Traduction de programmes impératifs dans un language fonctionnel à l'aide d'un système de types à base de capacités et de régions

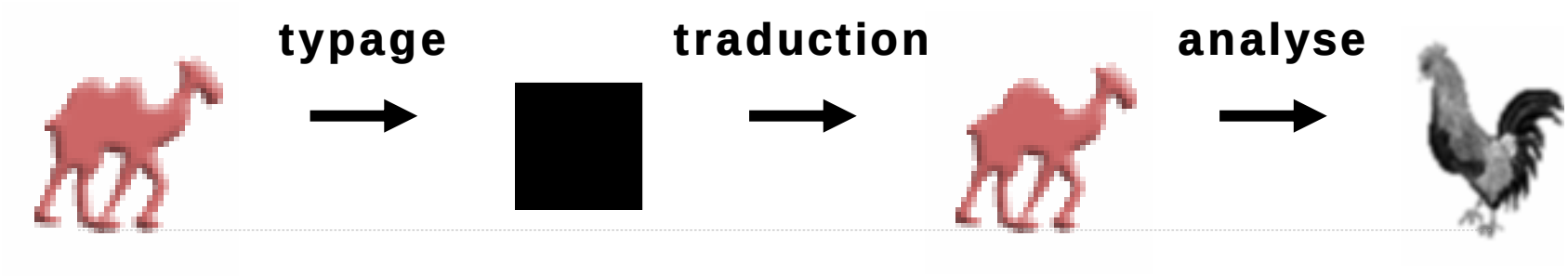
Arthur Charguéraud

avec François Pottier

Projet Gallium – INRIA-Rocquencourt

INRIA-Rocquencourt 14 Septembre 2007

Shéma général



CAML impératif | Typage capacités | CAML fonctionnel | Formules Coq

– Un système de types

Système ML + capacités statiques traitées linéairement

Capacité = droit exclusif d'accès et de modification

– Une traduction dirigée par le typage

Traduction vers programme fonctionnel équivalent

Mémoire représentée via la traduction des capacités

Plan

Vous êtes ici 

I Structurer la mémoire

II Typage et traduction

III Restructurer la mémoire

IV Empaquetage de régions

V Arguments de preuve

VI Travaux reliés et futurs

Ex : Références – typage

Source impératif :

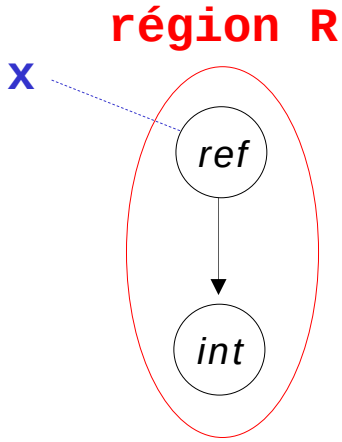
```
let x = ref 7
let y = get x
let _ = set (x,'c')
```

Typage des valeurs :

```
7 : int      x  : [R]
y : int    'c' : char
_ : unit
```

Typage des capacités :

```
let {R:ref int} x = ref {} 7
let {R:ref int} y = get {R:ref int} x
let {R:ref char} _ = set {R:ref int} (x,'c')
```



Typage des primitives :

$$
\begin{array}{lcl}
\text{ref} & : & \forall\,\alpha, \{\}\alpha \rightarrow \exists\,\rho.\{\rho : \text{ref}\ \alpha\}[\rho] \\
\text{get} & : & \forall\,\alpha\,\rho,\ \{\rho : \text{ref}\ \alpha\}[\rho] \rightarrow \{\rho : \text{ref}\ \alpha\}\alpha \\
\text{set} & : & \forall\,\alpha_1\,\alpha_2\,\rho,\ \{\rho : \text{ref}\ \alpha_1\}([\rho] \times \alpha_2) \rightarrow \{\rho : \text{ref}\ \alpha_2\}\text{unit}
\end{array}
$$

Ex : Références – traduction

Typage avec capacités :

```
let {R:ref int} x = ref {} 7
let {R:ref int} y = get {R:ref int} x
let {R:ref char} _ = set {R:ref int} (x,'c')
```

Traduction fonctionnelle :

```
let R1,x = (λ((),a).(a,()) ((),7)
let R2,y = (λ(a,()).(a,a)) (R1,x)
let R3,_ = (λ(a1,((),a2)).(a2,())) (R2,(x,'c'))
```

$$\text{set} \ : \ \forall\, \alpha_1\, \alpha_2\, \rho,\ \{\rho : \text{ref}\ \alpha_1\}([\rho] \times \alpha_2) \rightarrow \{\rho : \text{ref}\ \alpha_2\}\text{unit}$$

Même code une fois nettoyé :

```
let R1 = 7
let R2,y = R1,R1
let R3 = 'c'
```

x : [R]

Rappel du source impératif :

```
let x = ref 7
let y = get x
let _ = set (x,'c')
```

Ex : Matrices comme tableaux 2D

Type ML :

```
x : array (array int)
```

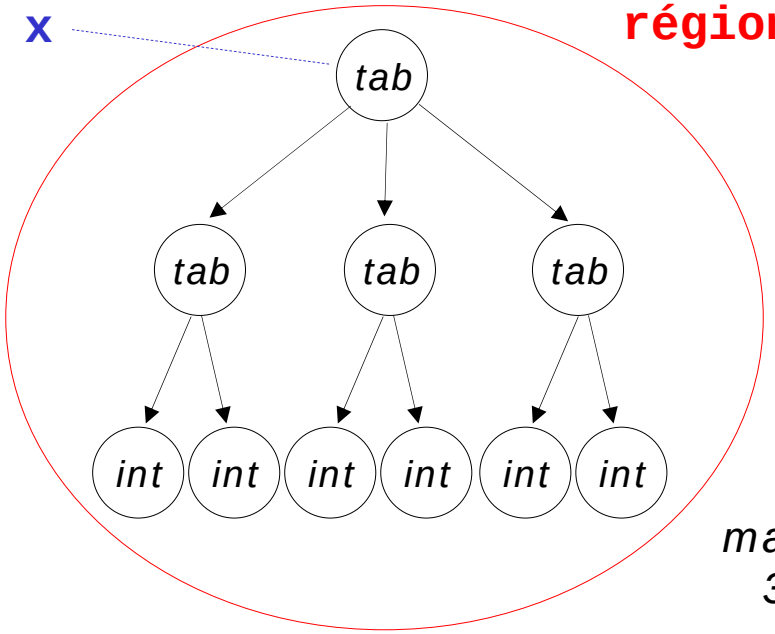
Type donné :

```
x : [R]
{R : array (array int)}
```

