Analyse de programmes par interprétation abstraite

Marc PEGON
marc.pegon@ensimag.imag.fr

Ensimag - Grenoble INP

Encadrants

David MONNIAUX Matthieu MOY

- Analyse statique : obtenir automatiquement des informations sur le comportement d'un programme, sans réellement l'exécuter
  - model-checking
  - **interprétation abstraite**
- Outils d'analyse utilisés dans l'industrie : Polyspace, Astrée, Coverity (?)..
- Contribution : développement d'un outil d'analyse de programmes C par interprétation abstraite

Plan

- **Prouver** des propriétés de sûreté (débordements arithmétiques, divisions par 0, ..)
- Etat indésirable = Point de programme + Valeurs des variables
- Idée : Déterminer toutes les valeurs prises par les variables
- Problème : Trop complexe (théorie)
- Solution : **Approximation** (Abstraction)

Calcul d'intervalles

Calculer un intervalle de valeurs possibles pour chaque variable

Exemple

```
// x ∈ [2, 5]
y = 2 * x;
// y ∈ [4, 10]
if (y <= 4) {
    z = y - 2;
    // z ∈ [2, 2]
} else {
    z = y + 2;
    // z ∈ [7, 12]
}
// z ∈ [2, 12]
```

Et avec des boucles ?

Rechercher un **invariant** :

Exemple

```
x = 2;

// [2,2], [2,4], .., [2,12], [2,12]
while (x <= 10) {
    x += 2;
    // [4,4], [4,6], .., [4,12]
}

// ∅, ∅, .., [11,12]
```

Elargissement

"Extrapolation" pour accélérer/permettre la convergence

Exemple

```
x = 2;

// [2,2], [2,4] → [2,+∞[, [2,12] → [2,+∞[
while (x <= 10) {
    x += 2;
    // [4,4], [4,12]
}

// ∅, [11,+∞[
```

Réduction de l'invariant

Par un pas de calcul supplémentaire

Exemple

```
x = 2;

// [2, +∞[, [2, 12]
while (x <= 10) {
    x += 2;
    // [4, 12], [4, 12]
}

// [11, +∞[, [11, 12]
```

Problème avec les intervalles : aucune relation entre les variables

Exemple

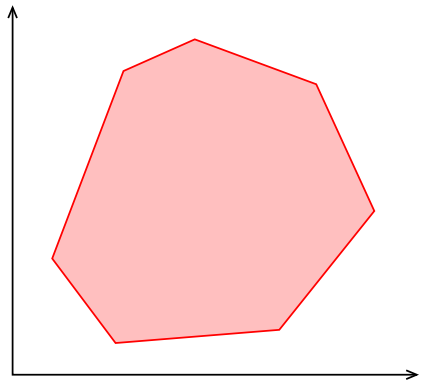
```
// x ∈ [0, 1]
y = x;
// y ∈ [0, 1]
z = x-y;
// z ∈ [-1, 1]
```

Problème avec les intervalles : aucune relation entre les variables

Exemple

```
// x ∈ [0, 1]
y = x;
// y ∈ [0, 1]
z = x-y;
// z ∈ [-1, 1]
```

Solution : **polyèdres** convexes



Plan

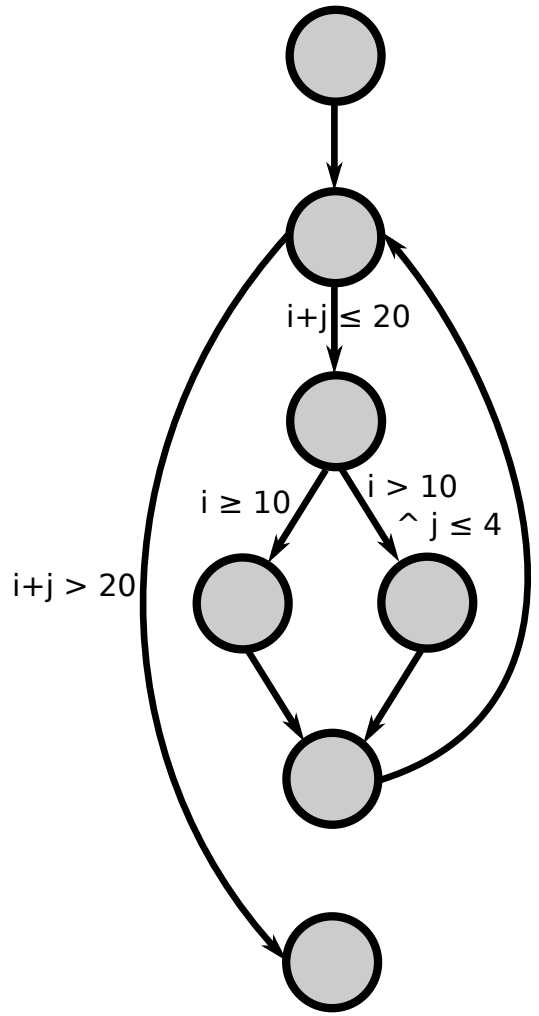
- **Vérifier** des propriétés : *check*
- Faire des **hypothèses** sur les variables : *assume*
- Analyseur correct (*sound*)
  - Ne se trompe jamais quand il ne signale pas d'erreur
  - Peut émettre des fausses alarmes

