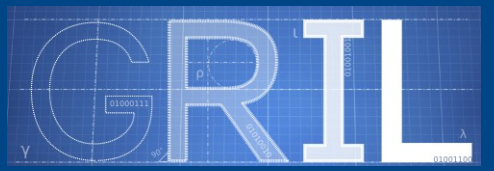
Automatic Generation of Vulnerability Tests for the Java Card Byte Code Verifier

Aymerick Savary

SAR-SSI, La Rochelle
May 18th, 2011



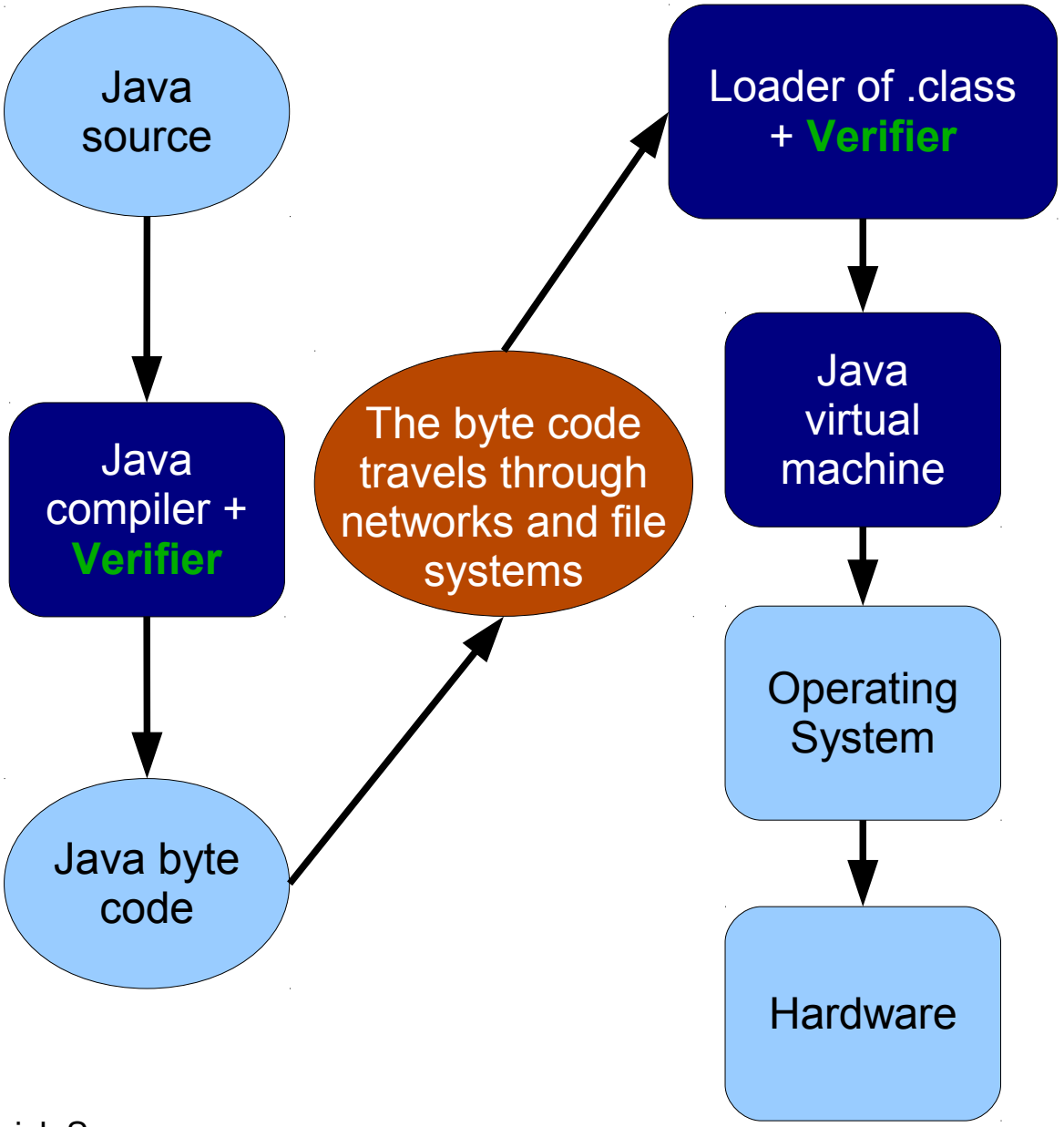
SSD team of Jean-Louis Lanet
Université de Limoges

