An Introduction to Formal Methods in Software Engineering.

PhD student:
Vincent Iampietro

PhD supervisors:
David Andreu, David Delahaye

December 4, 2019

Definitions.

Formal Methods.

Formal methods are techniques used to model *systems* as mathematical entities.

Systems can be

- Digital architectures
- Physical/Biological phenomenons
- Social interaction schemes
- and...

Definitions.

Formal Methods.

Formal methods are techniques used to model *systems* as mathematical entities.

Systems can be

- Digital architectures
- Physical/Biological phenomenons
- Social interaction schemes
- and. . . *softwares!*

Formal methods: perks and drawbacks.

Perks.

Mathematical approaches to model *softwares*:

- Improve the precision of the design.
- Enable proofs of correctness.
- Permit the extraction of properties.

Overall reliability of the software

Formal methods: perks and drawbacks.

Drawbacks.

- Conception time increases dramatically.
- Models are simpler than real softwares.
- Lack of qualified workforce.

Time & money spent to build a software

Formal methods: when do we need them?

- Development of safety-critical systems:
  - Avionics
  - Medicine
  - Automotive
  - Nuclear
  - ...

Formal methods: when do we need them?

- Development of safety-critical systems:
  - Avionics
  - Medicine
  - Automotive
  - Nuclear
  - ...
- Risks:
  - Loss of life/lives
  - Loss of resources (financial, natural...)

Formal methods: when do we need them?

Critical bugs are real!

- Therac-25 radiation therapy, excessive quantities of beta radiation (1980s).
- Patriot Missile, floating point rounding error (1991).
- Ariane 5 failure, conversion from 64-bit floating to 16-bit signed (1996).
- Hitomi astronomical satellite destruction, thruster orientation bug (2016).
- See P.G.Neumann's list of *Computer-related Risks* [Neumann, 1995].

What is bug? (baby don't hurt me...)

Context.

The *Airsafe* company orders an autopilot program to its *developer team* to control the Swiss new aircraft fleet.

What is bug?

Context.

The *Airsafe* company orders an autopilot program to its *developer team* to control the Swiss new aircraft fleet.

- Airsafe: **“I want an autopilot program that avoid plane crashes.”**

What is bug?

Context.

The *Airsafe* company orders an autopilot program to its *developer team* to control the Swiss new aircraft fleet.

- Airsafe: **“I want an autopilot program that avoid plane crashes.”**
- Result: *the plane never leaves the ground!*

What is bug?

Context.

The *Airsafe* company orders an autopilot program to its *developer team* to control the Swiss new aircraft fleet.

- Airsafe: **"I want an autopilot program that avoid plane crashes."**
- Result: *the plane never leaves the ground!*

- Question: **Is it a bug?**

What is bug?

Definition of a bug (failure).

"an undesired effect observed in the software's delivered service", [Abran, 2004].

Remark.

A bug is always related to the software *specification*.

About Specification.

Specification expressed in natural language can be:

- incomplete (e.g, the autopilot program).
- ambiguous: "I want an autopilot program that avoid plane crashes on the ground."

About Specification.

Formal specification.

- Removes the ambiguity.
- Limits: never sure about completeness.

Formal specification formats.

- Unified Modelling Language (UML).
- First Order Logic (FOL).
- Petri nets.
- Any formal languages...

Formal method techniques.

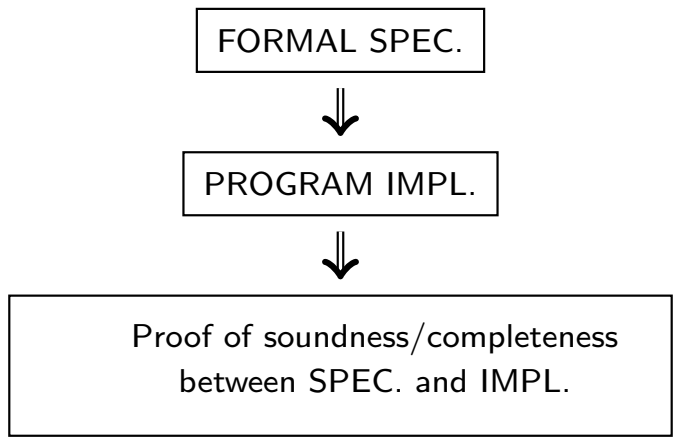
- Model checking.
- Automatic theorem proving.
- Deductive methods
- ...

Formal method techniques.

- Model checking.
- Automatic theorem proving.
- **Deductive methods.**
- ...

