

# Capacités, invariants et régions abstraites

Romain Bardou

CeProMi — Mars 2009

# Système d'invariant voulu

expressivité à la Spec#

- ▶ état **fermé** / **ouvert**

avec moins d'annotations

- ▶ **inférence** possible au moins partiellement

et des effets cachés

- ▶ **abstraction** du comportement interne

dans un contexte à la Why

# Maintenir les invariants

connaître **statiquement** l'état ouvert / fermé

- ▶ moins d'annotations

empêcher les **fuites** d'objets cachés

- ▶ risque de rompre un invariant

# Solution avec régions et capacités

régions : informations d'**aliasing**

- ▶ groupes de pointeurs

capacités : informations **linéaires** sur les régions

- ▶ **état** ouvert / fermé

# Ingrédients

régions groupes ou **singletons**

- ▶ opérations de **focus** / unfocus
- ▶ capacités “**implicatives**”  $\sigma \multimap \rho$

régions nommées **existentiellement**

- ▶ **création** de régions à la volée

# Problématiques orthogonales

expressivité du suivi des pointeurs ouverts

- ▶ bénéficie des régions singletons
- ▶ bénéficie des régions existentielles

abstraction des effets

⇒ cherchons d'abord sans les régions singletons et existentielles

## Ouverture locale

connaître le pointeur ouvert sans régions singletons

idée : portée de l'ouverture limitée statiquement

```
unpack  $e_1$  in  
   $e_2$   
end
```

problème : la région de  $e_1$  contient d'autres pointeurs

solution : sortir  $e_1$  de sa région temporairement

```
unpack  $e_1$  as  $x[r]$  in  
   $e_2$   
end
```

# Allocation

sans régions existentielles

- ▶ allocation dans une région **existante**
- ▶ allocation dans une région **groupe**

objet alloué **fermé**

- ▶ invariant à **vérifier**
- ▶ objet **initialisé**

syntaxe :

**new**  $C[\rho](e)$



## Syntaxe : classes

```
class  $\mathcal{C}$  =  
  own  $r, \dots, r$   
   $\tau$   
  inv( $x$ ) =  $P$   
end
```

avec :

$$\mathcal{C} ::= \langle r, \dots, r \rangle (\tau, \dots, \tau) \mathcal{C}$$

## Syntaxe : types

$\tau ::=$	unit, int, bool, ...	Type de base
	$\tau \times \dots \times \tau$	Tuple
	$\mathcal{C}[\rho]$	Pointeur
	$\alpha$	Variable de type
	$\alpha[\rho]$	Variable de type pointeur

## Syntaxe : définitions et déclarations

```
val  $f(x: \tau, \dots, x: \tau): \tau$   
  pre  $\{\Sigma, \dots, \Sigma\}, P$   
  post  $\{\Sigma, \dots, \Sigma\}, P$   
  assigns  $\{\rho, \dots, \rho\}$   
   $= e$ 
```

## Syntaxe : expressions

$e ::= v$	Valeur
$(e, \dots, e)$	Tuples
$e.i$	Projection
$x$	Variable
<b>let</b> $x = e$ <b>in</b> $e$	Variable locale
<b>new</b> $\mathcal{C}[\rho](e)$	Allocation
$e := e$	Affectation
<b>!</b> $e$	Déréférencement
<b>unpack</b> $e$ <b>as</b> $x[r]$ <b>in</b> $e$	Ouverture locale
$e; e$	Séquence
$f(e, \dots, e)$	Appel de fonction
<b>if</b> $e$ <b>then</b> $e$ <b>else</b> $e$	Test
<b>region</b> $r$ <b>in</b> $e$	Région locale

## Syntaxe : régions et capacités

$\rho ::= r$     Nom de région, variable de région  
 $\rho.r$     Région locale à une classe

$\Sigma ::= \rho^{\times}$     Fermée  
 $\rho^{\circ}$     Ouverte

## Exemple : calculateur de Morgan

```
class Morgan =  
  own  $\rho_{sum}, \rho_{count}$   
  (Long $[\rho_{sum}],$  Long $[\rho_{count}]$ )  
end
```

## Exemple : moyenne

```
val average(morgan: Morgan[ $\rho$ ]): Long[ $\rho'$ ]  
  pre { $\rho^{\times}$ },  $!(!morgan).2 > 0$   
  post { $\rho^{\times}$ }, !result =  $!(!morgan).1 / !(!morgan).2$   
  = new Long[ $\rho'$ ] ( $!(!morgan).1 / !(!morgan).2$ )
```