Traduction fonctionnelle :

```
R : arrayF (arrayF int)
```

Graphe mémoire correspondant :



matrice 3x2

Accès à la case (i,j) :

```
R : arrayF (arrayF int)
R[i] : arrayF int
R[i][j] : int
```

Avec traduction monadique naïve :

```
x : key
M[x] : arrayF key
M[M[x][i]] : arrayF int
M[M[x][i]][j] : int
```

Ex : Matrices à lignes aliasées

Type ML :

```
x : array (array int)
```

Type donné :

```
x : [R]
{R : array [ρ]}
{ρ*: array int}
```

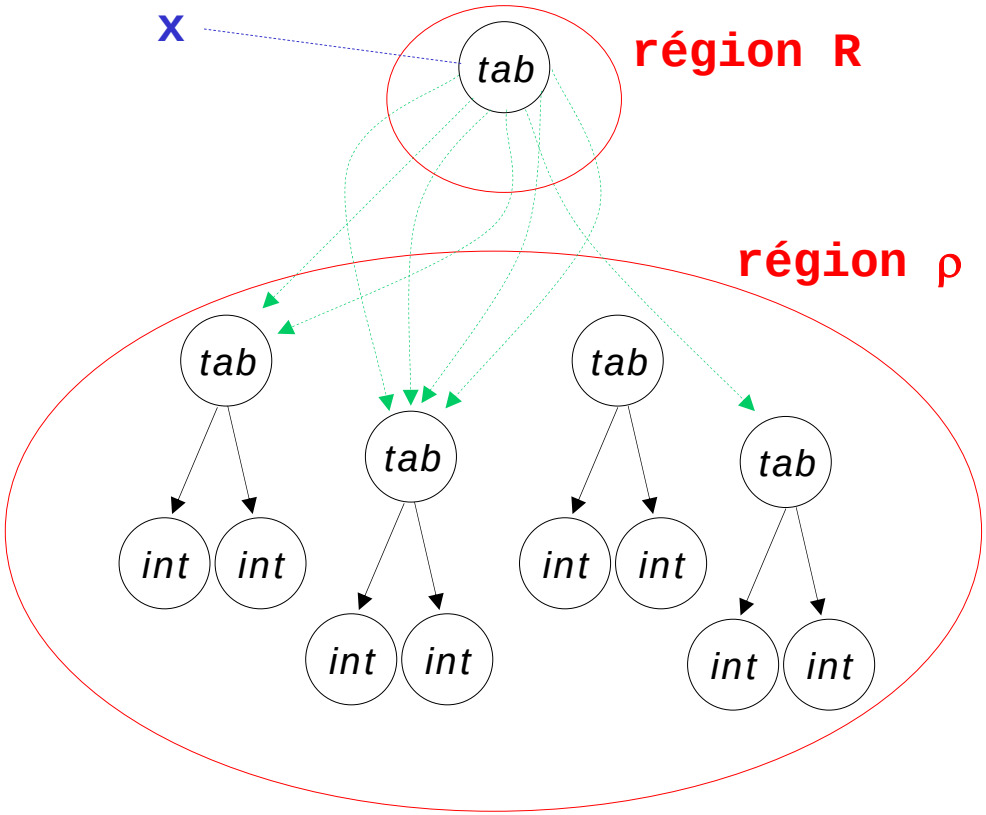
Traduction fonctionnelle :

```
R : arrayF key
ρ : map key (arrayF int)
```

Accès à la case (i,j) :

```
R[i] : key
ρ[M[i]] : arrayF int
ρ[M[i]][j] : int
```

Graphe mémoire correspondant :



Ex : Arbre de syntaxe abstraite

Type ML :

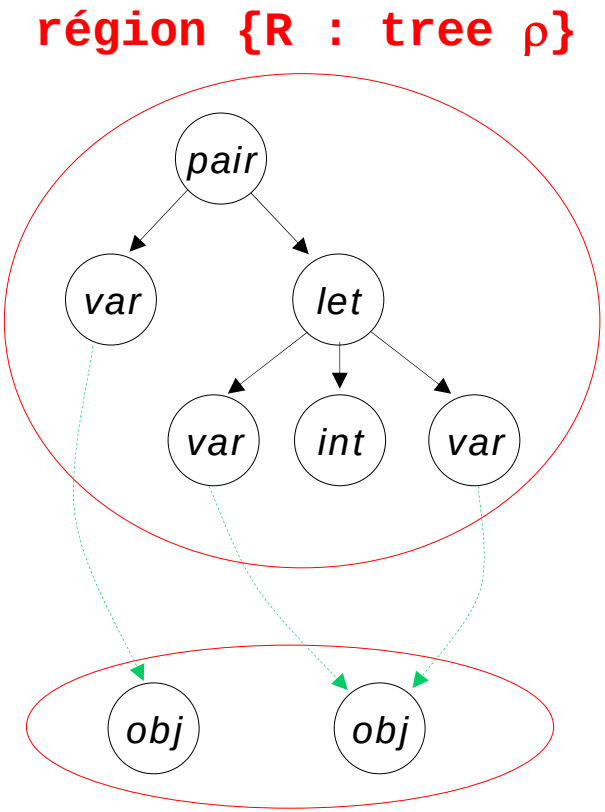
```
type tree =
  | Var of obj
  | Pair of tree * tree
  | Let of obj * tree * tree
```

Type donné :

```
type tree ρ =
  | Var of [ρ]
  | Pair of tree * tree
  | Let of [ρ] * tree * tree
```

Traduction fonctionnelle :

```
type tree =
  | Var of key
  | Pair of tree * tree
  | Let of key * tree * tree
```



Petit résumé

- Graphe mémoire partitionné en régions
- Régions contiennent des arbres typés
  - Région singleton (1 arbre) $\{\rho:\theta\}$
  - Région d'aliasing ($n \geq 0$ arbres) $\{\rho^*:\theta\}$
- Les objets d'une région ρ ont le type $[\rho]$

