Sémantiques mécanisées, cinquième cours

Un art abstrait: l'analyse statique par interprétation abstraite

Xavier Leroy

2020-01-17

Collège de France, chaire de sciences du logiciel

L'analyse statique (reprise du 3ième cours)

Inférer (sans aide du programmeur) statiquement (sans exécuter le programme) des propriétés qui sont vraies de toutes les exécutions possibles du programme.

Utilisations :

- Pour améliorer les performances du code en guidant les optimisations.
- Pour améliorer la fiabilité des programmes en montrant l'absence / détectant la présence possible d'erreurs de programmation.

Utiliser des analyses statiques pour la vérification

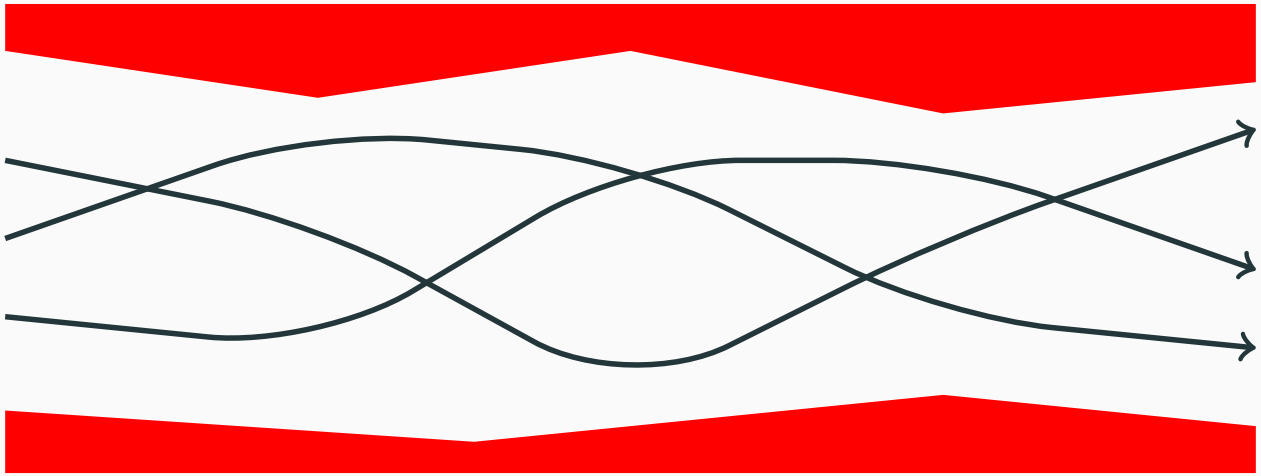
Utiliser les propriétés inférées par analyse statique pour montrer l'absence d'erreurs à l'exécution (p.ex. divisions par zéro, accès aux tableaux hors bornes).

$$b \in [n_1, n_2] \wedge 0 \notin [n_1, n_2] \implies a/b \text{ pas d'erreur possible}$$

$$\texttt{valid}(p[n_1 \ldots n_2]) \wedge i \in [n_1, n_2] \implies \texttt{p[i]} \text{ pas d'erreur possible}$$

Émettre une alarme là où on ne peut pas montrer l'absence d'erreurs.

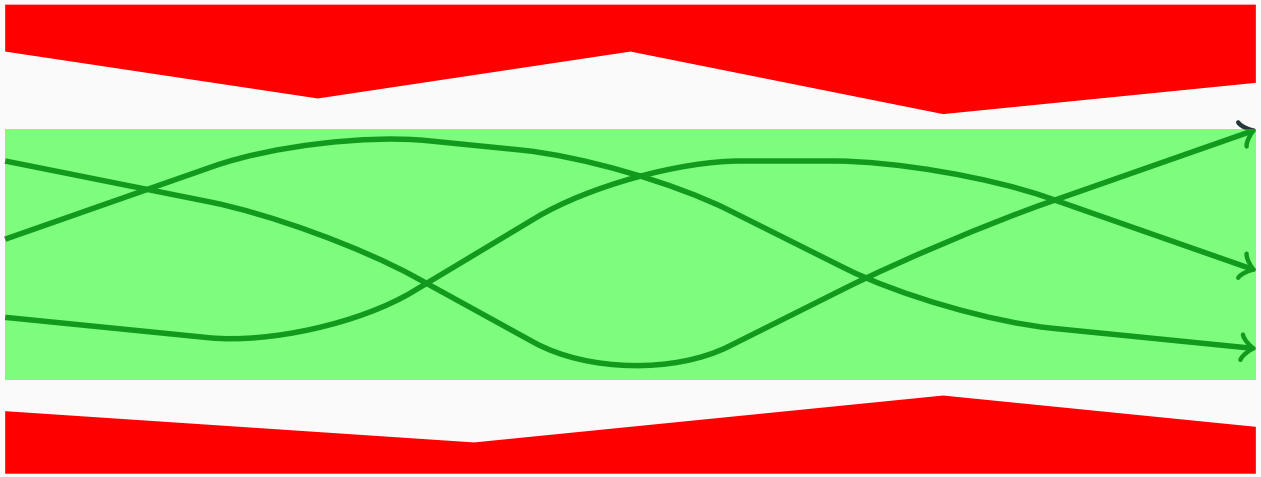
Analyse statique et vérification, graphiquement



Trajectoires = exécutions possibles du programme.
(P. ex. valeur d'une variable x au cours du temps t.)

Zone rouge = erreurs à l'exécution, comportements indésirables.

Analyse statique et vérification, graphiquement

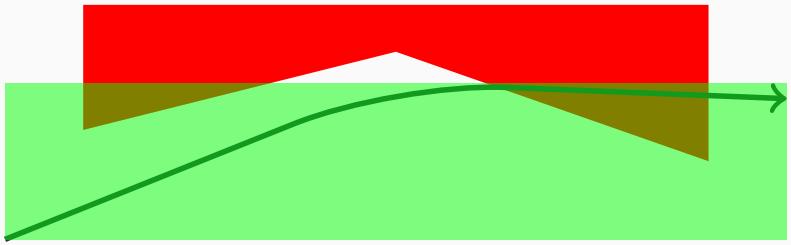


Trajectoires = exécutions possibles du programme.
(P. ex. valeur d'une variable x au cours du temps t.)

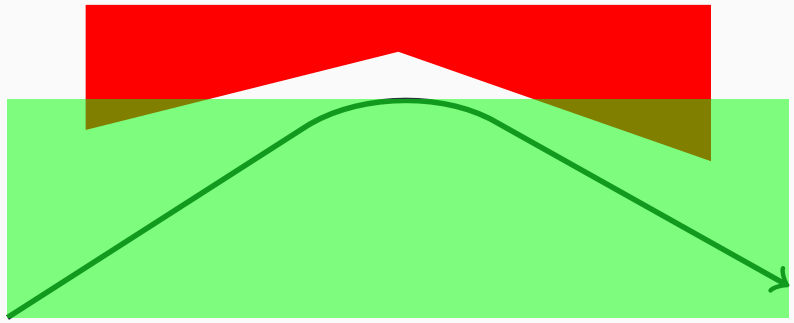
Zone rouge = erreurs à l'exécution, comportements indésirables.