Laboratory of Marc Frappier, PhD
Université de Sherbrooke

Outline

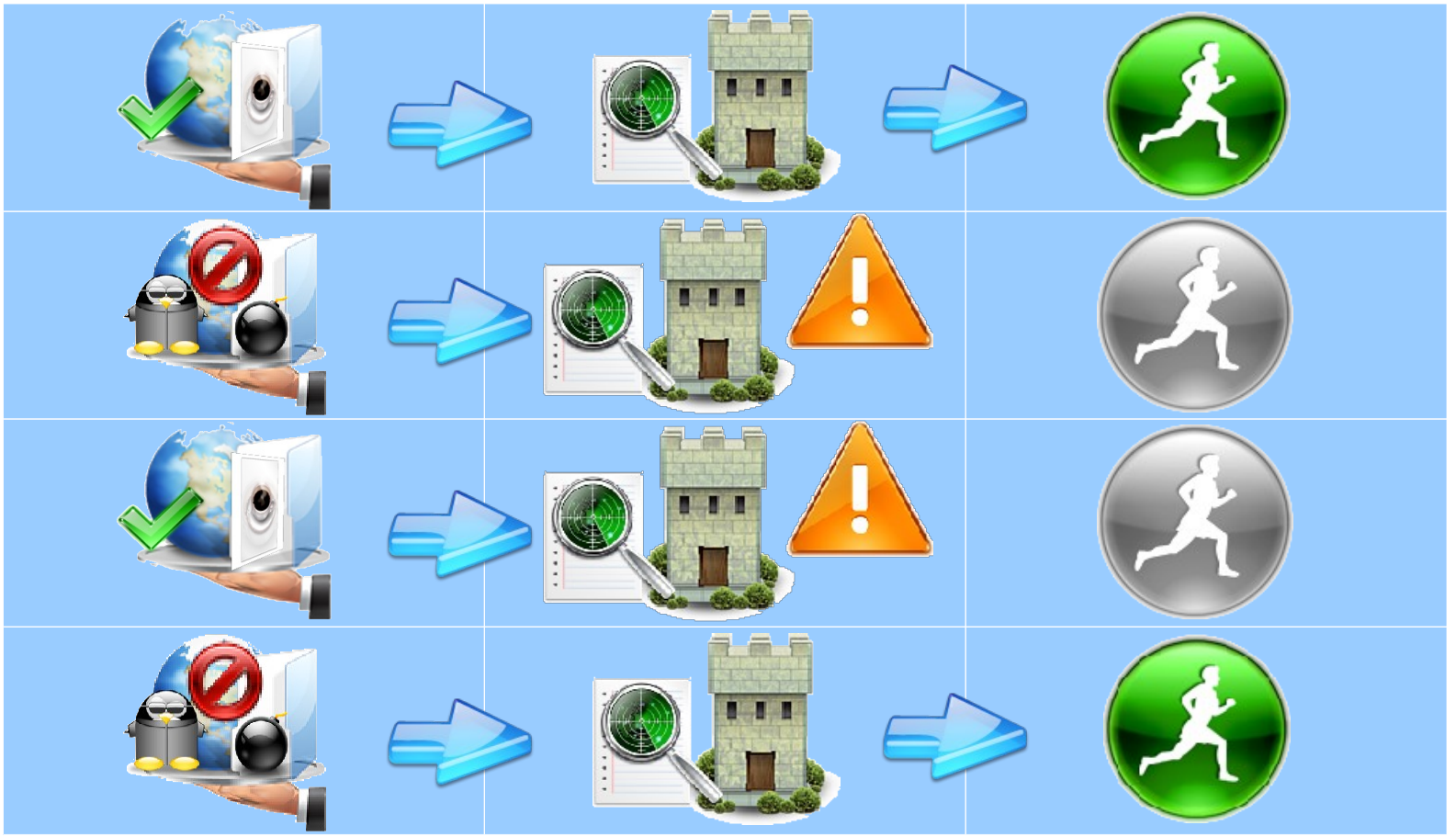
- Java, a strongly typed language
- Testing the implementation of the verifier
  - Definition of the vulnerability tests
  - Principle put forward
  - A few results
- Conclusions

Java, a safe language



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

Real behavior of a byte code verifier



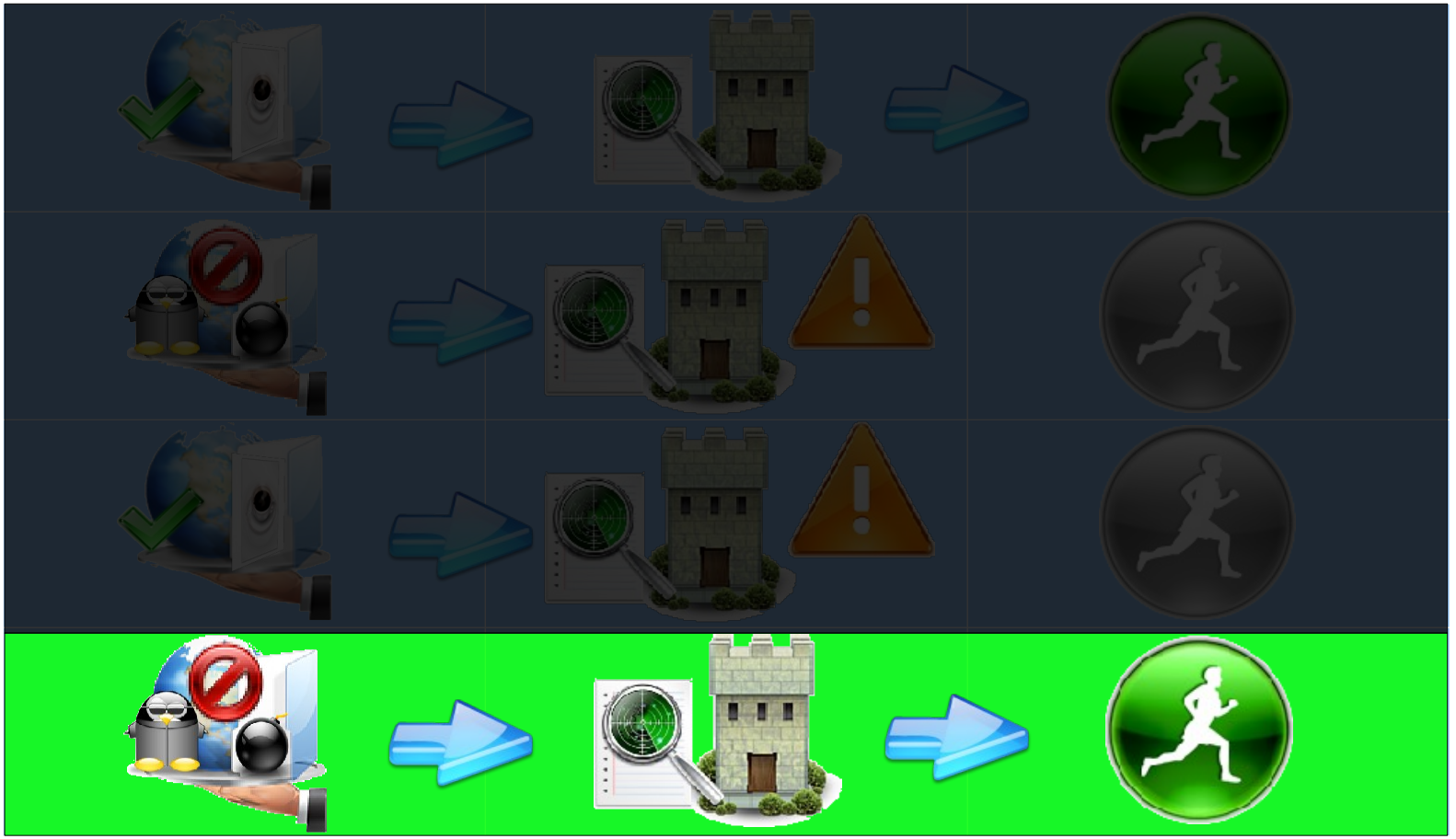
Valid code
Authorized
execution

Invalid code
Refused
execution

Valid code
Refused
execution

Invalid code
Authorized
execution

Real behavior of a byte code verifier



Valid code
Authorized execution

Invalid code
Refused execution

Valid code
Refused execution

Invalid code
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.

