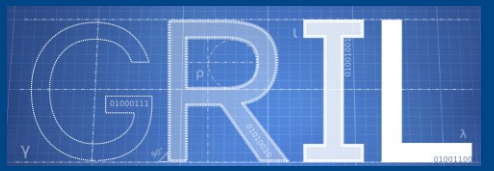
# Automatic Generation of Vulnerability Tests for the Java Card Byte Code Verifier

Aymerick Savary

e-smart

Sept, 22nd 2011

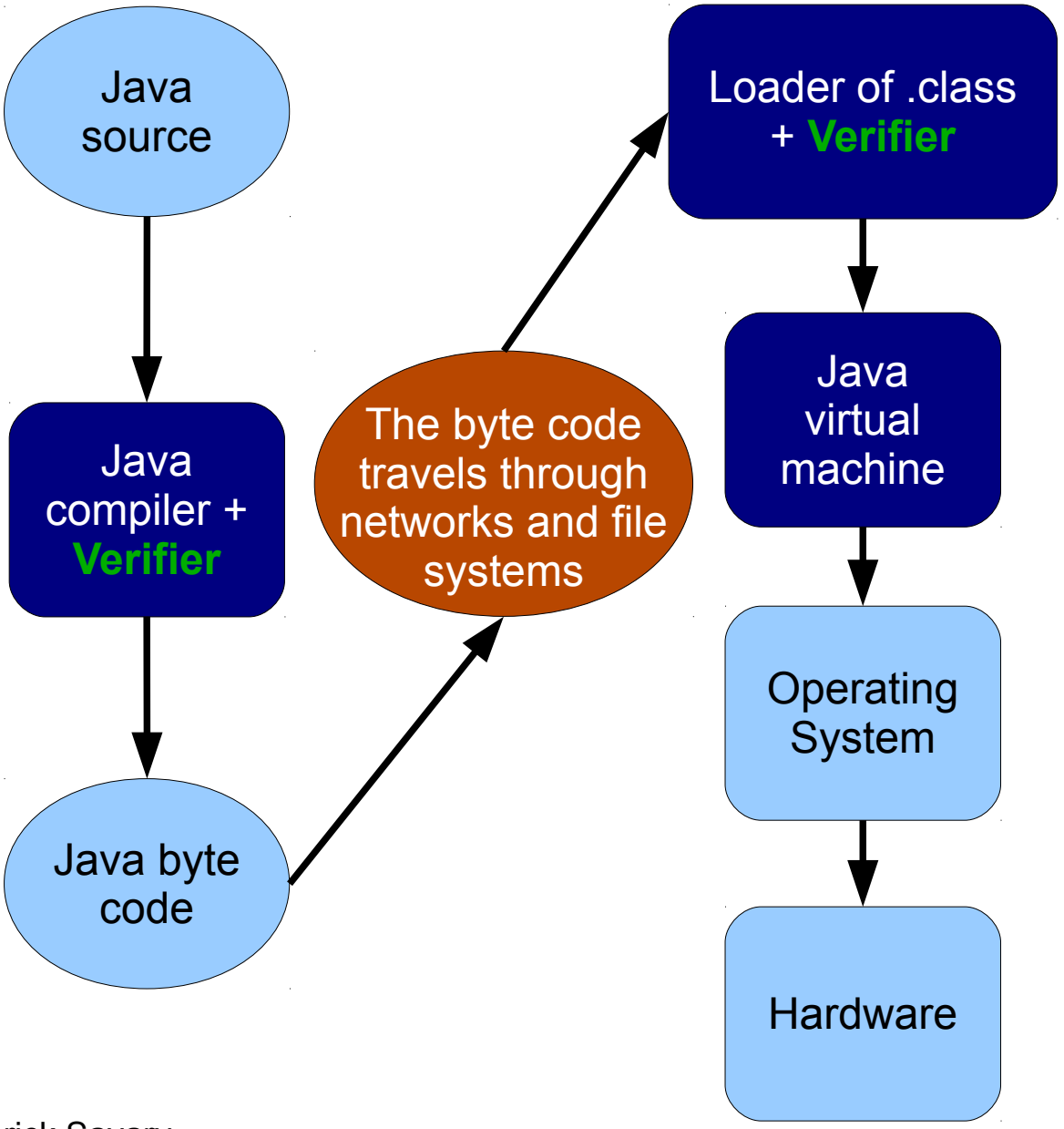

Université de Limoges