## Exemple : insertion

```
val insert(morgan: Morgan[ $\rho$ ], n: Long[ $\rho'$ ]): unit
pre { $\rho^{\times}$ }
post { $\rho^{\times}$ },
    (!(morgan)).1 = old(!(morgan)).1) + !n
     $\wedge$  (!(morgan)).2 = old(!(morgan)).2) + 1
assigns { $\rho.\rho_{sum}$ ,  $\rho.\rho_{count}$ } =
    unpack morgan as morgan[ $\rho''$ ] in
        unpack (!(morgan)).1 as sum[ $\rho_s$ ] in sum := !sum + !n;
        unpack (!(morgan)).2 as count[ $\rho_c$ ] in count := !count + 1
    end
```



# Typage

séparation entre typage et typage des capacités

- ▶ typage similaire à celui de ML
- ▶ capacités vues comme une analyse supplémentaire modulaire

## Typage : déréréfencement

$$\frac{\Gamma \vdash e : \mathcal{C}[\rho] \quad \mathcal{C} : \tau_1}{\Gamma \vdash !e : \tau_1} \text{Deref}$$

## Typage : allocation

$$\frac{\mathcal{C} : \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{new} \ \mathcal{C}[\rho](e) : \mathcal{C}[\rho]} \text{ Alloc}$$

## Typage : ouverture locale

$$\frac{\Gamma \vdash e_1 : \mathcal{C}[\rho] \quad \Gamma, x : \mathcal{C}[r] \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{unpack} \ e_1 \ \mathbf{as} \ x[r] \ \mathbf{in} \ e_2 : \tau_2} \text{Unpack}$$

## Typage des capacités

$$\frac{}{\{\bar{\Sigma}\} \times \{\bar{\Sigma}\}} \text{CVar} \qquad \frac{\{\bar{\Sigma}_1\} e \{\bar{\Sigma}_2, \Sigma\}}{\{\bar{\Sigma}_1\} e \{\bar{\Sigma}_2\}} \text{CWeaken}$$

$$\frac{\{\bar{\Sigma}_1\} e_1 \{\bar{\Sigma}_2\} \quad \{\bar{\Sigma}_2\} e_2 \{\bar{\Sigma}_3\}}{\{\bar{\Sigma}_1\} e_1; e_2 \{\bar{\Sigma}_3\}} \text{CSeq}$$

$$\frac{\{\bar{\Sigma}_1\} e_1 \{\bar{\Sigma}_2\} \quad \{\bar{\Sigma}_2\} e_2 \{\bar{\Sigma}_3\} \quad \{\bar{\Sigma}_2\} e_3 \{\bar{\Sigma}_3\}}{\{\bar{\Sigma}_1\} \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \{\bar{\Sigma}_3\}} \text{CIf}$$

## Capacités : affectation, déréréférencement

$$\frac{e_1 : \rho \quad \{\bar{\Sigma}_1\} e_1 \{\bar{\Sigma}_2\} \quad \{\bar{\Sigma}_2\} e_2 \{\bar{\Sigma}_3, \rho^\circ\}}{\{\bar{\Sigma}_1\} e_1 := e_2 \{\bar{\Sigma}_3, \rho^\circ\}} \text{CAffect}$$

$$\frac{\{\bar{\Sigma}_1\} e \{\bar{\Sigma}_2\}}{\{\bar{\Sigma}_1\} !e \{\bar{\Sigma}_2\}} \text{CDeref}$$

## Capacités : régions locales

$$\frac{\{\bar{\Sigma}_1, r^\times\} \in \{\bar{\Sigma}_2\}}{\{\bar{\Sigma}_1\} \text{ region } r \text{ in } \{\bar{\Sigma}_2 - r\}} \text{CRegion}$$

## Capacités : allocation

$$\frac{\{\bar{\Sigma}_1, \text{own}(C[\rho])\} e \{\bar{\Sigma}_2, \text{own}(C[\rho])\}}{\{\bar{\Sigma}_1\} \text{ new } C[\rho](e) \{\bar{\Sigma}_2\}} \text{ CAlloc}$$

Rappel :

```
class  $\mathcal{C}$  =  
  own  $r, \dots, r$   
   $\tau$   
  inv( $x$ ) =  $P$   
end
```



## Capacités : ouverture locale

$$\frac{e_1 : C[\rho] \quad \{\bar{\Sigma}_1\} e_1 \{\bar{\Sigma}_2, \rho^\times\} \quad \{\bar{\Sigma}_2, r^\circ, \mathbf{own}(C[r])\} e_2 \{\bar{\Sigma}_3, r^\circ, \mathbf{own}(C[r])\}}{\{\bar{\Sigma}_1\} \mathbf{unpack} e_1 \text{ as } x[r] \text{ in } e_2 \{\bar{\Sigma}_3, \rho^\times\}} \text{CUnpack}$$

