« Interprétation abstraite : application aux logiciels de l'A380 »

Patrick Cousot
Professeur d'informatique
École normale supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France

Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Exposé sur des questions d'actualité — Académie des Sciences — 6 juin 2006

Résumé

Les logiciels complexes comportant quasiment tous des erreurs, les chercheurs ont développé des méthodes de preuve de la correction des programmes. Ceci consiste à fournir une sémantique décrivant formellement les exécutions d'un programme puis à démontrer un théorème exprimant que ces exécutions ont une certaine propriété (comme par exemple qu'un résultat attendu est fourni en un temps fini). Des résultats mathématiques fondamentaux montrent qu'il n'est pas possible de faire faire ces preuves automatiquement par des ordinateurs.

Devant cette difficulté fondamentale, l'interprétation abstraite procède par approximation correcte de la sémantique. Si l'approximation est suffisamment grossière, elle est calculable par un ordinateur. Si elle est suffisamment fine, elle permet d'obtenir une preuve formelle de correction. L'objectif est donc de rechercher des approximations suffisamment précises et peu coûteuses à calculer.

Nous donnerons quelques éléments d'interprétation abstraite en expliquant comment formaliser l'abstraction de propriétés de sémantiques pour obtenir des approximations calculables conduisant à des algorithmes effectifs d'analyse statique des comportements possibles des programmes.

Finalement nous montrerons un exemple d'application de la théorie à la preuve de l'absence d'erreurs à l'exécution sur des programmes de contrôle/commande synchrones en soulignant les difficultés (comme le calcul flottant). Cette approche a été appliquée avec succès à la vérification du logiciel de commande de vol électrique de l'A380.

Abstract

Since almost any complex software has bugs, researchers have developed program correctness proof methods. This consists in defining a semantics formally describing the executions of a program and then in proving a theorem stating that these executions have a given property (for example that an expected result is provided in a finite time). Fundamental mathematical results show that these proofs cannot be done automatically by computers.

Confronted with this fundamental difficulty, abstract interpretation proceeds by correct approximation of the semantics. If the approximation is coarse enough, it is computable. If it is precise enough, it yields a correctness proof. The goal is therefore to find cheap approximations which are precise enough.

We will introduce a few elements of abstract interpretation and explain how to formalize the abstraction of semantic properties so as to obtain computable approximations leading to effective algorithms for the static analysis of the possible behaviours of programs.

Finally, we will describe an example of application of the theory to the proof of absence of runtime errors on synchronous control/command and underly the difficulties (such as floating point computations). This approach was applied with success to the verification of the electric flight control of the A380.

Plan

– L'importance du logiciel
– Pourquoi les logiciels sont-ils erronés ?
– Que faire des erreurs dans les logiciels ?
– Interprétation abstraite
  - (1) introduction très informelle
  - (2) quelques éléments
  - (3) quelques applications
  - (4) application aux logiciels de l'A380
– Perspectives

L'importance du logiciel

Le logiciel se cache partout





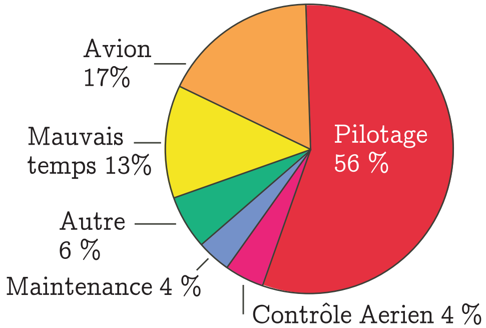


Origine des accidents (métro)

– L'accident de la ligne de métro n°12 [1] : le conducteur allait trop vite
– Le nouveau métro rapide de la ligne 14 (Météor) : complètement automatisé, pas d'opérateur


Origine des accidents (aviation)

Analyse mondiale de la première cause des accidents des vols commerciaux entre 1995 et 2004 comme déterminé par les autorités de contrôle [2] [1]



Référence

[1] D. Michaels & A. Pasztor. *Incidents Prompt New Scrutiny Of Airplane Software Giltches* citant la source Boeing. Wall Street Journal, Vol. CCXLVII, No 125, 30 mai 2006.

[1] Le 30 août 2000, la voiture motrice d'une rame du métro parisien s'est couchée à la station de métro Notre-Dame-de-Lorette, en s'arrêtant à un mètre d'une rame stationnant à quai en sens inverse (24 blessés).
[2] N'inclut que les accidents dont les causes sont connues.

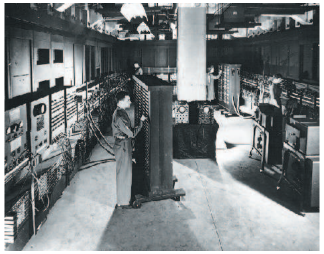
Le logiciel remplace les opérateurs humains

– Le contrôle par ordinateur est le moyen le plus sûr et le moins cher pour éviter de tels accidents
– Le logiciel est massivement présent dans tous les systèmes industriels complexes et critiques

(1) Les logiciels sont énormes

Pourquoi les logiciels sont-ils erronés ?

Alors que la capacité des matériels croît...

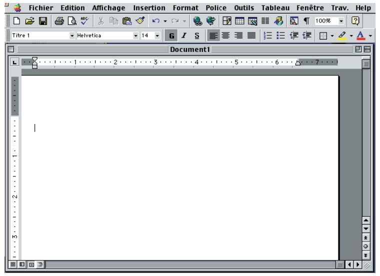


ENIAC
5,000 flops [3]

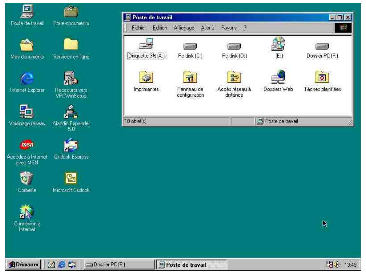
NEC Earth Simulator
35×10^{12} flops [4]

[3] Floating point operations per second (opérations flottantes par seconde)
[4] 10^{12} = mille milliards

la taille des logiciels croît...

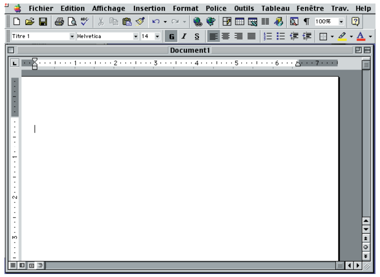


Éditeur de texte
1.700.000 lignes de C[5]



Système d'exploitation
35.000.000 de lignes de C[6]

... et donc le nombre d'erreurs croît



Éditeur de texte
1.700.000 lignes de C[5]
1.700 erreurs (estimation)

Système d'exploitation
35.000.000 de lignes de C[6]
30.000 erreurs connues

[5] 3 mois à plein temps pour une lecture rapide du code
[6] 5 ans à plein temps pour une lecture rapide du code
[5] 3 mois à plein temps pour une lecture rapide du code
[6] 5 ans à plein temps pour une lecture rapide du code

(2) Les ordinateurs sont finis

Les ordinateurs sont finis

- Les scientifiques raisonnent avec des structures mathématiques continues, infinies ($\mathbb{R}$ par exemple)
- Les ordinateurs ne peuvent calculer qu'en utilisant des structures discrètes, finies

Débordements

– Les nombres sont codés sur un nombre limité de bits (*digits binaires*)

– Certaines opérations peuvent déborder (exemple du produit d'entiers : 32 bits $\times$ 32 bits = 64 bits)

– L'utilisation de tailles de nombres différentes (16, 32, 64, ... bits) peut également être source de débordements

Le premier tir d'Ariane 5

– Premier tir d'Ariane 5, le 4 juin 1996 à Kourou

Le premier tir d'Ariane 5 se solde par un échec

– Premier tir d'Ariane 5, le 4 juin 1996 à Kourou

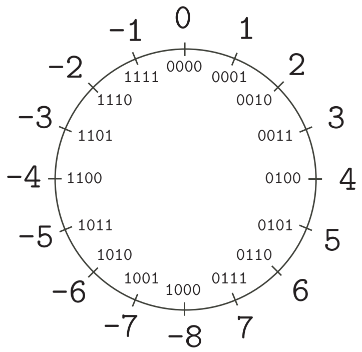
– Le lanceur fût détruit après 40 secondes de vol à cause d'un débordement logiciel[7]

[7] Du code d'Ariane 4 sur 16 bits fût réutilisé dans le code d'Ariane 5 sur 32 bits. Ceci fût à l'origine d'un débordement non traité, rendant finalement le lanceur incontrôlable.