Zone verte = résultat de l'analyse statique ; sur-approximation des exécutions possibles du programme.

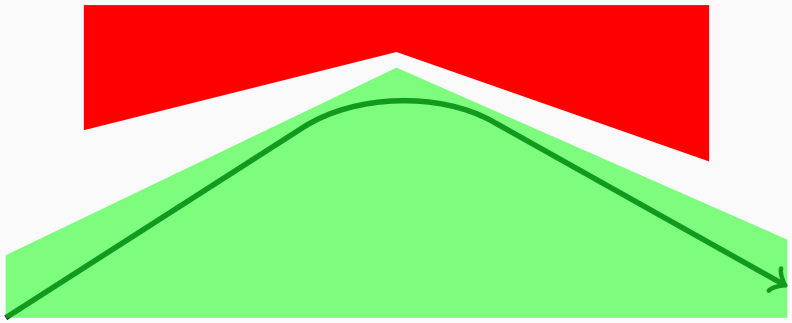
Vraies alarmes, fausses alarmes



Vraie alarme
(comportement erroné)



Fausse alarme
(analyse trop imprécise)



Analyse plus précise (p.ex. des polyèdres au lieu d'intervalles) :
fait disparaître la fausse alarme.

Quelques propriétés vérifiables par analyse statique

Absence d'erreurs à l'exécution :

- Tableaux et pointeurs :
  - Pas d'accès hors bornes de tableau.
  - Pas de déréférencement du pointeur nul.
  - Pas d'accès après désallocation explicite (`free`).
  - Contraintes d'alignment du processeur.
- Arithmétique entière :
  - Pas de divisions par zéro.
  - Pas de débordements arithmétiques (signés).
- Arithmétique en virgule flottante :
  - Pas de débordements (résultats infinis)
  - Pas d'opérations indéfinies (résultats *Not-a-Number*)
  - Pas de perte catastrophique de précision.

Quelques propriétés vérifiables par analyse statique

Valider des assertions ou des invariants du programme :

- Intervalles de variation pour les sorties numériques.
- Préconditions pour les fonctions de bibliothèque.
- Analyse de bonne forme (*shape analysis*) pour les structures de données chaînées.

Vérifier des politiques de sécurité :

- Analyse de flots d'information ; *tainting*.

Établir des propriétés ≪non fonctionnelles≫ :

- Borner la consommation en mémoire.
- Borner le pire temps d'exécution (WCET).

Quelles analyses statiques pour la vérification ?

Analyse statique de type «flot de données» (3e cours) :

- Entièrement automatique, algorithmiquement efficace.
- Infère des propriétés très simples, insuffisantes pour montrer l'absence d'erreurs à l'exécution.

Vérification déductive en logique de Hoare / de séparation (4e cours) :

- Vérifie des propriétés arbitrairement complexes.
- Pas automatique (invariants de boucles).

Et entre les deux ?

Interprétation abstraite classique

L'interprétation abstraite

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

(POPL, 1977)

Un formalisme très général pour décrire et implémenter des analyses statiques précises.

Intuition de l'interprétation abstraite

Analyser un programme, c'est l'exécuter avec une sémantique non-standard :

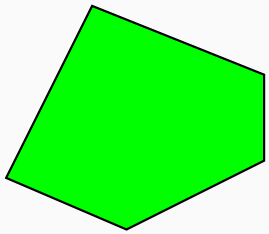
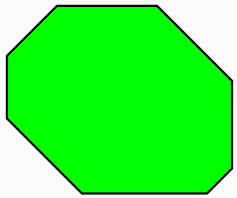
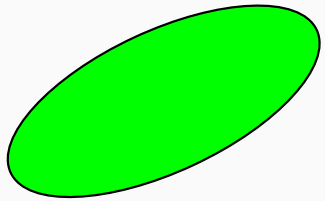
- Calcule dans les domaines abstraits des propriétés qu'il faut établir (p.ex. $\ll x \in [n_1, n_2] \gg$ pour l'analyse d'intervalle), au lieu d'objets concrets (nombres, valeurs, états).
- Garantit que le calcul abstrait est toujours une sur-approximation du calcul concret.

Exemples de domaines abstraits

Propriétés d'une variable numérique :

$x \in [a, b]$ (intervalles) $\qquad x \bmod a = b$ (congruences)

Relations entre plusieurs variables numériques :

polyèdres	octogones	ellipses
		
$\sum a_i x_i \leq c$	$\pm x_1 \pm x_2 \leq c$	$ax^2 + by^2 + cxy + dx + ey \leq f$

Structures de données chaînées (*shape analysis*) :

Interprétation abstraite avec des intervalles

Dans le concret	Dans l'abstrait
$\{ x = 3, y = 1 \}$	$\{ x^{\#} = [0, 9], y^{\#} = [-1, 1] \}$
`z = x + 2 * y;`	
$\{ z = 3 + 2 \times 1 = 5 \}$	$\{ z^{\#} = [0, 9] +^{\#} 2 \times^{\#} [-1, 1] = [-2, 11] \}$

(On note «#» les abstractions des variables et des opérateurs.)

Interprétation abstraite avec des intervalles

```
if x < 0 then
    s := -1
else if x > 0 then
    s := 1
else
    s := 0
```

$$x^{\#} = [0, 10]$$

$$s^{\#} = \emptyset$$

$$s^{\#} = [1, 1]$$

$$s^{\#} = [0, 0]$$

$$s^{\#} = \emptyset \cup [1, 1] \cup [0, 0] = [0, 1]$$

En général on ne peut pas résoudre statiquement les conditions
$\rightarrow$ exécution des branches `then` et `else` + union des abstractions.

Certaines conditions sont résolues statiquement
$\rightarrow$ on marque $\emptyset$ ou $\bot$ la branche non prise.

Interprétation abstraite avec des intervalles

En général on doit supposer que les boucles font un nombre arbitraire de tours.

```
x := 0;
while condition do
    x := x + 1
done
```

$$x^{\#} = [0, +\infty]$$

Certaines boucles comptées peuvent s'analyser plus finement.

