AST

Analyse Statique pour l'optimisation de programmes

David Pichardie

12 mars 2020

Rappel

Nous avons vu plusieurs analyses statiques exprimées comme des ensembles de *faits* attachés à chaque point du programme.

Variables vivantes : $L_{\mathsf{in}}(l), L_{\mathsf{out}}(l) \in \wp(Var)$

Définitions possibles : $DP_{\mathsf{in}}(l), DP_{\mathsf{out}}(l) \in \wp(Var \times Lab^{?})$

Expressions disponibles : $ED_{\mathsf{in}}(l), ED_{\mathsf{out}}(l) \in \wp(Exp)$

Chaque analyse était calculée comme le plus petit, ou plus grand, point fixe d'un système d'équations.

Généralisation

Peux-t-on attacher autre chose que des ensembles ?

$\Longrightarrow$ *nous allons attacher des propriétés, sans forcément les représenter en extension*

Comment décider si nous cherchons le plus petit ou le plus grand point fixe ?

$\Longrightarrow$ *tout dépend de quel ordre on parle !*

Le treillis des propriétés

L'ensemble $\mathcal{P}$ des propriétés doit être un *treillis :* il doit être muni

- d'un *ordre* $\sqsubseteq$
  $P_1 \sqsubseteq P_2$ si la propriétée P_1 contient plus d'informations (implique) que P_2
- d'un élément *minimum* $\perp$ (« bottom »)
  pour tout propriété P, on a $\perp \sqsubseteq P$
- d'une opération de *plus petite borne supérieure* (« join ») $\sqcup$.
  ($P_1 \sqsubseteq P$ et $P_2 \sqsubseteq P$) si et seulement si $P_1 \sqcup P_2 \sqsubseteq P$

Le treillis des propriétés des variables vivantes

Pour l'analyse de durée de vie,

$$\mathcal{P} = (\wp(Var), \subseteq, \emptyset, \cup)$$

$LV_{\text{out}}(n)$ doit *contenir* l'ensemble des variables v telles qu'il *est possible* que v soit utilisée avant d'être (re)définie après le noeud n.

L'ordre est bien l'inclusion $\subseteq$ car, par exemple $LV_{\text{out}}(n) = \{x\}$ *implique* $LV_{\text{out}}(n) = \{x, y\}$.

Remarque : Une fois $\sqsubseteq$ choisi, $\perp$ et $\sqcup$ en découle.

Le treillis des propriétés des définitions possibles

Pour l'analyse des définitions possibles,

$$\mathcal{P} = (\wp(\mathit{Var} \times \mathit{Lab}^?), \subseteq, \emptyset, \cup)$$

$DP_{\mathsf{in}}(n)$ doit *contenir* l'ensemble des couples (n_0, v) tels qu'il *est possible* que la variable v soit définie au noeud n_0 sans être redéfinie entre le noeud n_0 et le noeud n.

L'ordre est bien $\subseteq$ car, par exemple $DP_{\mathsf{in}}(n) = \{(x, n_1)\}$ *implique* $DP_{\mathsf{in}}(n) = \{(x, n_1), (x, n_2)\}$.

Remarque : on se contente d'une *approximation* (verbe *contenir*)

Le treillis des propriétés des expressions disponibles

Pour l'analyse des expressions disponibles

$$\mathcal{P} = (\wp(Exp), \supseteq, Var \times Exp, \cap)$$

Tous les expressions e de $ED_{\mathsf{in}}(n)$ doivent être tels que, au noeud n, tous les chemin menant en n on évaluer e sa,s redéfinir l'une de ses variables depuis.

L'ordre est bien $\supseteq$ car, par exemple $ED_{\mathsf{in}}(n) = \{e_1, e_2\}$ *implique* $ED_{\mathsf{in}}(n) = \{e_1\}$.

Remarque : l'élément minimum $\perp$ se retrouve être le maximum pour l'inclusion !

Treillis des propriétés : plus petit ou plus grand point fixe ?

