Analyse statique de logiciels :
du test exhaustif à la vérification automatique

Présentation du Séminaire

P. Cousot

DMI - École Normale Supérieure - Paris
45, rue d'Ulm, 75230 Paris cedex 05, France
<cousot@dmi.ens.fr> <http://www.dmi.ens.fr/~cousot>

Erreurs logicielles

- à peu près inévitables (rares et complexes) ;
- l'explosion de la taille des logiciels rend la vérification de leur fiabilité (test, simulation, validation de code) de plus en plus difficile et coûteuse ;
- conséquences imprévisibles voire catastrophiques (financièrement, humainement).

Notre responsabilité

- paradoxalement, la responsabilité des informaticiens n'est pas engagée (comparer à l'industrie automobile, l'aviation) ;
- l'échec informatique peut devenir une problème de société important (peur collective?).

$\Longrightarrow$ Nécessité d'élargir la panoplie de méthodes et outils utilisés pour lutter contre les erreurs logicielles.

Analyse statique de logiciels par interprétation abstraite

- Analyseur statique : outil entièrement automatisé permettant de prédire les comportements à l'exécution d'un programme quelconque par le seul examen de son texte ;
- Interprétation abstraite : méthodologie de conception d'analyseurs statiques par approximation correcte de la sémantique des programmes ;
- État de l'art :
  - bien étudié sur le plan académique,
  - tout début de l'industrialisation.

Objectifs du séminaire

- présenter les techniques d'analyse statique de logiciels par interprétation abstraite ;
- montrer leur utilisation pour la vérification formelle de logiciels ;
- montrer leur intérêt comme nouveau complément des méthodes industrielles existantes :

  test, simulation et validation de code.

Organisation de la journée

9h00–10h30 : Patrick Cousot, ENS, **Interprétation abstraite : fondements et applications** ;
10h30–10h45 : Pause ;
10h45–12h00 : Alain Deutsch, INRIA, **Vérification statique de logiciels critiques** ;
12h00–13h30 : Déjeuner ;
13h30–14h45 : Éric Goubault, CEA/Saclay, **L'interprétation abstraite au CEA** ;
14h45–16h00 : Arnaud Venet, ENS, **Analyse statique pour la sécurité du code mobile** ;
16h00–16h15 : Pause ;
16h15–17H30 : Nicolas Halbwachs, Vérimag, Grenoble, **Vérification des systèmes temporisés** ;
17H30–18H00 : *Discussion.*

Analyse statique de logiciels :
du test exhaustif à la vérification automatique

Interprétation Abstraite : Fondements et Applications

P. Cousot

DMI - École Normale Supérieure - Paris
45, rue d'Ulm, 75230 Paris cedex 05, France
<cousot@dmi.ens.fr> <http://www.dmi.ens.fr/~cousot>

Introduction

Analyseur statique

- Caractéristiques :
  - Analyseur : prédire les comportements à l'exécution d'un programme ;
  - Statique : à partir du texte du programme, sans exécution ;
  - Automatique : pas d'intervention humaine pendant l'analyse ;
- Objectifs :

  Répondre automatiquement à des questions sur les comportements possibles des programmes à l'exécution ;
- Idée de base :

  Approximation effective et pertinente de la sémantique des programmes.

Sémantique

- Sémantique d'un langage de programmation : définit la sémantique de chaque programme syntaxiquement correct du langage ;
- Sémantique d'un programme :
  - décrit formellement tous les comportements possibles du programme à l'exécution,
  - modèle mathématique qui représente les structures créées par le programme ainsi que leur évolution au cours de toutes les exécutions possibles du programme.

Difficulté essentielle : indécidabilité

- Théorèmes généraux d'indécidabilité (second théorème d'incomplétude de Gödel [1]) :

  Il est **impossible de calculer automatiquement et de manière exacte** la sémantique d'un programme [2].

1. Gödel, K., Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, *Monatshefte für Mathematik und Physik*, vol. 38 (1931), pp. 173–198.
2. Sauf évidemment pour des langages triviaux.

Comment surmonter les problèmes d'indécidabilité ?

- Tests :

  exploration (aléatoire ?) de quelques cas possibles
  → fondamentalement incomplet ;
- Vérification de modèles (model checking [3]) :

  exploration exhaustive de tous les cas possibles
  → explosion combinatoire du nombre de cas ;
- Méthodes formelles de preuve [4] :

  assistance à la preuve manuelle
  → explosion de la taille de la preuve.

3. http://www.cs.cmu.edu/~modelcheck/
4. http://www.comlab.ox.ac.uk/archive/formal-methods.html

Exemple : erreurs à l'exécution

- Une analyse par interprétation abstraite permet toujours de classer en un temps fini toutes les erreurs observées sans exception en :
  - certaines ;
  - impossibles ;
  - potentielles ← *incertitude liée à l'indécidabilité*.
- Deux caractéristiques :
  - approximatif : on ne sait pas toujours répondre affirmativement/négativement ;
  - correct : aucune erreur possible n'est oubliée parmi celles qui sont observées dans l'analyse.

Intérêt de l'analyse statique par interprétation abstraite

L'intérêt de la méthode doit être et a été justifié expérimentalement :

- coût d'analyse raisonnable (1 à 2 heures pour 10000 lignes sont des expérimentations typiques) ;
- taux d'incertitude faible (5 à 15% sont des expérimentations typiques).