Exemple

```
assume(x >= 5);
y = x / 2;
check(y > 0);
```

- On n'analyse pas directement du code C
- Représentation intermédiaire : bitcode LLVM
  - Programme : graphe de flot de contrôle
  - Noeuds : blocs de base
  - Arcs : sauts (conditionnels ou non)

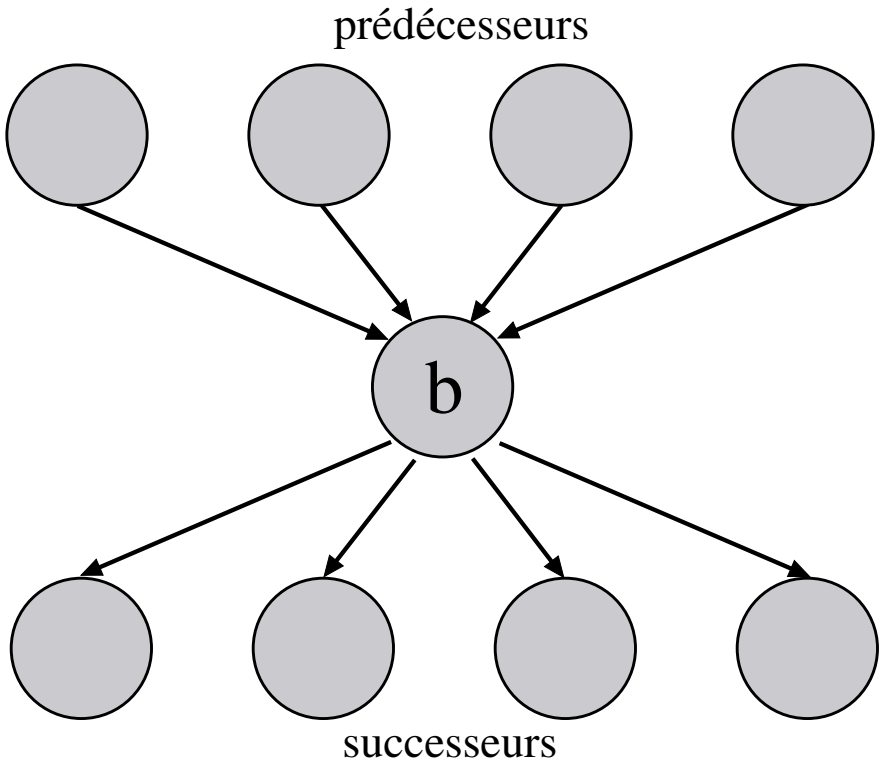
```
int i = 2;
int j = 0;

while (i+j <= 20){
    if (i >= 10) {
        i += 4;
    } else if (j <= 4){
        i += 2;
        j++;
    }
}
```



- Utilisation de Apron pour les calculs sur les polyèdres (enveloppe convexe, affectation, ..)
- Apron = bibliothèque de calcul pour les domaines numériques abstraits
  - Intervalles
  - Egalités linéaires
  - Octogones
  - **Polyèdres convexes**
- Interface **commune** entre différentes bibliothèques de calcul

  $\Rightarrow$ une ligne à changer dans le code de l'analyseur pour changer de domaine

- Calcul d'un polyèdre par bloc de base : approximation supérieure des états possibles du programme au niveau de ce bloc



- Arrêt quand tous les polyèdres sont stabilisés
- Phase de descente : visite des blocs sans élargissement

Plan

- Peu de lignes de code ($\simeq 1000$)
- Analyseur correct
- Limitations :
  - Seule la fonction `main` est analysée
  - Variables **numériques entières** uniquement
  - Entiers $=$ éléments de $\mathbb{Z}$

Exemple

```
int i = 2;
int j = 0;

while (i <= 10) {
    i = i + 2;
    j = j + 1;
    check(i == 2*(j+1));
}

check(j == 5);
}
```

Les deux propriétés sont **prouvées** automatiquement par l'analyseur.

Perspectives

- Analyse de plusieurs fonctions et gestion des appels de fonction
- Gestion des variables flottantes (possible grâce à Apron)
- Gestion des variables booléennes (model checking ?)
- Réduction des invariants (en élargissant en moins de points)
- Evaluation et amélioration des performances (réduire le nombre de dimensions, ..)

Merci