Tout les points fixes sont correctes mais nous préférons bien sûr le plus informatif : on veut *toujours* calculer le plus petit point fixe.

Attention : on parle ici du plus petit pour $\sqsubseteq$!

Remarque : si $\sqsubseteq$ est en fait $\supseteq$ (par exemple pour ED), alors le plus petit point fixe recherché est en fait le plus grand point fixe vis-à-vis de $\subseteq$.

Treillis des propriétés : terminaison de l'itération de point fixe

On exige de plus, qu'*il n'existe pas de suite strictement croissante infinie*, de façon à garantir la terminaison de l'analyse (qui croît à partir de $\perp$).

Pour cela, il est suffisant (mais non nécessaire)

- que $\mathcal{P}$ soit *de hauteur finie*,
- ou (plus fort), que $\mathcal{P}$ contienne un nombre *fini* de propriétés.

Valuations

Soit $\mathcal{N}$ l'ensemble des noeuds du graphe de flot de contrôle.

On manipule des *valuations* qui à chaque noeud n associent une propriété. Une valuation est donc un élément de $\mathcal{N} \to \mathcal{P}$.

On se donne une valuation X_{in} *à l'entrée* et une valuation X_{out} *à la sortie*.

Les valeurs acceptables pour les valuations sont spécifiées par un système d'équations récursives avant ou arrière, dont on cherche le plus petit point fixe.

Système d'équations

L'existence d'une arête entre n et n' dans le graphe de flot de contrôle est notée $n \to n'$.

On associe à chaque arête $n \to n'$ une *inéquation :*

$$X_{\mathsf{out}}(n) \sqsubseteq X_{\mathsf{in}}(n') \qquad \text{(si analyse en avant)}$$

ou

$$X_{\mathsf{in}}(n') \sqsubseteq X_{\mathsf{out}}(n) \qquad \text{(si analyse en arrière)}$$

Fonctions de transfert

L'effet de chaque instruction est modélisé par une *fonction de transfert* de $\mathcal{P}$ dans $\mathcal{P}$.

Toute fonction de transfert f doit être *monotone*, ce que l'on peut écrire de deux façons équivalentes :

$$p_1 \sqsubseteq p_2 \Rightarrow f(p_1) \sqsubseteq f(p_2)$$
$$f(p_1 \sqcup p_2) \sqsupseteq f(p_1) \sqcup f(p_2)$$

Cette condition signifie qu'une meilleure information à l'entrée d'une instruction doit donner une meilleure information à la sortie.

Fonctions de transfert

On associe à chaque noeud n une fonction de transfert notée transfert(n), d'où on déduit une *inéquation :*

$$\mathsf{transfert}(n)(X_{\mathsf{in}}(n)) \sqsubseteq X_{\mathsf{out}}(n) \qquad \text{(si analyse en avant)}$$

ou

$$\mathsf{transfert}(n)(X_{\mathsf{out}}(n)) \sqsubseteq X_{\mathsf{in}}(n) \qquad \text{(si analyse en arrière)}$$

Pour l'analyse de durée de vie, la fonction de transfert est donnée par :

$$\mathsf{transfert}(n)(S) = (S - \mathrm{kill}(n)) \cup \mathrm{gen}(n)$$

Conditions initiales

Pour certaines analyses, il est utile d'imposer des *conditions initiales* en certains points.

On se donne donc une fonction initiale de $\mathcal{N}$ dans $\mathcal{P}$, et on associe à chaque point une nouvelle *inéquation :*

$$\mathsf{initiale}(n) \sqsubseteq X_{\mathsf{in}}(n)$$

Pour l'analyse des expressions disponibles, on pose $\mathsf{initiale}(n) = \emptyset$ si n est le noeud d'entrée, $\perp$ sinon.

