Les méthodes formelles

État de l'art

Baptiste Pollien

2 juillet 2021

COVNI 2021

Développement d'un système

Le développement d'un système peut être divisé en 3 étapes :

1. **Spécification** des besoins fonctionnels et des contraintes.
2. **Implémentation** du système.
3. **Vérification** que le système correspond bien à la spécification.

Techniques de vérification :

- Revue de code,
- Tests,
- Méthodes formelles.

Les méthodes formelles

Les méthodes formelles

- Techniques de vérification basées sur des modèles mathématiques
- Utilisables dans l'avionique avec les normes DO-178C et DO-333
- Exemple : interprétation abstraite, méthodes déductives, model-checking

Objectifs de ma thèse

- Définir des processus de vérification avec des méthodes formelles,
- Appliquer ces méthodes à un autopilote de drone : Paparazzi.

Vérification d'un système

Deux types de propriétés qu'on souhaite vérifier :

- Propriétés **générales** : *erreurs à l'exécution, livelock, deadlock...*
- Propriétés **spécifiques** : *garantir des propriétés fonctionnelles.*

Familles de méthodes de vérification de programme :

- Statique : *interprétation abstraite...*
- Dynamique : *runtime monitoring...*

Limitation

Théorème de Rice

Toute propriété non triviale sur des programmes est indécidable.

Face à un problème indécidable, on peut abandonner :

- la terminaison ;
- la complétude ;
- l'automaticité.

D'où des compromis sur les outils de preuve de programme :

- entre « puissance » de l'outil et automaticité ;
- entre charge de travail du développeur et de l'utilisateur.

Interprétation abstraite

Interprétation abstraite

Objectifs : Vérifier qu'un état de la mémoire $x \in \mathcal{D}$ n'est pas accessible.
Exemple de domaine concret : $\mathcal{D} = 2^{(\mathbb{V} \rightarrow \mathbb{Z})}$

Idéalement,

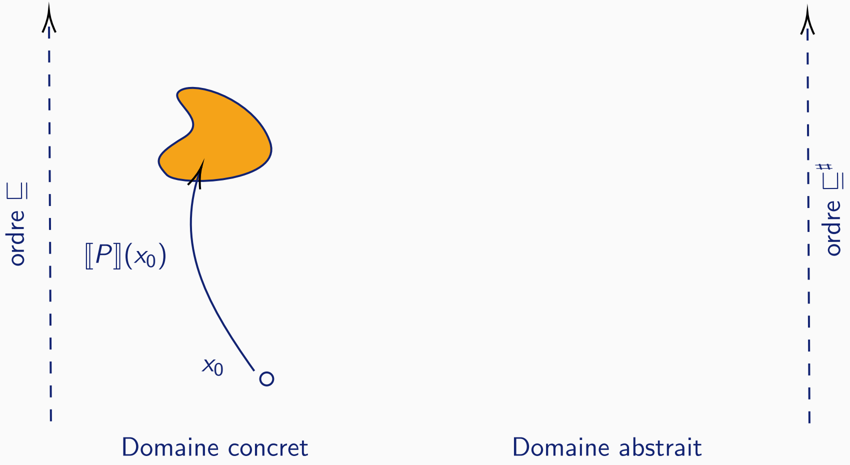
- à chaque point P du programme,
- on calcule l'ensemble des états possibles : $[\![P]\!](x_0),\ x_0 \in \mathcal{D}$
- et on vérifie si l'état x est accessible : $x \in [\![P]\!](x_0)$?

$\Longrightarrow$ Problème non décidable !

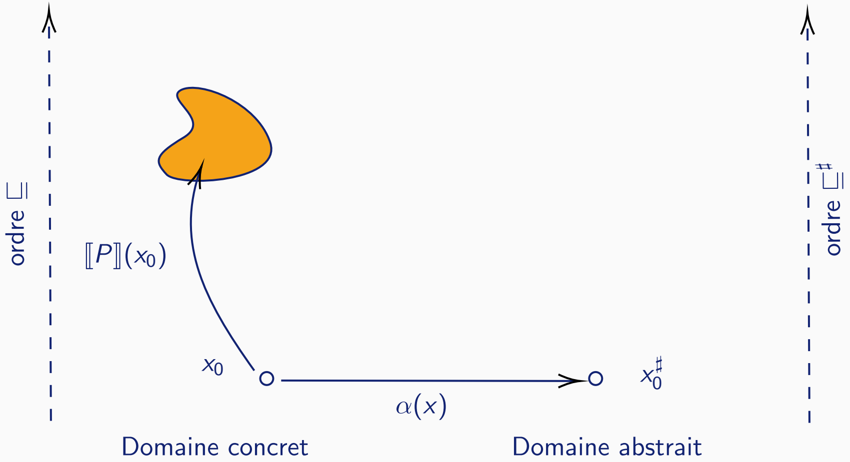
Interprétation abstraite : Abstraction des différentes valeurs possibles.