Université de Sherbrooke

# Outline

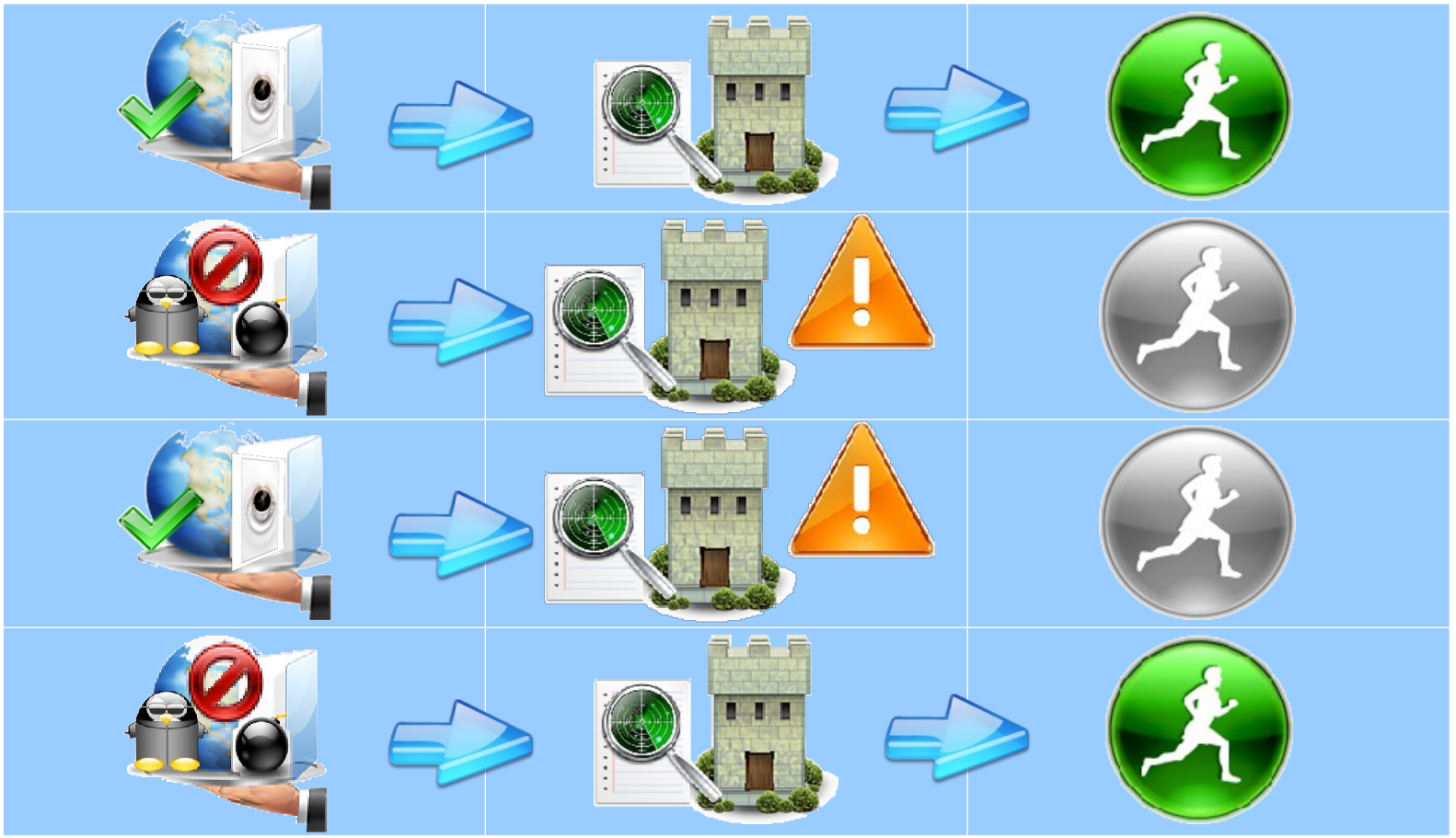
- Java, a strongly typed language
- Model-based testing
- Testing the implementation of the verifier
  - Principle put forward
  - A few results
- Conclusions