---

région singleton $\{\rho:\theta\}$	$\rightarrow$	traduction du contenu
région d'aliasing $\{\rho^*:\theta\}$	$\rightarrow$	map clés vers traductions
valeur de type $[\rho]$	$\rightarrow$	clé dans une map, ou unit
objet de type ref θ	$\rightarrow$	traduction du contenu
objet de type array θ	$\rightarrow$	array^F des traductions

Capacités et types

Capacités :

vide *paire* *singleton* *aliasing*

$$C \quad := \quad \{\} \mid C_1 \wedge C_2 \mid \{\rho : \theta\} \mid \{\rho^\star : \theta\}$$

Types des valeurs :

"at-rho"

$$\tau \quad := \quad \text{int} \mid \text{unit} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \sigma_1 \to \sigma_2 \mid [\rho]$$

Types des termes :

valeur *région* *capacité*

$$\sigma \quad := \quad \tau \mid \exists \rho.\sigma \mid C \wedge \sigma$$

Types des régions :

références

$$\theta \quad := \quad \text{int} \mid \text{unit} \mid \theta_1 \times \theta_2 \mid \theta_1 + \theta_2 \mid \sigma_1 \to \sigma_2 \mid [\rho] \mid \text{ref}\ \theta$$

Jugements de typage

Typage des valeurs :

$$\Gamma \vdash v : \tau$$

$x : \tau$

Typage des termes :

$$\Gamma ; C \vdash t : \sigma$$

capacités consommées

Rappel :

$$\sigma \ := \ \tau \mid \exists \rho . \sigma \mid C \wedge \sigma$$

Après normalisation :

capacités entrée *capacités sortie*

$$\Gamma ; C \vdash t : (\exists \overline{\rho} . C' \wedge \tau)$$

Résultat de l'évaluation du terme :

$$\Gamma \vdash v : \tau$$

Règles de typage

Typage des valeurs $\Gamma \vdash v : \tau$ et des termes $\Gamma ; C \vdash t : \sigma$

Valeur :

$$\frac{\Gamma \vdash v : \tau}{\Gamma ; \{\} \vdash v : \tau}$$

Abstraction :

$$\frac{(\Gamma, x : \tau) ; C \vdash t : \sigma \qquad \overline{\rho} \# \sigma, \Gamma}{\Gamma \vdash (\lambda x. t) : ((\exists \overline{\rho}. C \wedge \tau) \rightarrow \sigma)}$$

Contre-exemple :

```
{R : ref int}[R] -> (unit -> {R : ref int}[R])
```

Application :

$$\frac{\Gamma \vdash v : (\sigma_1 \rightarrow \sigma_2) \qquad \Gamma ; C \vdash t : \sigma_1}{\Gamma ; C \vdash (v\ t) : \sigma_2}$$

Règle frame

Règle frame :

$$\frac{\Gamma\,;\, C_2 \vdash t : \sigma}{\Gamma\,;\, (C_1 \wedge C_2) \vdash t : (C_1 \wedge \sigma)}$$

Frame avec let (règle dérivable) :

$$\frac{\Gamma\,;\, C_1 \vdash t_1 : (\exists\, \overline{\rho}.\, C_2 \wedge \tau) \qquad (\Gamma;\, x : \tau)\,;\, (C_2 \wedge C_3) \vdash t_2 : \sigma}{\Gamma\,;\, (C_1 \wedge C_3) \vdash (\text{let } x = t_1 \text{ in } t_2) : \sigma}$$

Exemple :

```
let x = ref 5 in
let y = ref 3 in
let a = get x in
let b = get y in
()
```

Capacités sur x et y disponibles, mais seule celle sur x est utile.

Définition de la traduction

Traduction des valeurs : $\Gamma \vdash v : \tau \triangleright u$

Traduction des termes : $\Gamma; C \triangleright w \vdash t : \sigma \triangleright u$

w traduit C — u traduit t:σ

Abstraction :

$$\frac{(\Gamma, x : \tau); C \triangleright y \vdash t : \sigma \triangleright u}{\Gamma \vdash (\lambda x.\, t) : ((\exists \overline{\rho}.\, C \wedge \tau) \to \sigma) \triangleright (\lambda(y, x).\, u)}$$

y est la traduction de la capacité *C*

Frame :

$$\frac{\Gamma; C_2 \triangleright y_2 \vdash t : \sigma \triangleright u}{\Gamma; (C_1 \wedge C_2) \triangleright w \vdash t : (C_1 \wedge \sigma) \triangleright (\text{let } (y_1, y_2) = w \text{ in } (y_1, u))}$$