Toute la recherche porte sur l'amélioration de ce rapport qualité/prix.

Comment surmonter les problèmes d'indécidabilité avec l'**interprétation abstraite ?**

L'interprétation abstraite permet de contourner les problèmes d'indécidabilité :

- en se restreignant à des ***sous-ensembles pertinents des informations observées sur les comportements des programmes*** (à la différence du model checking et des méthodes de preuve) ;
- ***sans jamais omettre aucun cas possible*** (la couverture est complète contrairement aux méthodes de test).

Pas de miracle :

- quel que soit le sous-ensemble d'informations observées par l'analyse, il existera toujours des ***questions restant sans réponse (pour cette analyse)***.

Principes de l'interprétation abstraite : Une présentation **très** simplifiée

Objectif de la théorie de l'interprétation abstraite

- Construire des analyseurs statiques cohérents et fiables
  à partir de :
  - la sémantique d'un langage de programmation,
  - la spécification des propriétés observables des comportements des programmes ;
- la théorie a d'autres champs d'application comme la conception de sémantiques [11], le typage [1], le model-checking abstrait [2], etc.

References

[1] P. Cousot. Types as abstract interpretations, papier invité. In *Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–331, Paris, France, janvier 1997. ACM Press, New York, New York, USA. http://www.dmi.ens.fr/~cousot/COUSOTpapers/POPL97.shtml 18

[2] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering Journal, special issue on Automated Software Analysis*, 6(1), 1999. À paraître. http://www.dmi.ens.fr/~cousot/COUSOTpapers/ASE-99.shtml 18

Sémantiques concrètes

Sémantique standard des programmes

- La sémantique standard d'un langage de programmation décrit l'exécution d'un programme quelconque dans un environnement d'exécution (entrées/sorties, synchronisations, ...) quelconque ;

  $\Longrightarrow$ *On définit ce qui peut se passer dans* ***chaque cas*** *possible.*

Un exemple simple de sémantique standard

```
1: X :=?; [5]
2: Y := 0;
3: while Y <= X do
4:   Y := Y + 1
   od
5:
```

↓ tirage au sort

$\langle 1, \langle \mathtt{X} = \Omega_i\ [6], \mathtt{Y} = \Omega_i \rangle\rangle \rightarrow \langle 2, \langle \mathtt{X} = 0, \mathtt{Y} = \Omega_i \rangle\rangle \rightarrow \langle 3, \langle \mathtt{X} = 0, \mathtt{Y} = 0 \rangle\rangle \rightarrow$
$\langle 4, \langle \mathtt{X} = 0, \mathtt{Y} = 0 \rangle\rangle \rightarrow \langle 3, \langle \mathtt{X} = 0, \mathtt{Y} = 1 \rangle\rangle \rightarrow \langle 5, \langle \mathtt{X} = 0, \mathtt{Y} = 1 \rangle\rangle$

...

↓ tirage au sort

$\langle 1, \langle \mathtt{X} = \Omega_i, \mathtt{Y} = \Omega_i \rangle\rangle \rightarrow \langle 2, \langle \mathtt{X} = 1073741823\ [7], \mathtt{Y} = \Omega_i \rangle\rangle \rightarrow$
$\langle 3, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 0 \rangle\rangle \rightarrow \langle 4, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 0 \rangle\rangle \rightarrow$
$\langle 3, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 1 \rangle\rangle \rightarrow \langle 4, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 1 \rangle\rangle \rightarrow$
$\langle 3, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 2 \rangle\rangle \ldots \rightarrow \langle 3, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 1073741823 \rangle\rangle \rightarrow$
blocage sur erreur Ω_a [8]

5. ? est le tirage aléatoire (par exemple lecture d'un entier en entrée).
6. Ω_i dénote la non initialisation.
7. 1073741823 est l'entier machine maximal.

Sémantique collectrice des programmes

- La sémantique concrète collectrice décrit les propriétés d'un programme quelconque valides pour toutes les exécutions du programme dans tous les environnements d'exécution (entrées/sorties, synchronisations, ...) possibles ;

  $\Longrightarrow$ *On définit ce qui se passe dans* ***tous les cas*** *possibles.*
- Les propriétés décrites dépendent des applications considérées : par exemple l'invariant le plus fort donne les valeurs possibles des variables en chaque point du programme ;
- La sémantique collectrice peut être définie rigoureusement d'un point de vue mathématique ;
- **La sémantique collectrice n'est pas calculable**.

8. Ω_a dénote un erreur de débordement arithmétique.

