Vérification en Coq d'analyseurs statiques par interprétation abstraite

Xavier Leroy

Inria Paris-Rocquencourt

JFLA 2014

Remerciements

David Pichardie

et toute la fine équipe du projet Verasco :

Sandrine Blazy, Vincent Laporte, André Maronèze (Rennes)
Jacques-Henri Jourdan, Jérôme Feret, Xavier Rival (Paris-Rocquencourt)
Alexis Fouilhé, David Monniaux, Michael Périn (Grenoble)
Jean Souyris (Airbus)

Plan

1. Introduction à l'analyse statique
2. L'analyse statique par interprétation abstraite
3. Un peu de Coq : un interprète abstrait ≪pédagogique≫
4. Passage à l'échelle : le projet Verasco
5. Conclusion et travaux futurs

L'analyse statique en bref

Inférer statiquement des propriétés d'un programmes qui sont vraies de toutes ses exécutions.

Au point p, on a $0 < x \leq y$ *et le pointeur p n'est pas* `NULL`.

Accent sur inférer : pas d'aide du programmeur.
(P.ex. pas besoin de marquer les invariants de boucles dans le source.)

Accent sur statiquement :

- Les entrées du programmes sont inconnues.
- L'analyse doit toujours terminer.
- L'analyse doit s'exécuter en temps et espace raisonnables.

Exemples de propriétés inférables

Propriétés de la valeur d'une variable : (analyse de valeurs)

$x = n$	propagation des constantes
$x > 0$ ou $x = 0$ ou $x < 0$	signes
$x \in [n_1, n_2]$	intervalles
$x = n_1 \pmod{n_2}$	congruences
$\mathtt{valid}(p[n_1 \ldots n_2])$	validité des pointeurs
p $\mathtt{pointsTo}$ x or $p \neq q$	(non-) aliasing entre pointeurs

(n, n_1, n_2 sont des constantes déterminées par l'analyse.)

Exemples de propriétés inférables

Propriétés de plusieurs variables : (analyse relationnelle)

$\sum a_i x_i \leq c$	polyèdres
$\pm x_1 \pm \cdots \pm x_n \leq c$	octogones
$expr_1 = expr_2$	équivalences de Herbrand

(a_i, c sont des constantes rationnelles déterminées par l'analyse.)

Propriétés «non fonctionnelles» :

- Consommation mémoire.
- Pire temps d'exécution (WCET).

Utiliser l'analyse statique pour l'optimisation

Appliquer des égalités algébriques lorsque leurs conditions sont satisfaites :

`x / 4`	$\rightarrow$	`x >> 2`	si l'analyse dit $\texttt{x} \geq 0$
`x + 1`	$\rightarrow$	`1`	si l'analyse dit $\texttt{x} = 0$

Optimiser les accès aux tableaux et via des pointeurs :

`a[i]=1; a[j]=2; x=a[i];` $\rightarrow$ `a[i]=1; a[j]=2; x=1;`
si l'analyse dit $\texttt{i} \neq \texttt{j}$

`*p = a; x = *q;` $\rightarrow$ `x = *q; *p = a;`
si l'analyse dit $\texttt{p} \neq \texttt{q}$

Parallélisation automatique :

$$loop_1; loop_2 \rightarrow loop_1 \parallel loop_2 \quad \text{si } polyh(loop_1) \cap polyh(loop_2) = \emptyset$$

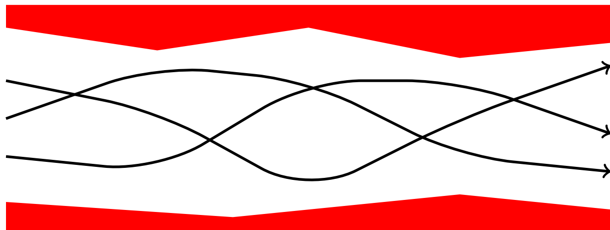
Utiliser l'analyse statique pour la vérification

Utiliser les résultats de l'analyse statique pour prouver l'absence de certaines erreurs à l'exécution :

$$b \in [n_1, n_2] \wedge 0 \notin [n_1, n_2] \implies a/b \text{ ne peut pas échouer}$$

$$\texttt{valid}(p[n_1 \ldots n_2]) \wedge i \in [n_1, n_2] \implies *(\texttt{p} + \texttt{i}) \text{ ne peut pas échouer}$$

Signaler une alarme dans le cas contraire.