Plan

Vous êtes là maintenant

I Structurer la mémoire

II Typage et traduction

III Restructurer la mémoire

IV Empaquetage de régions

V Arguments de preuve

VI Travaux reliés et futurs

Ex : Union-find

Type ML :

```
n : node
node = ref (1 + node)
```

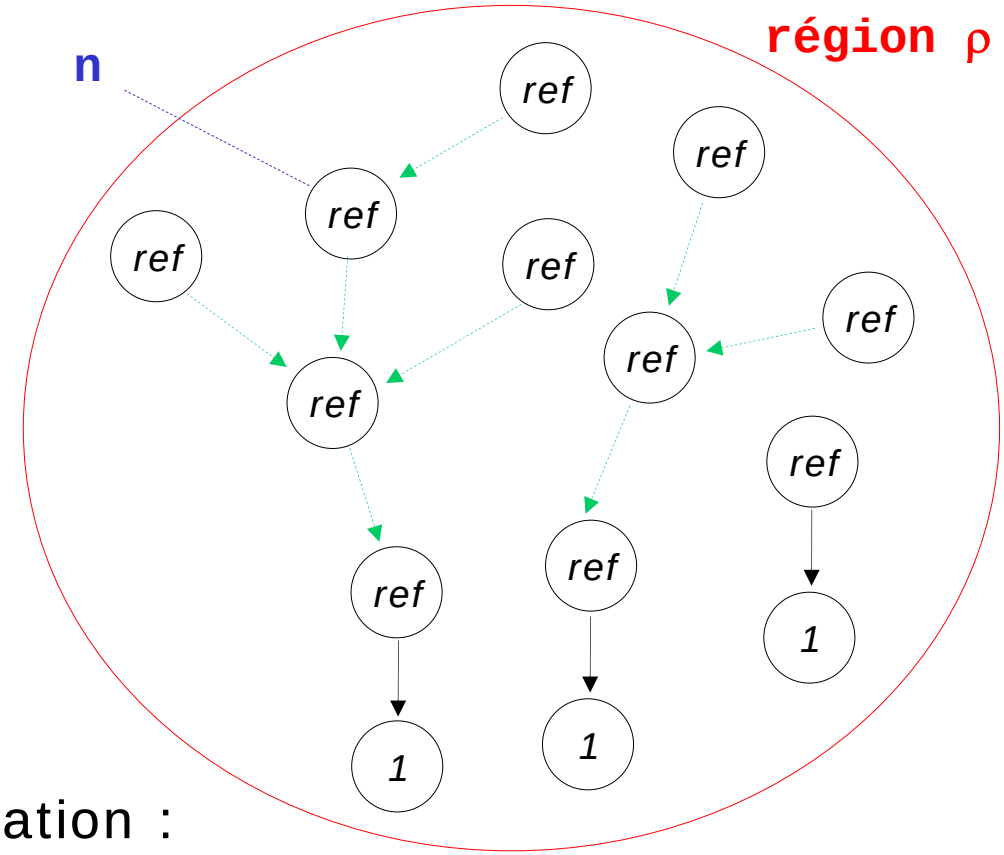
Type donné :

```
n : [ρ]
{ρ*: ref (1 + [ρ])}
```

Traduction fonctionnelle :

```
n : key
ρ : map key (1 + key)
```

Graphe mémoire correspondant :

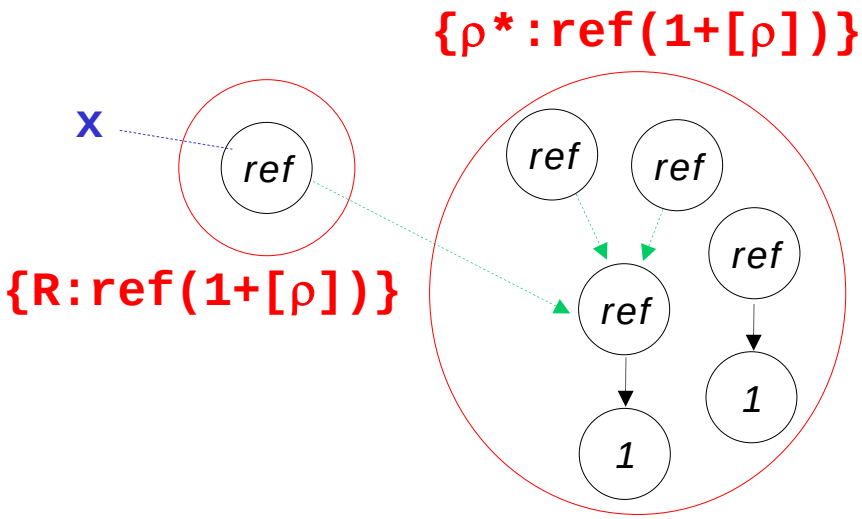


Types de la fonction d'unification :

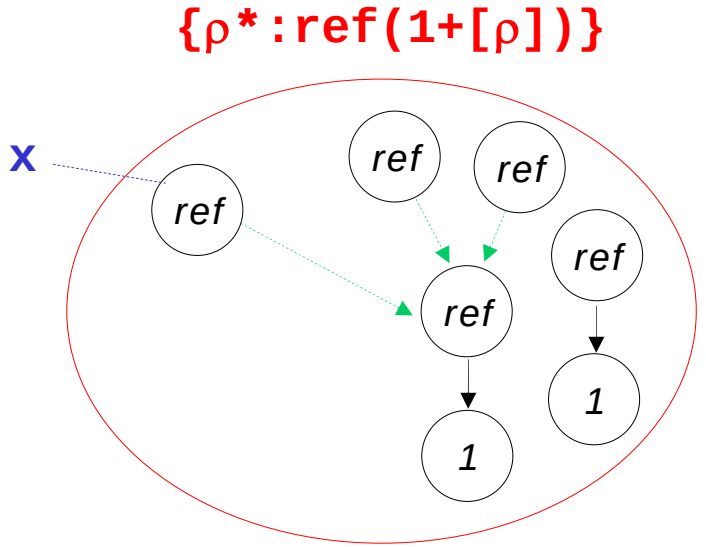
```
en ML :          node × node  ->          unit
ici   :  {ρ*:_}. [ρ] × [ρ]    ->  {ρ*:_}.unit
```

Adoption

Avant : **x : [R]**



Après : **x : [ρ]**



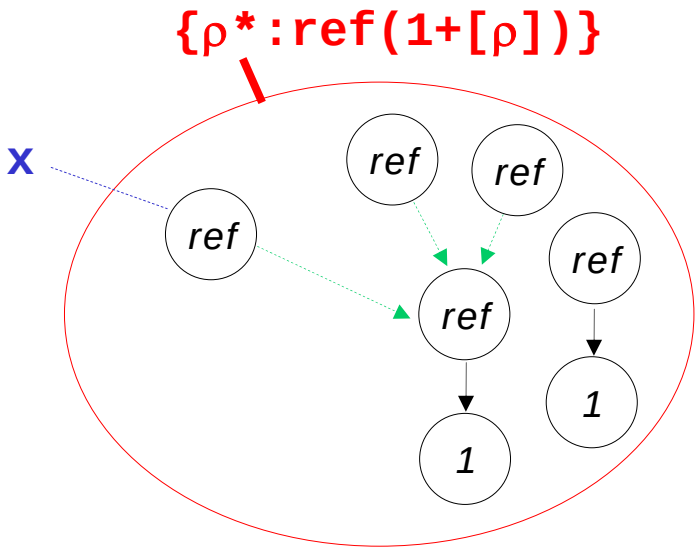
Formellement : $\{\rho_1{}^\star : \theta\}\{\rho_2 : \theta\}[\rho_2] \leq \{\rho_1{}^\star : \theta\}[\rho_1]$

