability wrt. code size	✗	✓	?

Outline

Implementation inside LTest



FRAMA-C plugin called LTEST

- sound detection !
- several modes : VA, WP, VA ⊕ WP
- based on PATHCRAWLER for DSE* and test generation

Service cooperation

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

Outline

Experiments

RQ1 : How effective are the static analyzers in detecting infeasible test requirements ?

RQ2 : To what extent can we improve test generation by detecting infeasible test requirements ?

Standard (test generation) benchmarks [Siemens, Verisec, Mediabench]

- 12 programs (50-300 loc), 3 criteria (**CC**, **MCC**, **WM**)
- 26 pairs (program, coverage criterion)
- 1,270 test requirements, 121 infeasible ones

RQ1 : detection power

	#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

- **Verif :** VA ⊕ WP perform better than VA or WP alone
- **Testing :** VA ⊕ WP achieves almost perfect detection

RQ2 : Impact on test generation

→ report a more accurate coverage ratio

	Coverage ratio reported by DSE*				
Detection method	**None**	**VA**	**WP**	**VA ⊕WP**	**Perfect***
Total	**90.5%**	96.9%	95.9%	**99.2%**	100.0%
Min **Max**	**61.54%** 100.00%	80.0% 100.0%	67.1% 100.0%	**91.7%** 100.0%	100.0% 100.0%
Mean	**91.10%**	96.6%	97.1%	**99.2%**	100.0%

* preliminary, manual detection of infeasible labels

→ speedup test generation

- Beware can be slower in the worse case
- Gain, max : 55x, mean :2.2x (wit RT)

Outline

Introduction

Overview

Checking assertion validity

Implementation

Experiments

Conclusion

Conclusion

Challenge

- detection of infeasible test requirements

Results

- automatic, sound and generic method ✓
  - rely on labels and a new combination VA $\oplus$ WP
- promising experimental results ✓
  - strong detection power [95%]
  - reasonable detection speed [$\leq$ 1s/obj.]
  - improve test generation [better coverage ratios, speedup]

Future work : scalability on larger programs

- explore trade-offs of VA $\oplus$ WP
- application for verification(safety), and security

$\rightarrow$ LTest available at `http://micdel.fr/ltest.html`

Questions ?

Direction de la Recherche Technologique
Département d'Ingénierie des Logiciels et des Systèmes
Laboratoire de Sûreté des Logiciels

Commissariat l'énergie atomique et aux énergies alternatives
Institut Carnot CEA LIST
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
Etablissement public à caractère industriel et commercial — RCS Paris B 775 685 019