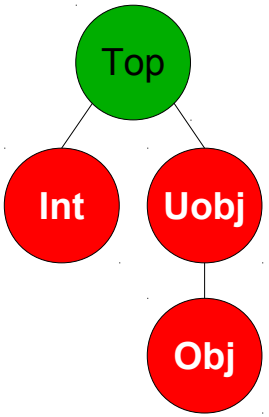
An example of vulnerability tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int



PC	Instruction	Stack
0	INIT	
1	Push0	Int
2	Inc	Int
3	Pop	
4	Halt	

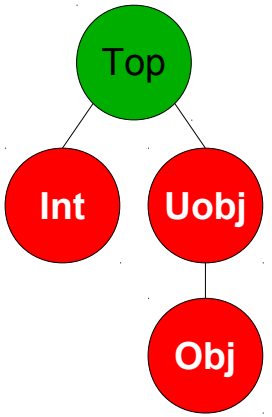
An example of vulnerability tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int



Preamble

IUT (Instruction Under Tests)

Postamble

PC	Instruction	Stack
0	INIT	
1	New	Uobj
2	Inc	***reject***
3	Pop	
4	Halt	

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

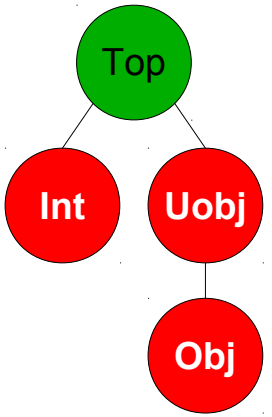
An example of vulnerability tests

Inc :

Manipulate the integer located on top of the stack

PRE
…, Int

POST
…, Int



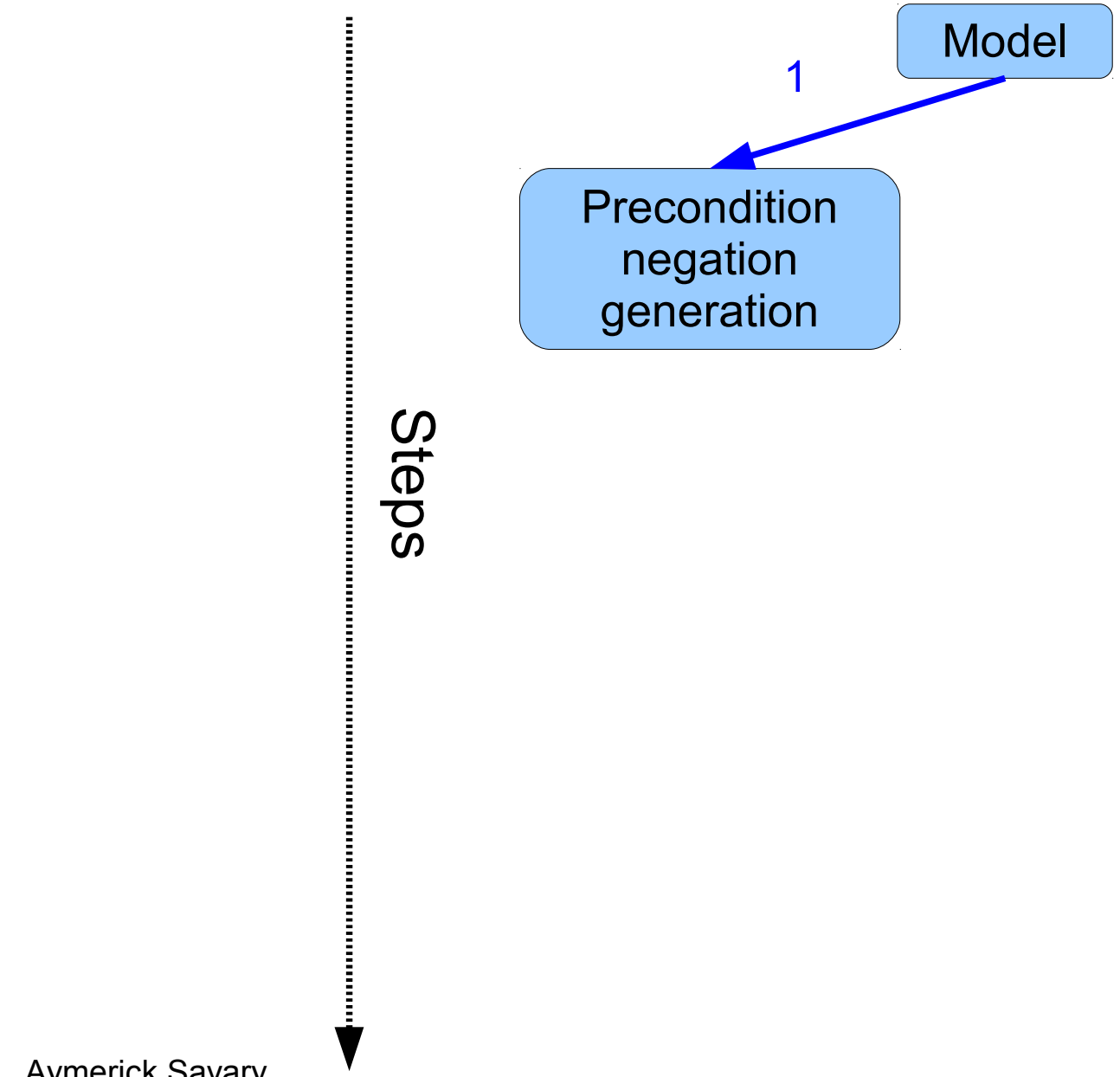
Preamble

IUT (Instruction Under Tests)