Traduction : $\triangleright \quad \lambda(h, (x, ())).\,(\mathrm{map\_add}\, h\, x)$

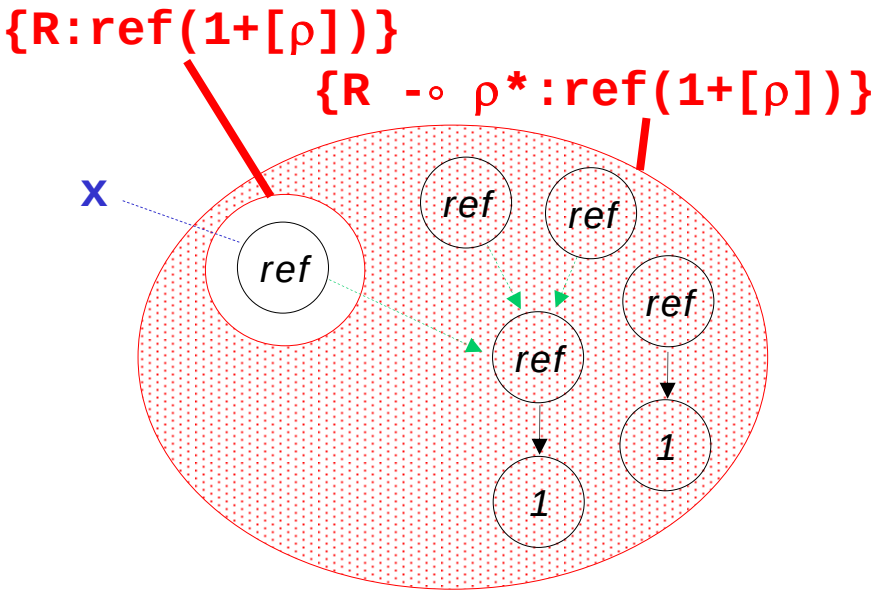
avec map_add : (map key obj) → obj → (map key obj) * key

Focus & Unfocus

Avant : `x : [ρ]`



Après : `x : [R]`



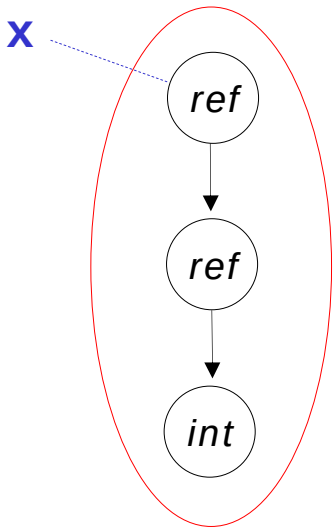
$$\text{FOCUS} \quad : \quad \{\rho_1{}^\star : \theta\}[\rho_1] \leq \exists \rho_2 . \{\rho_2 \multimap (\rho_1{}^\star : \theta)\}\{\rho_2 : \theta\}[\rho_2]$$

$$\text{UNFOCUS} \quad : \quad \{\rho_2 \multimap (\rho_1{}^\star : \theta)\}\{\rho_2 : \theta\} \leq \{\rho_1{}^\star : \theta\}$$

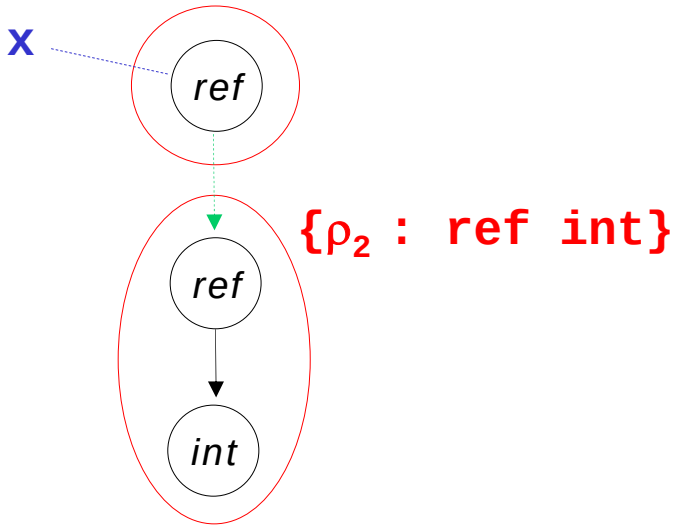
Focus se traduit par map_get et unfocus par map_set.

Open & Close

Avant : **{ρ_1 : ref ref int}**



Après : **{ρ_1 : ref [ρ_2]}**



Rappel :

$$\text{get} \quad : \quad \forall\, \alpha\, \rho,\ \{\rho : \text{ref}\ \alpha\}[\rho] \rightarrow \{\rho : \text{ref}\ \alpha\}\alpha$$

$$\text{OPEN-REF} \ : \ \{\rho_1 : \text{ref}\ \theta\} \ \equiv \ \exists \rho_2 . \{\rho_1 : \text{ref}\ [\rho_2]\} \wedge \{\rho_2 : \theta\}$$

Règles pour le sous-typage

Règles de sous-typage, à gauche et à droite :

$$\frac{\Gamma\,;\, C_2 \vdash t : \sigma \qquad C_1 \leq C_2}{\Gamma\,;\, C_1 \vdash t : \sigma} \qquad \frac{\Gamma\,;\, C \vdash t : \sigma_1 \qquad \sigma_1 \leq \sigma_2}{\Gamma\,;\, C \vdash t : \sigma_2}$$

Traductions associées :

