Counterexamples Validation in Why3 and SPARK

Claude MARCHÉ, Solène MOREAU

ProofInUse Meeting, November 21st, 2022

LMF, Inria, U. Paris-Saclay, etc., AdaCore

Context

Motivation: Explain Proof Failures

```
let test1 (x: int)
= let y = x + 1 in
  assert { y <> 43 }     ?
```

```
let f (x: int) : int
  ensures { result > x }     ✓
= x + 1

let test2 (x: int)
= let y = f x in
  assert { y = x + 1 }     ?
```

Motivation

How to help the user understand why proof fails?

Reminder: how SMT solvers are used by Why3

General workflow

- Verification Condition = declarations + hypotheses + goal
- Negate the goal → SMT solver, ask for satisfiability
  - if Unsat → VC is valid
  - if Sat → solver proposes a *model*

```
let test1 (x: int)
= let y = x + 1 in
  assert { y <> 43 }     ?
```

VC is `x,y : int ; H: y = x+1; G: y <> 43`

SMT query is `x,y : int ; H: y = x+1; G: y = 43`

→ Sat, model is `x=42, y=43`

Same with test2

```
let f (x: int) : int
  ensures { result > x }     ✓
= x + 1

let test2 (x: int)
= let y = f x in
  assert { y = x + 1 }     ?
```

VC is `x,y : int ; H: y > x; G: y = x+1`

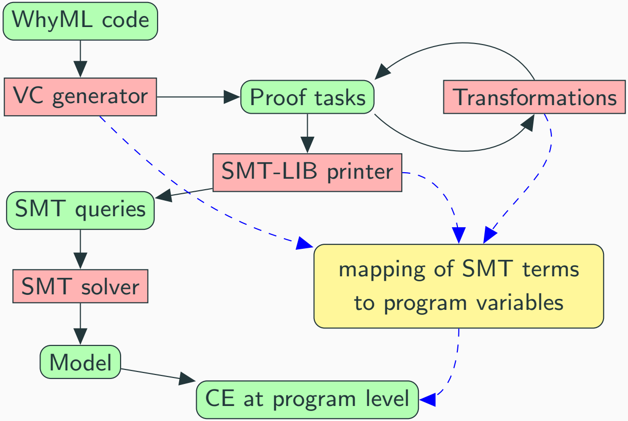
SMT query is `x,y : int ; H: y > x; G: y <> x+1`

→ Sat, model is `x=0, y=2`

Counterexamples in Why3 and in SPARK

Hauzar, Dailler, Marché, Moy [JLAMP 2018]

Turn the proposed model into a counterexample (CE) at the level of Why3, then at the level of SPARK



Counterexamples: the Good, the Bad, and the Ugly

DEMO

Counterexamples: the Good, the Bad, and the Ugly

DEMO

Main issue

Generated CEs are only *candidate* counterexamples

- incompleteness of the SMT solver (non-linear arithmetic, quantifiers)
- solver interrupted after a time limit

SPARK experience even worse

Experience showed CEs are *quite often wrong*, and thus *misleading* for the user

Identified need

It is desirable to *validate* the CE before giving it to the user

Need for Categorisation

Other identified need

Distinguish *true mistake in the code* versus *incomplete annotations*

Categories of proof failures [Petiot et al. 2018]

- *Non-conformity*
- *Sub-contract weakness*

Approach used: program transformations, symbolic execution (PathCrawler)

Validation and Categorisation [Becker, Lourenço, Marché, 2021]

Use *Runtime-Assertion-Checking* to check validity of CE and categorise

Runtime Assertion Checking (RAC) in Why3

- command

```
why3 execute
```

  exists for a long time, but *does not execute annotations*

- Benedikt added support for executing annotations (option `--rac`)
  - first by an ad-hoc partial evaluator
  - second by *calling a prover* to check them valid (option `--rac-prover`)

Small-step RAC

- execute programs, including function calls
- evaluate assertions when they are met
- evaluate pre-conditions at function call
- evaluate post-conditions at function exit
- evaluate loop invariants at each iterations of loops

```
let test1 (x: int)
= let y = x + 1 in
  assert { y <> 43 }
```

CE was proposing `x=42`

RAC of `test1` with `x=42`:

assertion violation

Small-step RAC not enough

```
let f (x: int) : int
  ensures { result > x }      ✓
= x + 1

let test2 (x: int)
= let y = f x in
  assert { y = x + 1 }      ?
```