Un exemple simple de sémantique collectrice

1: $\{\langle \mathtt{X} = \Omega_i, \mathtt{Y} = \Omega_i \rangle\}$
`X :=?` [9] `;`
2: $\{\langle \mathtt{X} = -1073741824, \mathtt{Y} = \Omega_i \rangle, \ldots, \langle \mathtt{X} = 1073741823, \mathtt{Y} = \Omega_i \rangle\}$
`Y := 0` [10] `;`
3: $\{\langle \mathtt{X} = -1073741824, \mathtt{Y} = 0 \rangle, \ldots, \langle \mathtt{X} = 0, \mathtt{Y} = 0 \rangle, \langle \mathtt{X} = 0, \mathtt{Y} = 1 \rangle, \ldots$
$\ldots, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 0 \rangle, \ldots, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 1073741822 \rangle,$
$\langle \mathtt{X} = 1073741823, \mathtt{Y} = 1073741823 \rangle\}$
`while Y <= X do`
4: $\{\langle \mathtt{X} = 0, \mathtt{Y} = 0 \rangle, \ldots, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 0 \rangle, \ldots$
$\langle \mathtt{X} = 1073741823, \mathtt{Y} = 1073741822 \rangle, \langle \mathtt{X} = 1073741823, \mathtt{Y} = 1073741823 \rangle\}$
`Y := Y + 1`
`od`
5: $\{\langle \mathtt{X} = -1073741824, 0 \rangle, \ldots, \langle \mathtt{X} = -1, 0 \rangle, \ldots, \langle \mathtt{X} = 0, \mathtt{Y} = 1 \rangle, \ldots,$
$\langle \mathtt{X} = 1073741822, \mathtt{Y} = 1073741823 \rangle\}$

9. Le tirage au sort retourne un entier compris entre le plus petit entier machine `min_int` $= -1073741824$ et le plus grand entier machine `max_int` $= -1073741823$.
10. La valeur d'un entier est supposée toujours la même. Ceci n'est pas vrai dans certains langages comme FORTRAN : on peut passer une constante entière en paramètre d'une sous-routine qui modifie sa valeur par effet de bord.

Définition formelle de la sémantique collectrice des programmes

- La sémantique collectrice d'une programme peut être définie mathématiquement comme la plus petite solution d'un système d'équations associé au programme ;
- On peut écrire un programme donnant le système d'équations associé à un programme ;
- On ne peut pas écrire de programme pour résoudre ce système d'équations (la sémantique collectrice n'est pas calculable).

Un exemple simple d'équations collectrices

```
1: P1
   X :=?;
2: P2
   Y := 0;
3: P3
   while Y <= X do
4: P4
     Y := Y + 1
   od
5: P5
```

$$P_1 = \{\langle \mathtt{X} = \Omega_i, \mathtt{Y} = \Omega_i \rangle\}$$

$$P_2 = \{\langle \mathtt{X} = x, \mathtt{Y} = y \rangle \mid \exists x' : \langle \mathtt{X} = x', \mathtt{Y} = y \rangle \in P_1 \wedge x \in [\mathtt{min\_int}, \mathtt{max\_int}]\}$$

$$P_3 = \{\langle \mathtt{X} = x, \mathtt{Y} = 0 \rangle \mid \exists y' : \langle \mathtt{X} = x, \mathtt{Y} = y' \rangle \in P_2\} \cup \{\langle \mathtt{X} = x, \mathtt{Y} = y+1 \rangle \mid \langle \mathtt{X} = x, \mathtt{Y} = y \rangle \in P_4 \wedge y \in [\mathtt{min\_int}, \mathtt{max\_int}] \wedge y + 1 \leq \mathtt{max\_int}\}$$

$$P_4 = \{\langle \mathtt{X} = x, \mathtt{Y} = y \rangle \mid \langle \mathtt{X} = x, \mathtt{Y} = y \rangle \in P_3 \wedge x,y \in [\mathtt{min\_int}, \mathtt{max\_int}] \wedge y \leq x\}$$

$$P_5 = \{\langle \mathtt{X} = x, \mathtt{Y} = y \rangle \mid \langle \mathtt{X} = x, \mathtt{Y} = y \rangle \in P_3 \wedge x,y \in [\mathtt{min\_int}, \mathtt{max\_int}] \wedge y > x\}$$

Deux idées

1. **Simplifier les équations** collectrices → abstraction ;
2. Résoudre les équations abstraites itérativement avec **accélération de la convergence** → élargissement/rétrécissement ;

Simplification des propriétés concrètes en propriétés abstraites

Propriétés concrètes des nombres entiers

- Une propriété concrète/exacte/mathématique P des nombres entiers est codée par l'ensemble des valeurs possédant cette propriété ;
- $P \in \mathbb{P}_e$,

où

$\mathbb{P}_e = \wp(\mathbb{V}_e)$	propriétés des entiers,
$\mathbb{V}_e = \mathbb{Z} \cup \{\Omega_i, \Omega_a\}$	valeurs des entiers,
$\mathbb{Z}$	entiers mathématiques,
Ω_i	erreur de non initialisation,
Ω_a	erreur arithmétique.

Exemples de propriétés concrètes des nombres entiers

- « entier valant 7 » : $\{7\}$,
- « entier naturel impair » : $\{1,3,5,\ldots,2n+1,\ldots\}$,
- « entier machine strictement positif » : $\{1,2,3,\ldots,\texttt{max\_int}\}$,
- « erreur arithmétique » : $\{\Omega_a\}$,
- « entier machine négatif sauf si non initialisation » :
  $\{\texttt{min\_int},\ldots,-1,0,\Omega_i\}$,
- « faux » : $\emptyset$;

Propriétés concrètes des variables

- Un environnement $\rho \in \mathbb{E}$ associe une valeur $\rho(\texttt{X})$ à toute variable $\texttt{X}$;
- Une propriété concrète/exacte/mathématique P des variables est codée par l'ensemble des environnements dans lesquels les valeurs des variables possédent cette propriété ;
- $P \in \mathbb{P}_v = \wp(\mathbb{E})$;
- Exemples :
  « X=Y » : $\{\rho \in \mathbb{E} \mid \rho(\texttt{X}) = \rho(\texttt{Y})\}$,
  « toutes les variables sont initialisées » :
  $\{\rho \in \mathbb{E} \mid \forall \texttt{Z} : \rho(\texttt{Z}) \in [\texttt{min\_int},\texttt{max\_int}]\}$;
- Les propriétés concrètes des variables sont des objets mathématiques infinis (en général) impossibles à représenter en machine.