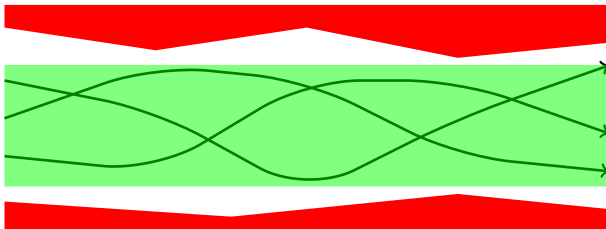
Utiliser l'analyse statique pour la vérification

Utiliser les résultats de l'analyse statique pour prouver l'absence de certaines erreurs à l'exécution :

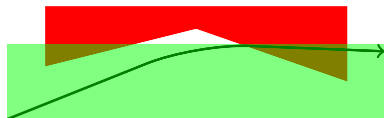
$$b \in [n_1, n_2] \wedge 0 \notin [n_1, n_2] \implies a/b \text{ ne peut pas échouer}$$

$$\texttt{valid}(p[n_1 \ldots n_2]) \wedge i \in [n_1, n_2] \implies *(\texttt{p} + \texttt{i}) \text{ ne peut pas échouer}$$

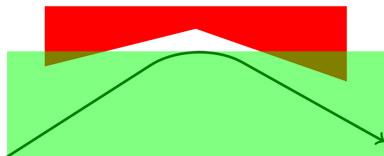
Signaler une alarme dans le cas contraire.



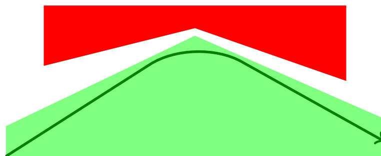
Vraies alarmes, fausses alarmes



Vraie alarme
(comportement dangereux)



Fausse alarme
(analyse trop imprécise)



Analyse plus précise (polyèdre au lieu d'intervalles) :
la fausse alarme disparaît.

Quelques propriétés vérifiables par analyse statiques

Absence d'erreurs à l'exécution :

- Tableaux et pointeurs :
  - Pas d'accès hors bornes.
  - Pas de déréférencement du pointeur nul.
  - Pas d'accès après un `free`.
  - Contraintes d'alignement du processeur.
- Arithmétique entière :
  - Pas de division par zéro.
  - Pas de débordements arithmétiques (signés).
- Arithmétique à virgule flottante :
  - Pas de débordements arithmétiques (résultats $\pm\infty$)
  - Pas d'opérations indéfinies (résultats *Not a Number*)
  - Pas de pertes de précision catastrophiques.

Assertions simples posées par le programmeur.

P.ex. `assert (0 <= x && x < sizeof(tbl))`.

Plan

1. Introduction à l'analyse statique
2. L'analyse statique par interprétation abstraite
3. Un peu de Coq : un interprète abstrait «pédagogique»
4. Passage à l'échelle : le projet Verasco
5. Conclusion et travaux futurs

Idée n° 1 :
analyser un programme,
c'est l'exécuter avec une sémantique non-standard

L'interprétation abstraite pour les nuls

$\ll$Exécuter$\gg$ le programme avec une sémantique qui :

- Calcule sur un domaine abstrait des propriétés désirées (p.ex. $\ll x \in [n_1, n_2] \gg$ pour l'analyse d'intervalles) au lieu de calculer sur des choses concrètes comme les valeurs et les états.
- Gère les conditionnelles même si elles ne peuvent être résolues statiquement.
  (Les branches `then` et `else` d'un `if` sont considérées comme prises toutes les deux. Les boucles `while` exécutent un nombre arbitraire d'itérations.)
- Termine toujours.

Exemple d'interprétation abstraite

Concret	Abstrait
$\{\ x = 3, y = 1\ \}$	$\{\ x^\# = [0,9], y^\# = [-1,1]\ \}$

```
z = x + 2 * y;
```

$\{\ z = 3 + 2 \times 1 = 5\ \}$	$\{\ z^\# = [0,9] +^\# 2 \times^\# [-1,1] = [-2,11]\ \}$
$\{\ b = \mathtt{true}, x = 3, y = 1\ \}$	$\{\ b^\# = \top, x^\# = [0,9], y^\# = [-1,1]\ \}$

```
z = (if b then x else y);
```

$\{\ z = 3\ \}$	$\{\ z^\# = [0,9] \sqcup [-1,1] = [-1,9]\ \}$

Idée n° 2 :
une variable peut avoir différentes abstractions
en différents points de programmes

Prise en compte du flot de contrôle (*flow sensitivity*)

(Au contraire des systèmes de types, où généralement une variable garde le même type en tout point de sa portée.)