1. Définition d'un domaine abstrait : $\left(\mathcal{D}^\sharp, \sqsubseteq^\sharp_{\mathcal{D}}\right)$,
   ex : domaine des intervalles avec l'inclusion comme ordre
2. Définition des opérateurs abstraits,
   ex : $[\![v_1 + v_2]\!]^\sharp(x^\sharp),\ v_1, v_2 \in \mathbb{V}$ et $x^\sharp \in \mathcal{D}^\sharp$
3. En cas de problèmes de convergence, définition d'opérateurs de *widening*.

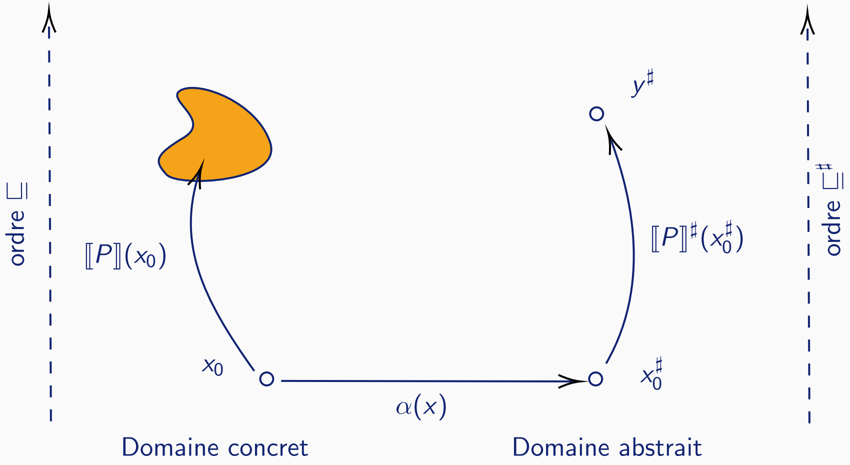
Principe de l'interprétation abstraite



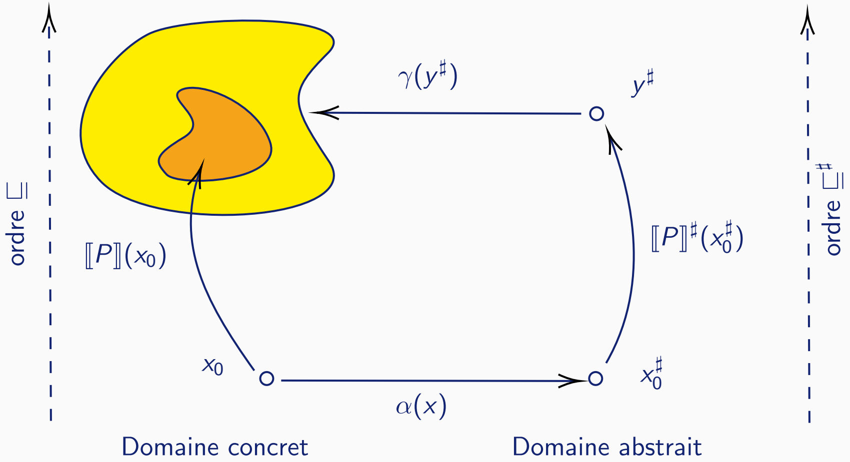
Principe de l'interprétation abstraite



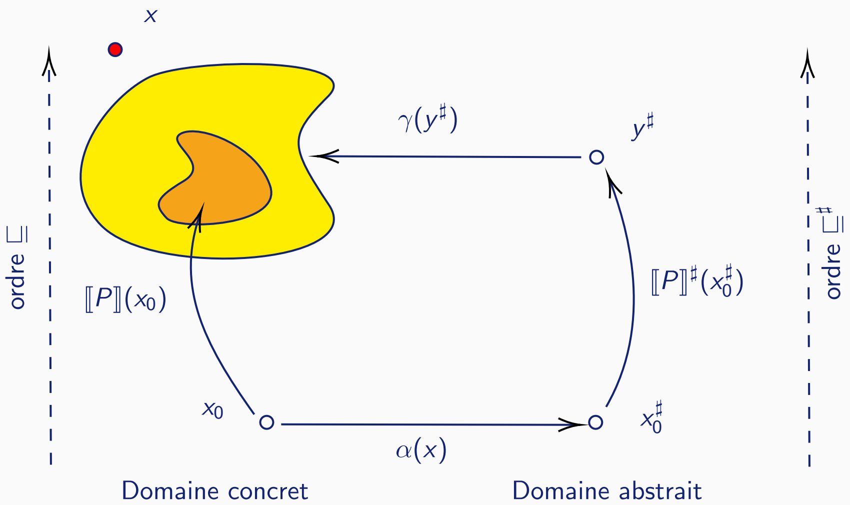
Principe de l'interprétation abstraite



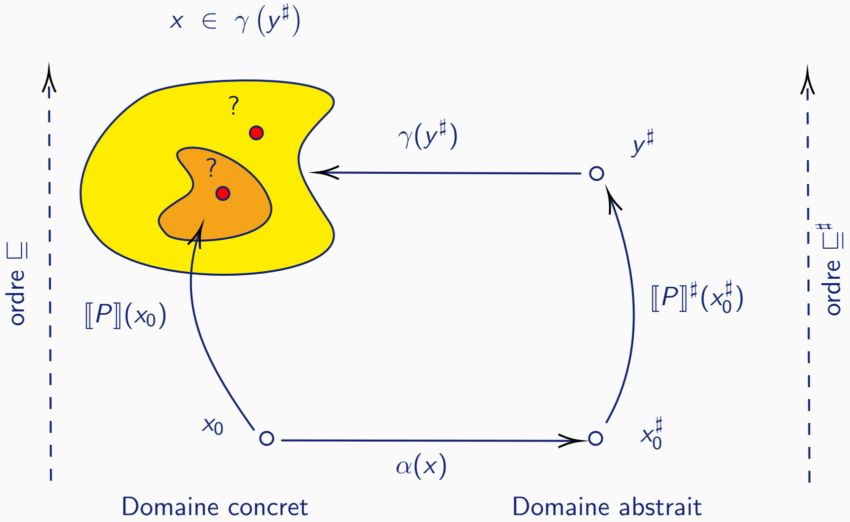
Principe de l'interprétation abstraite



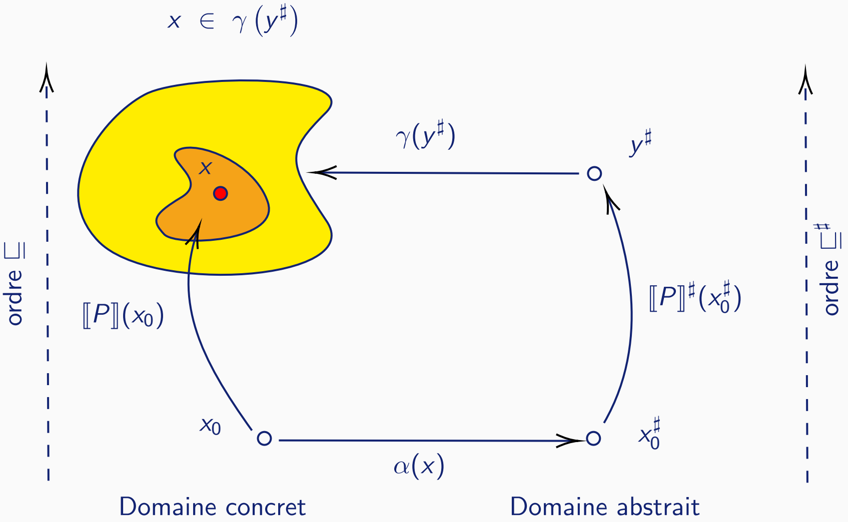
Principe de l'interprétation abstraite



Principe de l'interprétation abstraite

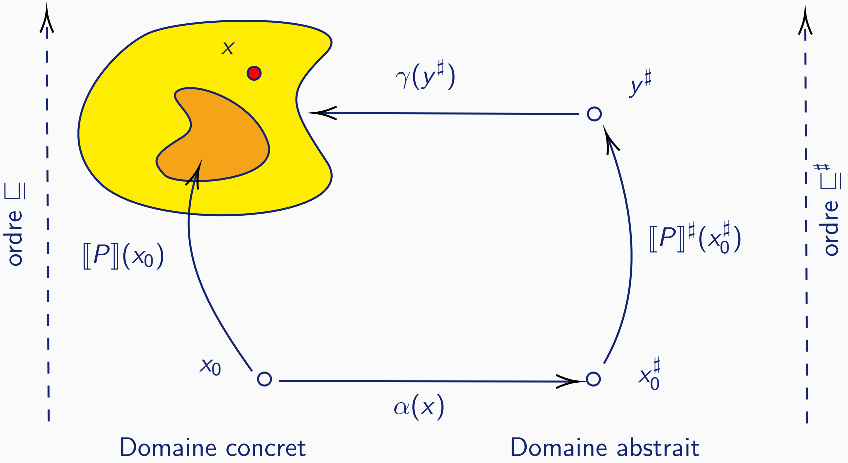


Principe de l'interprétation abstraite