Vers une inéquation unique

On peut transformer le système d'inéquations de l'analyse en avant [1] en *une seule inéquation* exprimée dans le treillis $(\mathcal{N} \to \mathcal{P})^2$:

$$\begin{pmatrix} n \mapsto \mathsf{initiale}(n) \sqcup \bigsqcup_{n' \to n} X_{\mathsf{out}}(n') \\ n \mapsto \mathsf{transfert}(n)(X_{\mathsf{in}}(n)) \end{pmatrix} \sqsubseteq \begin{pmatrix} X_{\mathsf{in}} \\ X_{\mathsf{out}} \end{pmatrix}$$

Cette inéquation est de la forme

$$F(X) \sqsubseteq X$$

où F est monotone.

1. L'analyse en arrière suit la même forme.

Vers une équation au point fixe

Théorème (Tarski). Soit F une fonction monotone d'un treillis *complet* vers lui-même. Alors l'équation $F(X) = X$ admet une plus petite solution, appelée *plus petit point fixe* de F et notée $\mathrm{lfp}\ F$. De plus, celle-ci coïncide avec la plus petite solution de l'inéquation $F(X) \sqsubseteq X$.

Nous ne détaillons pas ici la notion de treillis complet, mais elle est essentiellement équivalente à notre condition sur la convergence des suites croissantes.

Calcul par approximations successives

Le plus petit point fixe de F peut être calculé par *approximations inférieures successives :*

Théorème. Soit F une fonction monotone d'un treillis complet vers lui-même. Si la suite $(F^n(\perp))_n$ converge, alors,

$$\mathrm{lfp}\ F = \lim_{n\to\infty} F^n(\perp)$$

Point fixe obtenu (analyse avant)

Pour tout noeud n,

$$
\begin{array}{rcl}
X_{\mathsf{in}}(n) & = & \mathsf{initiale}(n) \sqcup \bigsqcup_{n' \to n} X_{\mathsf{out}}(n') \\
X_{\mathsf{out}}(n) & = & \mathsf{transfert}(n)(X_{\mathsf{in}}(n))
\end{array}
$$

Algorithme de recherche de plus petit point fixe

pour tout noeud n

$X_{\mathsf{in}}(n) = \mathsf{initiale}(n); X_{\mathsf{out}}(n) = \perp$

répéter

pour tout noeud n

$X'_{\mathsf{in}}(n) = X_{\mathsf{in}}(n)$;

$X'_{\mathsf{out}}(n) = X_{\mathsf{out}}(n)$;

$X_{\mathsf{out}}(n) = \mathsf{transfert}(X_{\mathsf{in}}(n))$;

$X_{\mathsf{in}}(n) = \mathsf{initiale}(n) \sqcup \bigsqcup_{n' \to n} X_{\mathsf{out}}(n')$

jusqu'à $X'_{\mathsf{in}}(n) = X_{\mathsf{in}}(n)$ et $X'_{\mathsf{out}}(n) = X_{\mathsf{out}}(n)$ pour tout noeud n

Application : analyse de constantes

Nous allons créer une analyse statique qui détermine, en chaque noeud n, les variables qui contiennent toujours la même valeur (constante) quelle que soit le chemin emprunté pour atteindre n.

Le treillis des propriété est de la forme $\mathcal{P} = \{\bot\} \cup (\mathit{Var} \rightarrow (\mathbb{Z} \cup \{\top\}))$

$CT_{\mathsf{in}}(n) = \rho^\sharp \in \mathit{Var} \rightarrow (\mathbb{Z} \cup \{\top\})$ signifie que quelque soit le chemin qui atteind le noeud n, pour toute variable v,

$$\begin{array}{rclll} \rho^\sharp(v) & = & i \in \mathbb{Z} & \text{signifie que} & v \textit{ contient la valeur } i \\ & & & & \quad\text{(ou n'est pas définie)} \\ \rho^\sharp(v) & = & \top & \text{signifie que} & v \textit{ a une valeur inconnue} \end{array}$$

$CT_{\mathsf{in}}(n) = \bot$ signifie que le noeud n n'est pas accessible depuis l'entrée de la fonction.