Affectation impérative de variables :

$\{\ x^\# = [0,9]\ \}$

```
x = x + 1;
```

$\{\ x^\# = [1,10]\ \}$

Prise en compte des conditions booléennes :

$\{\ x^\# = [0,9]\ \}$

```
if (x == 0) {
```

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

```
    ...
} else {
```

$\{\ x^\# = [1,9]\ \}$

```
    ...
}
```

Idée n° 3 :
on peut aussi inférer des relations
entre les valeurs de plusieurs variables

Analyse relationnelle ou pas ?

Analyse non relationnelle :

$$\textit{environnement abstrait} = \textit{variable} \mapsto \text{valeur abstraite}$$

(Comme les environnements de typage.)

Analyse relationnelle :
les environnements abstraits sont un domaine comme les autres, avec :

- structure de demi-treillis : $\bot$, $\top$, $\sqsubset$, $\sqcup$
- une opération abstraite d'affectation.

Exemple : les polyèdres $\sum a_i x_i \leq c$.

Idée n° 4 : l'élargissement :
on peut calculer des points fixes
même dans des domaines mal fondés

Les points fixes — le problème récurrent

Analyse statique d'une boucle :

```
while (...) {        { e# = X0 }
                     { e# = X }
    ...
                     { e# = Φ(X) }
}
```

$\{\ e^{\#} = X_0\ \}$

$\{\ e^{\#} = X\ \}$

$\{\ e^{\#} = \Phi(X)\ \}$

Étant donnés X_0 (l'état abstrait avant la boucle)
et Φ (la fonction d'analyse du corps de la boucle),
comment trouver X (l'invariant de boucle) ?

$$X \sqsupseteq X_0 \text{ (première itération)} \qquad X \sqsupseteq \Phi(X) \text{ (itérations suivantes)}$$

Solution : X est le plus petit point fixe de $F = X \mapsto X_0 \sqcup \Phi(X)$
ou au moins tout post-point fixe de F $\quad (X \sqsupseteq F(X))$.

Le paradis

Théorème (Tarski)

Soit $(A, \sqsubseteq, \bot)$ *un ensemble partiellement ordonné tel que* $\sqsupset$ *est bien fondé (pas de suites infiniment croissantes).*
Soit $F : A \to A$ *une fonction croissante.*
Alors F *admet un plus petit point fixe qui s'obtient par itération finie à partir de* $\bot$ *:*

$$\exists n, \quad \bot \sqsubset F(\bot) \sqsubset \ldots \sqsubset F^n(\bot) = F^{n+1}(\bot)$$

Paradis perdu

La majorité des domaines abstraits ne sont pas bien fondés. Exemples :

- Les intervalles entiers : $[0,0] \sqsubset [0,1] \sqsubset [0,2] \sqsubset \cdots \sqsubset [0,n] \sqsubset \cdots$
- Les environnements : *variable* $\mapsto$ *valeur abstraite*.

De plus, même lorsque l'itération de Tarski converge, elle converge souvent trop lentement :

```
x = 0;  while (x <= 10000) { x = x + 1; }
```

(Partant de $x^{\#} = [0,0]$, il faut 10000 itérations pour obtenir le point fixe $x^{\#} = [0,10000]$.)

Paradis retrouvé : l'élargissement (*widening*)

Un opérateur d'élargissement $\nabla : A \to A \to A$ calcule une borne supérieure de son second argument de telle manière que l'itération suivante converge toujours et converge rapidement :