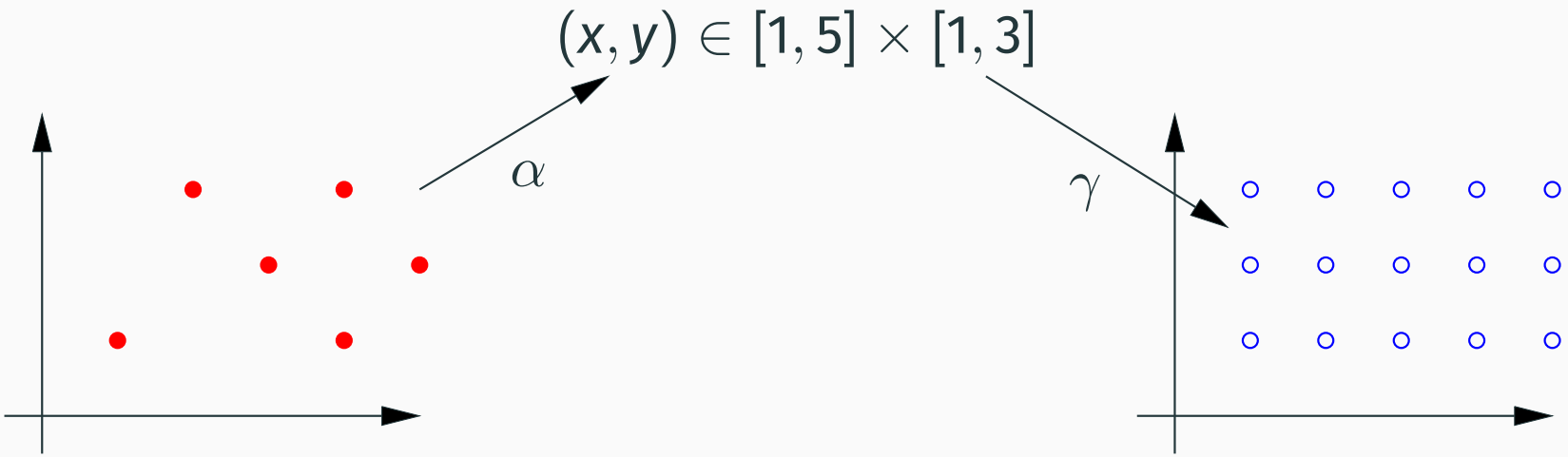
```
x := 0;
for i := 1 to 10 do
  if condition then x := x + 1
done
```

$$x^{\#} = [0, 10]$$

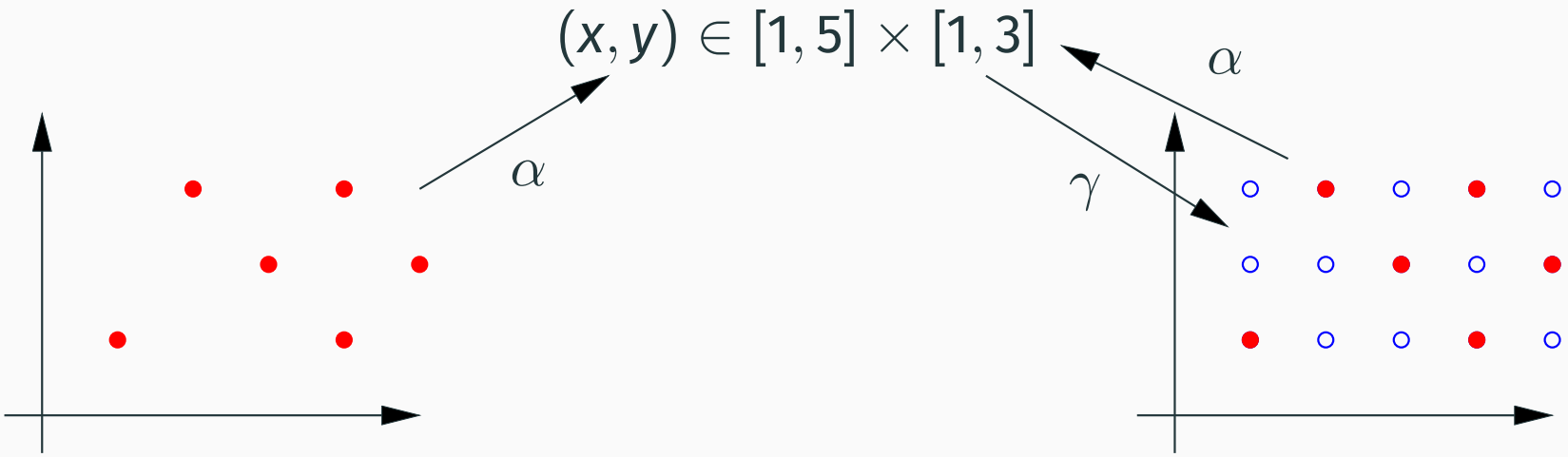
Formalisation des domaines abstraits

Un treillis $(A, \leq)$ de valeurs abstraites et deux fonctions :

- α, la fonction d'abstraction :
  ensemble de valeurs concrètes $\rightarrow$ valeur abstraite
- γ, la fonction de concrétisation :
  valeur abstraite $\rightarrow$ ensemble de valeurs concrètes.



Propriétés de la concrétisation et de l'abstraction



α et γ sont croissantes

$\wedge \quad \forall X, \; X \subseteq \gamma(\alpha(X))$ (sûreté)

$\wedge \quad \forall a, \; \alpha(\gamma(a)) \leq a$ (optimalité)

$(\mathcal{P}(C), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq)$ est une correspondance de Galois isotone.

Calculer les opérateurs abstraits

Pour tout opérateur concret $F : C \to C$, on définit son abstraction $F^\# : A \to A$ comme

$$F^\#(a) \stackrel{def}{=} \alpha\{F(x) \mid x \in \gamma(a)\}$$

Cet opérateur abstrait est :

- Correct : si $x \in \gamma(a)$, alors $F(x) \in \gamma(F^\#(a))$.
- Précis : $F(a)$ est la plus petite abstraction a' qui vérifie $x \in \gamma(a) \Rightarrow F(x) \in \gamma(a')$.

De plus, une implémentation effective de $F^\#$ peut être dérivée par calcul (symbolique) à partir de la définition.

Calculer $+^{\#}$ pour les intervalles

On a $\alpha(X) = [\inf X, \sup X]$ et $\gamma([a,b]) = \{n \mid a \leq x \leq b\}$. Donc :

$$\begin{aligned}
[a_1,b_1] +^{\#} [a_2,b_2] &= \alpha\{x_1 + x_2 \mid x_1 \in \gamma[a_1,b_1], x_2 \in \gamma[a_2,b_2]\} \\
&= [\, \inf\{x_1 + x_2 \mid a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2\}, \\
&\qquad \sup\{x_1 + x_2 \mid a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2\} \,] \\
&= [+\infty, -\infty] \text{ si } a_1 > b_1 \text{ ou } a_2 > b_2 \\
&= [a_1 + b_1, a_2 + b_2] \text{ sinon}
\end{aligned}$$

La définition intuitive $[a_1,b_1] +^{\#} [a_2,b_2] = [a_1 + b_1, a_2 + b_2]$ est correcte mais pas précise.