Notion de précision concrète

- L'implication concrète/logique fournit une notion de précision sur les propriétés concrètes [11] :
  - « entier valant 7 » est plus précis que « entier machine positif impair »
    $\{7\} \subseteq \{1,3,5,7,9,\ldots,\texttt{max\_int}\}$
  - « entier machine positif impair » est plus précis que « entier machine strictement positif »
    $\{1,3,5,7,9,\ldots,\texttt{max\_int}\} \subseteq \{1,2,3,4,5,\ldots,\texttt{max\_int}\}$
  - « entier machine positif pair » n'est pas comparable à « entier machine inférieur à 5 »
    $\{0,2,4,6,\ldots,\texttt{max\_int}-1\} \subsetneq \{-\texttt{min\_int},\ldots,0,1,2,3,4,5\}$
    $\{-\texttt{min\_int},\ldots,0,1,2,3,4,5\} \subsetneq \{0,2,4,6,\ldots,\texttt{max\_int}-1\}$
- Mathématiquement $\subseteq$ est un ordre partiel sur les propriétés concrètes.

11. dans ces exemples on suppose que max_int est impair.

Propriétés abstraites des nombres entiers

- Une propriété abstraite/approchée/observable des nombres entiers est donnée par :
  - Un codage fini $\widetilde{\mathbb{P}_e}$ des propriétés abstraites (représentables en machine),
  - Une fonction mathématique de concrétisation γ_e donnant la propriété exacte codée par la propriété abstraite :

$$\gamma_e \in \widetilde{\mathbb{P}_e} \longmapsto \mathbb{P}_e \ .$$

Exemple de propriétés abstraites : **intervalles d'entiers**

$\widetilde{\mathbb{P}_e}$ comprend les éléments suivants ($\texttt{min\_int} \leq a \leq b \leq \texttt{max\_int}$) :

$[a,b]$: « entiers machine compris entre a et b »,
$\gamma_e([a,b]) = \{x \in \mathbb{E} \mid a \leq x \leq b\} \cup \{\Omega_a\}$ [12] ;

$[a,b]?$: « entiers machine compris entre a et b sauf si erreur »,
$\gamma_e([a,b]?) = \{x \in \mathbb{E} \mid a \leq x \leq b\} \cup \{\Omega_a, \Omega_i\}$;

Ω : « erreur d'exécution »,
$\gamma_e(\Omega) = \{\Omega_a, \Omega_i\}$;

$\bot$: « faux »,
$\gamma_e(\bot) = \{\Omega_a\}$ [13].

12. On n'observe pas les valeurs (seulement leur maximum et leur minimum) ni les erreurs arithmétiques.
13. Comme on n'observe pas les erreurs arithmétiques, il n'y a pas moyen de distinguer $\{\Omega_*\}$ de $\emptyset$.

Abstraction

- L'abstraction consiste à approcher une propriété concrète exacte par une propriété abstraite observable moins précise ;
- Une fonction mathématique d'abstraction α_e donne la propriété abstraite approchant au mieux une propriété concrète :

$$\alpha_e \in \mathbb{P}_e \longmapsto \widetilde{\mathbb{P}_e} .$$

Propriétés abstraites des variables (exemple des intervalles)

- Une propriété abstraite/approchée/observable des variables est un environnement abstrait donnant la propriété abstraite des valeurs possibles de chaque variable :

$$\widetilde{\mathbb{P}_v} = \mathrm{Var} \longmapsto \widetilde{\mathbb{P}_e} ;$$

- Un environnement abstrait donne l'intervalle de valeurs possibles de chaque variable :

$$\gamma_v(\widetilde{\rho}) = \{\rho \mid \forall \mathtt{X} : \rho(\mathtt{X}) \in \gamma_e(\widetilde{\rho}(\mathtt{X}))\}$$

Exemple d'abstractions : intervalles d'entiers

Abstraction de :

- « entier valant 7 » et « entier valant 7 ou erreur arithmétique » :
  $\alpha_e(\{7\}) = \alpha_e(\{7,\Omega_a\}) = [7,7]$,
- « entier naturel impair » :
  $\alpha_e(\{1,3,5,\ldots,2n+1,\ldots\}) = [1,\texttt{max\_int}]$,
- « entier machine strictement positif » :
  $\alpha_e(\{1,2,3,\ldots,\texttt{max\_int}\}) = [1,\texttt{max\_int}]$,
- « erreur d'initialisation » :
  $\alpha_e(\{\Omega_i\}) = \Omega$,
- « entier machine négatif sauf si erreur » :
  $\alpha_e(\{\texttt{min\_int},\ldots,-1,0,\Omega_i\}) = [\texttt{min\_int},0]?$,
- « faux » :
  $\alpha_e(\emptyset) = \bot$;

Abstraction relationnelle/non relationnelle

- l'abstraction ignore les relations possibles entre les variables :
- Exemples :

  abstraction de « `X=Y` » :

  $$\widetilde{\rho} \text{ tel que } \forall \mathtt{Z} \in \mathrm{Var} : \widetilde{\rho}(\mathtt{Z}) = [\mathtt{min\_int}, \mathtt{max\_int}]?,$$

  abstraction de « toutes les variables sont initialisées » :

  $$\widetilde{\rho} \text{ tel que } \forall \mathtt{Z} \in \mathrm{Var} : \widetilde{\rho}(\mathtt{Z}) = [\mathtt{min\_int}, \mathtt{max\_int}] ;$$

- D'autres abstractions relationnelles, plus puissantes, permettent d'exprimer des relations entre variables (voir par exemple l'exposé de Nicolas Halbwachs).