(3) Les ordinateurs quotientent

L'arithmétique modulaire

– De nos jours, les ordinateurs évitent les débordements entiers grâce à l'arithmétique modulaire
– Exemple : codage sur 8 bits en complément à deux



L'arithmétique modulaire n'est pas très intuitive

```
# -1073741823 / -1;;
- : int = 1073741823
# -1073741824 / -1;;
- : int = -1073741824
```

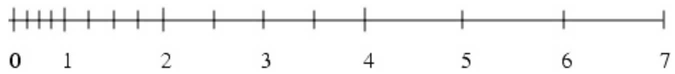

(4) Les ordinateurs arrondissent

Nombres flottants

– Les réels sont appliqués sur les flottants (arithmétique flottante)

$$\pm d_0.d_1 d_2 \ldots d_{p-1} \beta^e$$ [8]

– Par exemple sur 6 bits (avec $p = 3$, $\beta = 2$, $e_{\min} = -1$, $e_{\max} = 2$), il y a 32 nombres flottants normalisés. Les 16 nombres positifs sont



[7] où - $d_0 \neq 0$,
- p est le nombre de digits significatifs,
- β est la *base* (2), et
- e est l'*exposant* ($e_{\min} \leq e \leq e_{\max}$)

Arrondis

- Les résultats des calculs retournant des réels qui ne sont pas des flottants, doivent être arrondis
- La plupart des identités mathématiques sur $\mathbb{R}$ ne sont plus valides sur les flottants
- Les erreurs d'arrondi peuvent se compenser ou s'accumuler dans de longs calculs
- Des calculs convergeant dans les réels peuvent diverger sur les flottants (et finalement déborder)

Exemple d'erreur d'arrondi

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```

$(x + a) - (x - a) \neq 2a$

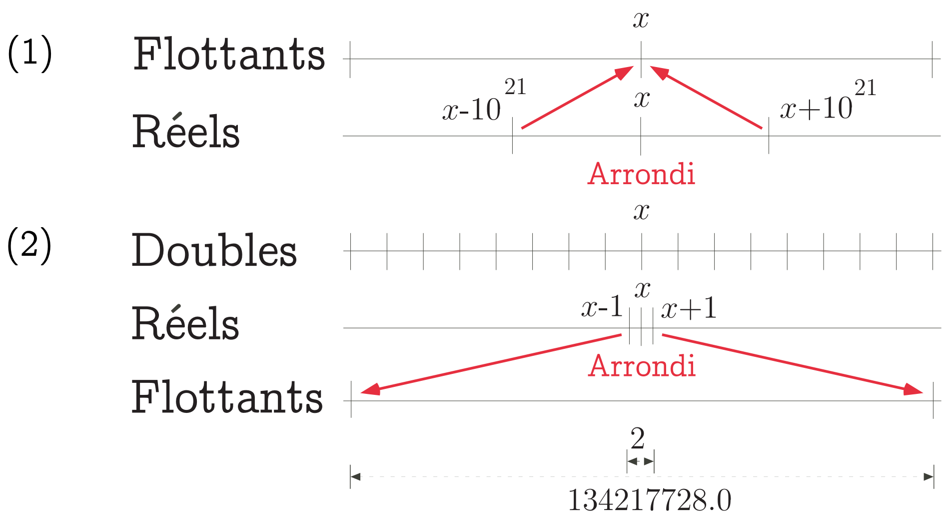
Exemple d'erreur d'arrondi

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
}
% gcc float-error.c
% ./a.out
0.000000
```

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951487.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
0.000000
```

$(x + a) - (x - a) \neq 2a$

Explication de l'énorme erreur d'arrondi



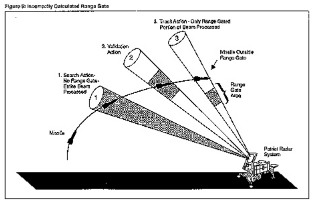
Exemple d'accumulation d'une petite erreur d'arrondi

```
% ocaml
        Objective Caml version 3.08.1
# let x = ref 0.0;;
val x : float ref = {contents = 0.}
# for i = 1 to 1000000000 do
      x := !x +. 1.0/.10.0
  done; x;;
- : float ref = {contents = 99999998.7454178184}
```

since $(0.1)_{10} = (0.0001100110011001100\ldots)_2$

L'échec de l'antimissile « Patriot »

- "Le 25 février 1991, un antimissile « Patriot » … échoua dans la poursuite et l'interception d'un missile « Scud »[9]."
- L'erreur logicielle était due à un cumul d'erreurs d'arrondi[10]



[8] Ce missile « Scud » explosa ensuite sur une baraque, tuant 28 soldats.

[9]
- "Le temps est conservé continuellement par l'horloge interne du système en dixièmes de secondes"
- "Le système était en fonction depuis plus de 100 heures consécutives"
- "À cause de ce long fonctionnement ininterrompu, l'imprécision résultante dans le calcul du temps fut à l'origine d'un décalage de la fenêtre de suivi de sorte que le système ne put suivre correctement l'arrivée du Scud"

Que faire des erreurs dans les logiciels ?

Absence de Garantie

Extrait d'une licence pour les logiciels libres :

LIMITATION DE GARANTIE

Article 11.

Parce que l'utilisation de ce Programme est libre et gratuite, aucune garantie n'est fournie, comme le permet la loi. Sauf mention écrite, les détenteurs du copyright et/ou les tiers fournissent le Programme en l'état, sans aucune sorte de garantie explicite ou implicite, y compris les garanties de commercialisation ou d'adaptation dans un but particulier. Vous assumez tous les risques quant à la qualité et aux effets du Programme. Si le Programme est défectueux, Vous assumez le coût de tous les services, corrections ou réparations nécessaires.

On n'a rien pour rien !

Absence de Garantie

Extrait d'une licence d'un logiciel commercial :

> EXCLUSION DE GARANTIE. . . . *DANS TOUTE LA MESURE PERMISE PAR LA RÉGLEMENTATION APPLICABLE, MICROSOFT ET SES FOURNISSEURS VOUS FOURNISSENT LES COMPOSANTS OS, AINSI QUE, LE CAS ÉCHÉANT, TOUT SERVICE D'ASSISTANCE RELATIF À CES COMPOSANTS (LES "SERVICES D'ASSISTANCE"), EN L'ÉTAT, AVEC LEURS IMPERFECTIONS. EN OUTRE, MICROSOFT ET SES FOURNISSEURS EXCLUENT PAR LES PRÉSENTES TOUTE AUTRE GARANTIE LÉGALE, EXPRESSE OU IMPLICITE RELATIVE AUX COMPOSANTS OS ET AUX SERVICES D'ASSISTANCE, NOTAMMENT (LE CAS ÉCHÉANT), TOUTE GARANTIE : DE PROPRIÉTÉ, D'ABSENCE DE CONTREFAÇON, DE QUALITÉ, D'ADÉQUATION À UN USAGE PARTICULIER, D'ABSENCE DE VIRUS, DE PRÉCISION, D'EXHAUSTIVITÉ DES RÉPONSES, DES RÉSULTATS OBTENUS, D'ABSENCE DE NÉGLIGENCE, OU DE DÉFAUT DE FABRICATION, DE JOUISSANCE PAISIBLE, D'ABSENCE DE TROUBLE DE POSSESSION ET DE CONFORMITÉ À LA DESCRIPTION. VOUS ASSUMEZ L'ENSEMBLE DES RISQUES DÉCOULANT DE L'UTILISATION OU DU FONCTIONNEMENT DES COMPOSANTS OS ET DES SERVICES D'ASSISTANCE. . . .*

On n'a rien de plus pour son argent !