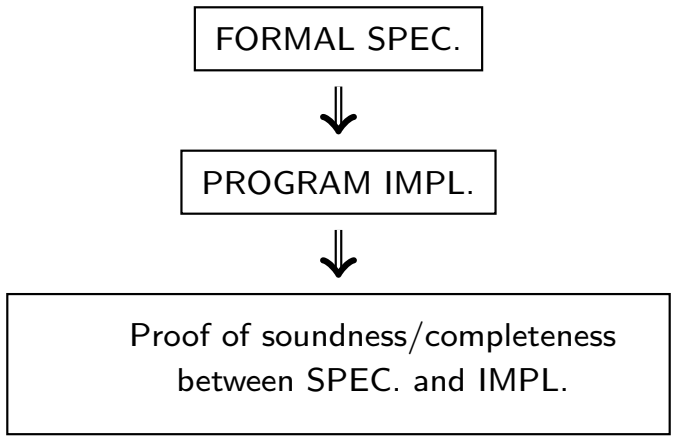
Formal method techniques.

Deductive methods.



Formal method techniques.

Deductive methods.



Human-executed although mechanized

An example of program.

Natural language specification.

A bank:
"I want a program that debits a non-negative amount from an account if there's enough money in the account."

An example of program.

Natural language specification.

A bank:
"I want a program that **debits** a **non-negative amount** from an account if there's **enough money in the account**."

An example of program.

Formal specification.

$$\texttt{Debit} \in \mathbb{Z} \to \mathbb{Z} \to (\mathbb{Z} \cup \texttt{Err}) \to \{\top, \bot\} \equiv$$
$$\forall \mathit{acc}, \mathit{amnt} \in \mathbb{Z},$$
$$\quad | \ \mathit{acc} \geq \mathit{amnt} > 0 \Rightarrow \texttt{Debit}(\mathit{acc}, \mathit{amnt}, \mathit{acc} - \mathit{amnt})$$
$$\quad | \ \mathit{amnt} \leq 0 \Rightarrow \texttt{Debit}(\mathit{acc}, \mathit{amnt}, \texttt{Err})$$
$$\quad | \ \mathit{amnt} > \mathit{acc} \Rightarrow \texttt{Debit}(\mathit{acc}, \mathit{amnt}, \texttt{Err})$$

An example of program.

Formal specification.

$$\texttt{Debit} \in \mathbb{Z} \to \mathbb{Z} \to (\mathbb{Z} \cup \texttt{Err}) \to \{\top, \bot\} \equiv$$
$$\forall acc, amnt \in \mathbb{Z},$$
$$\quad | \; acc \geq amnt > 0 \Rightarrow \texttt{Debit}(acc, amnt, acc - amnt)$$
$$\quad | \; amnt \leq 0 \Rightarrow \texttt{Debit}(acc, amnt, \texttt{Err})$$
$$\quad | \; amnt > acc \Rightarrow \texttt{Debit}(acc, amnt, \texttt{Err})$$

Program implementation.

$$\forall acc, amnt \in \mathbb{Z},$$

$$debit\_impl(acc, amnt) = \begin{cases} acc - amnt & \texttt{if } (acc \geq amnt > 0) \\ \texttt{Err} & otherwise \end{cases}$$

An example of program.

Proofs.

An example of program.

Proofs.

- Soundness:
  $\forall acc, amnt \in \mathbb{Z}, acc' \in \mathbb{Z} \cup \texttt{Err},$
  $debit\_impl(acc, amnt) = acc' \Rightarrow Debit(acc, amnt, acc').$

An example of program.

Proofs.

- Soundness:
  $\forall acc, amnt \in \mathbb{Z}, acc' \in \mathbb{Z} \cup \texttt{Err},$
  $debit\_impl(acc, amnt) = acc' \Rightarrow Debit(acc, amnt, acc').$
- Completeness:
  $\forall acc, amnt \in \mathbb{Z}, acc' \in \mathbb{Z} \cup \texttt{Err}$
  $Debit(acc, amnt, acc') \Rightarrow debit\_impl(acc, amnt) = acc'.$

Another example of program.

Formal specification

$$\texttt{Sum} \in \mathbb{N} \to \mathbb{N} \in \{\top, \bot\} \equiv \forall n, m \in \mathbb{N},\ m = \frac{n(n+1)}{2} \Rightarrow \texttt{Sum}(n, m)$$

Program implementation.

$$\forall n \in \mathbb{N},\ sumf(n) = \begin{cases} 0 & \texttt{if } (n = 0) \\ n + sumf(n-1) & otherwise \end{cases}$$

Another example of program.

Formal specification

$\texttt{Sum} \in \mathbb{N} \to \mathbb{N} \in \{\top, \bot\} \equiv \forall n, m \in \mathbb{N},\ m = \frac{n(n+1)}{2} \Rightarrow \texttt{Sum}(n, m)$