Sémantique abstraite

Sémantique abstraite

- La sémantique abstraite d'un programme est l'abstraction de la sémantique collectrice concrète ;
- La sémantique abstraite d'un programme est une propriété observable qui est moins précise que la sémantique collectrice concrète ;
- Les propriétés abstraites données par la sémantique abstraite sont un **sur-ensemble** [14] des propriétés concrètes : aucun cas possible ne peut être oublié.

14. Avec un sous-ensemble, la sémantique abstraite permettrait d'oublier certains cas, comme avec les méthodes classiques de test, ce qui serait incomplet/non exhaustif.

Définition de la sémantique abstraite

- La sémantique abstraite d'un programme est la plus petite solution d'un système d'équations abstraites associé au programme ;
- Les équations abstraites s'obtiennent formellement par abstraction des équations concrètes ;
- L'analyseur construit le système d'équations puis calcule une solution effective.

Un exemple simple de sémantique abstraite

```
{ X:_O_ [15]; Y:_O_ }
  X :=?;
{ X:[-oo,+oo]; Y:_O_ }
  Y := 0;
{ X:[-oo,+oo]; Y:[0,+oo] }
  while (X <= Y) do
    { X:[-oo,+oo]; Y:[0,+oo] }
      Y := (Y + 1)
    { X:[-oo,+oo]; Y:[1,+oo] }
  od
{ X:[1,+oo]; Y:[0,1073741822] }
```

15. _O_ dénote Ω, _|_ dénote $\bot$, -oo dénote min_int et +oo dénote max_int = 1073741823.

Un exemple simple d'équations abstraites

```
1: P1
   X :=?;
2: P2
   Y := 0;
3: P3
   while Y <= X do
4: P4
     Y := Y + 1
   od
5: P5
```

$P_1 = \langle \mathtt{X} : \Omega, \mathtt{Y} : \Omega \rangle$

$P_2 = \langle \mathtt{X} : [\mathtt{min\_int},\mathtt{max\_int}], \mathtt{Y} : P_1(\mathtt{Y}) \rangle$

$P_3 = \langle \mathtt{X} = P_2(\mathtt{X}), \mathtt{Y} : [0,0] \rangle \sqcup \langle \mathtt{X} = P_4(\mathtt{X}), \mathtt{Y} = P_4(\mathtt{Y}) \oplus [1,1] \rangle$

$P_4 = P_3 \setminus \mathtt{X} \leq \mathtt{Y}$

$P_5 = P_3 \setminus \mathtt{X} > \mathtt{Y}$

Opérations abstraites

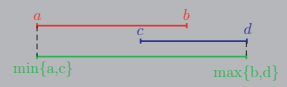
union abstraite

$x \sqcup y = y \sqcup x, \quad x \sqcup x = x \ , \quad \bot \sqcup x = x$

$\Omega \sqcup [a,b] = \Omega \sqcup [a,b]? = [a,b]?$

$[a,b] \sqcup [c,d] = [\min\{a,c\}, \max\{b,d\}]$

$[a,b]? \sqcup [c,d] = [\min\{a,c\}, \max\{b,d\}]?$



...

addition abstraite

$\bot \oplus x = x \oplus \bot = \bot$

$\Omega \oplus x = x \oplus \Omega = \Omega$

$[a,b] \oplus [c,d] = \Omega$ si $a + c > \mathtt{max\_int}$

$[a,b] \oplus [c,d] = \Omega$ si $b + d < \mathtt{min\_int}$

$[a,b] \oplus [c,d] = [\max\{a + c, \mathtt{min\_int}\}, \min\{b + d, \mathtt{max\_int}\}]$ si $\max\{a + c, \mathtt{min\_int}\} \leq \min\{b + d, \mathtt{max\_int}\}$

...

Résolution itérative des équations abstraites avec accélération de la convergence

Un exemple simple de **convergence très lente**

$P_1 = \langle \mathtt{X} : \Omega, \mathtt{Y} : \Omega \rangle$
$P_2 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : P_1(\mathtt{Y}) \rangle$
$P_3 = \langle \mathtt{X} = P_2(\mathtt{X}), \mathtt{Y} : [0,0] \rangle \sqcup \langle \mathtt{X} = P_4(\mathtt{X}), \mathtt{Y} = P_4(\mathtt{Y}) \oplus [1,1] \rangle$
$P_4 = P_3 \setminus \mathtt{X} \leq \mathtt{Y}$
$P_5 = P_3 \setminus \mathtt{X} > \mathtt{Y}$

$P_1 = P_2 = P_3 = P_4 = P_5 = \bot$
$P_1 = \langle \mathtt{X} : \Omega, \mathtt{Y} : \Omega \rangle$
$P_2 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : \Omega \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,0] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty,0], \mathtt{Y} : [0,0] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,1] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty,1], \mathtt{Y} : [0,1] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,2] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty,2], \mathtt{Y} : [0,2] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,3] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty,3], \mathtt{Y} : [0,3] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,4] \rangle$
...
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0, +\infty] \rangle$

Pour accélérer la convergence de 0 à $+\infty = 1073741823$, extrapoler les bornes instables pour les têtes de boucles.