Interprétation abstraite des commandes

Soient une sémantique opérationnelle $c/s \Downarrow s'$
et une abstraction des états mémoire $(\mathcal{P}(\mathit{state}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq)$.

On définit l'interprétation abstraite $\mathtt{exec}^{\#}(c) : A \to A$
d'une commande comme une fonction de l'état abstrait
«avant» vers l'état abstrait «après» :

$$\mathtt{exec}^{\#}(c) \overset{\mathit{def}}{=} \lambda a.\ \alpha\{s' \mid s \in \gamma(a),\ c/s \Downarrow s'\}$$

(Approche plus classique : passer par une sémantique collectrice,
point de programme $\to$ ensemble d'états concrets.)

Interprétation abstraite des commandes

$$\texttt{exec}^{\#}(c) \stackrel{def}{=} \lambda a.\, \alpha\{s' \mid s \in \gamma(a),\ c/s \Downarrow s'\}$$

On peut alors dériver par calcul des équations pour $\texttt{exec}^{\#}$, comme par exemple

$$\begin{aligned}
\texttt{exec}^{\#}(\texttt{skip}) &= \lambda a.\, a \\
\texttt{exec}^{\#}(c_1; c_2) &= \texttt{exec}^{\#}(c_2) \circ \texttt{exec}^{\#}(c_1) \\
\texttt{exec}^{\#}(\texttt{while } b \texttt{ do } c) &= \texttt{exec}^{\#}(\texttt{if } b \texttt{ then c; while } b \texttt{ do } c \texttt{ else skip})
\end{aligned}$$

Interprétation abstraite et logique de Hoare

$$\texttt{exec}^{\#}(c) \stackrel{def}{=} \lambda a.\ \alpha\{s' \mid s \in \gamma(a),\ c/s \Downarrow s'\}$$

On peut lire un état abstrait a comme une assertion de la logique de Hoare : l'assertion ≪l'état courant appartient à $\gamma(a)$≫.

Avec cette interprétation, $\texttt{exec}^{\#}$ satisfait le triplet de Hoare

$$\{\, a \,\}\ c\ \{\, \texttt{exec}^{\#}(a) \,\}$$

De plus, $\texttt{exec}^{\#}(a)$ est la plus forte post-condition de c et a qui est exprimable dans le domaine abstrait des états.

Interprétation abstraite constructive

Correspondances de Galois : un problème en théorie des types

Les correspondances de Galois $(\mathcal{P}(C), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq)$ ne sont généralement pas définissables en théorie des types.

La concrétisation γ se modélise aisément comme

$$\gamma : A \to (C \to \texttt{Prop}) \qquad \text{(une relation entre } A \text{ et } C\text{)}$$

mais α est généralement non calculable dès que C est infini :

$\alpha : (C \to \texttt{Prop}) \to A$ seulement des fonctions constantes ?
$\alpha : (C \to \texttt{bool}) \to A$ ne peut dépendre que d'un nombre fini de C

Exemple : $\alpha(S) = [\inf S, \sup S]$ n'est pas calculable, pas davantage que inf et sup sur des ensembles infinis d'entiers.

Correspondances de Galois : un problème de fond

Pour certains domaines, la fonction d'abstraction α n'existe pas !
(L'optimalité $a \leq \alpha(\gamma(a))$ ne peut pas être satisfaite.)

Exemple 1 : intervalles rationnels.

$$\alpha\{x \mid x^2 \leq 2\} = ???$$

Il n'y a pas d'approximation rationnelle optimale de $[-\sqrt{2}, \sqrt{2}]$.

Correspondances de Galois : un problème de fond

Pour certains domaines, la fonction d'abstraction α n'existe pas !
(L'optimalité $a \leq \alpha(\gamma(a))$ ne peut pas être satisfaite.)

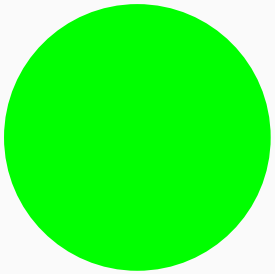
Exemple 1 : intervalles rationnels.

$$\alpha\{x \mid x^2 \leq 2\} = ???$$

Il n'y a pas d'approximation rationnelle optimale de $[-\sqrt{2}, \sqrt{2}]$.

Exemple 2 : polyèdres.

$$\alpha\{(x, y) \mid x^2 + y^2 \leq 1\} = ???$$



Correspondances de Galois : un problème de fond

Pour certains domaines, la fonction d'abstraction α n'existe pas !
(L'optimalité $a \leq \alpha(\gamma(a))$ ne peut pas être satisfaite.)

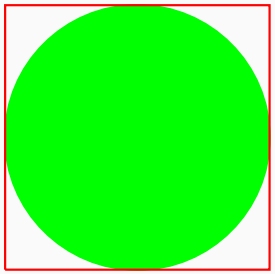
Exemple 1 : intervalles rationnels.

$$\alpha\{x \mid x^2 \leq 2\} = ???$$

Il n'y a pas d'approximation rationnelle optimale de $[-\sqrt{2}, \sqrt{2}]$.

Exemple 2 : polyèdres.

$$\alpha\{(x, y) \mid x^2 + y^2 \leq 1\} = ???$$



Correspondances de Galois : un problème de fond

Pour certains domaines, la fonction d'abstraction α n'existe pas !
(L'optimalité $a \leq \alpha(\gamma(a))$ ne peut pas être satisfaite.)

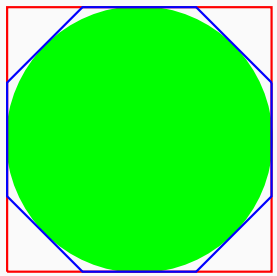
Exemple 1 : intervalles rationnels.

$$\alpha\{x \mid x^2 \leq 2\} = ???$$

Il n'y a pas d'approximation rationnelle optimale de $[-\sqrt{2}, \sqrt{2}]$.

Exemple 2 : polyèdres.

$$\alpha\{(x, y) \mid x^2 + y^2 \leq 1\} = ???$$



L'interprétation abstraite sans fonction d'abstraction

(P. Cousot et R. Cousot, *Abstract interpretation frameworks*, JLC, 1992.)
(D. Pichardie, *Interprétation abstraite en logique intuitionniste : extraction d'analyseurs Java certifiés*, thèse, U. Rennes 1, 2005.)