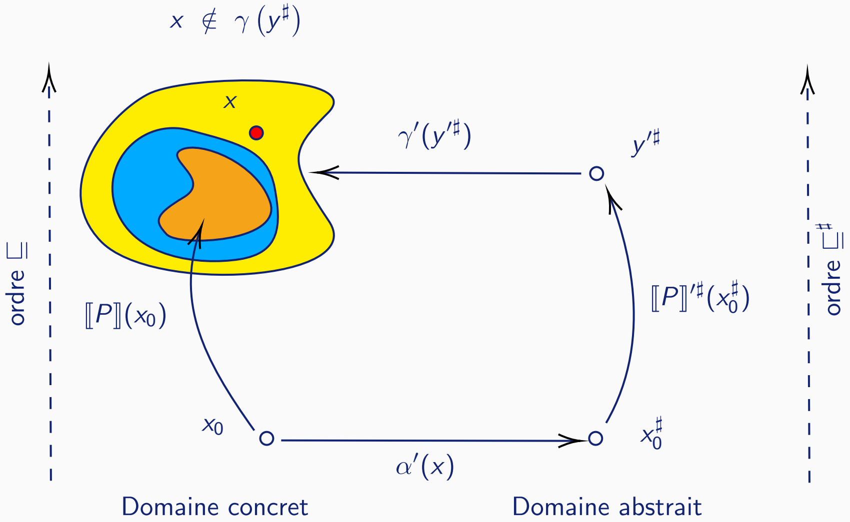


Principe de l'interprétation abstraite

$$x \in \gamma\left(y^\sharp\right)$$



Principe de l'interprétation abstraite



Domaines abstraits

Exemples de domaines abstraits non relationnels :

- Domaine des signes,
- Domaines des constantes,
- Domaine des intervalles.

Exemples de domaines abstraits relationnels :

- Domaine des polyèdres,
  ex : $a_1 v_1 + a_2 v_2 + a_3 v_3 < b,\ \ a_i, b \in \mathbb{R}, v_i \in \mathbb{V}$
- Domaines des octogones,
  ex : $\pm v_1 \pm v_2 \leq b,\ \ b \in \mathbb{R}, v_i \in \mathbb{V}$

Opérateurs abstraits

Exemples : Opérateurs abstraits génériques.

$$[\![P; Q]\!]^\sharp(x^\sharp) := [\![Q]\!]^\sharp \left([\![P]\!]^\sharp(x^\sharp)\right)$$

$$[\![E_1 + E_2]\!]^\sharp(x^\sharp) := +^\sharp \left([\![E_1]\!]^\sharp(x^\sharp), [\![E_2]\!]^\sharp(x^\sharp)\right)$$

$$[\![\textbf{if } C \textbf{ then } P \textbf{ else } Q \textbf{ fi}]\!]^\sharp(x^\sharp) := [\![P]\!]^\sharp \left([\![C]\!]^\sharp(x^\sharp)\right) \sqcup^\sharp [\![Q]\!]^\sharp \left([\![\neg C]\!]^\sharp(x^\sharp)\right)$$

Exemple : Opérateur abstrait $+$ dans le domaine des intervalles.

$$+^\sharp : (x, y) \mapsto \begin{cases} \emptyset & \text{quand } x = \emptyset \text{ ou } y = \emptyset \\ [a + c, b + d] & \text{quand } x = [a, b] \text{ et } y = [c, d] \end{cases}$$

Résumé interprétation abstraite

Interprétation abstraite (ex. : Astrée, Polyspace, Frama-C/EVA)

+ automatique
+ propriétés numériques (vs logique booléenne)
− outils complexes et spécialisés
− parfois incapable de conclure

Méthodes déductives

Methodes déductive

“*Deductive program verification is the art of turning the correctness of a program into a mathematical statement and then proving it.*”

Jean-Christophe Filliâtre dans *Deductive Program verification*.