# Java, a safe language




- Strongly typed language
- Virtual machine execution
- Sandbox
- **Byte code verifier**

# Real behaviour of a byte code verifier



Valid behaviour
Authorized execution

Invalid behaviour
Refused execution

Valid behaviour
Refused execution

Invalid behaviour
Authorized execution

# Real behaviour of a byte code verifier

**Valid behaviour**
**Authorized execution**

**Invalid behaviour**
**Refused execution**

**Valid behaviour**
**Refused execution**

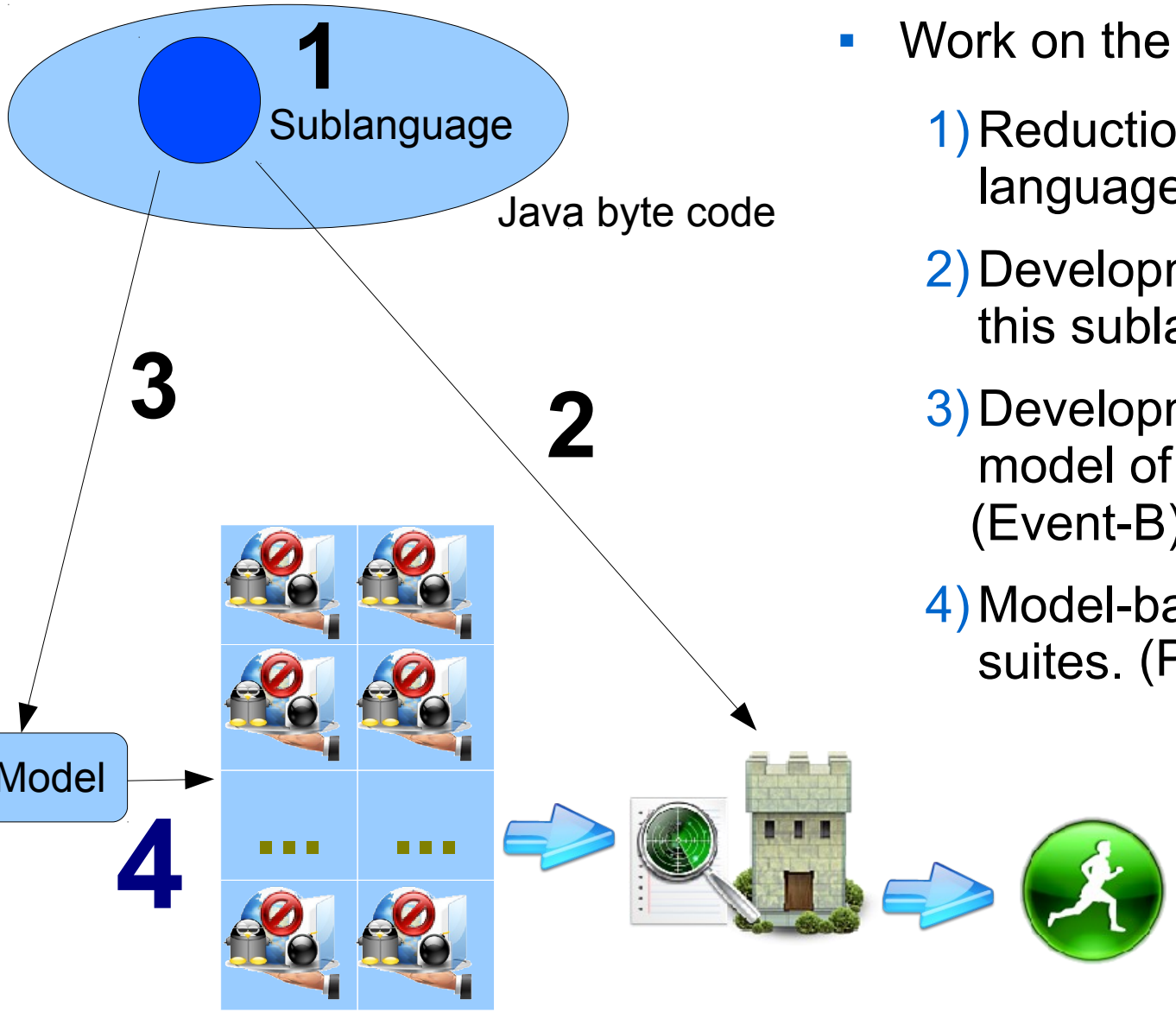
**Invalid behaviour**
**Authorized execution**