Méthodes traditionnelles de validation de logiciels

– La loi n'impose pas plus que les « meilleures pratiques »
– Les méthodes manuelles de validation de logiciels (revues de code, simulations, tests, etc.) ne passent pas à l'échelle
– La capacité intellectuelle des informaticiens reste essentiellement la même
– La taille des équipes de programmation ne peut pas croître significativement sans perte d'efficacité

Les mathématiques et les ordinateurs peuvent aider

– Les comportements des logiciels peuvent être formalisés mathématiquement → sémantique
– Les ordinateurs peuvent faire des vérifications basées sur la sémantique → analyse statique
– Les ordinateurs sont finis avec des limitations intrinsèques (indécidabilité, complexité) → l'automatisation totale de vérifications quelconques est impossible
– Les approximations sémantiques sont nécessaires → c'est l'idée de l'interprétation abstraite

Interprétation abstraite

Il y a deux concepts fondamentaux en informatique (et en sciences en général) :

– L'**abstraction** : pour raisonner sur des systèmes complexes
– L'**approximation** : pour rendre effectifs des calculs indécidables

Ces concepts sont formalisés par l'interprétation abstraite

Références

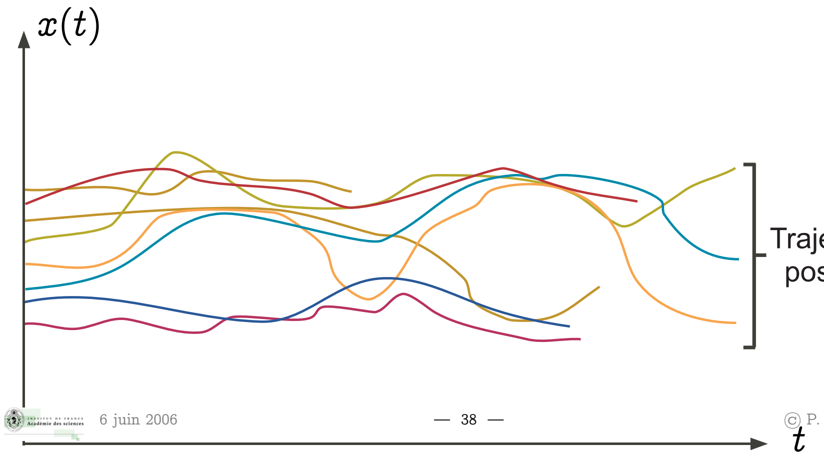
[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *ACM POPL*.

[Thesis '78] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse ès sci. math. Grenoble, march 1978.

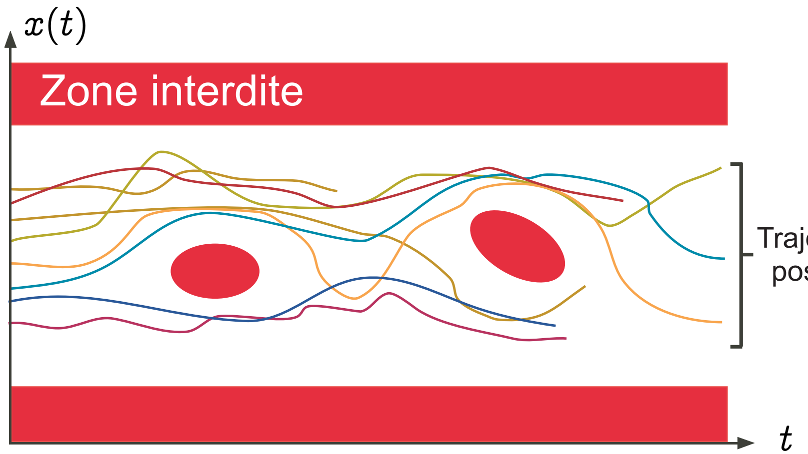
[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *ACM POPL*.

Interprétation abstraite (1) introduction très informelle

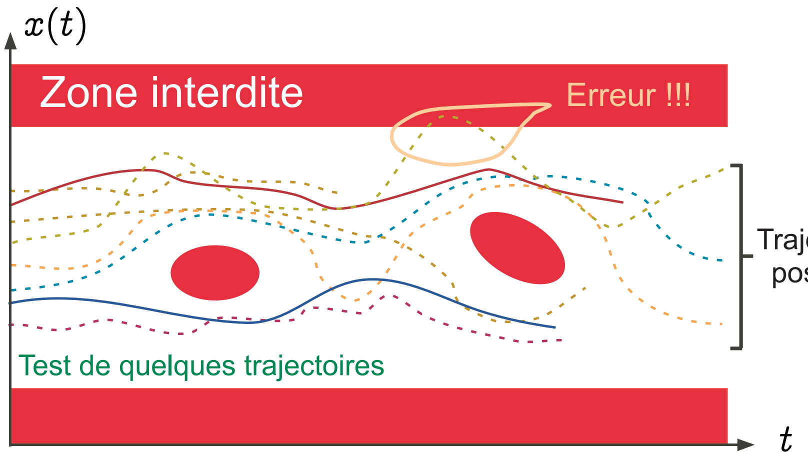
Sémantique opérationnelle



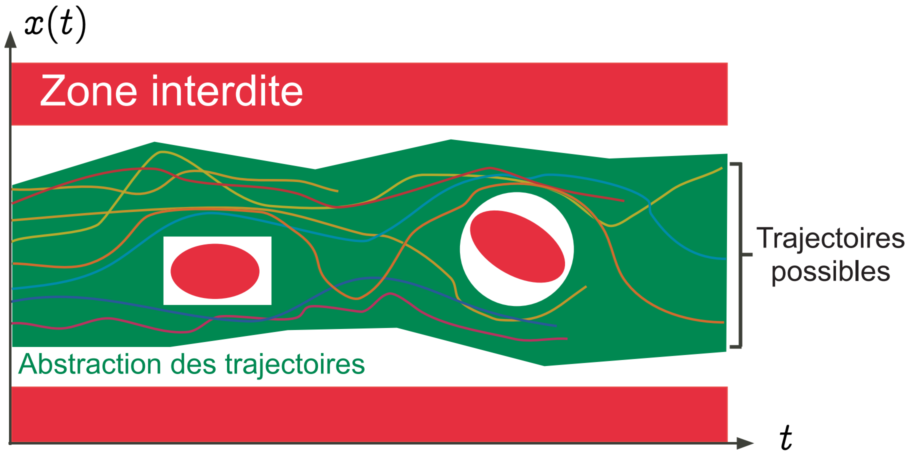
Propriétés de sûreté



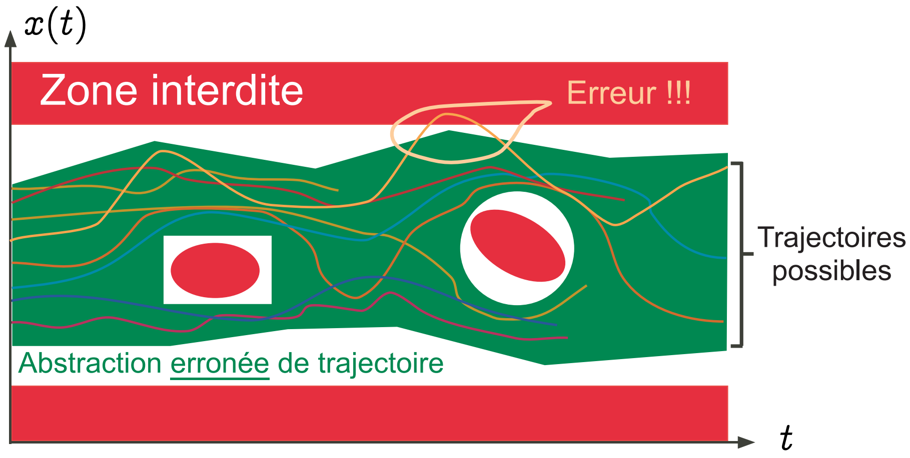
Le test n'est pas sûr



L'interprétation abstraite est sûre

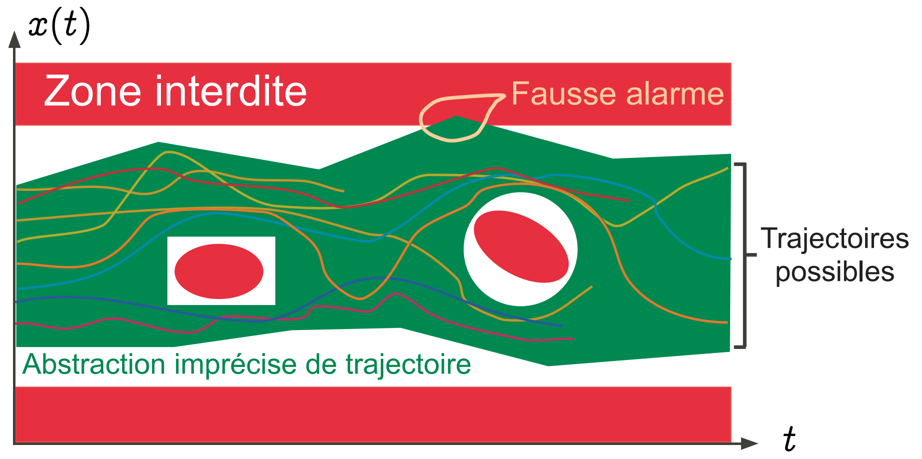


Exigence de sûreté : abstraction erronée [11]

