VTG - Vulnerability Test Generator, a Plug-in for Rodin

Aymerick Savary[1,2] Jean-Louis Lanet[1] Marc Frappier[2]
Tiana Razafindralambo[1] Josselin Dolhen[1]

[1]Université de Limoges

[2]Université de Sherbrooke

29-02-2012

Outline

1. Definition and Tests Critera
2. Model Derivation
3. Trace Search
4. Experimentations
5. Conclusion and Future Works

Definition

Vulnerability Test

A vulnerability test confirms that a behaviour rejected by the model will also be rejected by its implementation.

- A test is a trace
- A fault is an event executed while its guard is unsatisfied
- Fault can occur at any point in the trace
- EUT for Event Under Test

Test Generation Strategy

- One fault per trace
  - Preamble
  - Faulty event
  - Postamble
- Traces selected using an LTL formula
- Tests criteria are determined by user-defined rules
- Implemented using ProB

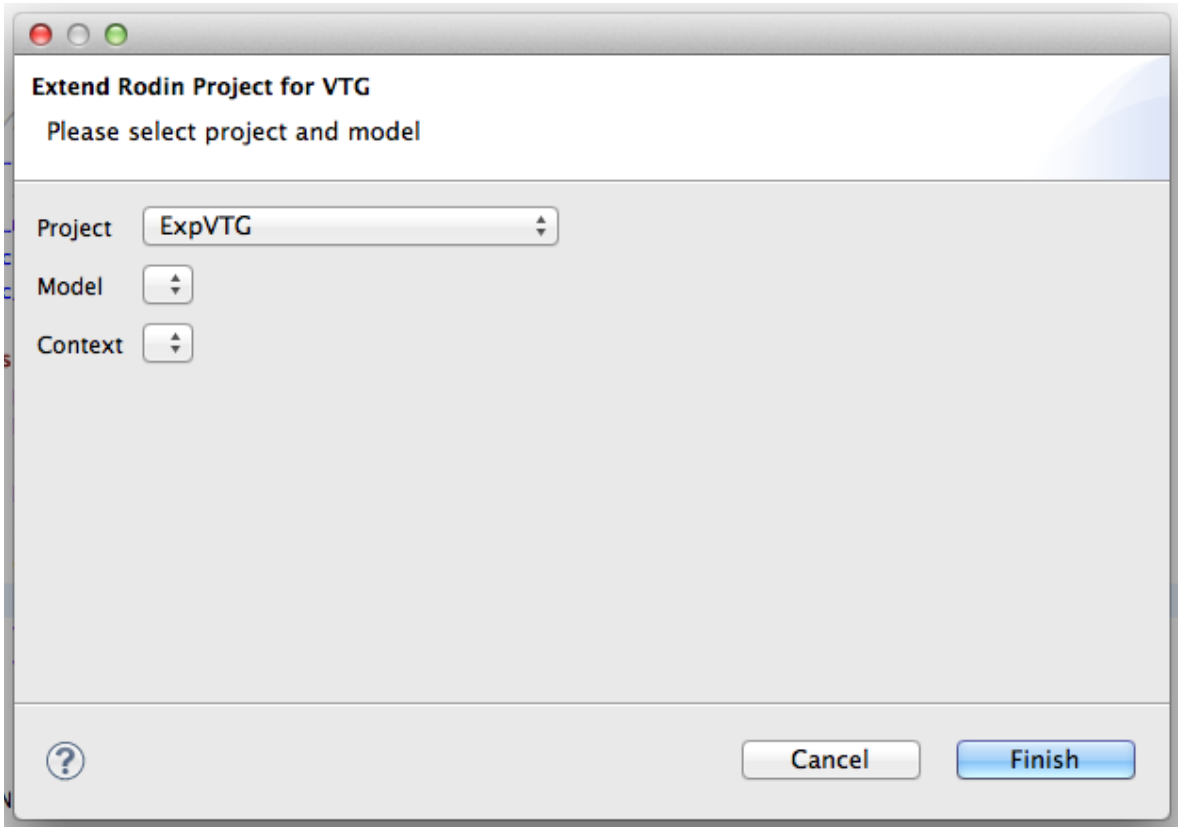
EUT - Event Under Test

- The faulty event is obtained by negating the guard of the EUT
- Negation are computed using rewrite rules
- Rewrite rules support usual test criteria
  - Combinatorial testing
  - Random testing
  - Boundary testing