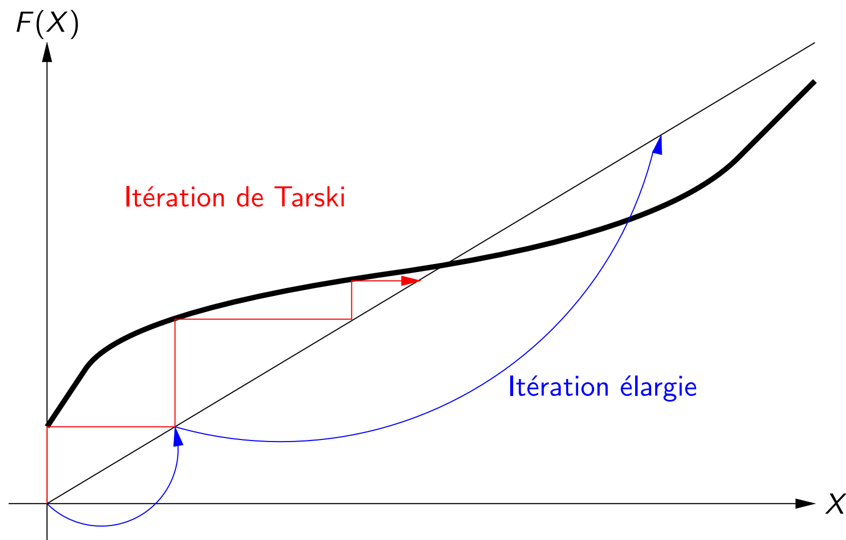
$$X_0 = \bot \qquad X_{i+1} = \begin{cases} X_i & \text{si } F(X_i) \sqsubseteq X_i \\ X_i \nabla F(X_i) & \text{sinon} \end{cases}$$

La limite X de cette suite est un post-point fixe : $F(X) \sqsubseteq X$.

Exemple : élargissement pour les intervalles :

$$[l_1, u_1] \nabla [l_2, u_2] = \; [\texttt{if } l_2 < l_1 \texttt{ then } -\infty \texttt{ else } l_1, \\ \texttt{if } u_2 > u_1 \texttt{ then } \infty \texttt{ else } u_1]$$

L’élargissement en action



Raffiner le post-point fixe

La qualité du post-point fixe peut être améliorée en itérant F un ou plusieurs tours de plus :

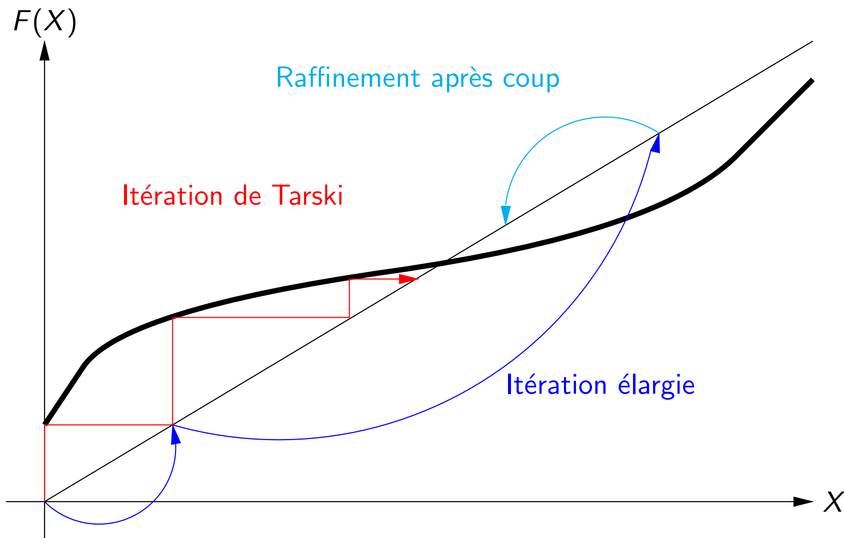
$$Y_0 = \text{un post point fixe} \qquad Y_{i+1} = F(Y_i)$$

Si F est croissante, chacun des Y_i est un post point fixe : $F(Y_i) \sqsubseteq Y_i$.

Souvent, $Y_i \sqsubset Y_0$, ce qui donne un meilleur résultat.

On peut stopper l'itération lorsque Y_i est un point fixe, ou à tout moment.