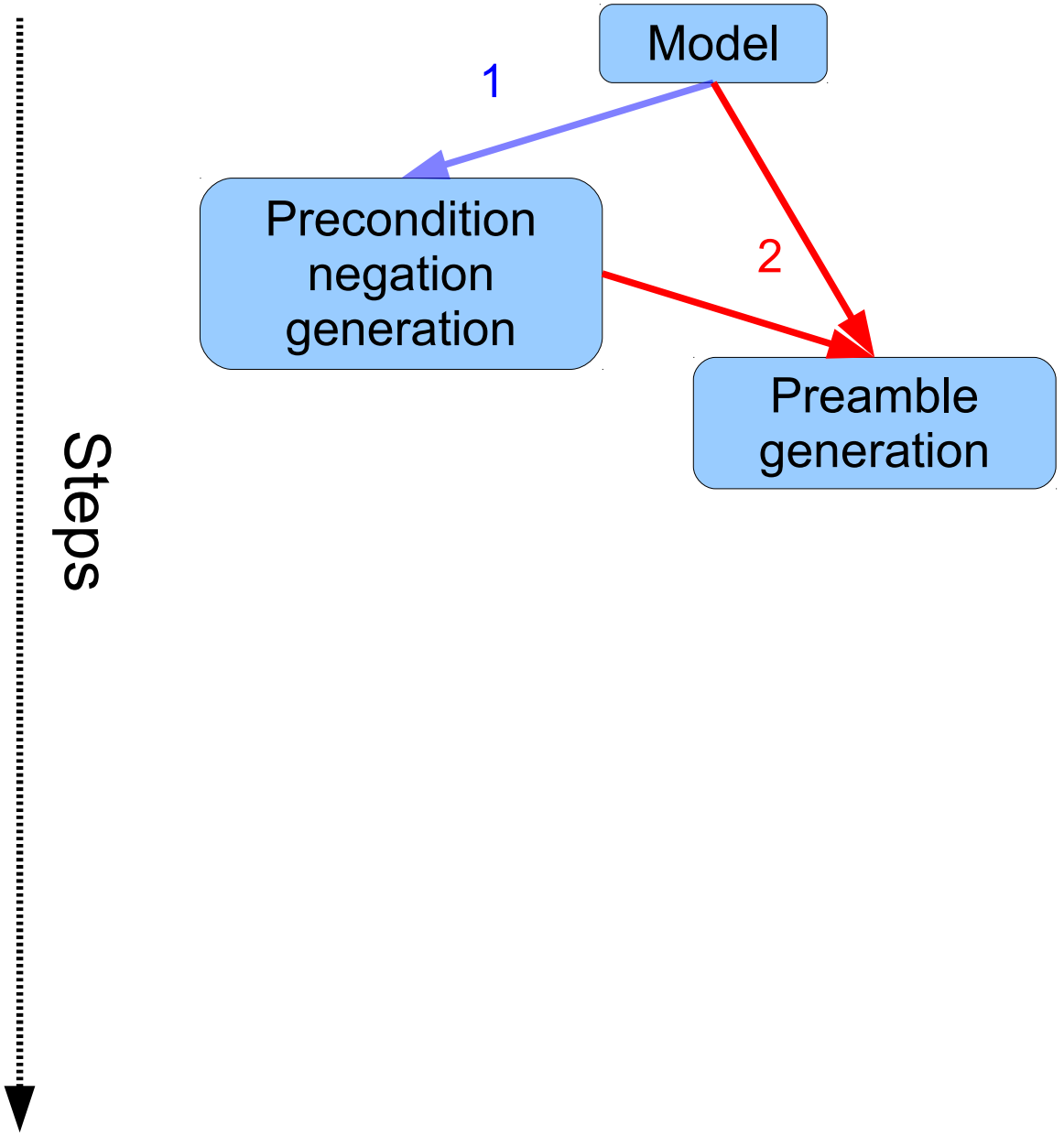
Postamble

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

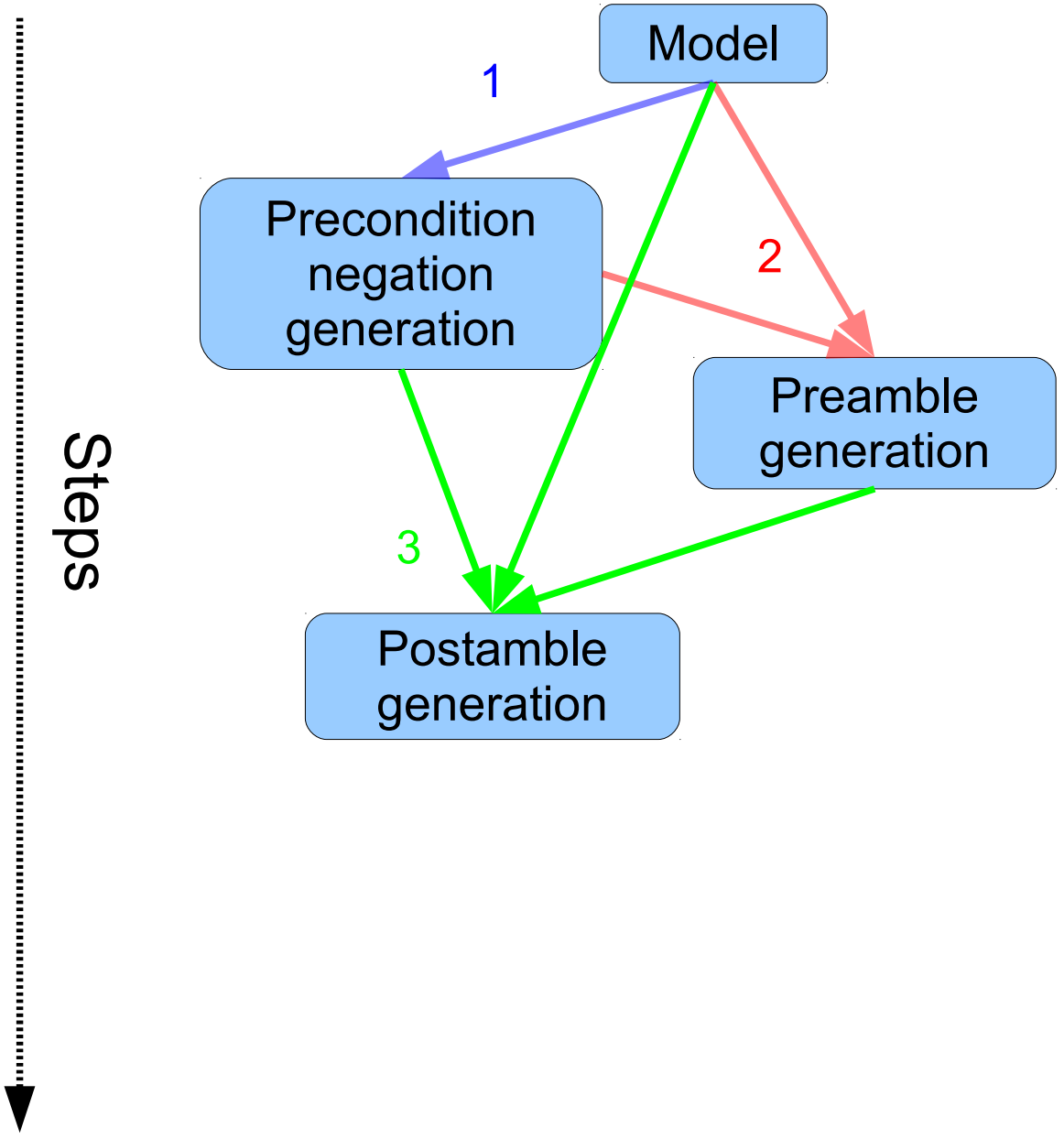
Working mechanisms



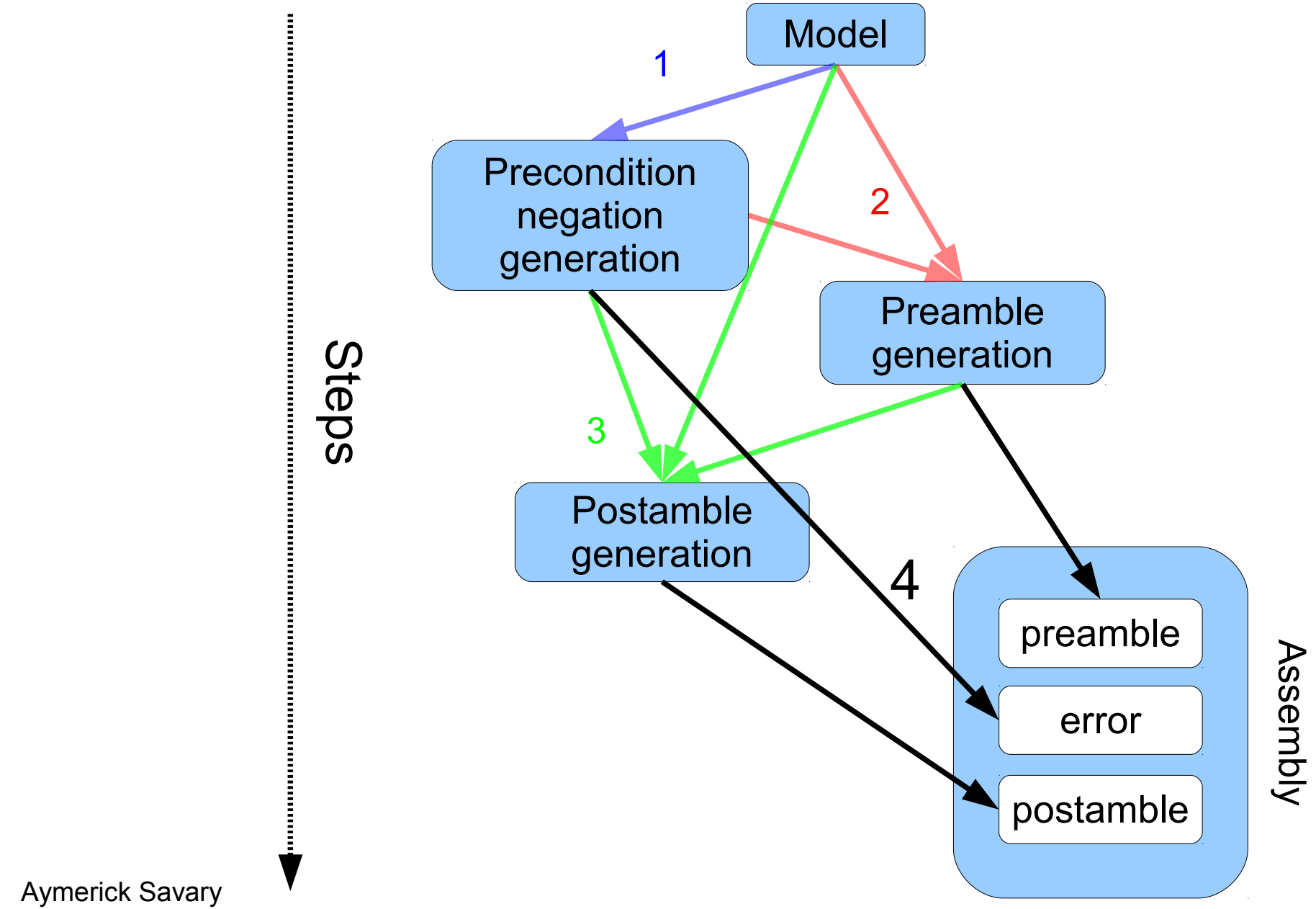
Working mechanisms



Working mechanisms



Working mechanisms



Precondition negation generation

- Grouping of clauses subsets :

$$\neg((C11 \wedge C12) \wedge C2) \rightarrow \begin{cases} (\neg (C11 \wedge C12) \wedge C2) \\ ((C11 \wedge C12) \wedge \neg C2) \end{cases}$$