CE was proposing `x=0`

RAC of `test2` with `x=0`: everything fine

Towards giant-step RAC

Need to “mimic” the VC generation, which considers the function call as an “atomic” operation

Giant-step RAC briefly

Giant steps are what you take
Proving on the moon
I hope my code don't break
Proving on the moon

- Execution of a function call:
    - checks pre-conditions
    - take values of *modified variables* and *result* from the CE (oracle)
    - checks post-conditions
- Execution of a loop: similar idea, only *one arbitrary iteration*

(see details in `https://hal.inria.fr/hal-03213438`)

Example

```
let f (x: int) : int
  ensures { result > x }       ✓
= x + 1

let test2 (x: int)
= let y = f x in
  assert { y = x + 1 }       ?
```

Giant-step RAC of `test2` with `x=0, y=2`

- call `f x` : pre OK, result taken from model = 2, post OK
- assertion: failed!

Small-step OK, giant-step failed → *subcontract weakness*

Categorisation

- Small-step RAC failure : *non-conformity*
- Small-step RAC OK, Giant-step RAC Failed : *subcontract-weakness*
- otherwise : bad counterexample, don't show it to the user

Taking also *incomplete* results for both RAC:

- lots of sources of incompleteness, including
  - solver unable to decide a formula
  - missing value from the CE
- we may answer : *non-conformity or sub-contract weakness*

What's next

Experimental results from SPARK

Small-step RAC in SPARK

Small-step in WhyML not good for SPARK

- WhyML generated contains many `val`, that is functions with contracts but no bodies
- Small-step RAC is thus not convenient for categorisation

Small-step directly in Ada

- Dedicated interpreter for Ada/SPARK code, in Ada
- Started by Benedikt
- Continued later by Viviane, intern at AdaCore

Outline

Context

Current State in SPARK, Statistics

Work In Progress

Future work

Current State in SPARK, Statistics

Situation on January 2022

SPARK testsuite:

2256 test programs, 915 of which containing unproved VC.

Candidate counterexamples for $\sim$ 4000 VC.

Situation on January 2022

SPARK testsuite:

2256 test programs, 915 of which containing unproved VC.

Candidate counterexamples for $\sim$ 4000 VC.

Repartition of the categorization for these candidate CE:

Bad counterexample	19.4 %	
Non-conformity	18.7 %	
Subcontract weakness	2.2 %	
Non-conformity or subcontract weakness	14.5 %	
Incomplete	**45.2 %**	
missing return value		36.5 %
cannot decide		29.5 %
unsupported values		15.9 %
other reasons		18.2 %

Improvements since January 2022 (1)

Small-step RAC (on the SPARK side) **widely extended**.

Improvements since January 2022 (1)

Small-step RAC (on the SPARK side) **widely extended**.

Lack of **return values** in CE models: needed by the giant-step RAC to extract results of function calls.

- Was missing for `val` functions.
- More return values when pushing `let...in...` in the context.

Improvements since January 2022 (2)

RAC should also execute goals.

- Error `Model term has no location`.

Improvements since January 2022 (2)

RAC should also execute goals.

- Error `Model term has no location`.

Do not forget to specify `--rac-prover`.

Improvements since January 2022 (2)

RAC should also execute goals.

- Error `Model term has no location`.

Do not forget to specify `--rac-prover`.

Some technical bug fixes.

- *E.g.* parsing errors of SMT models when switching to cvc5.

Improvements since January 2022 (2)

RAC should also execute goals.

- Error `Model term has no location`.

Do not forget to specify `--rac-prover`.

Some technical bug fixes.

- *E.g.* parsing errors of SMT models when switching to cvc5.

(WIP) **Collection of functional values** (more details later).

Updates of statistics...

...or how to try **not** comparing apples and oranges.

Some **improvements and inaccuracy fixes on statistics**:

- more accurate analysis of **cases without categorization**,
- checking of CE **reactivated** on several tests from the SPARK testsuite,
- more accurate analysis of **incompleteness** reasons,
- more **automation** to compute statistics.