Un exemple simple d'accélération de la convergence par **élargissement**

$P_1 = \langle \mathtt{X} : \Omega, \mathtt{Y} : \Omega \rangle$
$P_2 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : P_1(\mathtt{Y}) \rangle$
$P_3 = P_3 \triangledown \langle \mathtt{X} = P_2(\mathtt{X}), \mathtt{Y} : [0,0] \rangle \sqcup \langle \mathtt{X} = P_4(\mathtt{X}), \mathtt{Y} = P_4(\mathtt{Y}) \oplus [1,1] \rangle$
$P_4 = P_3 \setminus \mathtt{X} \leq \mathtt{Y}$
$P_5 = P_3 \setminus \mathtt{X} > \mathtt{Y}$

Élargissement :
$\bot \triangledown x = x \triangledown \bot = x$
$\Omega \triangledown [a,b] = [a,b] \triangledown \Omega = [a,b]?$
$[a,b] \triangledown [c,d] = [(c < a? -\infty : a),(d > b? +\infty : b)]$

perte de précision.

$P_1 = P_2 = P_3 = P_4 = P_5 = \bot$
$P_1 = \langle \mathtt{X} : \Omega, \mathtt{Y} : \Omega \rangle$
$P_2 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : \Omega \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,0] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty,0], \mathtt{Y} : [0,0] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0,0] \triangledown [0,1] \rangle$
$= \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0, +\infty] \rangle$
$P_4 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0, +\infty] \rangle$
$P_3 = \langle \mathtt{X} : [-\infty, +\infty], \mathtt{Y} : [0, +\infty] \rangle$
stabilité

Applications de l'interprétation abstraite à l'**analyse statique**

(I) Recherche d'**erreurs à l'exécution**

- Recherche d'erreurs à l'exécution liées à la sémantique du langage de programmation utilisé ;
- Exemples classiques d'erreurs à l'exécution :
  - défaut d'initialisation,
  - dépassements de capacité (par exemple le dépassement des bornes de tableaux),
  - opérations indéfinies (par exemple division par zéro),
  - pointeurs pendants,
  - défauts de synchronisation, etc ;
- classement des erreurs en certaines, potentielles ou impossibles à l'exécution.

Principe de la recherche statique d'erreurs à l'exécution

- On effectue une analyse statique du programme (relative aux valeurs des variables pouvant créer des erreurs à l'exécution) ;
- Pour chaque opération dans le texte du programme pouvant conduire à une erreur à l'exécution, on utilise l'information statique pour déterminer si elle est certaine, impossible ou potentielle ;
- Aucun cas possible n'est omis (couverture complète).

Un exemple simple de recherche d'erreurs à l'exécution...

```
{ i:_0_; j:_0_ }
  i := 1;                                  débordement de 1 impossible
{ i:[1,+oo]; j:[1,+oo]? }
  while (i < 1000) do                      non initialisation de i impossible
    { i:[1,999]; j:[1,+oo]? }
      j := 1;                              débordement de 1 impossible
    { i:[1,+oo]; j:[1,+oo] }
      while (j < i) do                     non initialisation de i ou j impossible
        { i:[2,+oo]; j:[1,1073741822] }
          j := (j + 1)                     non initialisation de j impossible, débordement
        { i:[2,+oo]; j:[2,+oo] }           de j + 1 impossible
      od;
    { i:[1,+oo]; j:[1,+oo] }
      i := (i + 1);                        ⚠ non initialisation de i impossible,
    { i:[2,+oo]; j:[1,+oo] }               débordement de i + 1 POTENTIEL
  od
{ i:[1000,+oo]; j:[1,+oo]? }
```


Avec une extrapolation plus précise ...

```
{ i:_0_; j:_0_ }
  i := 1;                                  débordement de 1 impossible
{ i:[1,1000]; j:[1,999]? }
  while (i < 1000) do                      non initialisation de i impossible
    { i:[1,999]; j:[1,999]? }
      j := 1;                              débordement de 1 impossible
    { i:[1,999]; j:[1,999] }
      while (j < i) do                     non initialisation de i ou j impossible
        { i:[2,999]; j:[1,998] }
          j := (j + 1)                     non initialisation de j impossible, débordement de j + 1
        { i:[2,999]; j:[2,999] }           impossible
      od;
    { i:[1,999]; j:[1,999] }
      i := (i + 1);                        non initialisation de i impossible, débordement de i + 1
    { i:[2,1000]; j:[1,999] }              impossible
  od
{ i:[1000,1000]; j:[1,999]? }              imprécision
```


Sur la méthode d'extrapolation plus précise

- Ne pas extrapoler pour un composant du système d'équations qui est calculé pour la première fois depuis la dernière extrapolation ;
- Sinon, ne pas extrapoler pour une variable dont la valeur abstraite n'a pas été modifiée [16] depuis la dernière extrapolation.

16. plus précisément modifiée dans une équation abstraite correspondant à une affectation à la variable dans le programme.