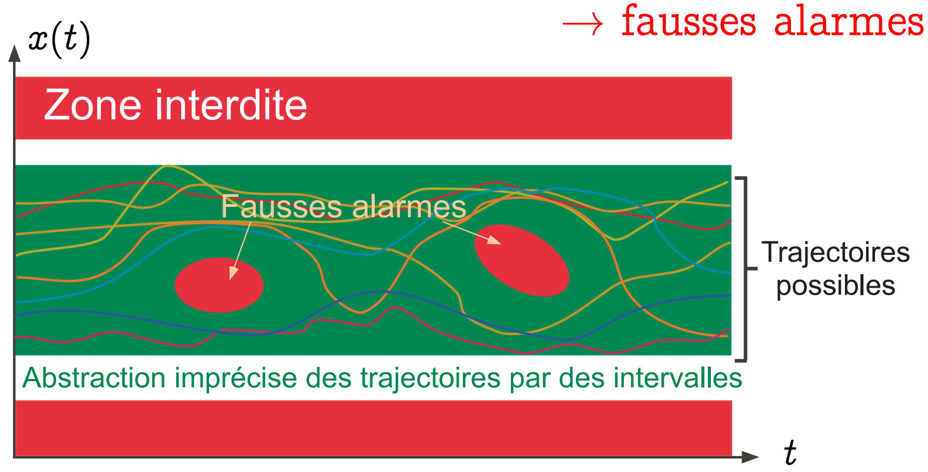


10 Cette situation est toujours exclue en analyse statique par interprétation abstraite.

L'imprécision entraîne des fausses alarmes

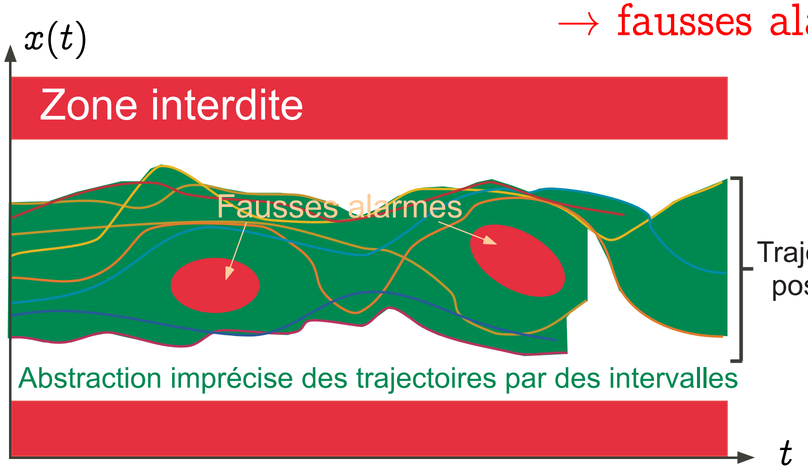


Abstraction globale par intervalles $\rightarrow$ fausses alarmes



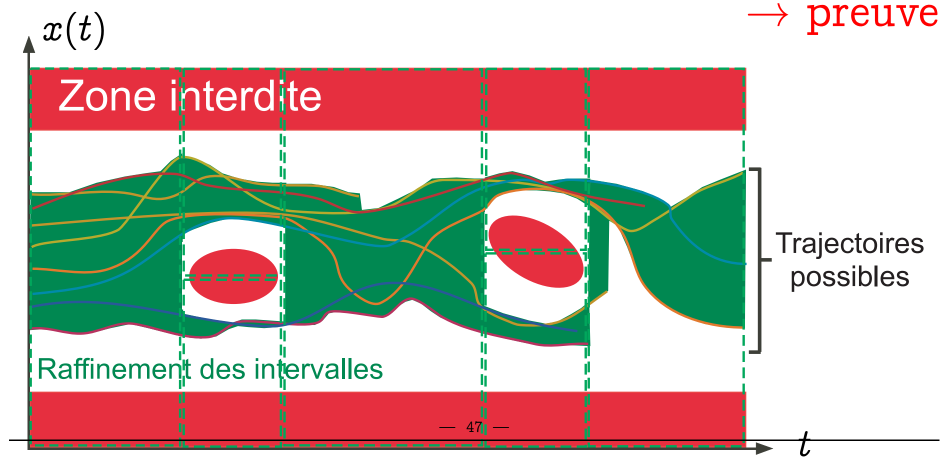
Abstraction locale par intervalles

$\longrightarrow$ fausses alarmes

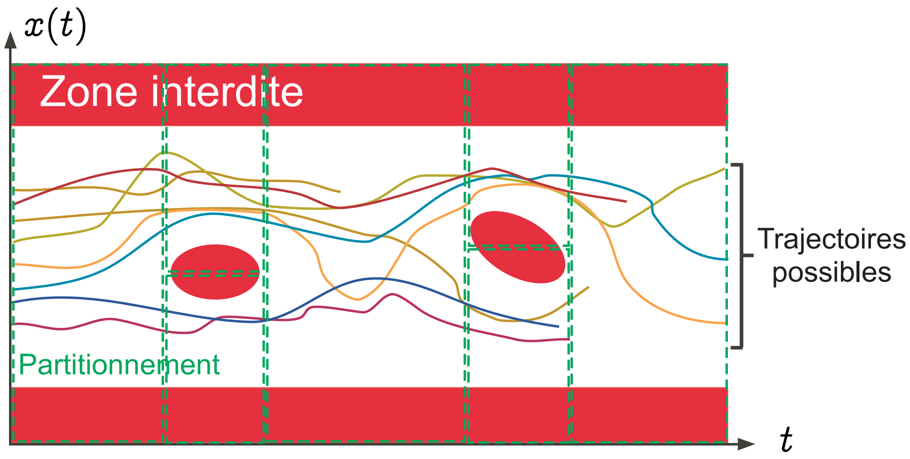


Intervalles avec partitionnement

$\longrightarrow$ preuve



Raffinement par partitionnement



Interprétation abstraite (2) quelques éléments

(2.1) Sémantique des programmes

Description d'un pas de calcul

– Système de transition $\langle \Sigma,\ \tau \rangle$, états $\Sigma = \{\bullet, \ldots, \odot \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \odot \ldots\}$

– Exemple

- États : $\langle p,\ v \rangle$, p est un point de programme, v est la valeur des variables
- Transitions $\langle p,\ v \rangle \longrightarrow \langle p',\ v' \rangle$ pour l'affectation :

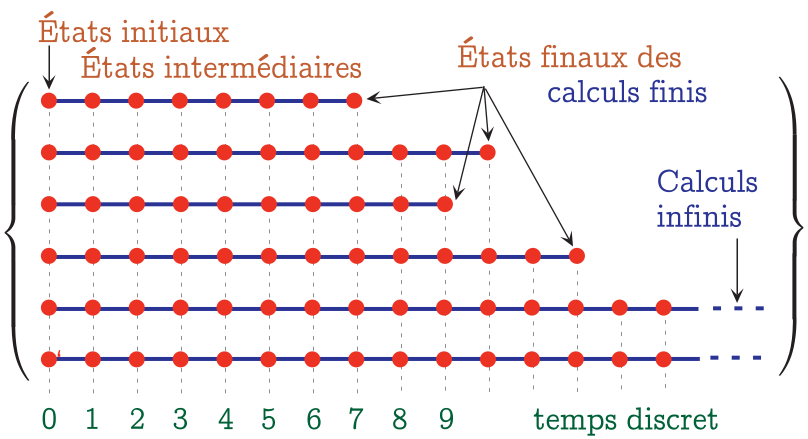
```
p:
   X = X + 1;
p'
```

$v'(\texttt{X}) = v(\texttt{X}) + 1$ si $v(\texttt{X}) < \texttt{maxint}$

$v'(\texttt{Y}) = v(\texttt{Y})$ si $\texttt{Y} \neq \texttt{X}$

État de blocage ($\odot$) si $v(\texttt{X}) \geqslant \texttt{maxint}$.