- Logic transformations :

$$\neg(a \wedge b) \rightarrow \begin{cases} (\neg a \wedge b) \\ (a \wedge \neg b) \\ (\neg a \wedge \neg b) \end{cases}$$

$$\neg(a \vee b) \longrightarrow (\neg a \wedge \neg b)$$

Precondition negation generation

- Arithmetic transformations :

¬(a = b) → a < b, a > b

¬(a > b) → a < b, a = b

¬(a ≠ b) → a = b

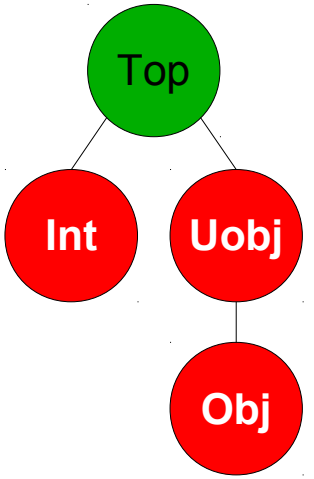
¬(a < b) → a > b, a = b

- Types transformations :

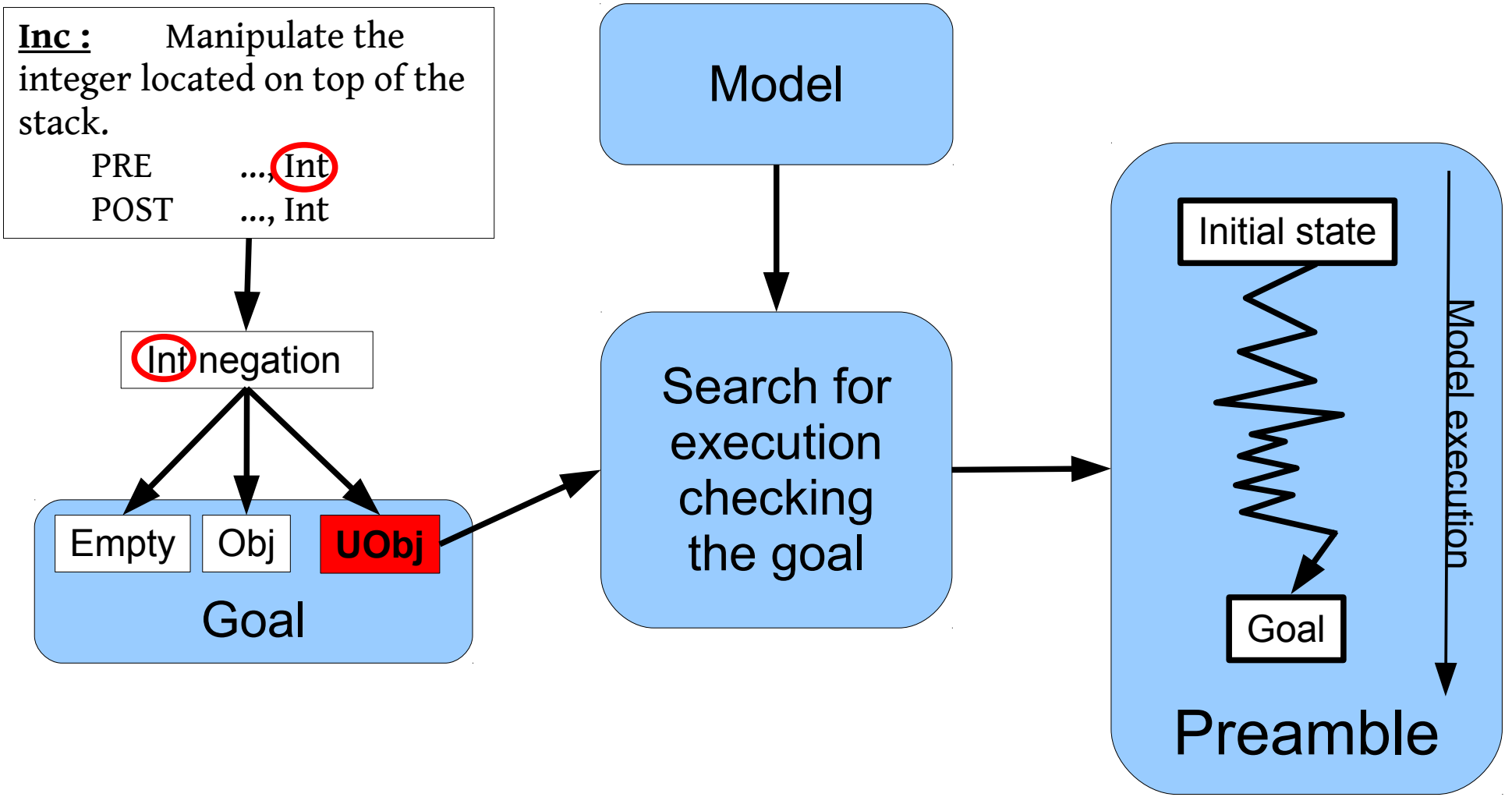