La fonction d'abstraction α est nécessaire seulement pour calculer les opérateurs abstraits. Étant donné un opérateur abstrait $F^\#$ pour l'opérateur F, la concrétisation γ suffit à vérifier

- la correction sémantique de $F^\#$:

$$x \in \gamma(a) \Rightarrow F(x) \in \gamma(F^\#(a))$$

- en option, l'optimalité relative de $F^\#$:

$$(\forall x \in \gamma(a),\ F(x) \in \gamma(a')) \Rightarrow F^\#(a) \leq a'$$

Cette approche s'exprime parfaitement en théorie des types.

La correction est essentielle, l'optimalité optionelle

L'optimalité n'est pas nécessaire pour obtenir un analyseur fiable (pas d'alarmes $\Rightarrow$ pas d'erreurs à l'exécution).

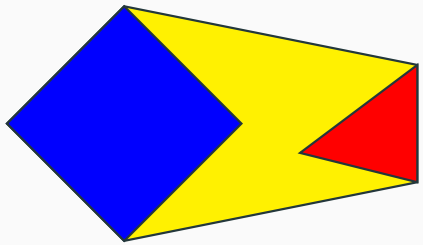
Rendre l'optimalité optionnelle permet de simplifier un analyseur abstrait et sa preuve de correction :

- Opérateurs abstraits qui produisent des sur-approximations (ou juste $\top$) dans des cas rares ou trop coûteux.
- Opérateurs de borne supérieure $\sqcup$ qui produisent un majorant des arguments mais pas forcément le plus petit.
- Itérations de point fixe qui produisent un post-point fixe mais pas forcément le plus petit point fixe.
- L'approche par validation a posteriori.

Validation a posteriori

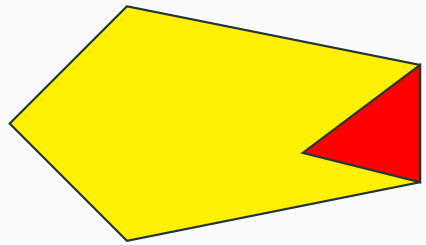
Certaines opérations abstraites peuvent être implémentées par un oracle externe non vérifié s'il est facile de valider le résultat a posteriori par un validateur. Seul le validateur doit être prouvé correct.

Exemple : la borne supérieure $\sqcup$ de deux polyèdres.



Calcul du sup
(enveloppe convexe)

vs.



Test d'inclusion
(Formule de Presburger)

Mécanisation d'un interprète abstrait pour IMP

Modélisation des domaines abstraits

Par des interfaces de modules :

- `VALUE_ABSTRACTION` : abstraction des nombres entiers
- `STORE_ABSTRACTION` : abstraction des états mémoires.

Chaque interface déclare :

- Un type `t` des ≪choses abstraites≫.
- Un prédicat `In` reliant choses concrètes et abstraites.
  (`In` c a équivaut à $c \in \gamma(a)$)
- Des opérations abstraites sur le type `t`
  (opérations arithmétiques ; `get` et `set` pour les états).
- Les énoncés de correction sémantique de ces opérations.

(Voir le fichier Coq `AbstrInterp`.)

Interprétation abstraite des expressions arithmétiques

Soient `ST` une abstraction des états et `V` son abstraction des nombres.

```
Fixpoint Aeval (a: aexp) (S: ST.t) : V.t :=
  match a with
  | CONST n => V.const n
  | VAR x => ST.get x S
  | PLUS a1 a2 => V.add (Aeval a1 S) (Aeval a2 S)
  | MINUS a1 a2 => V.sub (Aeval a1 S) (Aeval a2 S)
  end.
```

(Définition très naturelle.)

Interprétation abstraite des commandes

Calcule l'état abstrait «après» l'exécution de la commande c en fonction de l'état abstrait «avant» S.

```
Fixpoint Cexec (c: com) (S: ST.t) : ST.t :=
  match c with
  | SKIP => S
  | ASSIGN x a => ST.set x (Aeval a S) S
  | SEQ c1 c2 => Cexec c2 (Cexec c1 S)
  | IFTHENELSE b c1 c2 => ST.join (Cexec c1 S) (Cexec c2 S)
  | WHILE b c => postfixpoint (fun X => ST.join S (Cexec c X))
  end.
```

Interprétation abstraite des commandes

```
Fixpoint Cexec (c: com) (S: ST.t) : ST.t :=
  match c with
  | SKIP => S
  | ASSIGN x a => ST.set x (Aeval a S) S
  | SEQ c1 c2 => Cexec c2 (Cexec c1 S)
  | IFTHENELSE b c1 c2 => ST.join (Cexec c1 S) (Cexec c2 S)
  | WHILE b c => postfixpoint (fun X => ST.join S (Cexec c X))
  end.
```

Pour le moment, l'analyse n'essaye pas de déterminer la valeur d'une expression booléenne

$\rightarrow$ on exécute abstraitement les 2 branches `then` et `else`

$\rightarrow$ on prend la borne supérieur de leurs états finaux

Interprétation abstraite des commandes

```
Fixpoint Cexec (c: com) (S: ST.t) : ST.t :=
  match c with
  | SKIP => S
  | ASSIGN x a => ST.set x (Aeval a S) S
  | SEQ c1 c2 => Cexec c2 (Cexec c1 S)
  | IFTHENELSE b c1 c2 => ST.join (Cexec c1 S) (Cexec c2 S)
  | WHILE b c => postfixpoint (fun X => ST.join S (Cexec c X))
  end.
```

Soit X l'état abstrait ≪avant≫ le corps de boucle c.

- Première itération : état S, donc $S \leq X$.
- Itérations suivantes : état `Cexec` c X, donc `Cexec` c $X \leq X$.

On résout ces deux contraintes en calculant un post-point fixe.

Résultats de correction sémantique

On montre sans peine que le résultat d'une exécution concrète appartient au résultat de l'exécution abstraite correspondante.

```
Lemma Aeval_sound:
  forall s S a,
  ST.In s S -> V.In (aeval a s) (Aeval a S).

Theorem Cexec_sound:
  forall c s s' S,
  ST.In s S -> cexec s c s' -> ST.In s' (Cexec c S).
```

Un exemple d'abstraction des états

Paramétrisé par une abstraction `V` des valeurs.