# Definition of vulnerability tests

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

- The goal is to generate actions in order to out-pass the limits defined by the model.
- By opposition to functional tests, vulnerability tests simply test the negation of the specification.

# Methodology




- Work on the sublanguage :
  1) Reduction of language in a sub language (Freud and Michel)
  2) Development of a verifier for this sublanguage. (ProB)
  3) Development of an Event-B model of this sublanguage. (Event-B)
  4) Model-based generation of test suites. (Plug-in Rodin)

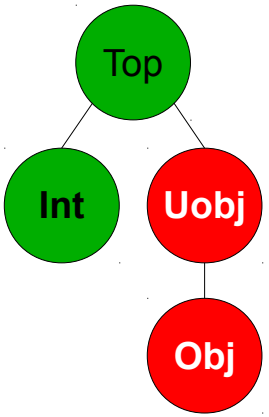
# An example of functional tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int




| PC | Instruction | Stack |
|---|---|---|
| 1 | Push0 | Int |
| 2 | Inc | Int |
| 3 | Pop | |
| 4 | Halt | |

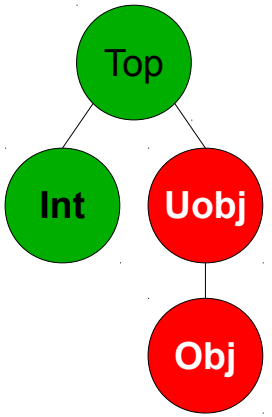
# An example of vulnerability tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int




Preamble

IUT (Instruction Under Tests)

Postamble

| PC | Instruction | Stack |
|---|---|---|
| 1 | Inc | ***reject*** |
| 2 | Pop | |
| 3 | Halt | |

| PC | Instruction | Stack |
|---|---|---|
| 1 | New | Uobj |
| 2 | Inc | ***reject*** |
| 3 | Pop | |
| 4 | Halt | |

| PC | Instruction | Stack |
|---|---|---|
| 1 | New | Uobj |
| 2 | Init | Obj |
| 3 | Inc | ***reject*** |
| 4 | Pop | |
| 5 | Halt | |

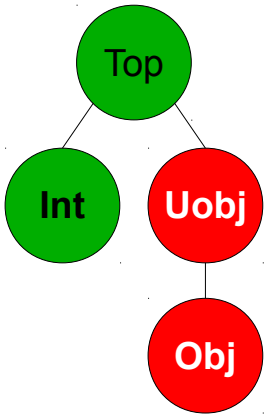
# An example of vulnerability tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int




Preamble

IUT (Instruction Under Tests)

Postamble

- Assessment of the preamble
- Set of test cases :
  - Empty
  - Uobj
  - Obj
- Assessment of the postamble

| PC | Instruction | Stack |
|---|---|---|
| 2 | Inc | *reject* |
| 3 | Pop | |
| 4 | Halt | |

| PC | Instruction | Stack |
|---|---|---|
| 1 | New | Uobj |
| 2 | Inc | *reject* |
| 3 | Pop | |
| 4 | Halt | |

| PC | Instruction | Stack |
|---|---|---|
| 1 | New | Uobj |
| 2 | Init | Obj |
| 3 | Inc | *reject* |
| 4 | Pop | |
| 5 | Halt | |