(II) Test abstrait

- Permettre de prendre en compte une spécification donnée au programmeur ;
- Déterminer des conditions nécessaires pour que cette spécification soit toujours satisfaite ;

$\Rightarrow$ Recherche d'erreurs logiques invalidant la spécification du programme.

Principe du test abstrait

- Un jeux de données est remplacé par une spécification partielle,
- Une exécution avec ce jeux de données est remplacé par une analyse statique par interprétation abstraite ;

$\Longrightarrow$

- l'activité de test abstrait est proche du test traditionnel ;
- on remplace une exécution particulière par un ensemble d'exécutions ;
- les jeux de test sont engendrés par l'analyse.

Spécification d'**invariance**/sureté (safety)

Une spécification d'invariance `always`B en un point ℓ du programme est satisfaite si et seulement si la condition booléenne B est toujours vraie à chaque fois que l'exécution passe en ce point ℓ du programme.

Exemple : spécification de non terminaison

```
x :=?;  [17]
while (0 < x) do
    x := (x + 1);
od;
always false
```

- le programme est donné ;
- la spécification est la non terminaison [18].

17. ? est l'affectation aléatoire.
18. sauf peut-être la terminaison après une erreur à l'exécution.

Spécification d'**inévitabilité**/vivacité (liveness)

Une spécification d'inévitabilité `sometime`B en un point ℓ du programme est satisfaite si et seulement si l'exécution du programme passer en ce point au moins une fois avec la condition booléenne B qui est vraie.

Exemple : spécification de terminaison

```
x :=?;  [19]
while (0 < x) do
    x := (x + 1);
od;
sometime true
```

- le programme est donné ;
- la spécification (de terminaison) est donnée.

19. `?` l'affectation aléatoire.

Débogeur abstrait

Un ***débogeur abstrait*** est un programme qui
- prend en entrée :
  - le texte d'un *programme*,
  - une *spécification* de conditions d'invariance et d'inévitabilité qui sont nécessaires pour que l'exécution du programme soit correcte ;
- qui termine tout le temps et retourne comme résultat :
  - des conditions nécessaires sur l'environnement d'exécution du programme et les objets manipulés par le programme (variables, etc.) pour que la spécification soit satisfaite au cours d'une exécution quelconque ;
  - des invariants qui sont toujours vrais lors d'une exécution correcte.

Exemple: condition nécessaire de terminaison

```
{ x:_O_ }  [20]
  x :=?;  [21]
x??? { x:[-oo [22],0] }
  while (0 < x) do
    _|_  [23]
      x := (x + 1);
    _|_
  od;
{ x:[-oo,0] }
  sometime true
{ x:[-oo,0] }
```

- le programme est donné ;
- la spécification (de terminaison) est donnée ;
- la condition nécessaire de satisfaction de la spécification inférée par l'analyse est :
  sometime terminate $\Longrightarrow$ `x` ≤ 0;
- les assertions inférées sont invariantes (quand la condition de satisfaction de la spécification est satisfaite) ;

20. `_O_` dénote la non initialisation.
21. `?` est l'affectation aléatoire.
22. `-oo` est l'entier machine le plus petit `min_int` = -1073741824.
23. `_|_` est « faux ».

Exemple (suite) : condition necéssaire de non terminaison

```
{ x:_0_ }
  x :=?;
x??? { x:[1,1073741821] }
  while (0 < x) do
    { x:[1,1073741821] } [24]
      x := (x + 1);
    { x:[2,1073741822] }
  od;
_|_
  always false
_|_
```

- la spécification est la non terminaison [25];
- la condition de satisfaction de la spécification inférée par l'analyse est :
  never terminate $\Rightarrow$ x ≥ 1

24. +oo = max_int = 1073741823.
25. sauf peut-être à la suite d'une erreur à l'exécution.

Exemple (fin) : condition necéssaire et suffisante de terminaison

- par la condition nécessaire de terminaison :
  sometime terminate $\Rightarrow$ x ≤ 0 ;
- par la condition necéssaire de non terminaison :
  never terminate $\Rightarrow$ x ≥ 1
  $\Longleftrightarrow$ x $\leq 0 \Rightarrow \neg$(never terminate) $\Leftrightarrow$ sometime terminate ;
- par conjonction des deux conditions :
  sometime terminate $\Leftrightarrow$ x ≤ 0 ;
- par déterminisme (pas d'autre possibilité [26]) :
  always terminate $\Leftrightarrow$ x ≤ 0

26. aucune erreur à l'exécution n'étant possible.

Exemple de combinaison de spécifications d'invariance et d'inévitabilité

```
{ x:_0__ }
  x :=?;
x??? { x:[0,+oo] }
  while (0 <= x) & (x < 1000)) do
    { x:[0,999] }
      x := (x + 1);
    { x:[1,1000] }
  od;
{ x:[1000,+oo] }
  always (x > 0);
{ x:[1000,+oo] }
  sometime true
{ x:[1000,+oo] }
```

x < 0 est nécéssaire pour que la spécification soit violée.