Exemple

Fichier ExCP1.rtl

```
func Main()
  entry:
    a = 1
    b = 1
    goto test
  test:
    t.0 = Lt(b 10)
    if t.0 goto corps else fin
  corps:
    b = Add (a b)
    a = Sub(2 a)
    c = Sub(a 1)
    d = b
    d = Mul(c d)
    c = Add(d c)
                         // a=1 b=TOP c=0 d=0
    goto test
  fin:
    ret c
```

Ce qu'il reste à faire

1. définir $\sqsubseteq$
2. définir $\perp$
3. définir $\sqcup$
4. définir initiale
5. définir transfert

Exercice : définir $\sqsubseteq$

$$\mathcal{P} = \Big(Var \rightarrow \mathbb{Z}^{\top} \Big)_{\perp}$$

cet ordre doit mimer l'implication logique

Pour tout $\rho_1^\sharp, \rho_2^\sharp \in \mathcal{P}$,
$\rho_1^\sharp \sqsubseteq \rho_2^\sharp$ si et seulement si ...

Exercice : définir $\perp$

$$\mathcal{P} = \Big(Var \to \mathbb{Z}^{\top}\Big)_{\perp}$$

cet opérateur doit être cohérent avec notre choix de $\sqsubseteq$

Exercice : définir $\sqcup$

$$\mathcal{P} = \Big(Var \to \mathbb{Z}^\top \Big)_\perp$$

cet opérateur doit être cohérent avec $\sqsubseteq$

Pour tout $\rho_1^\sharp, \rho_2^\sharp \in \mathcal{P}$,
$\rho_1^\sharp \sqcup \rho_2^\sharp = \ldots$

Exercice : définir initiale

$$\mathcal{P} = \Big(Var \to \mathbb{Z}^{\top}\Big)_{\perp}$$

dans une analyse en avant, on utilise souvent la fonction initiale pour parler de l'état initial du programme

$$Var = \{x_1, \ldots, x_n\}$$

Pour tout noeud n,

$$\begin{array}{lcll} \text{initiale}(n) & = & [x_1 \mapsto \top, \ldots, x_n \mapsto \top] & \text{si } n \text{ entrée de graphe} \\ \text{initiale}(n) & = & \perp & \text{sinon} \end{array}$$

Exercice : définir transfert

$$\mathcal{P} = \left(Var \rightarrow \mathbb{Z}^{\top} \right)_{\perp}$$

dans une analyse en avant, transfert(n) doit transformer une propriétée vraie avant le noeud n en une propriétée vraie juste après son exécution

Pour tout noeud n, $\rho^{\sharp} \in \mathcal{P}$,

$instr(n)$	transfert$(n)(\rho^{\sharp})$
x $=$ i	
x $=$ y	
x $=$ Add(y z)	
x $=$ Sub(y z)	
x $=$ Mul(y z)	
x $=$ [y $+$ o]	
x $=$ **call** f$(\ldots)$	

$\ldots$

TP5

Programmer une analyse de constantes pour RTL

Classe IntOrTop (immutable)

```
// construction de TOP
static public IntOrTop buildTop();

// construction d'une constante entière
static public IntOrTop buildInt(int i);

// teste si this est TOP
public boolean isTop();

// renvoie i si this est un entier i,
// échoue avec une exception si this est TOP
public int getInt();

// renvoie une nouvelle valeur égal au join de this et v
public IntOrTop join(IntOrTop v);

public String toString();

public boolean equals(Object o);
```

Classe `ConstMap` (immutable) 1/2

```
// construction de BOT
static public ConstMap buildBot();

// teste si this est égal à BOT
public boolean isBot();

// construction d'une fonction de domaine dom,
// où chaque ident est associé à TOP
static public ConstMap buildTop(Set<Ident> dom);

// renvoie le join de this est mp
public ConstMap join(ConstMap mp);
```

Classe ConstMap (immutable) 2/2

```
// applique la fonction this sur id,
// échoue si this est égal à BOT
public IntOrTop get(Ident id);

// renvoie une nouvelle fonction égal à la fonction this,
// sauf pour l'ident id qui est associé à v;
// échoue si this est égal à BOT
public ConstMap set(Ident id, IntOrTop v);

public boolean equals(Object o);

public String toString();
```