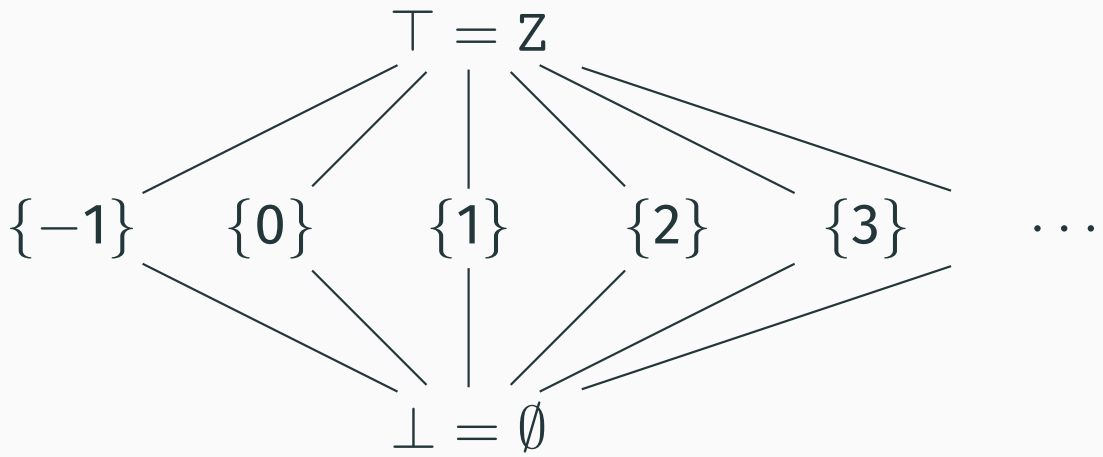
États abstraits = des fonctions `ident` $\to$ `V.t`
telles que l'inclusion $\forall x$, `V.le` $(f\ x)\ (g\ x)$ soit décidable.

Représentation : $\bot$ | fonction finie `ident` $\to$ `V.t`
(valeur par défaut : `V.top`)

Convient à toutes les analyses non-relationnelles.

Un exemple d'abstraction des valeurs

Domaine abstrait = le treillis plat des entiers :



Les opérations abstraites sont triviales :

$$\bot +^{\#} x = x +^{\#} \bot = \bot \quad \{n_1\} +^{\#} \{n_2\} = \{n_1 + n_2\} \quad \top +^{\#} x = x +^{\#} \top = \top$$

Améliorer l'analyse des conditionnelles

Jusqu'ici, nous n'analysons pas les expressions booléennes
$\rightarrow$ les branches `then` et `else` d'un `if` sont supposées exécutées.

Nous pouvons mieux faire si l'information abstraite permet de résoudre statiquement la condition du `if`. Exemple :

```
x := 0;
if x = 0 then y := 1 else y := 2
```

Notre analyse de constante infère $y^{\#} = \{1\} \sqcup \{2\} = \top$.

Puisque $x^{\#} = \{0\}$ avant le `if`, la branche `else` n'est jamais exécutée, et on devrait avoir $y^{\#} = \{1\}$ à la fin.

Améliorer l'analyse des conditionnelles

Même lorsque l'expression booléenne ne peut être calculée statiquement, l'analyse peut apprendre des propriétés intéressantes en regardant quelle branche du `if` est prise.

```
if x = 0 then
    y := x + 1
else
    y := 1
```

$x^\# = \top$ au début

on apprend que $x^\# = \{0\}$

donc $y^\# = \{1\}$

ici aussi $y^\# = \{1\}$

donc $y^\# = \{1\}$ et non $\top$

Améliorer l'analyse des boucles

On peut aussi tirer des enseignements du fait qu'une boucle `while` termine :

$x^\# = \top$ au début

```
while not (x = 10) do
    x := x + 1
done
```

on apprend que $x^\# = \{10\}$

Exemple plus réaliste avec des intervalles au lieu de constantes :

$x^\# = \top = [-\infty, +\infty]$ au début

```
while x <= 1000 do
    x := x + 1
done
```

on apprend que $x^\# = [1001, +\infty]$

Analyse inverse des expressions

`assume_test` *b res S*

calcule un état abstrait $S' \leq S$ reflétant le fait que *b* (une expression booléenne) s'évalue à *res* (`true` ou `false`).

`assume_eval` *a Res S*

calcule un état abstrait $S' \leq S$ reflétant le fait que *a* (une expression arithmétique) s'évalue en un entier qui appartient à *Res* (une valeur abstraite).

Exemples :

$\texttt{assume\_test}\ (x = 0)\ \texttt{true}\ (x \mapsto \top) = (x \mapsto \{0\})$

$\texttt{assume\_test}\ (x = 0)\ \texttt{true}\ (x \mapsto \{1\}) = \bot$

$\texttt{assume\_eval}\ (x + 1)\ \{10\}\ (x \mapsto \top) = (x \mapsto \{9\})$

Analyse inverse des expressions

On ajoute au domaine abstrait des valeurs des tests abstraits inverses `eq_inv`, `ne_inv`, `le_inv`, `gt_inv`.

Formellement :

$$\begin{aligned} \texttt{le\_inv}\ N_1\ N_2 \quad = \quad & (\alpha\{x \mid x \in \gamma(N_1), y \in \gamma(N_2), x \leq y\}, \\ & \alpha\{y \mid x \in \gamma(N_1), y \in \gamma(N_2), x \leq y\}) \end{aligned}$$

Exemples avec des intervalles :

```
eq_inv [0,5] [4,9] = ([4,5], [4,5])
le_inv [0,9] [2,5] = ([0,5], [2,5])
```

(Voir fichier Coq `AbstrInterp2`.)

Analyse améliorée des commandes

```
Fixpoint Cexec (c: com) (S: ST.t) : ST.t :=
  match c with
  | SKIP => S
  | ASSIGN x a => ST.set x (Aeval a S) S
  | SEQ c1 c2 => Cexec c2 (Cexec c1 S)
  | IFTHENELSE b c1 c2 =>
      ST.join (Cexec c1 (assume_test b true S))
              (Cexec c2 (assume_test b false S))
  | WHILE b c =>
      assume_test b false
        (postfixpoint
          (fun X => ST.join S (Cexec c (assume_test b true X))))
  end.
```

Calcul de points fixes et élargissement

Améliorer le calcul des points fixes

L'analyse des boucles nécessite le calcul d'un post-point fixe.

Au 3e cours nous avons vu deux approches :

1. L'approche du théorème de Knaster-Tarski :
   itération qui termine toujours pourvu qu'il n'y ait pas de suite infinie croissante de valeurs abstraites $N_0 < N_1 < \cdots$
2. L'approche avec fuel :
   on renvoie $\top$ au bout de N itérations infructueuses.

1- ne s'applique pas toujours ou est trop lente.

2- est trop imprécise.

Domaines abstraits pas bien fondés

De nombreux domaines abstraits intéressants ont des suites infinies croissantes. Par exemple les intervalles :

$$[0,0] < [0,1] < [0,2] < \cdots < [0,n] < \cdots$$

C'est un problème pour analyser des boucles non comptées :

```
x := 0;
while cond do x := x + 1
```

$x^\#$ est successivement $[0,0]$, puis $[0,1]$, puis $[0,2]$, puis ...