Situation in November 2022

Overview picture:

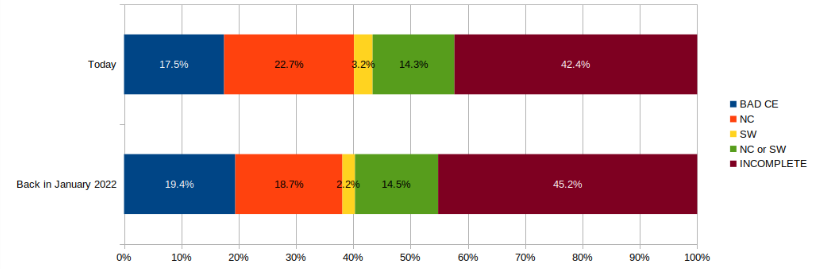
- a candidate CE is checked for $\sim$ 70 % of unproved VC with CE checking requested,
- (for the remaining $\sim$ 30 %, there is no candidate CE).

Situation in November 2022

Overview picture:

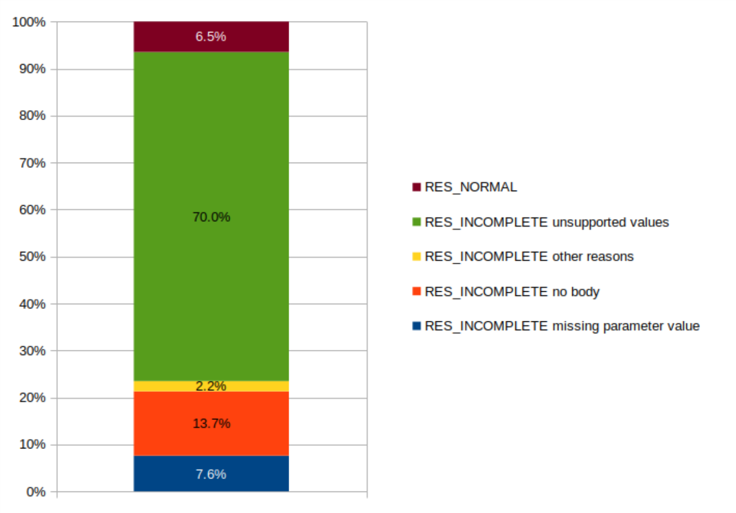
- a candidate CE is checked for ∼ 70 % of unproved VC with CE checking requested,
- (for the remaining ∼ 30 %, there is no candidate CE).

Repartition of the categorization for these candidate CE:



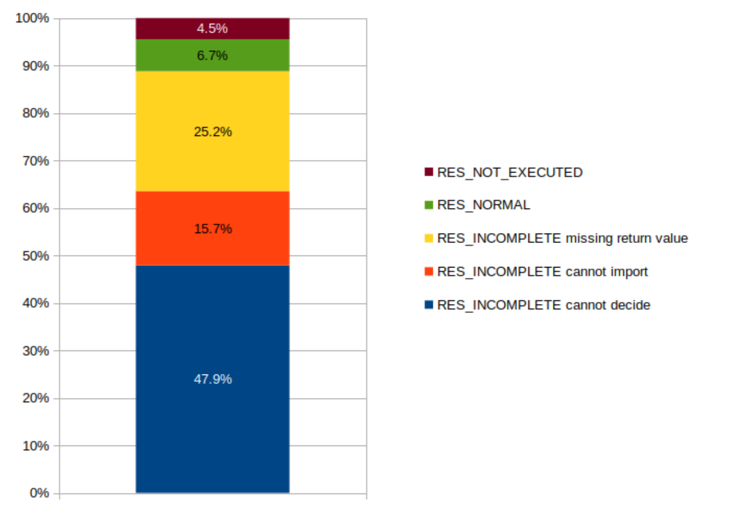
Situation in November 2022 — in more details

Small-step RAC results when the categorization is incomplete.



Situation on November 2022 — in more details

Giant-step RAC results when the categorization is incomplete.