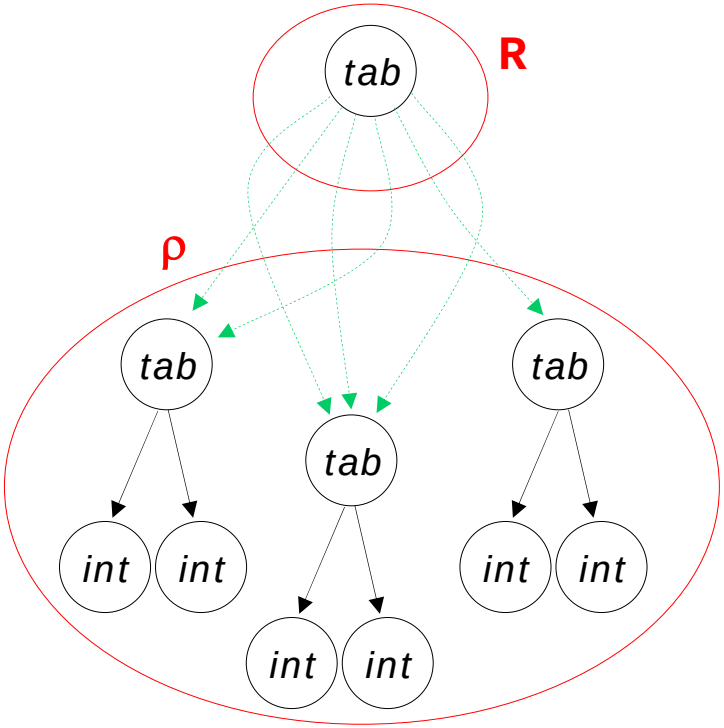
$$\frac{\Gamma;\, C_2 \rhd (f\ w) \vdash t : \sigma \rhd u \qquad C_1 \leq_f C_2}{\Gamma;\, C_1 \rhd w \vdash t : \sigma \rhd u}$$

$$\frac{\Gamma;\, C \rhd w \vdash t : \sigma_1 \rhd u \qquad \sigma_1 \leq_f \sigma_2}{\Gamma;\, C \rhd w \vdash t : \sigma_2 \rhd (f\ u)}$$

Ex : Matrice avec partage – suite

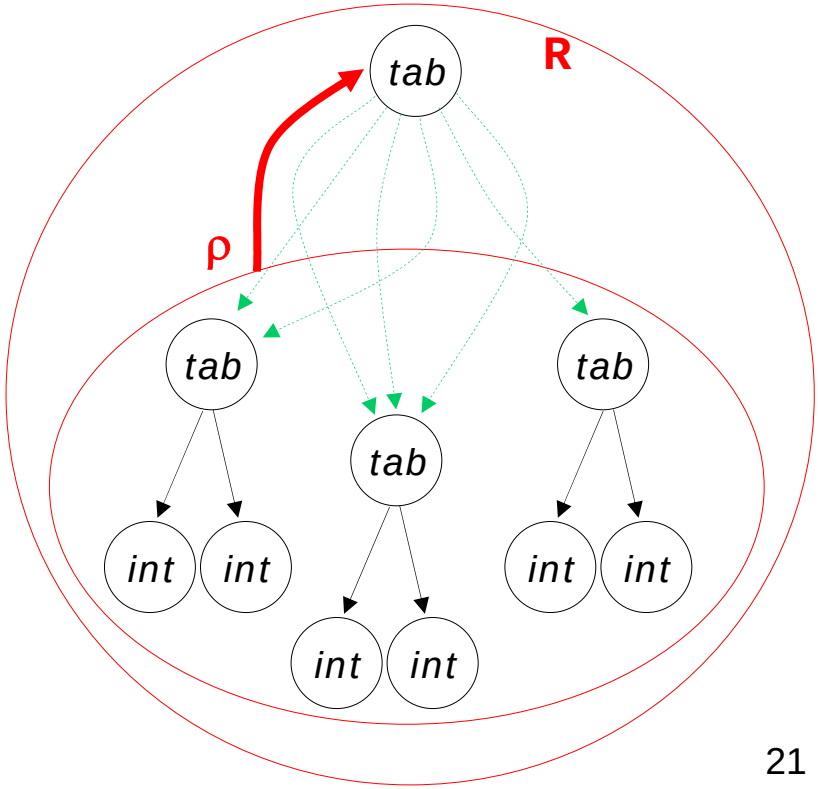
Typage précédent :

```
x : [R]
{R : array [ρ]}
{ρ*: array int}
```



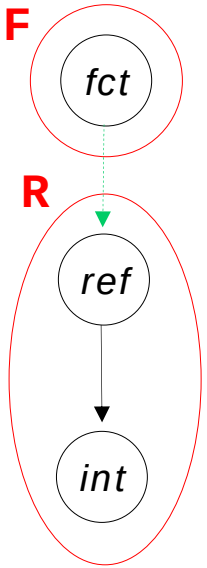
Avec région empaquetée :

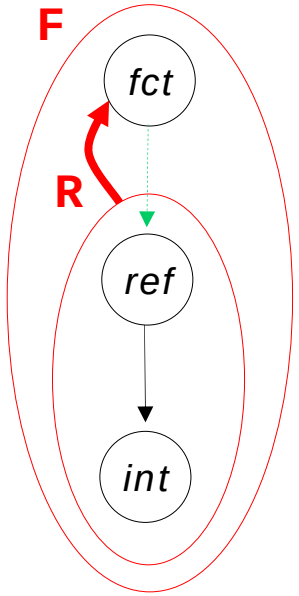
```
{R: ∃ρ.{ρ*:array int}.array[ρ]}
```



Ex : Clôture avec état interne

```
let new_gen () =
    let x = ref 0
    let next () =
        let _ = incr x
        let y = get x
        y in
    next
```





$$\text{FUNC} : \quad \tau \equiv \exists\rho.\{\rho : \tau\}[\rho]$$

Typage : à vous de jouer !

```
x : [R]   avec    {R:ref int}
next : {R:ref int}.unit -> {R:ref int}.int
```

```
new_gen : unit -> ∃R.{R:ref int}.(type of next)
new_gen : unit -> ∃F.{F: ∃R.{R:ref int}.(type of next)}.[F]
```

Extensions pour l'empaquetage

Types des régions et capacités encapsulées :

$$\theta \quad := \quad \ldots \quad | \quad \exists \rho.\theta \quad | \quad C \wedge \theta$$

Règles de conversion pack et unpack :

$$\begin{array}{lcl} \{\rho : C \wedge \theta\} & \equiv & C \wedge \{\rho : \theta\} \\ \{\rho_1 : \exists \rho_2.\theta\} & \equiv & \exists \rho_2.\{\rho_1 : \theta\} \end{array}$$

Traduction de $\{\rho : C \wedge \theta\}$ par une paire, exactement comme $C \wedge \{\rho : \theta\}$.