Élargissement plus raffinement en action

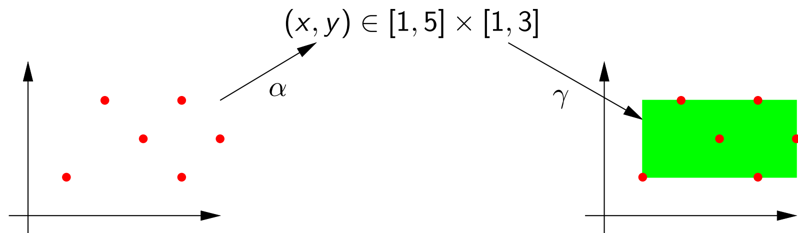


Idée trop belle pour être vraie :
on peut calculer les opérateurs abstraits
de manière systématique, correcte par construction, et optimale

Les correspondances de Galois

Un treillis $\mathcal{A}, \sqsubseteq$ d'états abstraits et deux fonctions :

- Fonction d'abstraction α : ensemble états concrets $\rightarrow$ état abstrait
- Fonction de concrétisation γ : état abstrait $\rightarrow$ ensemble états concrets



α et γ croissantes ; $X \subseteq \gamma(\alpha(X))$; et $x^{\#} \sqsubseteq \alpha(\gamma(x^{\#}))$.

Calculer les opérateurs abstraits

Pour chaque opération du langage, on peut calculer l'opération abstraite correspondante, de manière systématique.

Exemple : pour l'opérateur $+$, on définit

$$a_1 +^{\#} a_2 = \alpha\{n_1 + n_2 \mid n_1 \in \gamma(a_1), n_2 \in \gamma(a_2)\}$$

Après, on simplifie dans cette définition jusqu'à obtenir un algorithme, p.ex.

$$[l_1, u_1] +^{\#} [l_2, u_2] = [l_1 + l_2, u_1 + u_2]$$

L'opérateur $+^{\#}$ ainsi obtenu est :

- Correct par définition :
  si $n_1 \in \gamma(a_1)$ et $n_2 \in \gamma(a_2)$ alors $n_1 + n_2 \in \gamma(a_1 +^{\#} a_2)$
- Le plus précis possible par rapport à notre choix de α.

Problèmes avec les correspondances de Galois

(en vue d'une implémentation Coq)

1- L'approche ≪calculatoire≫ est peu pratique en Coq
(cf. stage DEA David Monniaux).

2- La fonction α est généralement non calculable
Exemple des intervalles : $\alpha(X) = [\inf X, \sup X]$
où X est un ensemble possiblement infini de nombres.

3- La fonction α n'existe pas toujours
Exemple des intervalles sur les rationnels :

$$\alpha\{x \in \mathbf{Q} \mid x^2 \leq 2\} = [??, ??]$$

(pas possible de prendre $[-\sqrt{2}, \sqrt{2}]$).

Idée piétonne :
définir manuellement les opérateurs abstraits
et les prouver corrects ensuite

Plan B

Oublions les α et l'optimalité. Focalisons-nous sur les γ et la correction.

$$\gamma : A \to \mathcal{P}(C) \equiv A \to C \to \texttt{Prop} \equiv C \to A \to \texttt{Prop}$$

Chaque domaine abstrait vient avec une relation $\in$:

$$\textit{chose-concrète} \in \textit{chose-abstraite}$$

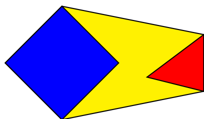
Les opérateurs abstraits sont définis «au pif» ou par calcul sur papier, puis prouvés corrects ensuite :

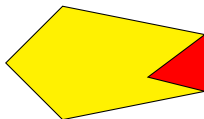
```
Definition aplus (a1 a2: A) : A := ...

Theorem aplus_sound:
  forall v1 a1 v2 a2,
  v1 ∈ a1 -> v2 ∈ a2 -> (v1 + v2) ∈ (aplus a1 a2).
```

Validation a posteriori

Certaines opérations abstraites peuvent être implémentées de manière non prouvées s'il est facile de valider leurs résultats *a posteriori* par un validateur prouvé correct.

Exemple : la borne supérieure $\sqcup$ pour les polyèdres.





Calculer l'union vs. Tester l'inclusion

(enveloppe convexe) (formule de Presburger)

Validation a posteriori de l'opération $\sqcup$

```
Parameter ajoin_impl: A -> A -> A.

Definition aincl_b (a1 a2: A) : A := ...

Theorem aincl_b_sound:
  forall v a1 a2, aincl_b a1 a2 = true -> v ∈ a1 -> v ∈ a2.
Proof. (* un peu de travail *) Qed.

Definition ajoin (a1 a2: A) : A :=
  let a := ajoin_impl a1 a2 in
  if aincl_b a1 a a&& aincl_b a2 a then a else top.

Theorem ajoin_sound:
  forall v a1 a2, v ∈ a1  v ∈ a2 -> v ∈ (ajoin a1 a2).
Proof. (* immédiat *) Qed.
```

Validation a posteriori de $\sqsubseteq$ sur les polyèdres

Le polyèdre $A_1 \wedge \cdots \wedge A_n$ est inclus dans $B_1 \wedge \cdots \wedge B_m$
ssi la formule $A_1 \wedge \cdots \wedge A_n \Rightarrow B_1 \wedge \cdots \wedge B_m$ est toujours vraie
ssi les formules $A_1 \wedge \cdots \wedge A_n \wedge \neg B_i$ sont insatisfiables.