Convergence trop lente

Dans d'autres cas, l'itération de Knaster-Tarski termine, mais prend trop de temps.

```
x := 0;
while x <= 1000 do x := x + 1
```

En partant de $x^{\#} = [0, 0]$, il faut 1000 itérations pour atteindre le point fixe $x^{\#} = [0, 1000]$.

Convergence trop imprécise

L'approche avec fuel converge en un nombre fixé N d'itérations.
Mais dans le cas où elle retourne $\top$ on perd trop d'information.

```
x := 0;
y := 0;
while x <= 1000 do x := x + 1
```

Dans l'état abstrait final, non seulement $x^{\#} = \top$ mais aussi $y^{\#} = \top$.

L'élargissement (*widening*)

Un opérateur d'élargissement $\nabla : A \to A \to A$ calcule un majorant de ses deux arguments, choisi suffisamment grand pour que l'itération ci-dessous converge toujours et converge rapidement :

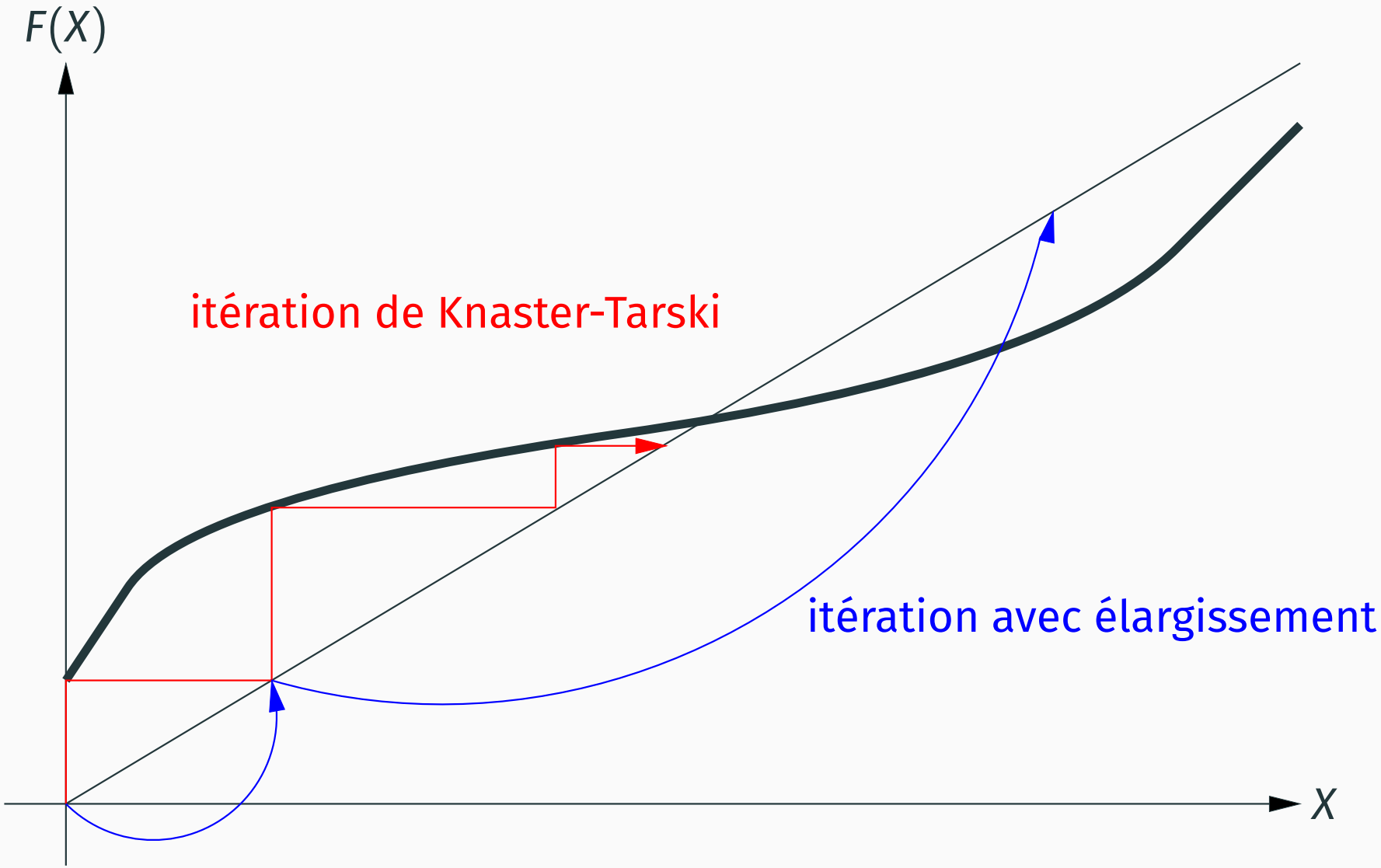
$$X_0 = \bot \qquad X_{i+1} = \begin{cases} X_i & \text{si } F(X_i) \leq X_i \\ X_i \nabla F(X_i) & \text{sinon} \end{cases}$$

La limite de cette suite est un post-point fixe de F.

Exemple : l'élargissement standard pour les intervalles.

$$[l_1, h_1] \nabla [l_2, h_2] = [\,\texttt{if } l_2 < l_1 \texttt{ then } -\infty \texttt{ else } l_1,\ \texttt{if } h_2 > h_1 \texttt{ then } +\infty \texttt{ else } h_1\,]$$

L'élargissement en action



Analyse statique avec élargissement

```
x := 0;
while x <= 1000 do x := x + 1
```

L'abstraction de `x` est un post-point fixe de l'opérateur
$F(X) = [0, 0] \cup (X \cap [-\infty, 1000]) + 1$.

$X_0 = \bot$
$X_1 = X_0 \nabla F(X_0) = \bot \nabla [0, 0] = [0, 0]$
$X_2 = X_1 \nabla F(X_1) = [0, 0] \nabla [0, 1] = [0, +\infty]$
X_2 est un post-point fixe : $F(X_2) = [0, 1001] \leq [0, +\infty]$.

État abstrait final : $x^{\#} = [0, +\infty] \cap [1001, +\infty] = [1001, +\infty]$.