# Working mechanisms

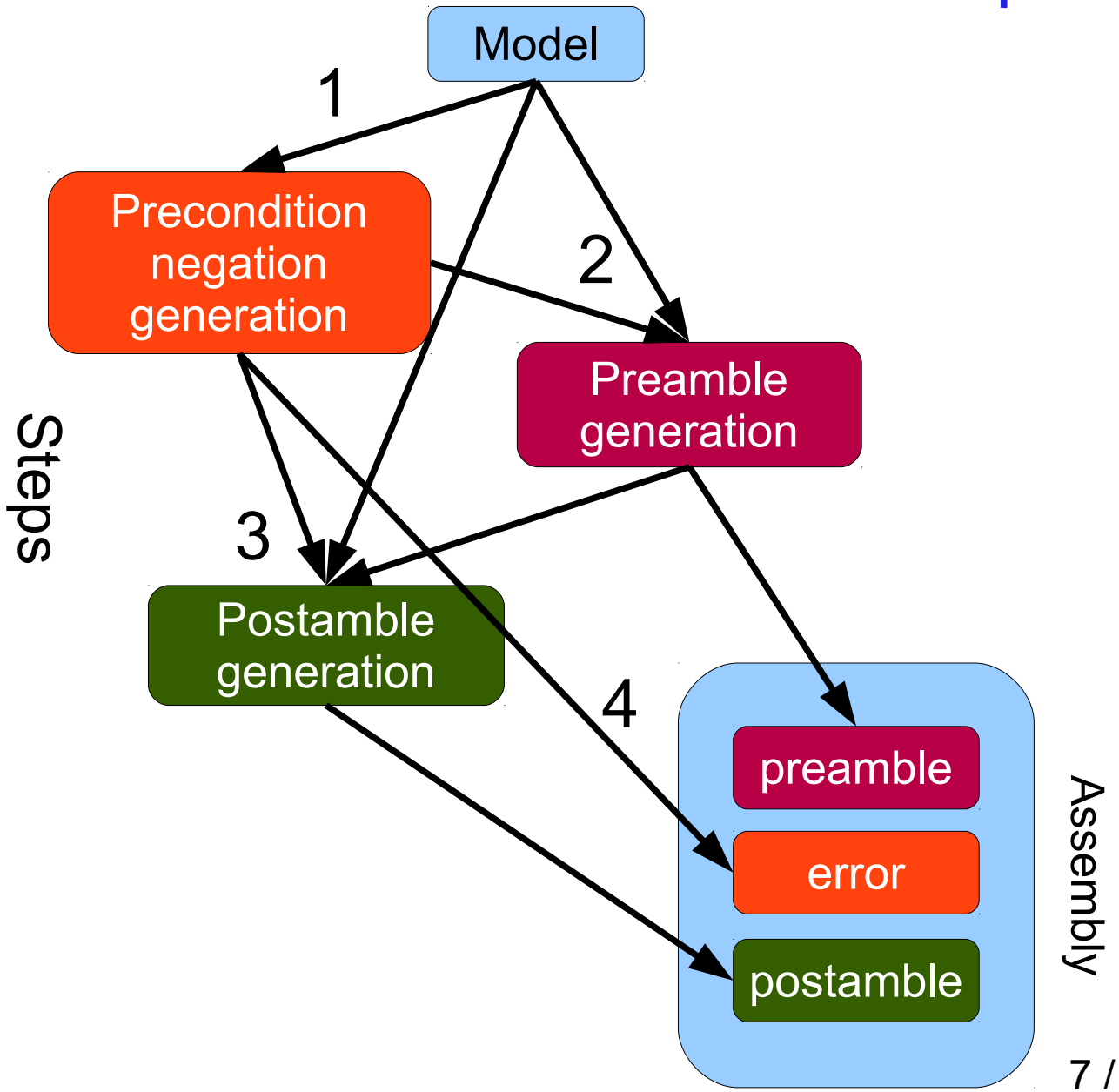
**Instruction:** Inc
**Pre:** Stack(TOS) = Int
**Post:** Stack(TOS) = Int