Lemme (Farkas)

Un système d'inéquations linéaires $A_1 \wedge \cdots \wedge A_n$ est insatisfiable ssi il existe des coefficients $k_1, \ldots, k_n$ tels que la combinaison linéaire $k_1 A_1 + \cdots + k_n A_n$ est égale à $0 \geq 1$.

Algorithme :

- Faire calculer les coefficients k_i par une implémentation externe non prouvée du Simplex.
- Vérifier que $k_1 A_1 + \cdots + k_n A_n \equiv (0 \geq 1)$.

Plan

1 Introduction à l'analyse statique

2 L'analyse statique par interprétation abstraite

3 Un peu de Coq : un interprète abstrait «pédagogique»

4 Passage à l'échelle : le projet Verasco

5 Conclusion et travaux futurs

Un interprète abstrait pour Imp

Le mini-langage Imp de *Software Foundations* :

Arithmetic expressions :

$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$$

Boolean expressions :

$$b ::= \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \texttt{not}\ b \mid b_1\ \texttt{and}\ b_2$$

Commands (statements) :

$c ::=$	$\texttt{SKIP}$	(do nothing)
$\mid$	$x ::= a$	(assignment)
$\mid$	$c_1; c_2$	(sequence)
$\mid$	$\texttt{IFB}\ b\ \texttt{THEN}\ c_1\ \texttt{ELSE}\ c_2\ \texttt{FI}$	(conditional)
$\mid$	$\texttt{WHILE}\ b\ \texttt{DO}\ c\ \texttt{END}$	(loop)

Notes de cours : *Proving a Compiler : Mechanized verification of program transformations and static analyses*, école d'été Oregon, 2010–2012, `http://gallium.inria.fr/~xleroy/courses/Eugene-2012/`.

Plan

1. Introduction à l'analyse statique
2. L'analyse statique par interprétation abstraite
3. Un peu de Coq : un interprète abstrait «pédagogique»
4. Passage à l'échelle : le projet Verasco
5. Conclusion et travaux futurs

Le projet Verasco

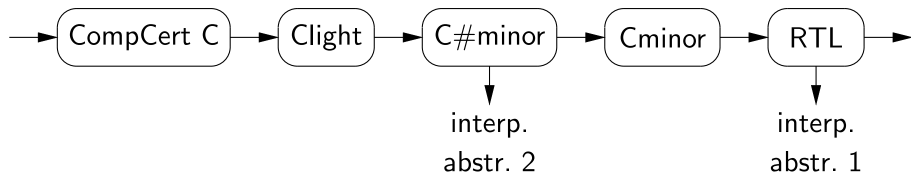
Inria Celtique, Gallium, Abstraction, Toccata + Verimag + Airbus

Objectif : développer et vérifier en Coq un analyseur statique par interprétation abstraite qui soit réaliste :

- Langage analysé : le sous-ensemble C de CompCert.
- Domaines abstraits non triviaux, incl. domaines relationnels.
- Architecture modulaire inspirée d'Astrée.
- Rapport d'alarmes.

Slogan : si ≪CompCert = 1/10e de GCC mais prouvé≫,
de même ≪Verasco = 1/10e d'Astrée mais prouvé≫.

Couche haute : les interprètes abstraits



Se branchent sur les langages intermédiaires du compilateur CompCert.

Paramétrés par une domaine abstrait pour les états (environnement + état mémoire + pile d'appels).

1. Interprète abstrait pour RTL (Blazy, Maronèze, Pichardie, SAS 2013)
   Contrôle non structuré $\rightarrow$ points fixes fonction par fonction, algorithme de Bourdoncle.
2. Interprète abstrait pour C#minor (Jourdan, en cours)
   Points fixes locaux pour les boucles, par fonction pour les `goto`.