Exemple simple de test abstrait

- Par des choix judicieux des spécification d'invariance et d'inévitabilité, on peut tester les propriétés d'ensembles d'exécution possibles (et non pas une seule à la fois comme dans le test classique) ;
- Exemple (squelette du tri par remontée des bulles) :

```
n :=?; i := n;
while (i <> 0) do
    j := 0;
    while (j <> i) do
        j := (j + 1)
    od;
    i := (i - 1);
od;
```

1) test de terminaison correcte

```
{ n:_0_; i:_0_; j:_0_ }
  n :=?;
n??? { n:[0,+oo]; i:_0_; j:_0_ }
  i := n;
  while (i <> 0) do
    { n:[0,+oo]; i:[1,+oo]; j:[1,+oo]? }
      j := 0;
      while (j <> i) do
        j??? { n:[0,+oo]; i:[1,+oo]; j:[0,1073741822] }
          j := (j + 1)
      od;
      i := (i - 1);
  od;
  sometime true
{ n:[0,+oo]; i:[0,0]; j:[1,+oo]? }
```


2) Analyse d'erreur

```
  n :=?;
n??? { n:[-oo,-1]; i:_0_; j:_0_ }
  always (n < 0);
  i := n;
  while (i <> 0) do
      j := 0;
      while (j <> i) do
        j??? { n:[-oo,-1]; i:[-oo,-1]; j:[0,1073741821] }
          j := (j + 1)
      od;
    _|_
      i := (i - 1);
  od
_|_
```

$\Longrightarrow$ la boucle interne ne termine jamais.

Conclusion

Exposé introductif

- Exposé introductif très élémentaire sur l'interprétation abstraite :
- Pour approfondir :
  - Consulter la bibliographie pour les aspects théoriques, bien développés [8, 9] ;
  - Les exposés qui suivent vont explorer plus largement le champ d'applications possibles.

Difficultés et perspectives

- La conception et la réalisation d'analyseurs statique est complexe (plus difficile que d'écrire un compilateur) ;
- Des réalisations académiques existent (avec un succès certain par exemple en programmation logique) ;
- Des expériences industielles ont eu lieu avec grand succès (voir l'exposé d'Alain Deutsch) ;
- Début d'industrialisation va rendre disponible à court terme des analyseurs statiques utilisables dans un environnement de production pour des langages de programmation couramment utilisés.

Pourquoi l'interprétation abstraite va réussir ?

- Il faut des compléments aux méthodes de validation classiques (tests, simulations, ...) qui n'offrent pas de garantie de couverture :
  - Le model-checking a connu de grands succès mais atteint ses limites quand il s'agit de logiciels complexes ;
  - Les preuves formelles sont trop coûteuses ;
- L'interprétation abstraite, grâce à la notion d'approximation, offre :
  - une méthodologie compréhensible par les programmeurs,
  - une garantie de couverture,
  - un compromis coût/efficacité,

  qui permettent de détecter un grand nombre d'erreurs de programmation (mais certainement pas toutes les erreurs logiques profondes, sauf à développer des analyseurs spécialisés).

Quelques références bibliographiques

(I) Articles fondateurs

[3] **Patrick Cousot & Radhia 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*, pages 238–252, Los Angeles, Californie, 1977. ACM Press, New York, usa.

[4] **Patrick Cousot & Radhia Cousot**.
Systematic design of program analysis frameworks.
In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, usa.

(II) Articles introductifs élémentaires

[5] **Patrick Cousot**. ***Abstract interpretation***.
Symposium on Models of Programming Languages and Computation, ACM Computing Surveys, 28(2):324–328, 1996.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/CS96.shtml

[6] **Patrick Cousot**. ***Program Analysis: The Abstract Interpretation Perspective***.
ACM Computing Surveys, Vol. 28, No. 4es (Dec. 1996), pages 165-es.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/CSA96.shtml

[7] **Patrick Cousot**.
Semantic foundations of program analysis.
In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1981.

(III) Théorie de l'interprétation abstraite

[8] **Patrick Cousot & Radhia Cousot**.
Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper.
In M. Bruynooghe & M. Wirsing, éds., *Programming Language Implementation and Logic Programming*, Proc. Fourth Int. Symp., PLILP'92, Louvain, Belgique, 13–17 août 1992, Lecture Notes in Computer Science 631, pp. 269–295. Springer-Verlag, 1992.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/PLILP92.shtml 68

[9] **Patrick Cousot & Radhia Cousot**.
Abstract interpretation and application to logic programs.
Journal of Logic Programming, 13(2–3):103–179, 1992.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/JLP92.shtml 68

(IV) Théorie de l'interprétation abstraite (articles avancés)

[10] **Patrick Cousot & Radhia Cousot**.
Abstract interpretation frameworks.
Journal of Logic and Computation, 2(4):511–547, August 1992.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/JLC92.shtml

[11] **Patrick Cousot**.
Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation.
Electronic Notes in Theoretical Computer Science, 6 (1997), 25 pages.
http://www.elsevier.nl/locate/entcs/volume6.html &
http://www.dmi.ens.fr/~cousot/COUSOTpapers/ENTCS-97.shtml
18

(V) Exemple pédagogique d'analyseur statique

[12] **Patrick Cousot**.
The Calculational Design of a Generic Abstract Interpreter. In *Calculational System Design*, M. Broy & R. Steinbrüggen, editors, NATO ASI Series F. IOS Press, Amsterdam, 1999.
http://www.dmi.ens.fr/~cousot/COUSOTpapers/Marktoberdorf98.shtml
, novembre 1998.
[13] **Patrick Cousot**.
The Marktoberdorf'98 generic abstract interpreter.
http://www.dmi.ens.fr/~cousot/Marktoberdorf98.shtml , novembre 1998.