Un système formel

- Un **langage formel** pour exprimer les propriétés à vérifier.
- Un **système déductif** (ou d'inférence) afin de construire les preuves.
  exemple :

$$\frac{A \qquad B}{A \wedge B}\ (\text{Conj})$$

Logique de Floyd-Hoare

Triplet de Hoare : le langage formel dans la logique de Floyd-Hoare

$$\{\varphi\}\ P\ \{\psi\}$$

P : un programme,

φ : **précondition** pour le programme P,

ψ : **postcondition** pour le programme P,

Système formel de Floyd-Hoare

Système d'inférence : Un ensemble d'axiomes et de règles utilisé pour dériver des expressions de la logique à partir d'autres expressions.

Exemples de règles d'inférence :

$$\frac{}{\{\varphi[v/E]\}\ \texttt{v := E}\ \{\varphi\}}\ (:=)$$

$$\frac{\{\varphi\}\ \texttt{P}\ \{\gamma\} \qquad \{\gamma\}\ \texttt{Q}\ \{\psi\}}{\{\varphi\}\ \texttt{P;Q}\ \{\psi\}}\ (\text{Seq})$$

$$\frac{\varphi \Rightarrow \varphi' \qquad \{\varphi'\}\ \texttt{P}\ \{\psi'\} \qquad \psi' \Rightarrow \psi}{\{\varphi\}\ \texttt{P}\ \{\psi\}}\ (\text{Cons})$$

$$\frac{\{\varphi \wedge C\}\ \texttt{P}\ \{\psi\} \qquad \{\varphi \wedge \neg C\}\ \texttt{Q}\ \{\psi\}}{\{\varphi\}\ \texttt{if C then P else Q fi}\ \{\psi\}}\ (\text{Cond})$$

Remarque : Il n'est pas toujours trivial de construire une preuve à partir des règles d'inférence.

WP (Weakest Precondition)

La fonction *wp* effectue un calcul de **plus faible précondition**.

$$\{wp(P, \psi)\}\ P\ \{\psi\}$$

$\Longrightarrow$ *wp* est définie telle que le triplet précédent est toujours vérifié.

Relation entre un triplet de Hoare et *wp* :

$$\{\varphi\}\ P\ \{\psi\} \Longleftrightarrow (\varphi \implies wp(P, \psi))$$

La vérification de cette formule peut être automatisée par des prouveurs.

Exemple

On cherche à prouver le triplet :

$$\{x \geq 4\}\ x := x + 1\ \{x \geq 4\}$$
$$\iff (\ (x \geq 4) \implies wp(x := x + 1, x \geq 4))$$

On a la définition suivante de *wp* pour les affectations :

$$wp(v := E,\ \psi) := \psi[v/E]$$

On calcule *wp* :

$$\begin{aligned} \implies wp(x := x + 1, x \geq 4) &= (x \geq 2)[x/x + 1] \\ &= (x + 1 \geq 4) \end{aligned}$$

Il reste donc à prouver que :

$$(x \geq 4) \implies (x + 1 \geq 4)$$
$$\iff (x \geq 4) \implies (x \geq 3)$$

$\square$

Définition de *wp*

La fonction *wp* peut être calculée à partir des formules suivantes :

$$wp(v := E,\ \psi) := \psi[v/E]$$

$$wp(P; Q,\ \psi) := wp(P, wp(Q, \psi))$$

$$\begin{aligned} wp(\texttt{if}\ C\ \texttt{then}\ P\ \texttt{else}\ Q\ \texttt{fi},\ \psi) := & \quad (C \rightarrow wp(P, \psi)) \\ \wedge & \quad (\neg C \rightarrow wp(Q, \psi)) \end{aligned}$$

wp se calcule mécaniquement, sauf pour les boucles

$$\begin{aligned} wp(\texttt{while}\ C\ \texttt{do}\ P\ \texttt{od}, \psi) := & \quad I \\ \wedge & \quad (C \wedge I) \rightarrow wp(P, I) \\ \wedge & \quad (\neg C \wedge I) \rightarrow \psi \end{aligned}$$

Résumé méthodes déductives

Méthodes déductives (ex. : B, Caveat, Frama-C/WP)

+ outils beaucoup plus génériques

− moins automatique

MBSE

MBSE

MBSE : *Model Based Systems Engineering*

- Méthode de développement basée sur l'utilisation de modèles,
- Modélisation des systèmes à différents niveaux de complexité.
  ex : interaction entre les composants matériels ou logiciels.