Réduire le post-point fixe

On peut obtenir un meilleur post-point fixe en itérant quelques tours supplémentaires :

$$Y_0 = \text{un post-point fixe} \qquad Y_{i+1} = F(Y_i)$$

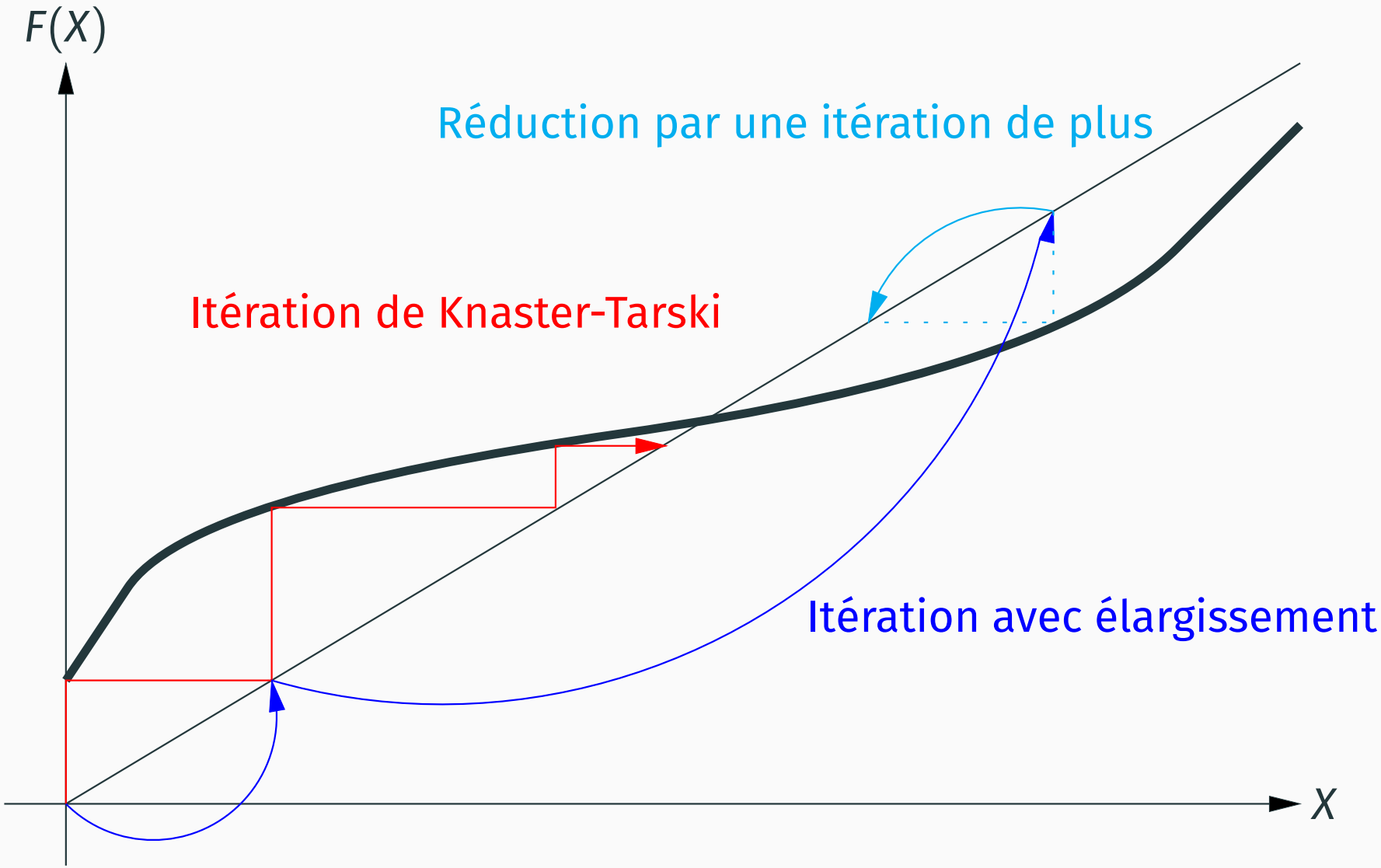
Si F est croissante, chaque Y_i est un post-point fixe : $F(Y_i) \leq Y_i$.

Souvent, $Y_i < Y_0$, ce qui donne un post-point fixe plus précis.

On peut arrêter l'itération quand Y_i est un point fixe, ou au bout d'un nombre fini de tours, ou encore utiliser un opérateur de rétrécissement (*narrowing*) pour accélérer la convergence.

(P. Cousot et R. Cousot, *Comparing the Galois connection and widening/narrowing approaches to abstract interpretation*, PLILP 1992)

Élargissement plus réduction en action



Analyse statique avec élargissement et réduction

```
x := 0;
while x <= 1000 do x := x + 1
```

L'abstraction de `x` est un post-point fixe de l'opérateur $F(X) = [0, 0] \cup (X \cap [-\infty, 1000]) + 1$.

Le post-point fixe trouvé par itération élargie est $[0, +\infty]$.

$Y_0 = [0, +\infty]$
$Y_1 = F(Y_0) = [0, 1001]$
$Y_2 = F(Y_1) = [0, 1001]$

Le post-point fixe final est $Y_1 = [0, 1001]$ (c'est un point fixe).

État abstrait final : $x^{\#} = [0, 1001] \cap [1001, +\infty] = [1001, 1001]$.

Mise en œuvre dans le développement Coq

On ajoute aux interfaces `VALUE_ABSTRACTION` et `STORE_ABSTRACTION` un opérateur `widen` et les propriétés qui garantissent que l'ordre d'itération avec élargissement

```
Definition widen_order (S S1: t) :=
  exists S2, S = widen S1 S2 /\ ble S2 S1 = false.
```

est bien fondé.

On implémente le calcul du post-point fixe par récursion noetherienne sur cet ordre.

(Voir fichier Coq `AbstrInterp2`.)

Point d’étape

Point d'étape

L'approche par interprétation abstraite produit des analyseurs statiques très modulaires :

- Un interprète abstrait générique pour chaque langage.
- Des domaines abstraits largement indépendants du langage.
- Des mécanismes pour combiner plusieurs domaines.

La correction vis-à-vis de la sémantique concrète est soit par construction (dans l'approche ≪par calcul≫), soit vérifiable de manière également modulaire.

Les analyses relationnelles sont plus difficiles (et plus précises !) que les analyses non-relationnelles vues ici.

À suivre dans le séminaire de David Pichardie le 30 janvier.

Bibliographie

Bibliographie

Les bases de l'interprétation abstraite :

- P. Cousot, *Interprétation abstraite*, TSI(19), 2000.
- P. Cousot, *Abstract Interpretation Based Formal Methods and Future Challenges*, LNCS(2000), 2001.

Quelques applications industrielles :

- D. Kästner et al, *Astrée : Proving the Absence of Runtime Errors*, ERTS 2010.
- M. Fähndrich, F. Logozzo, *Static Contract Checking with Abstract Interpretation*, FoVeOOS 2010.
- C. Ferdinand, R. Heckmann, R. Wilhelm, *Analyzing the Worst-Case Execution Time by Abstract Interpretation of Executable Code*, ASWSD 2004.

Mécanisation d'interpréteurs abstraits :

- T. Nipkow, G. Klein, *Concrete Semantics*, chap. 13.
- J.-H. Jourdan, V. Laporte, S. Blazy, X. Leroy, D. Pichardie, *A Formally-Verified C Static Analyzer*, POPL 2015.