Description d'un calcul complet par une trace



États $\Sigma = \{\bullet, \ldots, \odot \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \odot \ldots\}$

Sémantique de traces par plus petit point fixe

$$\begin{aligned}
\textbf{Traces} = \ & \{\odot \mid \odot \text{ est un état final}\} \\
\cup\ & \{\bullet \longrightarrow \bullet \longrightarrow \ldots \longrightarrow \odot \mid \bullet \longrightarrow \bullet \text{ est un pas de calcul } \& \\
& \quad \bullet \longrightarrow \ldots \longrightarrow \odot \in \textbf{Traces}^{+}\} \\
\cup\ & \{\bullet \longrightarrow \bullet \longrightarrow \ldots \longrightarrow \ldots \mid \bullet \longrightarrow \bullet \text{ est un pas de calcul } \& \\
& \quad \bullet \longrightarrow \ldots \longrightarrow \ldots \in \textbf{Traces}^{\infty}\}
\end{aligned}$$

- En général, l'équation a plusieurs solutions
- Choisir la plus petite pour l'ordre de calcul:

« *plus de traces finies & moins de traces infinies* ».

Calcul itératif de la sémantique de traces

Itérés	Traces finies	Traces infinies
F^0	$\emptyset$	$\{\bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \cdots\}$
F^1	$\{\bullet_0\}$	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \cdots\}$
F^2	$\{\bullet_0, \bullet_0 \xrightarrow{\tau} \bullet_1\}$	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \xrightarrow{\tau} \bullet_2 \xrightarrow{?} \bullet \xrightarrow{?} \bullet \xrightarrow{?} \bullet \cdots\}$
F^3	$\{\bullet_0, \bullet_0 \xrightarrow{\tau} \bullet_1, \bullet_0 \xrightarrow{\tau} \bullet_1 \xrightarrow{\tau} \bullet_2\}$	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \xrightarrow{\tau} \bullet_2 \xrightarrow{\tau} \bullet_3 \xrightarrow{?} \bullet \xrightarrow{?} \bullet \cdots\}$
$\ldots$	$\ldots$	$\ldots$
F^n	$\{\bullet_0, \ldots, \bullet_0 \xrightarrow{\tau} \bullet_1 \cdots \bullet_{n-2} \xrightarrow{\tau} \bullet_{n-1}\}$	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \cdots \bullet_{n-1} \xrightarrow{\tau} \bullet_n \xrightarrow{?} \bullet \xrightarrow{?} \bullet \cdots\}$
$\ldots$	$\ldots$	$\ldots$
F^ω	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \cdots \bullet_{n-1} \xrightarrow{\tau} \bullet_n \mid n \geqslant 0\}$	$\{\bullet_0 \xrightarrow{\tau} \bullet_1 \cdots \bullet_n \xrightarrow{\tau} \bullet_{n+1} \xrightarrow{\tau} \bullet_{n+2} \xrightarrow{\tau} \bullet_{\ldots} \cdots\}$

Sémantique de traces

– Traces d'un système de transitions $\langle \Sigma, \tau \rangle$:

finies $\Sigma^+ \stackrel{\text{def}}{=} \bigcup_{n>0} [0, n[\mapsto \Sigma$, infinies $\Sigma^\omega \stackrel{\text{def}}{=} \mathbb{N} \mapsto \Sigma$

– $\mathcal{D} = \wp(\Sigma^+ \cup \Sigma^\omega)$ domaine sémantique

– $\mathcal{S}[\![\langle \Sigma, \tau \rangle]\!] = \mathbf{pppf}^{\sqsubseteq}\ F \in \mathcal{D}$ sémantique de traces

– $F(X) = \{s \in \Sigma^+ \mid s \in \Sigma \wedge \forall s' \in \Sigma : \langle s, s' \rangle \notin \tau\}$

$\cup \{ss'\sigma \mid \langle s, s' \rangle \in \tau \wedge s'\sigma \in X\}$ opérateur

– $X \sqsubseteq Y \stackrel{\text{def}}{=} (X \cap \Sigma^+) \subseteq (Y \cap \Sigma^+) \wedge (X \cap \Sigma^\omega) \supseteq (Y \cap \Sigma^\omega)$ ordre de calcul

(2.2) Propriétés des programmes

Propriétés des programmes & analyse statique

– Une propriété de programme $\mathcal{P} \in \wp(\mathcal{D})$ est un ensemble de sémantiques pour ce programme (et donc un sous-ensemble du domaine sémantique $\mathcal{D}$)

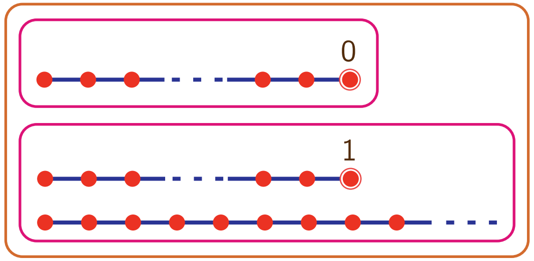
– La plus forte propriété d'un programme[12] est $\{\mathcal{S}[\![P]\!]\} \in \wp(\mathcal{D})$

– Une analyse statique consiste à approcher effectivement la plus forte propriété d'un programme :

Calculer $\mathcal{P} \in \wp(\mathcal{D})$: $\{\mathcal{S}[\![P]\!]\} \subseteq \mathcal{P}$

[11] également appelée *sémantique collective*

Exemple de propriété de programme

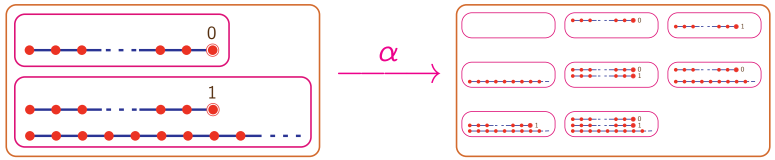


- Implantations correctes : `print 0`, `print 1`, `[print 1|loop]`, ...
- Implantations incorrectes : `[print 0|print 1]`

(2.3) Abstraction des propriétés des programmes

Abstraction

- Remplacer une propriété concrète $\mathcal{P} \in \wp(\mathcal{D})$ par une propriété abstraite $\alpha(\mathcal{P})$ approchée
- Exemple :
  - $\mathcal{D} = \wp(\Sigma^{+} \cup \Sigma^{\omega})$ domaine sémantique
  - $\mathcal{P} \in \wp(\mathcal{D})$ propriété concrète
  - $\alpha(\mathcal{P}) \stackrel{\text{def}}{=} \wp(\bigcup P)$ propriété abstraite



Propriétés communément requises pour l'abstraction

- [Dans cet exposé,] nous considérons des surapproximations : $\mathcal{P} \subseteq \alpha(\mathcal{P})$
  - Si la propriété abstraite $\alpha(\mathcal{P})$ est vraie alors la propriété concrète $\mathcal{P}$ l'est également
  - Si la propriété abstraite $\alpha(\mathcal{P})$ est fausse alors la propriété concrète $\mathcal{P}$ peut être vraie[13] ou fausse !
- Toute l'information est perdue en une seule fois:

$$\alpha(\alpha(\mathcal{P})) = \alpha(\mathcal{P})$$

- L'abstraction de propriétés plus précises est plus précise :

$$\text{si } \mathcal{P} \subseteq \mathcal{Q} \text{ alors } \alpha(\mathcal{P}) \subseteq \alpha(\mathcal{Q})$$

[12] Dans ce cas, on parle de *"fausse alarme"*.