Les modèles peuvent être représentés :

- semi formellement : UML, SysML. . .
- formellement : automates finis, réseaux de Pétri. . .

Spécification et vérification d'un modèle

Les spécifications utilisent des languages formels.
$\Longrightarrow$ Supprimer l'ambiguïté du langage naturel.

Exemples de langages formels utilisés :

- **Logique de Floyd-Hoare**
  - Spécification du comportement statique d'une fonction ou d'un programme,
  - Vérification de l'implémentation avec les méthodes déductives.
- **Logique temporelle**
  - Spécification de propriétés pour décrire le comportement dynamique d'un système,
  - Vérification des propriétés sur le modèle par **model-checking**,
  - Génération de code à partir du modèle.

Model-Checking

Model-Checking

Vérification de propriétés temporelles sur des automates finis ou des réseaux de Petri.

Principalement 2 types de propriétés peuvent être vérifiées :

- Propriétés de **sureté** (*safety*) : états d'erreur qui doivent être inatteignables,
- Propriétés de **vivacité** (*liveness*) : tous les états qui doivent être atteints pendant l'exécution.

Remarque : En model-checking, un "model" n'a pas le même sens qu'un modèle en MBSE.

Algorithmes de model-checking

Historiquement, des algorithmes énumératifs étaient utilisés.

- **CTL** (exécutions possibles) : Algorithme de marquage d'états,
  Exemple de propriété : $AX(\varphi \wedge \psi)$
- **PLTL** (chemins possibles) : ω-expressions régulières.
  Exemple de propriété : $G(\neg\psi \rightarrow X\psi)$

$\Longrightarrow$ **Problème d'explosion du nombre d'états**

Model-checking symbolique

Représenter symboliquement des groupes d'états et de transitions.

- **BDD** (*Binary Decision Diagrams*)
- **Formules logiques**
  - utilisées notamment pour le BMC (*Bounded Model Checking*),
  - associées à des prouveurs SAT/SMT.

Résumé model-checking

Model checking (explicite, BDD, SAT/SMT)
(ex. : Spin, NuSMV, Prover verifier, TLA+)

+ relativement automatique
+ logique booléenne (vs propriétés numériques)
− fonctionne rarement sur du code : développement manuel d'un modèle haut niveau (coûteux et potentiellement erroné)

Assistants de preuve

Assistants de preuve

Logiciel pour écrire et vérifier des preuves formelles

- Preuves de théorèmes mathématiques,
- Preuves de propriétés sur des programmes.

Assistants de preuve (ex. Coq, Isabelle/HOL, PVS, Lean)

+ très haut niveau de confiance
+ parfaitement générique
− extrêmement manuel

Projets majeurs

- CompCert : un compilateur C prouvé en Coq,
- Vélus : un compilateur Lustre prouvé en Coq,
- Sel4 : un micronoyau prouvé en Isabelle.

Conclusion

Résumé des techniques de vérification

- **Interprétation abstraite** (ex. : Astrée, Polyspace, Frama-C/EVA)
  - + automatique
  - + propriétés numériques (vs logique booléenne)
  - – outils complexes et spécialisés
  - – parfois incapable de conclure
- **Méthodes déductives** (ex. : B, Caveat, Frama-C/WP)
  - + outils beaucoup plus génériques
  - – moins automatique
- **Model checking** (ex. : Spin, NuSMV, Prover verifier)
  - + relativement automatique
  - + logique booléenne (vs propriétés numériques)
  - – fonctionne rarement sur du code : développement manuel d'un modèle haut niveau (coûteux et potentiellement erroné)
- **Assistants de preuve** (ex. Coq, Isabelle/HOL, PVS, Lean)
  - + très haut niveau de confiance
  - + parfaitement générique
  - – extrêmement manuel

Conclusion

Cette présentation est un résumé du rapport :

Formal Verification for Autopilot – Preliminary state of the art

Disponible sur HAL :

`https://hal.archives-ouvertes.fr/hal-03255656/document`

Merci de votre attention

References i

P. Cousot and R. Cousot.
Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *POPL*, pages 238–252, 1977.

J.-C. Filliâtre.
Deductive Program Verification.
Thèse d'habilitation, Université Paris-Sud, Dec. 2011.

C. A. R. Hoare.
An axiomatic basis for computer programming.
Commun. ACM, 12(10) :576–580, 1969.