¬(Int) → Uobj, Obj

¬(Obj) → Uobj, Int

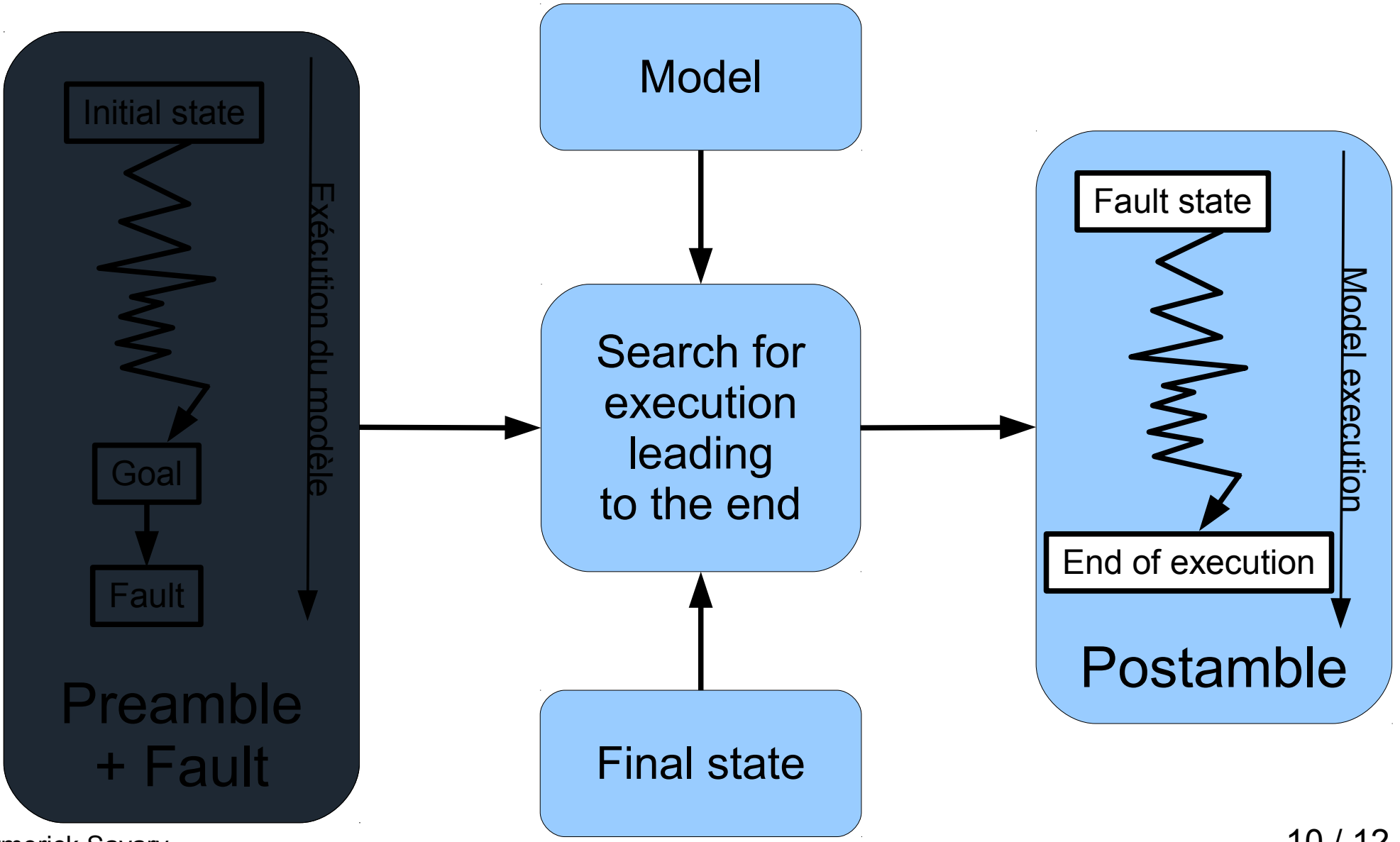
¬(Uobj) → Int



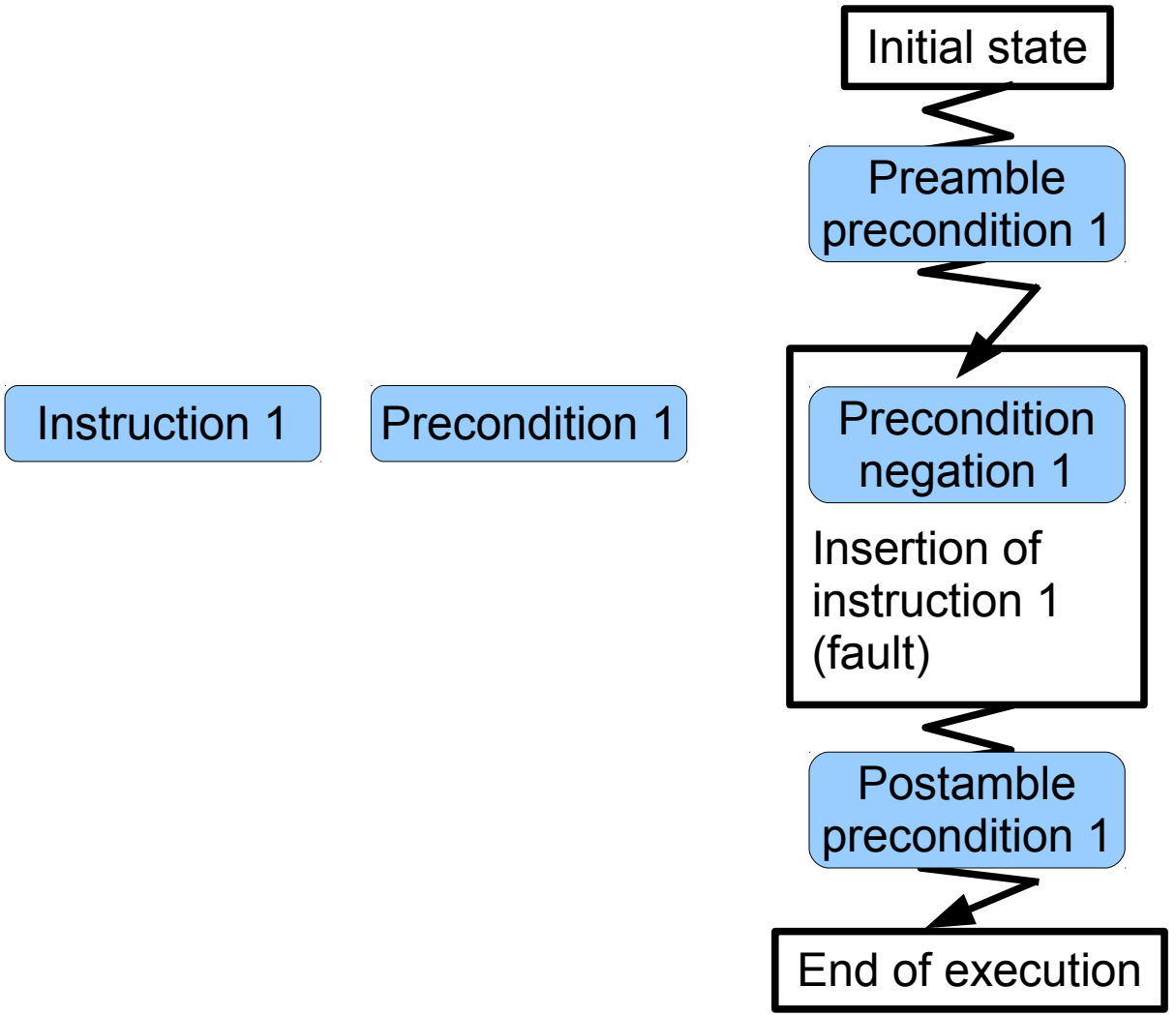
Preamble generation



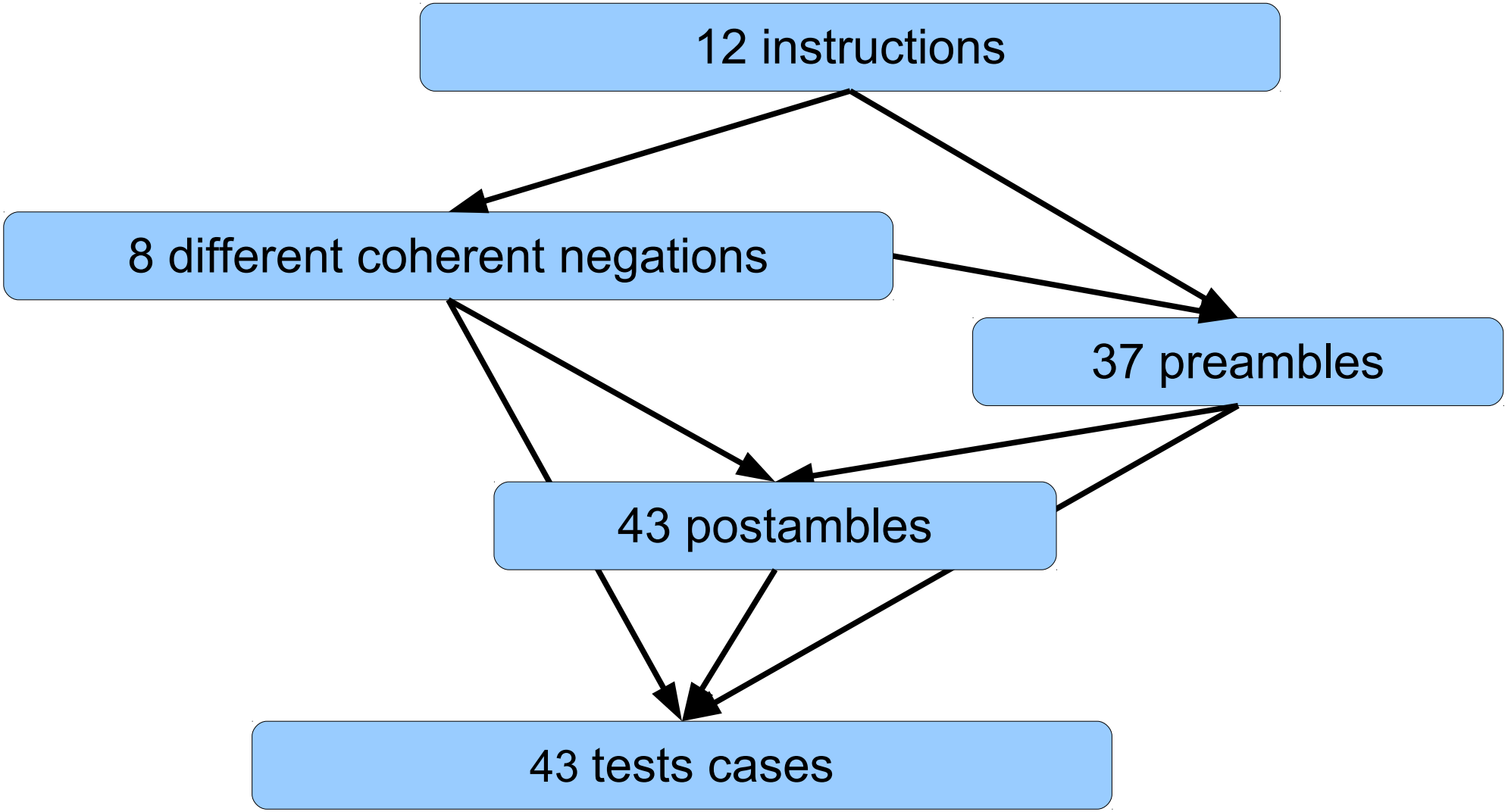
Postamble generation



Components assembly



A few results...



Conclusion and future works

- We demonstrated the validity of our proof of concept on a reduced language set of Java
- We have been able to generate automatically the preamble and the post-amble

Next steps

- Adapt the approach on the complete Java language
- Characterize a Java Card embedded byte code verifier
- Find and exploit vulnerabilities
- Apply it to other application domains

Thank you !

- Marc Frappier, PhD
- Jean-Louis Lanet
- Laboratory members
- Founding organization

Contact informations

- Aymerick Savary
- Mail : aymerick.savary@gmail.com

  aymerick.savary@usherbrooke.ca

  aymerick.savary@etu.unilim.fr
- Phone (Canada) : +1 819 432 2062
- Phone (France) : +(33)6 61 81 88 39