Work In Progress

Towards less incomplete verdicts

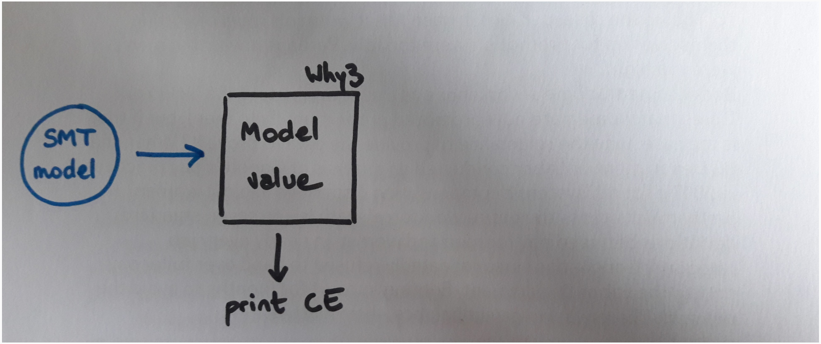
On the Why3 side, the main reason for incomplete verdicts is currently **“cannot decide”**, *i.e.* when the RAC prover cannot evaluate a formula.

Idea

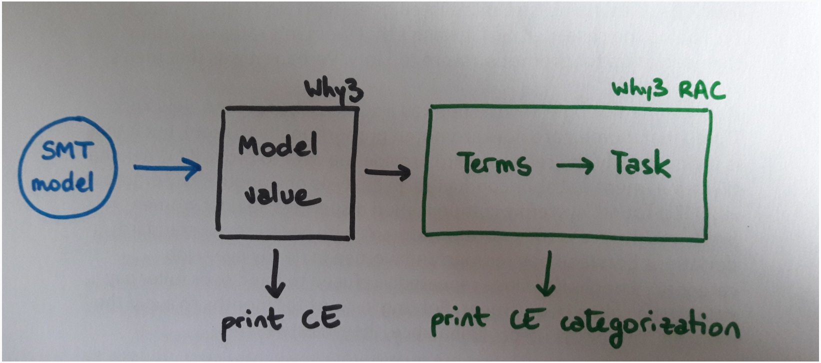
Collect **functional values** in SMT models to help the RAC prover.

DEMO

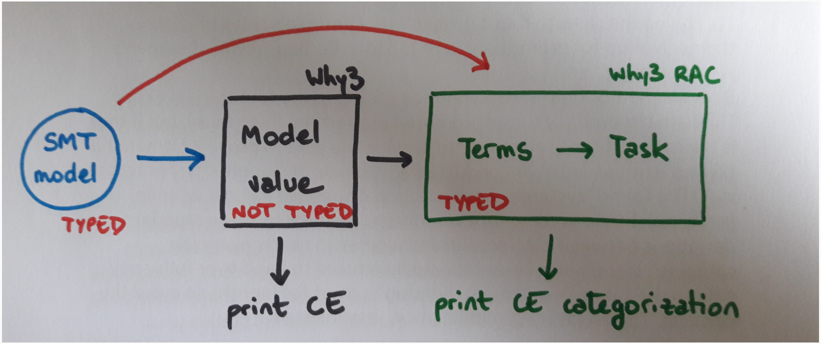
A bit of history



A bit of history



A bit of history



```
(define-fun c () Int 0)

(define-fun f ((x1 Int) (x2 Int)) Int (ite (= x1 x2) 1 0))
```

Simpler and stronger model parser for CE values

Collect **functional values** in SMT models to help the RAC prover.

Main issue

Complex transmission chain from the SMT output to the terms in the Why3 task for the RAC, “forgetting” the type of values for historical reasons.

Simpler and stronger model parser for CE values

Collect **functional values** in SMT models to help the RAC prover.

Main issue

Complex transmission chain from the SMT output to the terms in the Why3 task for the RAC, “forgetting” the type of values for historical reasons.

First steps (work in progress):

- interpret and check types in SMT models,
- simplify the transmission chain to directly translate SMT outputs to Why3 terms.

Side-effect: extended support for arrays, floats, reals.

Future work

Future work

Incompleteness in the Why3 small- and giant-step RAC: collect functional values from SMT models.

Future work

Incompleteness in the Why3 small- and giant-step RAC:
collect functional values from SMT models.

Incompleteness in the SPARK small-step RAC:
extend the range of supported values.

Future work

Incompleteness in the Why3 small- and giant-step RAC:
collect functional values from SMT models.

Incompleteness in the SPARK small-step RAC:
extend the range of supported values.

No counterexample for $\sim$ 30 % of unproved VC in the SPARK testsuite:
extend the fuzzing mechanism allowing to generate candidate CE values when the CE returned by SMT solvers are absent or bad.

Thank you for your attention!