Rappel :

$$\frac{\Gamma \vdash e_1 : C[\rho] \quad \Gamma, x : C[r] \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{unpack} e_1 \text{ as } x[r] \text{ in } e_2 : \tau_2} \text{Unpack}$$

## Appel de fonction : exemple

```
val f(x: Long[ $\rho_1$ ], y: Long[ $\rho_2$ ]): unit
  pre { $\rho_1^x, \rho_2^x$ }
  post { $\rho_1^x, \rho_2^x$ },
    !x = old(!x) + 1
    ^ !y = old(!y) + 1
  assigns { $\rho_1, \rho_2$ } =
    unpack x as x[r] in x := !x + 1;
    unpack y as y[r] in y := !y + 1
```

si  $\rho_1$  et  $\rho_2$  unifiés

- ▶ besoin de deux fois la même capacité
- ▶ appel de  $f$  impossible

## Appel de fonction : typage

$$\frac{f(\tau_1, \dots, \tau_n) : \tau \quad \Gamma \vdash e_1 : \tau_1 \sigma \quad \dots \quad \Gamma \vdash e_n : \tau_n \sigma}{\Gamma \vdash f(e_1, \dots, e_n) : \tau \sigma} \text{Call}$$

où  $\sigma$  est la substitution permettant d'instancier  $f$

- ▶ substitution des variables de type polymorphes
- ▶ substitution des variables de région

## Appel de fonction : capacités

$$\frac{\{\bar{\Sigma}_1\} e_1 \{\bar{\Sigma}_2\} \quad \dots \quad \{\bar{\Sigma}_n\} e_n \{\bar{\Sigma}\}}{\{\bar{\Sigma}_1\} f(e_1, \dots, e_n) \{\bar{\Sigma}'\}} \text{CCall}$$

$\bar{\Sigma} = \bar{\Sigma}'' \uplus \mathbf{pre}(f) \sigma \quad \bar{\Sigma}' = \bar{\Sigma}'' \uplus \mathbf{post}(f) \sigma$

où  $\uplus$  est l'union des multi-ensembles

# Sémantique

sémantique à **petits pas**

**tas**  $H$  : adresses  $\rightarrow_{\rho}$  valeurs

affectation et déréférencement :

$$\begin{aligned} H[p \mapsto v], p := v &\longrightarrow H[p \mapsto v'], () \\ H[p \mapsto v], !p &\longrightarrow H[p \mapsto v], v \end{aligned}$$

## Sémantique : allocation

si  $p$  est une adresse fraîche :

$$H, \mathbf{new} \ C[\rho](v) \longrightarrow H[p \mapsto_{\rho} v], p$$

## Sémantique : ouverture

$$H, \text{unpack } p \text{ as } x[r] \text{ in } v \longrightarrow H, v$$

règle particulière de **passage au contexte** :

$$\frac{H[p \mapsto_{\rho} v][p \mapsto_r v], e[p/x] \longrightarrow H'[p \mapsto_{\rho} v'][p \mapsto_r v'], e'[p/x]}{H[p \mapsto_{\rho} v], \text{unpack } p \text{ as } x[r] \text{ in } e \longrightarrow H'[p \mapsto_{\rho} v'], e'}$$

**problème** : affectation simultanée dans deux régions

- ▶ changer la sémantique de l'affectation ?
- ▶ imposer des capacités pour la lecture ?

## Sémantique : appel de fonction

### Définition (satisfaction du contrat d'une fonction)

$$H, H' \models f(v_1, \dots, v_n) \longrightarrow v$$

si, et seulement si :

$$\left\{ \begin{array}{l} H \models_{\mathbf{pre}} f(v_1, \dots, v_n) \\ H, H' \models_{\mathbf{post}} f(v_1, \dots, v_n) \longrightarrow v \\ H, H' \models_{\mathbf{assigns}} f \end{array} \right.$$

si  $H, H' \models f(v_1, \dots, v_n) \longrightarrow v$  :

$$H, f(v_1, \dots, v_n) \longrightarrow H', v$$



## Correction

**Théorème (Correction)** Si :

$$\left\{ \begin{array}{l} \Gamma \vdash e: \tau \\ \{\bar{\Sigma}_1\} e \{\bar{\Sigma}_2\} \\ coh(H, \bar{\Sigma}_1) \\ H, e \longrightarrow H', e' \end{array} \right.$$

alors il existe  $\bar{\Sigma}'_1$  et  $\bar{\Sigma}'_2$  tels que :

$$\left\{ \begin{array}{l} \Gamma \vdash e': \tau \\ \{\bar{\Sigma}'_1\} e' \{\bar{\Sigma}'_2\} \\ coh(H', \bar{\Sigma}'_1) \end{array} \right.$$