Correspondance de Galois

– On obtient une correspondance de Galois :

$$\langle \wp(\mathcal{D}),\ \subseteq\rangle \underset{\alpha}{\overset{1}{\leftrightarrows}} \langle \wp(\mathcal{D}),\ \subseteq\rangle$$

$$\begin{array}{cc} \uparrow & \uparrow \\ \text{Propriétés concrètes} & \text{Propriétés abstraites} \end{array}$$

– Avec une représentation mathématique/informatique isomorphe :

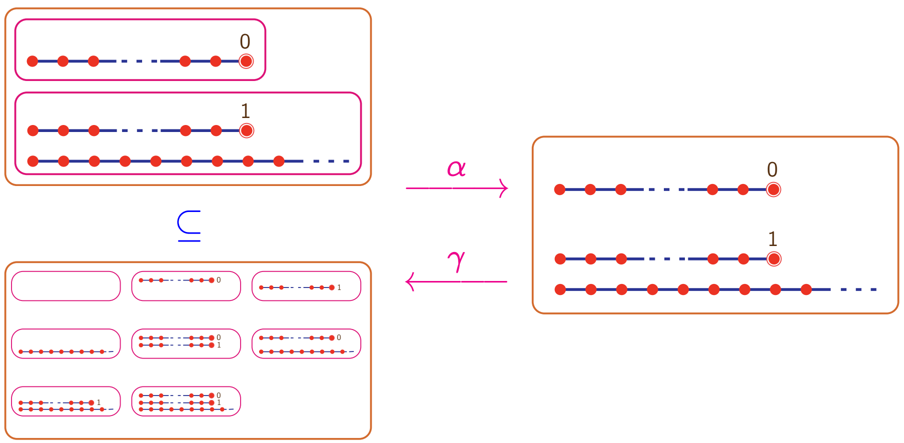
$$\langle \wp(\mathcal{D}),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp,\ \sqsubseteq\rangle$$

$$\begin{array}{cc} \uparrow & \uparrow \\ \text{Propriétés concrètes} & \text{Domaine abstrait} \end{array}$$

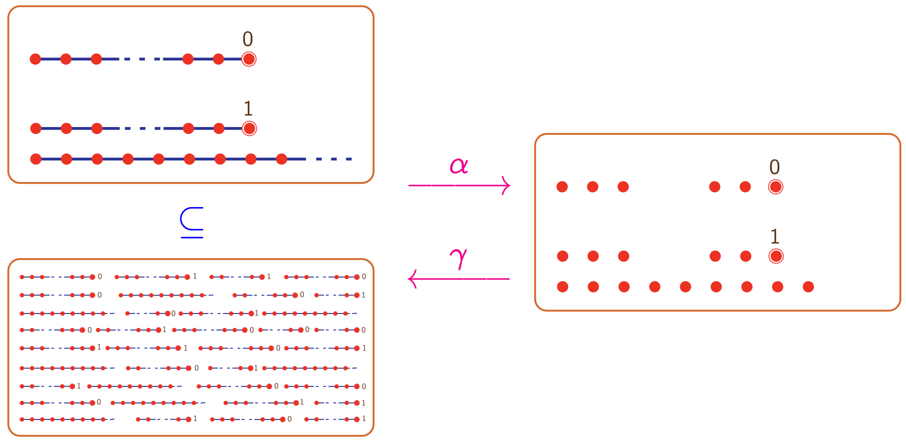
$$\forall \mathcal{P} \in \wp(\mathcal{D}) : \forall \mathcal{Q} \in \mathcal{D}^\sharp : \alpha(\mathcal{P}) \sqsubseteq \mathcal{Q} \iff \mathcal{P} \subseteq \gamma(\mathcal{Q})$$

— 61 —

Exemple 1 de correspondance de Galois



Exemple 2 de correspondance de Galois



— 63 —

Abstraction de fonctions

– Soit $\langle \wp(\mathcal{D}),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp,\ \sqsubseteq\rangle$

– Comment abstraire un *opérateur* $F \in \wp(\mathcal{D}) \overset{\leqslant}{\longmapsto} \wp(\mathcal{D})$?

– La surapproximation la plus précise est

$$F^\sharp \in \mathcal{D}^\sharp \overset{\leqslant}{\longmapsto} \mathcal{D}^\sharp$$
$$F^\sharp = \alpha \circ F \circ \gamma$$

– C'est une correspondance de Galois

$$\langle \wp(\mathcal{D}) \overset{\leqslant}{\longmapsto} \wp(\mathcal{D}),\ \subseteq\rangle \underset{\lambda F \cdot \alpha \circ F \circ \gamma}{\overset{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha}{\leftrightarrows}} \langle \mathcal{D}^\sharp \overset{\leqslant}{\longmapsto} \mathcal{D}^\sharp,\ \sqsubseteq\rangle$$

Abstraction de points fixes

– Soit $\langle \wp(\mathcal{D}),\ \subseteq \rangle \xleftarrow[\alpha]{\gamma}\!\!\to \langle \mathcal{D}^\sharp,\ \sqsubseteq \rangle$

– Comment abstraire une *propriété de point fixe* $\mathbf{pppf}^{\subseteq}\ F$ où $F \in \wp(\mathcal{D}) \stackrel{\leqslant}{\longmapsto} \wp(\mathcal{D})$?

– Abstraction correcte approchée :

$$\mathbf{pppf}^{\subseteq}\ F \subseteq \gamma(\mathbf{pppf}^{\sqsubseteq}\ \alpha \circ F \circ \gamma)$$

– Abstraction complète: si $\alpha \circ F = F^\sharp \circ \alpha$ alors

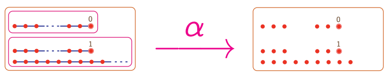
$$F^\sharp = \alpha \circ F \circ \gamma, \text{ et}$$
$$\alpha(\mathbf{pppf}^{\subseteq}\ F) = \mathbf{pppf}^{\sqsubseteq}\ F^\sharp$$

Exemple : états accessibles

– Système de transition : $\langle \Sigma,\ \tau \rangle$

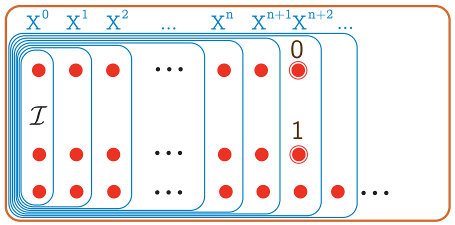
– États initiaux : $\mathcal{I} \subseteq \Sigma$

– Abstraction :



– États accessibles : $\mathbf{pppf}^{\subseteq}\ F^\sharp$,

$$F^\sharp(X) = \mathcal{I} \cup \{s' \mid \exists s \in X : \langle s,\ s' \rangle \in \tau\}$$



Accélération de la convergence du calcul itératif de points fixes

– Le point fixe $\mathbf{pppf}^{\sqsubseteq}\ F^\sharp$, $F^\sharp \in \mathcal{D}^\sharp \stackrel{\leqslant}{\longmapsto} \mathcal{D}^\sharp$ est calculé itérativement [14] :

$$X^0 = \bot \qquad X^{n+1} = F^\sharp(X^n) \qquad X^\omega = \bigsqcup_{n \geqslant 0} X^n$$

– Pour des systèmes d'équations $\mathcal{D}^\sharp = \prod_{i=1}^{n} \mathcal{D}^\sharp_n$, on utilise des itérations asynchrones

– Des techniques d'accélération de la convergence ont été développées pour approcher supérieurement la limite.

Analyse statique par interprétation abstraite

1. Définir la sémantique du language $\mathcal{S} \in \mathcal{L} \mapsto \mathcal{D}$ et les propriétés concrètes $\wp(\mathcal{D})$;
2. Soit à démontrer une propriété $\mathcal{Q} \in \wp(\mathcal{D})$ d'un programme P : $\mathcal{S}[\![P]\!] \in \mathcal{Q}$
3. Choisir l'abstraction $\langle \wp(\mathcal{D}),\ \subseteq \rangle \xleftarrow[\alpha]{\gamma}\!\!\to \langle \mathcal{D}^\sharp,\ \sqsubseteq \rangle$
4. La théorie de l'interprétation abstraite permet de construire formellement une sémantique abstraite $\mathcal{S}^\sharp[\![P]\!] \sqsupseteq \alpha(\{\mathcal{S}[\![P]\!]\})$
5. L'algorithme d'analyse statique est le calcul de la sémantique abstraite (donc correct par construction)

13 $\langle \mathcal{D}^\sharp,\ \sqsubseteq \rangle$ est un ensemble partiellement ordonné, $F^\sharp$ est croissante, $\bot$ est l'infimum, la borne supérieure $\sqcup$ doit exister pour les itérés (en général transfinis).

6. Le résultat du calcul est
   - $\mathcal{S}[\![P]\!] \in \gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}$ (preuve de correction), ou
   - $\gamma(\mathcal{S}^\sharp[\![P]\!]) \not\subseteq \mathcal{Q}$ (propriété non satisfaite ou approximation trop grossière)
7. L'abstraction doit être choisie en fonction de la propriété $\mathcal{Q}$ à démontrer
   - suffisamment grossière pour être calculable par un ordinateur,
   - suffisamment précise pour obtenir une preuve formelle de correction : $\gamma(\mathcal{S}^\sharp[\![P]\!]) \subseteq \mathcal{Q}$;

Interprétation abstraite (3) quelques applications

Applications de l'interprétation abstraite

Tout raisonnement sur des systèmes informatiques complexes se fait sur une approximation correcte de leur comportements formalisée par l'interprétation abstraite [5, 20, 21, 34]

- Syntaxe des langages de programmation [30]
- Sémantique des langages de programmation [13, 27]
- Preuves de programmes [11, 12]
- Typage et inférence de types [18]

- Analyse statique de langages [3, 7, 15, 16, 22, 26]
  - impératifs [2, 4, 6, 9, 19]
  - parallèles [10, 8]
  - logiques [14]
  - fonctionnels [17]
- Vérification de modèles[15] [23, 28, 31]
- Transformation de programmes [29]
- Stéganographie [33]
- . . .