Plan

I Structurer la mémoire

II Typage et traduction

III Restructurer la mémoire

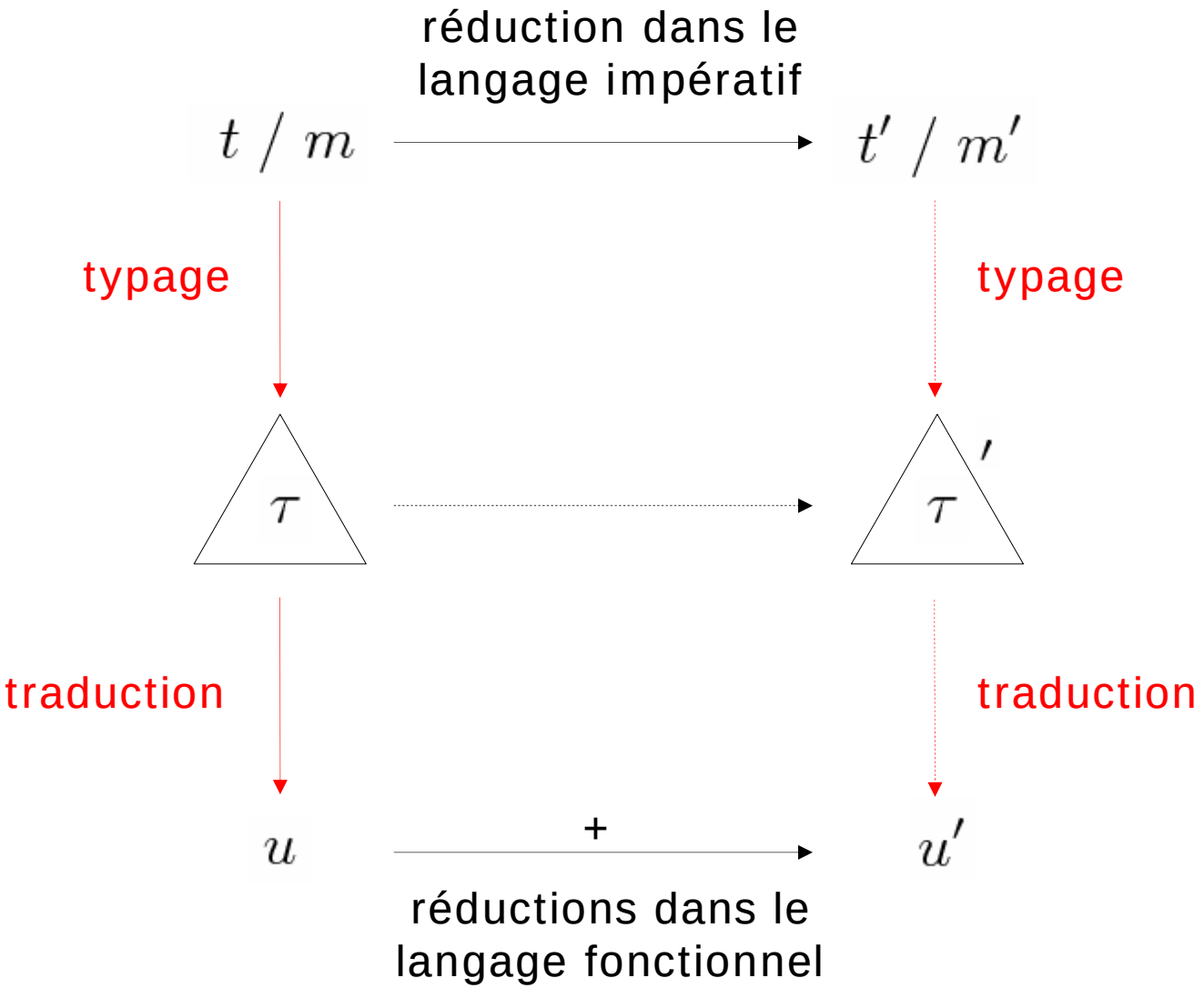
IV Empaquetage de régions

Nous sommes ici maintenant

V Arguments de preuve

VI Travaux reliés et futurs

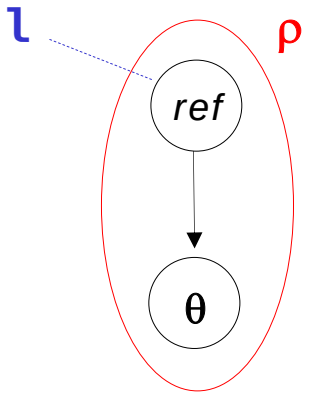
Preuve de correction



Extension du typage aux états

Règle "loc" pour ML+reference :

$$\frac{\Phi[l] = \theta}{\Gamma;\, \Phi \vdash l \,:\, \mathrm{ref}\ \theta}$$



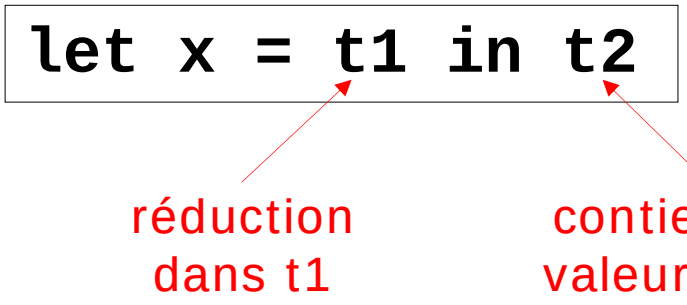
Règle "in-region" utilisée ici :

$$\frac{v \in \mu[\rho]}{\Gamma;\, \mu \vdash v \,:\, [\rho]}$$

où μ décrit le contenu des régions.