Couche basse : les domaines numériques «idéaux»

Non-relationnels :

- Intervalles entiers (sur **Z**).
- Congruences entières.
- Intervalles flottants (au dessus de la bibliothèque Flocq).

Relationnels :

- La bibliothèque VPL (Fouilhé, Monniaux, Périn, SAS 2013) : polyèdres à coefficients rationnels, implémentés en OCaml, avec production de certificats vérifiables en Coq.
- Intégration en cours dans Verasco.
  (Transparent suivant : l'interface d'un domaine relationnel idéal.)

```
Class ab_ideal_env (var t:Type) `{EqDec var}: Type := {
  id_wl:> weak_lattice t;
  id_gamma:> gamma_op t (var->ideal_num);
  id_adom:> adom t (var->ideal_num) id_wl id_gamma;
  get_itv: iexpr var -> t -> IdealIntervals.abs+⊥;
  assign: var -> iexpr var -> t -> t+⊥;
  forget: var -> t -> t+⊥;
  assume: iexpr var -> bool -> t -> t+⊥;
  get_itv_correct: forall e ρ ab,
    ρ ∈ γ ab ->
    eval_iexpr ρ e ⊆ γ (get_itv e ab);
  assign_correct: forall x e ρ n ab,
    ρ ∈ γ ab ->
    n ∈ eval_iexpr ρ e ->
    (upd ρ x n) ∈ γ (assign x e ab);
  forget_correct: forall x ρ n ab,
    ρ ∈ γ ab ->
    (upd ρ x n) ∈ γ (forget x ab);
  assume_correct: forall c ρ ab b,
    ρ ∈ γ ab ->
    (INz (if b:bool then 1 else 0)) ∈ eval_iexpr ρ c ->
    ρ ∈ γ (assume c b ab)
}.
```

Transformateurs de domaines numériques

Domaine non-relationnel idéal $\rightarrow$ domaine relationnel idéal :

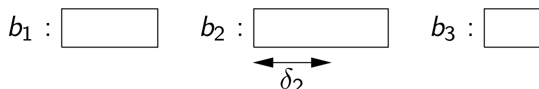
- Pour `assign` : interp. abstraite des expressions
- Pour `assume` : interp. abstraite inverse.

Domaine relationnel idéal $\rightarrow$ domaine relationnel ≪machine≫ :

- Idée : $x \in \mathbf{Z} \approx n \in$ `int` ssi $n = z \bmod 2^{32}$.
- Compatible avec addition, soustraction, multiplication.
- Pour les autres opérations (comparaisons, divisions, ...) :
  essayer de réduire l'entier idéal dans l'intervalle $[0, 2^{32}[$ ou $[-2^{31}, 2^{31}[$
  selon que l'opération est signée ou non.

Couche médiane : abstraction de la mémoire et des états

Le modèle mémoire CompCert : adresse mémoire = bloc b × offset δ.



Abstraction des offsets → domaine numérique.

Abstraction des blocs :

- Solution courante : 1 bloc concret = 1 bloc abstrait
  ≪variable globale x≫ ou ≪variable locale y de la fonction f≫.
- Récursion, allocation dynamique → besoin de blocs abstraits imprécis (abstrayant plusieurs blocs concrets).
- Besoin de fusionner différents offsets d'un même bloc (gros tableau p.ex.).

Plan

1. Introduction à l'analyse statique
2. L'analyse statique par interprétation abstraite
3. Un peu de Coq : un interprète abstrait «pédagogique»
4. Passage à l'échelle : le projet Verasco
5. Conclusion et travaux futurs

Travaux futurs

Beaucoup reste à faire pour atteindre un ≪vrai≫ analyseur statique :

- ≪Bonnes≫ approximations de la mémoire.
- Davantage de (combinaisons de) domaines abstraits :
  égalités symboliques, produits réduits, partitionnement de traces.
- Efficacité algorithmique :
  - Préservation du partage entre représentations d'états abstraits.
  - Repartage a posteriori (*hash-consing* ?)
  - Domaines relationnels en $O(N)$, $N =$ nombre de variables...
    $\Rightarrow$ partitionnement en groupes de $n \ll N$ variables.
- Rapport d'alarmes.
- Débugger la précision des analyses.

Petit à petit...

... on se rapproche d'une vérification formelle des outils intervenant dans la production et la vérification des logiciels embarqués critiques.