14 « *model-checking* » en anglais.

Interprétation abstraite (4) application aux logiciels de l'A380

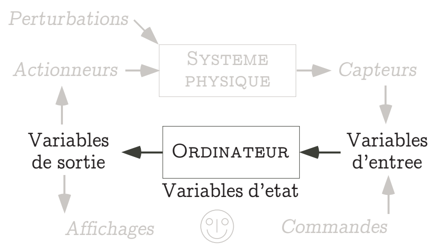
(4.1) L'analyseur statique ASTRÉE

www.astree.ens.fr [25, 32, 35]

ASTRÉE est un analyseur statique spécialisé

– Programmes C embarqués de contrôle/commande temps-réel synchrone :

```
déclarer et initialiser les variables
d'état;
loop forever
  lire les variables d'entrée volatiles,
  calculer les variables de sortie et
  d'état,
  écrire les variables de sortie;
  attendre le prochain tick d'horloge
end loop
```

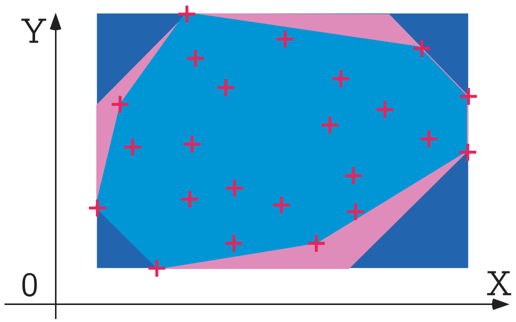


Objectif d'ASTRÉE

– Démontrer automatiquement l'absence d'erreurs à l'exécution :
  - Pas de division par 0, NaN, accès à un élément de tableau en dehors des bornes
  - Pas de débordements entiers/flottants
  - Vérification des propriétés définies par l'utilisateur (par exemple dépendant de l'architecture machine)
– Exigences :
  - efficacité (analyse faisable sur une station de travail)
  - précision (peu de fausses alarmes)
– Pas d'alarme → certification complète

(4.2) Exemples d'abstractions

Domaines abstraits numériques d'usage général



Approximation d'un ensemble de points

Intervalles : [2]

$$\bigwedge_{i=1}^{n} a_i \leqslant x_i \leqslant b_i$$

Octogones :

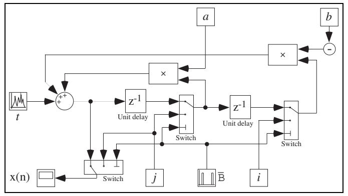
$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \pm x_i \pm y_j \leqslant a_{ij}$$

Polyèdres : [6]

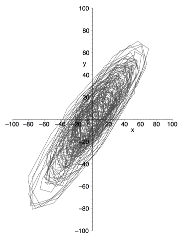
$$\bigwedge_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ji} x_i\right) \leqslant b_j$$

Domaine abstrait ellipsoïdal

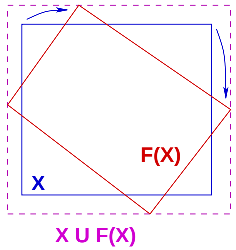
Filtre digital du 2ème ordre :



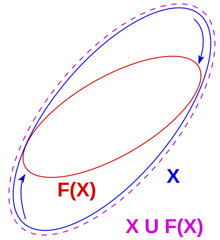
- Calcule $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- Quelle abstraction peut démontrer que le calcul est borné ?
- Pas d'intervalle, octogone ou polyèdre stable
- Le volume le plus simple est un ellipsoïde [36]



trace d'exécution



intervalle instable



ellipsoïde stable

Exemple de filtre

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
            + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```

Divergences lentes par cumuls d'arrondis

```
X = 1.0;
while (TRUE) {
  X = X / 3.0;
  X = X * 3.0;
}
```

– Sur les réels $\mathbb{R}$: $x = 1.0$ en
– Sur les flottants : erreurs d'arrondi
– Cumul des erreurs d'arrondi : cause possible de divergence

Solution [35] : borner l'erreur d'arrondi cumulée en fonction du nombre d'itérés par des progressions arithmético-géométriques :

– Relation $|\texttt{X}| \leq a \cdot b^n + c$, où a, b, c sont des constantes déterminées par l'analyse, n est le numéro d'itéré
– Nombre d'itérations borné par N : $|\texttt{X}| \leq a \cdot b^N + c$

(4.3) Résultats

Application aux A 340/ A 380

– Logiciel primaire de contrôle de vol du système de commande de vol électrique de la famille des Airbus A340 et de l'A380

– programme C, automatiquement engendré à partir d'une spécification de haut niveau (à la Simulink/SCADE)
– A340 : 100.000 à 250.000 lignes
– A380 : 400.000 à 1.000.000 lignes

Une première mondiale

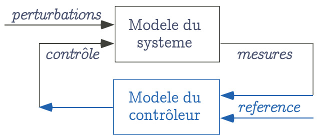
Analyse de 400.000 lignes de code C[16]

temps	mémoire	fausses alarmes
13h 52mn	2,2 Go	**0**

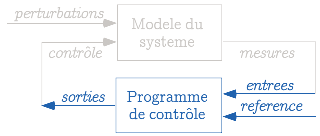
15 sur un AMD Opteron 248, 64 bits, un seul processeur

Perspectives

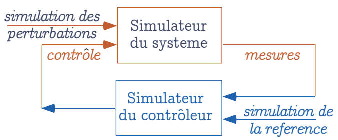
La situation actuelle [17]



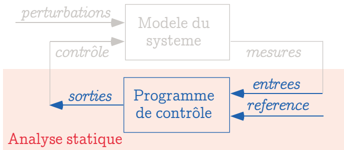
(1) Modélisation



(3) Implantation



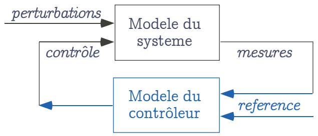
(2) Simulation



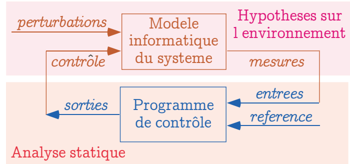
(4) Analyse du programme

16 grandement simplifié, la fiabilité par redondance est simplement ignorée !

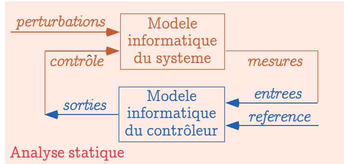
Une approche pluridisciplinaire



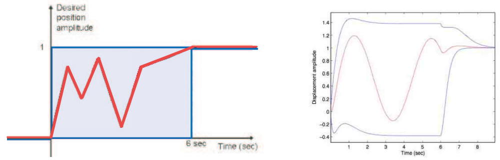
(1) Modélisation



(3) Analyse du programme



(2) Analyse du modèle



Exemple (analyse de réponse)

Fin, merci de votre attention

Références sur la toile : www.di.ens.fr/~cousot.

17 grandement simplifié, la fiabilité par redondance est simplement ignorée !

Références bibliographiques

[2] P. Cousot and R. Cousot. – Static determination of dynamic properties of programs. *In : Proceedings of the Second International Symposium on Programming*, Paris, 1976. pp. 106–130. – Dunod, Paris.

[3] P. Cousot and R. Cousot. – Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In : Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, Californie, 1977. pp. 238–252. – ACM Press, New York, New York, USA.

[4] P. Cousot and R. Cousot. – Static determination of dynamic properties of recursive procedures. *In : IFIP Conference on Formal Description of Programming Concepts, St-Andrews, N.B., Canada*, edited by E. Neuhold. pp. 237–277. – North-Holland Pub. Co., Amsterdam, Pays-Bas, 1977.

[5] P. Cousot. – *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* – Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 21 mars 1978.

[6] P. Cousot and N. Halbwachs. – Automatic discovery of linear restraints among variables of a program. *In : Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Tucson, Arizona, 1978. pp. 84–97. – ACM Press, New York, New York, USA.