Stack = Empty
Stack(TOS) = Uobj
Stack(TOS) = Obj

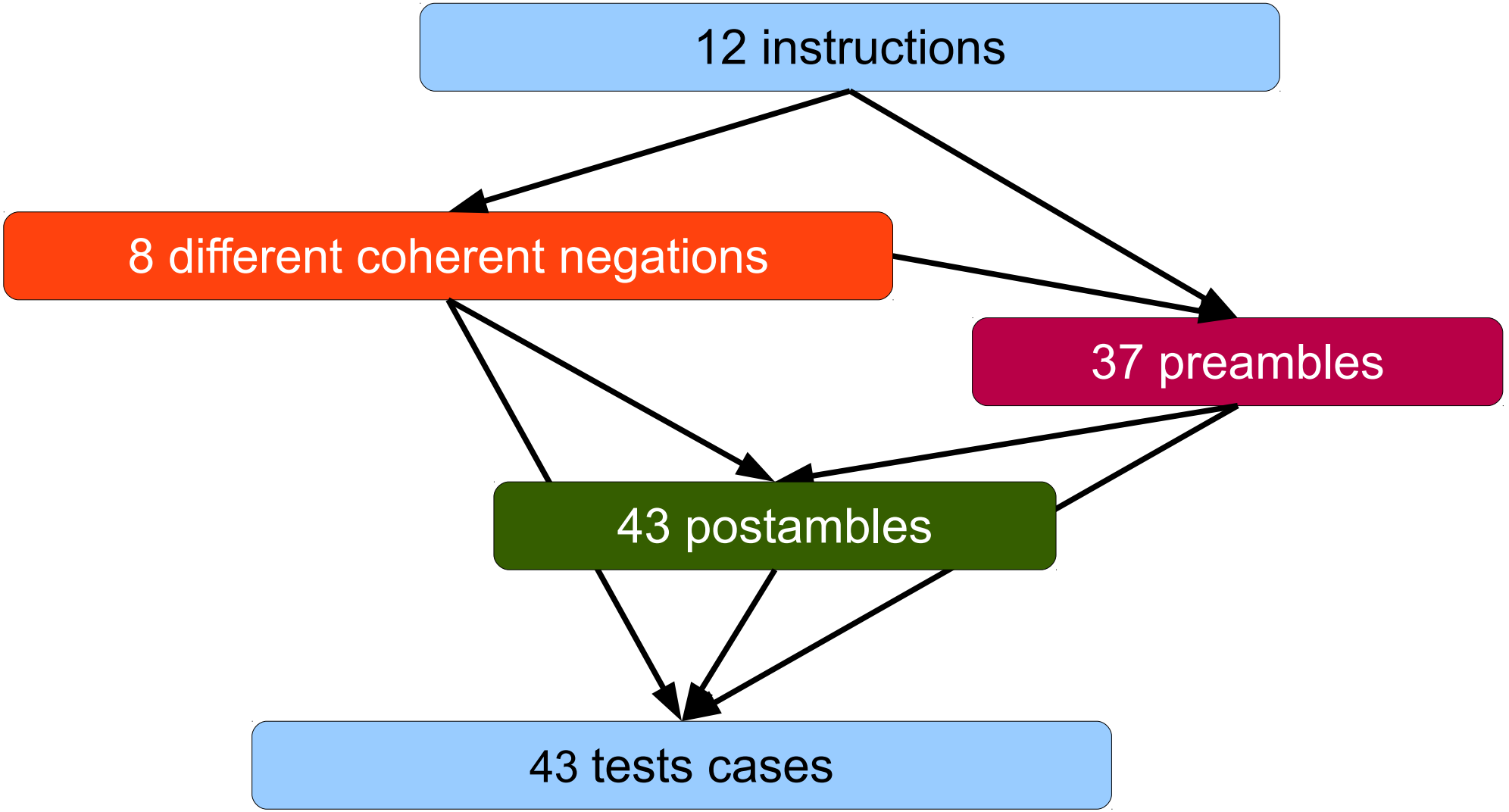
Ø
New
New; Init

Pop; Halt
Pop; Halt
Pop; Halt

Inc; Pop; Halt
New; Inc; Pop; Halt
New; Init; Inc; Pop; Halt



# Some metrics

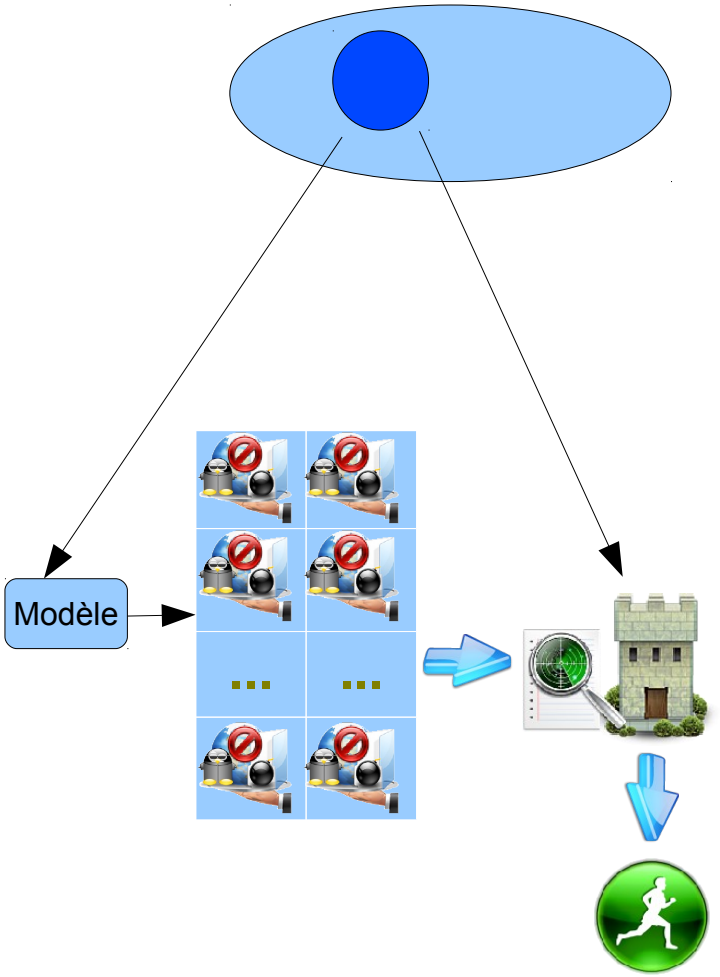


# Conclusions and future work

- Generation of Vulnerability Tests for the Java Card Byte Code Verifier
  - Successful of Proof Of Concept
  - Applying on applet vulnerability tests

Future work

- Automate the process using software.
- Start my PhD on : Apply our research to Java Card language.
- Find a way to generalise the negation of preconditions for all types of data structures.



# Achievements

This method can generate minimal test set able to characterize vulnerabilities of Java Card byte code verifier.

| 2011/12 | 2012/06 | 2012/12 | 2013/06 | 2013/12 | 2014/06 | 2014/12 | 2015/06 |
|---|---|---|---|---|---|---|---|
| M2 | | PHD | | | | | |
| Implementation for the sublanguage | stress the system | Model the Java Card language | | Generate the test set | | Evaluate the methodology on java Cards | |

# Contact information


- Aymerick Savary
- aymerick.savary@etu.unilim.fr
- aymerick.savary@usherbrooke.ca


CRYPTIS
FORMATIONS EN CRYPTOLOGIE
ET SÉCURITÉ DE L'INFORMATION