or

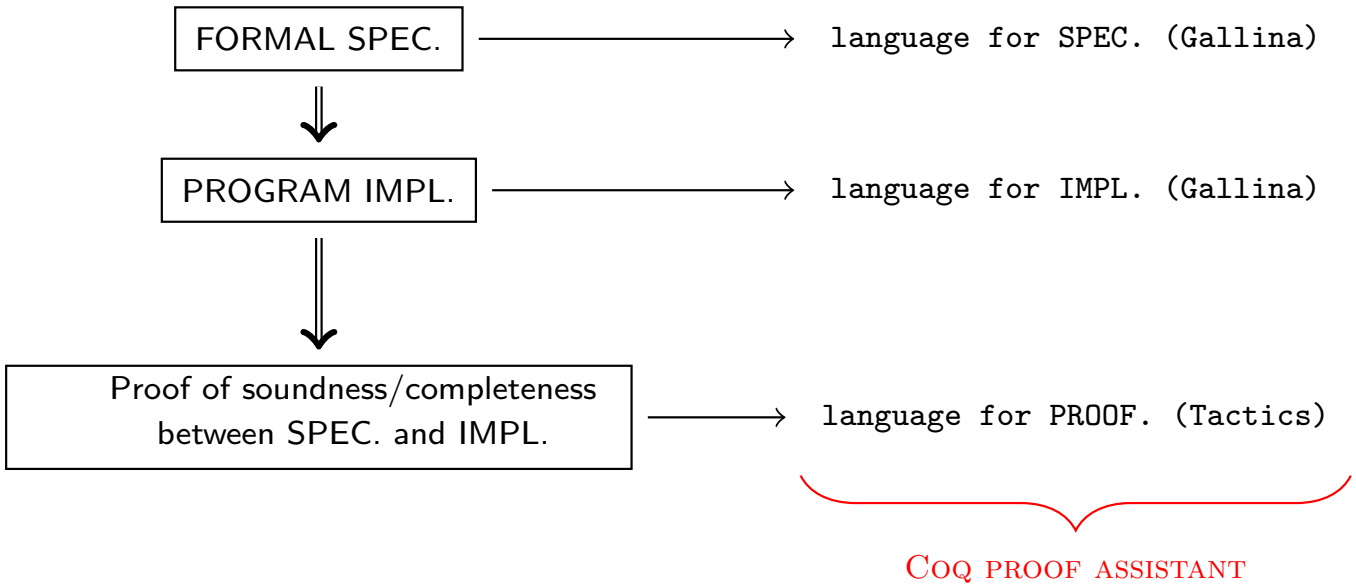
$\forall n, m \in \mathbb{N},\ \texttt{Sum}(n, m) \equiv m = \frac{n(n+1)}{2}$

Program implementation.

$$\forall n \in \mathbb{N},\ sumf(n) = \begin{cases} 0 & \texttt{if } (n = 0) \\ n + sumf(n-1) & otherwise \end{cases}$$

The Coq proof assistant.

Coq is one formal proof verifier among many (Isabelle, HOL, PVS...).



The Coq proof assistant.

Coq is one formal proof verifier among many (Isabelle, HOL, PVS...).

Use cases [Leroy, 2014].

- Basic mathematics theories (geometry, algebra...):
  e.g, *Four Color Theorem proof* [Gonthier, 2008].
- Complex algorithms, and protocols verification:
  e.g, *Boyer-Moore majority vote algorithm*, formalized by C.Paulin-Mohring.
- Program verification (written in Coq!).

Coq demo.

Conclusion.

- Real need for formal methods for safety-critical softwares (especially as software *complexity increases*).

Conclusion.

- Real need for formal methods for safety-critical softwares (especially as software *complexity increases*).
- Adoption of formal methods, a 2-axis process:
  1. Training a qualified workforce.
  2. Helping uninitiated developers (automatic theorem provers for main-stream programming languages).
     e.g the *Frama-C* framework [Cuoq et al., 2012].

Conclusion.

- Real need for formal methods for safety-critical softwares (especially as software *complexity increases*).
- Adoption of formal methods, a 2-axis process:
  1. Training a qualified workforce.
  2. Helping uninitiated developers (automatic theorem provers for main-stream programming languages).
     e.g the *Frama-C* framework [Cuoq et al., 2012].
- *Formal methods are not only for software engineering!*

  e.g. Applications to digital architecture design.

Bibliography.

Abran, A., editor (2004).

Guide to the software engineering body of knowledge, 2004 version: SWEBOK ; a project of the IEEE Computer Society Professional Practices Committee.

IEEE Computer Society, Los Alamitos, Calif.

OCLC: 934432015.

Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., and Yakobowski, B. (2012).

Frama-C.

In Eleftherakis, G., Hinchey, M., and Holcombe, M., editors, *Software Engineering and Formal Methods*, Lecture Notes in Computer Science, pages 233–247, Berlin, Heidelberg. Springer.

Gonthier, G. (2008).

The Four Colour Theorem: Engineering of a Formal Proof.

In Kapur, D., editor, *Computer Mathematics*, Lecture Notes in Computer Science, pages 333–333, Berlin, Heidelberg. Springer.

Leroy, X. (2014).

Spcification et vrification formelles avec l'assistant la preuve Coq.

Neumann, P. (1995).

Computer-related risks.

ACM Press ; Addison-Wesley, New York, New York : Reading, Mass.