[7] P. Cousot and R. Cousot. – Systematic design of program analysis frameworks. *In : Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Antonio, Texas, 1979. pp. 269–282. – ACM Press, New York, New York, USA.

[8] P. Cousot and R. Cousot. – Semantic analysis of communicating sequential processes. *In : Seventh International Colloquium on Automata, Languages and Programming*, edited by J. de Bakker and J. van Leeuwen. *Lecture Notes in Computer Science 85*, pp. 119–133. – Springer, Berlin, Allemagne, juillet 1980.

[9] P. Cousot. – Semantic Foundations of Program Analysis, chapitre invité. *In : Program Flow Analysis: Theory and Applications*, edited by S. Muchnick and N. Jones, Chapter 10, pp. 303–342. – Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1981.

[10] P. Cousot and R. Cousot. – Invariance Proof Methods and Analysis Techniques For Parallel Programs, chapitre invité. *In : Automatic Program Construction Techniques*, edited by A. Biermann, G. Guiho and Y. Kodratoff, Chapter 12, pp. 243–271. – Macmillan, New York, New York, USA, 1984.

[11] P. Cousot and R. Cousot. – 'À la Floyd' induction principles for proving inevitability properties of programs, chapitre invité. *In : Algebraic Methods in Semantics*, edited by M. Nivat and J. Reynolds, Chapter 8, pp. 277–312. – Cambridge University Press, Cambridge, Royaume Uni, 1985.

[12] P. Cousot. – Methods and Logics for Proving Programs, chapitre invité. *In : Formal Models and Semantics*, edited by J. van Leeuwen, Chapter 15, pp. 843–993. – Elsevier Science Publishers B.V., Amsterdam, Pays-Bas, 1990, *Handbook of Theoretical Computer Science*, Vol. B.

[13] P. Cousot and R. Cousot. – Inductive Definitions, Semantics and Abstract Interpretation. *In : Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Albuquerque, Nouveau Mexique, USA, 1992. pp. 83–94. – ACM Press, New York, New York, USA.

[14] P. Cousot and R. Cousot. – Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, Vol. 13, n° 2–3, 1992, pp. 103–179. – (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.)

[15] P. Cousot and R. Cousot. – Abstract Interpretation Frameworks. *Journal of Logic and Computation*, Vol. 2, n° 4, août 1992, pp. 511–547.

[16] P. Cousot and R. Cousot. – Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation, papier invité. *In : Proceedings of the Fourth International Symposium Programming Language Implementation and Logic Programming, PLILP '92*, edited by M. Bruynooghe and M. Wirsing. *Louvain, Belgique, 26–28 août 1992, Lecture Notes in Computer Science 631*, pp. 269–295. – Springer, Berlin, Allemagne, 1992.

[17] P. Cousot and R. Cousot. – Higher-Order Abstract Interpretation (and Application to Comportment Analysis Generalizing Strictness, Termination, Projection and PER Analysis of Functional Languages), papier invité. *In : Proceedings of the 1994 International Conference on Computer Languages*, Toulouse, 16–19 mai 1994. pp. 95–112. – IEEE Computer Society Press, Los Alamitos, Californie, USA.

[18] P. Cousot. – Types as Abstract Interpretations, papier invité. *In : Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, janvier 1997. pp. 316–331. – ACM Press, New York, New York, USA.

[19] P. Cousot. – The Calculational Design of a Generic Abstract Interpreter, chapitre invité. *In : Calculational System Design*, edited by M. Broy and R. Steinbrüggen, pp. 421–505. – NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, Amsterdam, Pays-Bas, 1999, Volume 173.

[20] P. Cousot. – Interprétation abstraite. *Technique et science informatique*, Vol. 19, n° 1-2-3, janvier 2000, pp. 155–164.

[21] P. Cousot. – Abstract Interpretation Based Formal Methods and Future Challenges, chapitre invité. *In : « Informatics — 10 Years Back, 10 Years Ahead »*, edited by R. Wilhelm, pp. 138–156. – Springer, Berlin, Allemagne, 2001, *Lecture Notes in Computer Science* , Vol. 2000.

[22] P. Cousot. – Partial Completeness of Abstract Fixpoint Checking, papier invité. *In : Proceedings of the Fourth International Symposium on Abstraction, Reformulation and Approximation, SARA '2000*, edited by B. Choueiry and T. Walsh, pp. 1–25. – Springer, Berlin, Allemagne, 26–29 juillet 2000, *Horseshoe Bay, Texas, USA, Lecture Notes in Artificial Intelligence 1864*.

[23] P. Cousot and R. Cousot. – Temporal Abstract Interpretation. *In : Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, USA, janvier 2000. pp. 12–25. – ACM Press, New York, New York, USA.

[24] P. Cousot and R. Cousot. – Static Analysis of Embedded Software: Problems and Perspectives, papier invité. *In : Proceedings of the First International Workshop on Embedded Software, EMSOFT '2001*, edited by T. Henzinger and C. Kirsch. *Lecture Notes in Computer Science* , Vol. 2211, pp. 97–113. – Springer, Berlin, Allemagne, 2001.

[25] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software, chapitre invité. *In : The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, edited by T. Mogensen, D. Schmidt and I. Sudborough, pp. 85–108. – Springer, Berlin, Allemagne, 2002, *Lecture Notes in Computer Science 2566*.

[26] P. Cousot and R. Cousot. – Modular Static Program Analysis, papier invité. *In : Proceedings of the Eleventh International Conference on Compiler Construction, CC '2002*, edited by R. Horspool, Grenoble, 6–14 avril 2002. pp. 159–178. – Lecture Notes in Computer Science 2304, Springer, Berlin, Allemagne.

[27] P. Cousot. – Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science*, Vol. 277, nº 1—2, 2002, pp. 47–103.

[28] P. Cousot and R. Cousot. – On Abstraction in Software Verification, papier invité. *In : Proceedings of the Fourteenth International Conference on Computer Aided Verification, CAV '2002*, edited by E. Brinksma and K. Larsen. *Copenhague, Danemark, Lecture Notes in Computer Science 2404*, pp. 37–56. – Springer, Berlin, Allemagne, 27–31 juillet 2002.

[29] P. Cousot and R. Cousot. – Systematic Design of Program Transformation Frameworks by Abstract Interrpetation. *In : Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, USA, janvier 2002. pp. 178–190. – ACM Press, New York, New York, USA.

[30] P. Cousot and R. Cousot. – Parsing as Abstract Interpretation of Grammar Semantics. *Theoretical Computer Science*, Vol. 290, nº 1, janvier 2003, pp. 531–544.

[31] P. Cousot. – Verification by Abstract Interpretation, chapitre invité. *In : Proceedings of the International Symposium on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, edited by N. Dershowitz, pp. 243–268. – Taormina, Italie, Lecture Notes in Computer Science 2772, Springer, Berlin, Allemagne, 29 juin – 4 juillet 2003.

[32] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – A Static Analyzer for Large Safety-Critical Software. *In : Proceedings of the ACM SIGPLAN '2003 Conference on Programming Language Design and Implementation (PLDI)*, San Diego, Californie, USA, 7–14 juin 2003. pp. 196–207. – ACM Press, New York, New York, USA.

[33] P. Cousot and R. Cousot. – An Abstract Interpretation-Based Framework for Software Watermarking. *In : Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Venise, Italie, 14–16 janvier 2004. pp. 173–185. – ACM Press, New York, New York, USA.

[34] P. Cousot and R. Cousot. – Basic Concepts of Abstract Interpretation% invitedchapter. *In : Building the Information Society*, edited by P. Jacquart, Chapter 4, pp. 359–366. – Kluwer Academic Publishers, Dordrecht, Pays-Bas, 2004.

[35] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival. – The ASTRÉE analyser. *In : Proceedings of the Fourteenth European Symposium on Programming Languages and Systems, ESOP '2005, Édimbourg, Écosse*, edited by M. Sagiv, pp. 21–30. – Springer, Berlin, Allemagne, 2–10 avril 2005, *Lecture Notes in Computer Science* , Vol. 3444.

[36] J. Feret. – Static analysis of digital filters. – *ESOP'04*, Barcelone, Espagne, *Lecture Notes in Computer Science* , Vol. 2986, pp. 33—-48, Springer, Berlin, Allemagne, 2004.

[37] A. Miné. – Relational abstract domains for the detection of floating-point run-time errors. *ESOP'04*, Barcelone, Espagne, *Lecture Notes in Computer Science* , Vol. 2986, pp. 3—-17, Springer, Berlin, Allemagne, 2004.