Steps

- Model Derivation
  - The user identifies the guard elements to test
  - The tool rewrites the guard in two guard elements
  - The tool applies the rewrite rules
  - The tool creates a new model for each possible rewrite
  - The tool adds a new boolean variable denoting the execution of the faulty event
- Trace Search
  - The user choses an LTL formula and the max length of the trace
  - The tool generates the traces

Project, Model and Context Wizard



- Work in a new project

Identification of Guards to Negate

Original Event

Event $nomEvent \ \widehat{=}$

when

grd1 : a

grd2 : b

grd1_t : ca

grd2_t : cb

then

act1 : $a := a + 1$

act2 : $b := a + b$

end

Rewrite the guard in 2 elements

Event $nomEvent \ \widehat{=}$

when

grd : $a \wedge b$

grd_t : $ca \wedge cb$

then

act1 : $a := a + 1$

act2 : $b := a + b$

end

Negation of the EUT

Rewrite the guard in 2 elements

Event $nomEvent \;\widehat{=}$

when

$\texttt{grd} : \; a \wedge b$

$\texttt{grd\_t} : \; ca \wedge cb$

then

$\texttt{act1} : \; a := a + 1$

$\texttt{act2} : \; b := a + b$

end

Apply the rewrite rules

$\neg(c1_1 \wedge c1_2) \rightsquigarrow \neg c1_1 \wedge c1_2$

Event $nomEvent\_EUT \;\widehat{=}$

when

$\texttt{grd} : \; a \wedge b$

$\texttt{grd\_t} : \; \neg\mathbf{ca} \wedge cb$

then

$\texttt{act1} : \; a := a + 1$

$\texttt{act2} : \; b := a + b$

end

$\neg(c1_1 \wedge c1_2) \rightsquigarrow c1_1 \wedge \neg c1_2$

Event $nomEvent\_EUT \;\widehat{=}$

when

$\texttt{grd} : \; a \wedge b$

$\texttt{grd\_t} : \; ca \wedge \neg\mathbf{cb}$

then

$\texttt{act1} : \; a := a + 1$

$\texttt{act2} : \; b := a + b$

end

$\neg(c1_1 \wedge c1_2) \rightsquigarrow \neg c1_1 \wedge \neg c1_2$

Event $nomEvent\_EUT \;\widehat{=}$

when

$\texttt{grd} : \; a \wedge b$

$\texttt{grd\_t} : \; \neg\mathbf{ca} \wedge \neg\mathbf{cb}$

then

$\texttt{act1} : \; a := a + 1$

$\texttt{act2} : \; b := a + b$

end

Negation of the EUT

Apply the rewrite rules

Event $nomEvent\_EUT \mathrel{\widehat{=}}$
  when
    grd : $a \wedge b$
    grd_t : $\neg \mathbf{ca} \wedge cb$
  then
    act1 : $a := a + 1$
    act2 : $b := a + b$
  end

Add a new boolean variable

Event $nomEvent\_EUT \mathrel{\widehat{=}}$
  when
    grd : $a \wedge b$
    grd_t : $\neg ca \wedge cb$
    grd_eut : $eut = FALSE$
  then
    act1 : $a := a + 1$
    act2 : $b := a + b$
    act3 : $eut := TRUE$
  end

Derivation Rules

Logicals:

- $\neg(c_{1_1} \wedge c_{1_2}) \rightsquigarrow \{\neg c_{1_1} \wedge c_{1_2},\ c_{1_1} \wedge \neg c_{1_2},\ \neg c_{1_1} \wedge \neg c_{1_2}\}$.
- $\neg(c_{1_1} \vee c_{1_2}) \rightsquigarrow \{\neg c_{1_1} \wedge \neg c_{1_2}\}$.

Arithmetics predicates:

- $\neg(a = b) \rightsquigarrow \{a < b,\ a > b\}$
- $\neg(a > b) \rightsquigarrow \{a = b,\ a < b\}$
- $\neg(a < b) \rightsquigarrow \{a = b,\ a > b\}$
- $\neg(a \neq b) \rightsquigarrow \{a = b\}$

Set predicates:

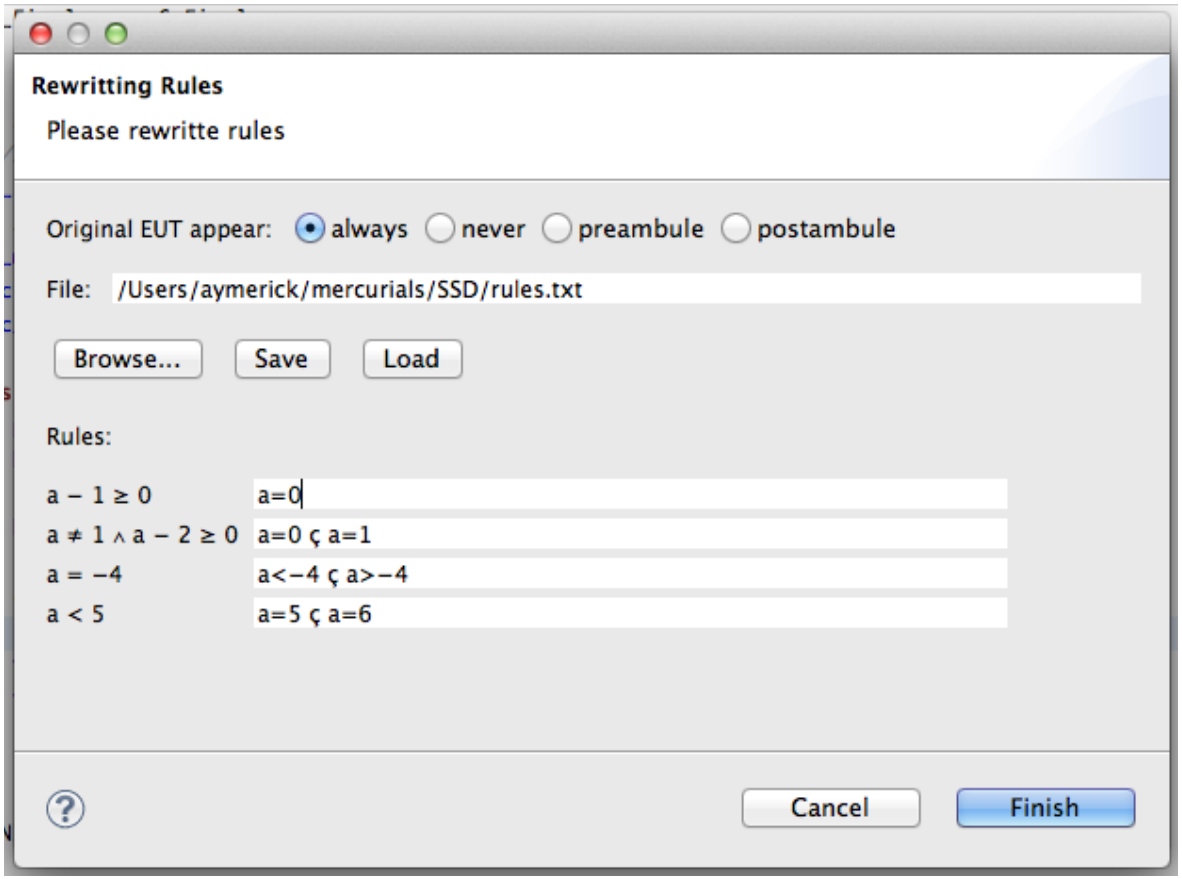
- $\neg(a \in B) \rightsquigarrow \{a \notin B\}$
- $\neg(a \notin B) \rightsquigarrow \{a \in B\}$

Summary

MACHINE M
VARIABLES
a
INVARIANTS
inv1 : $a \in -5 \mathinner{..} 5$
EVENTS
Initialisation
begin
act1 : $a := 3$
end
Event $add \mathrel{\widehat{=}}$
when
grd1 : $a \geq 0$
grd2_t : $a < 5$
then
act1 : $a := a + 1$
end
...
END