## Correction des définitions de fonction

**Corollaire (Correction d'une fonction)** Soit une fonction  $f$  définie par :

$$\mathbf{val} f(x_1: \tau_1, \dots, x_n: \tau_n): \tau \dots = e$$

et soit un tas  $H$  et des valeurs  $v_1, \dots, v_n$  de types respectifs  $\tau_1, \dots, \tau_n$ , tels que  $H \models_{\mathbf{pre}} f(v_1, \dots, v_n)$ . Pour toute valeur  $r$  de type  $\tau$  telle que  $e[v_1/x_1, \dots, v_n/x_n] \longrightarrow^* r$ , on a :

$$H, H' \models f(v_1, \dots, v_n) \longrightarrow r$$

# Abstraction

cacher des effets<sup>1</sup>

modules à la Caml

abstraction du contrat d'une fonction

- ▶ types
- ▶ pré-condition et ses capacités
- ▶ post-condition et ses capacités
- ▶ clause assigns

---

<sup>1</sup>uniquement les effets personnels, pas les feutres pour tableau blanc

## Abstraction : exemple (version concrète)

```
class Morgan =  
  own  $\rho_{sum}, \rho_{count}$   
  (Long[ $\rho_{sum}$ ], Long[ $\rho_{count}$ ], set)  
  inv(x) =  
    !x.1 = setsum(x.3)  
     $\wedge$  !x.2 = setcount(x.3)  
end  
  
val average(morgan: Morgan[ $\rho$ ]): Long[ $\rho'$ ]  
  pre { $\rho^x$ }, !(!morgan).2 > 0  
  post { $\rho^x$ }, !result = !(!morgan).1 / !(!morgan).2  
  = new Long[ $\rho'$ ] (!(!morgan).1 / !(!morgan).2)
```

## Abstraction : exemple (version abstraite)

```
class Morgan =
```

```
  set
```

```
end
```

```
val average(morgan: Morgan[ $\rho$ ]): Long[ $\rho'$ ]
```

```
  pre { $\rho^x$ }, setcount(!morgan) > 0
```

```
  post { $\rho^x$ }, !result = setsum(!morgan) / setcount(!morgan)
```

## Abstraction : exemple (insertion)

```
val insert(morgan: Morgan[ $\rho$ ], n: Long[ $\rho'$ ]): unit
  pre { $\rho^{\times}$ }
  post { $\rho^{\times}$ },
     $!(!morgan).1 = \mathbf{old}(!(!morgan).1) + !n$ 
     $\wedge !(!morgan).2 = \mathbf{old}(!(!morgan).2) + 1$ 
  assigns { $\rho.\rho_{sum}, \rho.\rho_{count}$ } =  $\dots$ 
```

```
val insert(morgan: Morgan[ $\rho$ ], n: Long[ $\rho'$ ]): unit
  pre { $\rho^{\times}$ }
  post { $\rho^{\times}$ },  $!morgan = \mathbf{setadd}(!morgan, !n)$ 
  assigns  $\emptyset = \dots$ 
```

## Abstraction : obligation de preuve

$$\begin{aligned} \forall m_a, \mathbf{pre}_a(m_a) &\Rightarrow \\ \exists m, \mathbf{pre}(m) \wedge & \\ \exists m', \mathbf{post}(m, m') &\Rightarrow \\ \exists m_a', \mathbf{post}_a(m_a, m_a') & \end{aligned}$$

obligation de preuve

- ▶ modulo les régions
- ▶ **pre** et **post** sous-entendent aussi les invariants
- ▶ avec parfois  $m = m'$  et/ou  $m_a = m_a'$

## Abstraction : correction

instance (?) de la règle de conséquence :

$$\frac{\{P \Rightarrow P', P'\} e \{Q', Q' \Rightarrow Q\}}{\{P\} e \{Q\}}$$



# Inférence

inférables :

- ▶ ouvertures locales (**unpack** ... **in**)
- ▶ capacités **ouvertes**

non-inférables :

- ▶ clauses **assigns**
- ▶ égalités des régions
- ▶ capacités **fermées**

discutable :

- ▶ **portée** des ouvertures locales

# Conclusion

syntaxe et typage d'un système d'invariants à base de capacités  
avec sa sémantique et des possibilités d'abstractions

adaptable pour C, Java, ...

lien avec Spec# assez clair

faire les **preuves**

mieux **formaliser** l'inférence

étendre avec régions **singletons** et **existentielles**

**implémenter** (dans Jessie ?)

Essayez **Melt** pour écrire vos articles et présentations avec :

- ▶ la puissance de mise en forme de **L<sup>A</sup>T<sub>E</sub>X** ;
- ▶ l'expressivité de **Caml** !

`https://forge.ocamlcore.org/projects/melt/`