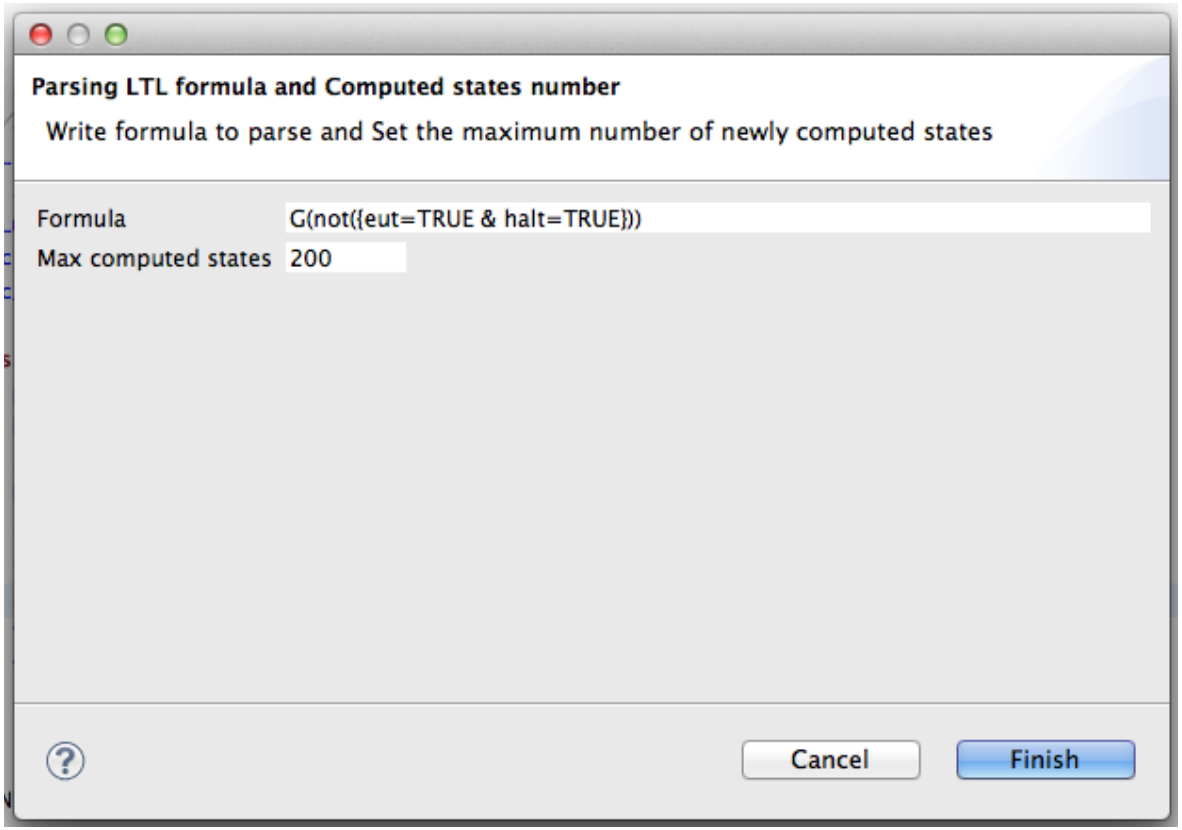
MACHINE M
VARIABLES
a
eut
INVARIANTS
inv2 : $eut \in BOOL$
EVENTS
Initialisation
begin
act2 : $eut := FALSE$
end
Event $add \mathrel{\widehat{=}}$
...
Event $add\_eut \mathrel{\widehat{=}}$
when
grd : $a \geq 0$
grd_t : $a = 5$
grd_eut : $eut = FALSE$
then
act1 : $a := a + 1$
act2 : $eut := TRUE$
end
...
END

Rewrite Wizard



- Parameters about the original EUT
- Load/Save rewrite file
- Modify/Edit each rewrite

Trace Search Wizard



- ProB API
- LTL formula
- Traces saved in a file

Cases studies

- Apply on JavaCard byte code verifier
  - Use the Freud and Mitchell language (on 10 instructions)
  - 13 events
  - Generation of 43 mutated models in 540 ms
  - Between 1.5 s and 15 s for one trace (8 s in average)
  - 6.5 min for all traces
  - 5 time more space used on hard drive
  - 1,4Mb of RAM
- Limits
  - Write rewrite rules
  - Adapt the model for ProB
  - All options are not yet in the Java API
- Time to obtain the tests from the start
  - Modeling : 3 weeks
  - Write rewrite rules : 1 hour
  - Generate tests : 7 minutes

Conclusion and Future Works

Conclusion:

- First prototype ready and integrated in Rodin platform
- The model may need to be adapted for the test
- Flexibility for selecting test criteria

Future Works:

- Use the GOAL algorithm of ProB
- Implement derivation rules
- Test it on large models (eg, full JavaCard Bytecode language)

Contact

- Aymerick Savary :
  - aymerick.savary@etu.unilim.fr
  - aymerick.savary@usherbrooke.ca
- Jean-Louis Lanet :
  - jean-louis.lanet@unilim.fr
- Marc Frappier :
  - marc.frappier@usherbrooke.ca