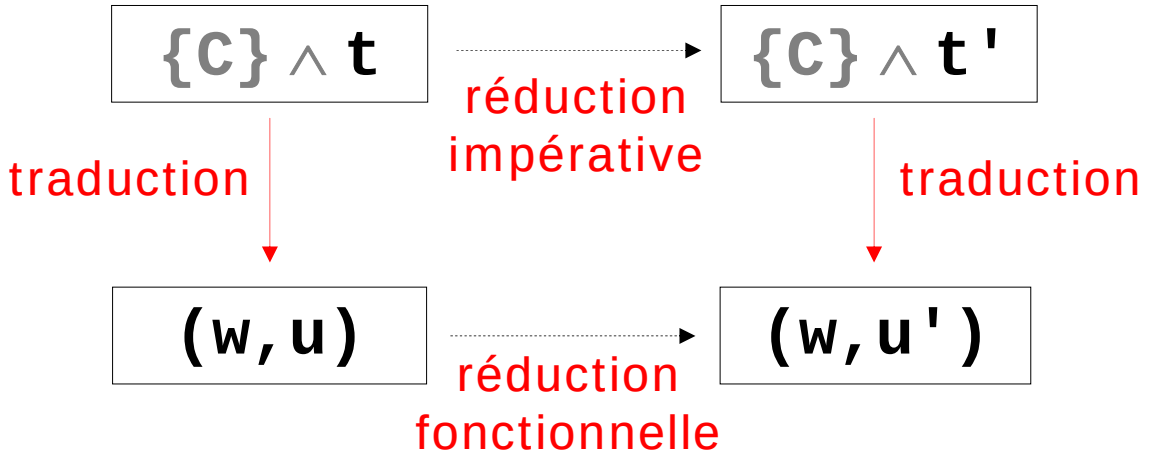
Arguments de stabilité

1– Stabilité du contenu des régions



on veut que v soit toujours dans la région r après réduction dans t1

2– Stabilité de la mémoire couverte par une capacité laissée de côté par frame



Travaux reliés de près

Ligne de travaux sur les régions et les capacités :

94 – Tofte & Talpin : allocation dans des régions qui s'empilent

99 – Calcul des Capacités : capacité = droit de désallouer des régions

00 – Alias Types : types dans les capacités sur des régions singletons

02 – Adoption and Focus : régions d'aliasing, adoption et focus

04 – Monnier : Calcul des Capacités + Alias Types + CiC

05 – Boyland : adoption champ par champ

Travaux reliés de plus loin

– Types linéaires $\theta^{\bullet} \equiv \exists \rho.\{\rho : \theta\}[\rho]$

– Logique de séparation $\dfrac{\{P\}\, t\, \{P'\}}{\{P * Q\}\, t\, \{P' * Q\}}$

– Ownership et invariant de classes sur les objets

– Autres traductions fonctionnelles : monadique, Why

Conclusions

Contribution : validation du concept de "traduction des capacités"

– consolide et étend des travaux sur des calculs de capacités,

– propose une traduction fonctionnelle dirigée par le typage.

Travaux futurs :

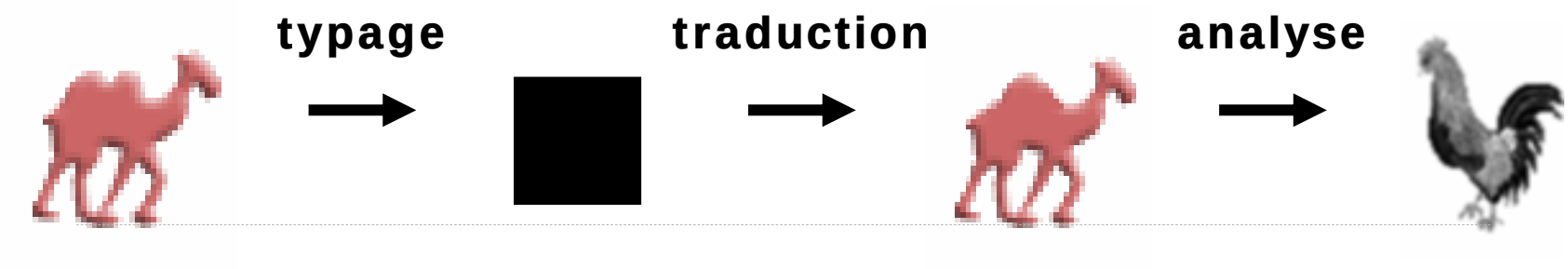
– augmenter la taille du langage source : records, tableaux, etc...

– augmenter l'expressivité : autres règles de sous-typage

– ajouter un système de module, un moteur d'inférence partielle

– obligations de preuve sortant des dérivations de typage

Obligations de preuves



Petite dépendance circulaire :

– la traduction est correcte si toutes les obligations sont prouvés,

– les obligations sont prouvées par analyse du code traduit.

Merci !

Bonus

Ex : Arguments aliasés

Sources impératifs :

```
let x = ref 7
let y = x
let z = ref 6
```

```
let copy (a,b) =
    let v = get a
    let _ = set b v
```

```
1: copy(x,z)
2: copy(y,z)
3: copy(x,y)
```

Typage :

```
x : [R1]    y : [R1]    z : [R2]
```

ok pour :

```
copy : ∀α.∀β.∀A.∀B.    {A:ref α}{B:ref β}([A]*[B])
                    ->  {A:ref α}{B:ref α}(unit)
```

1,2

```
copy : ∀α.∀A. {A:ref α}([A]*[A]) -> {A:ref α}(unit)
```

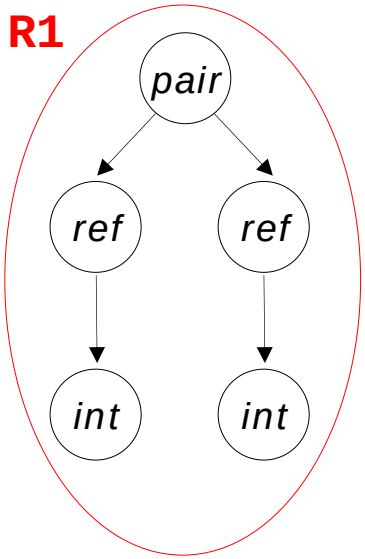
3

```
copy : ∀α.∀R. {R*:ref α}([R]*[R]) -> {R*:ref α}(unit)
```

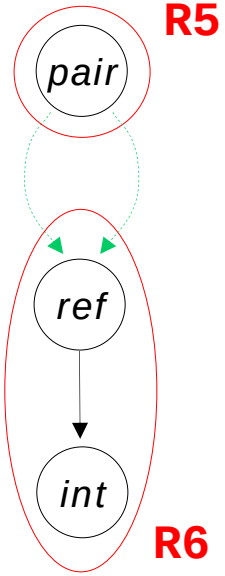
1,2,3

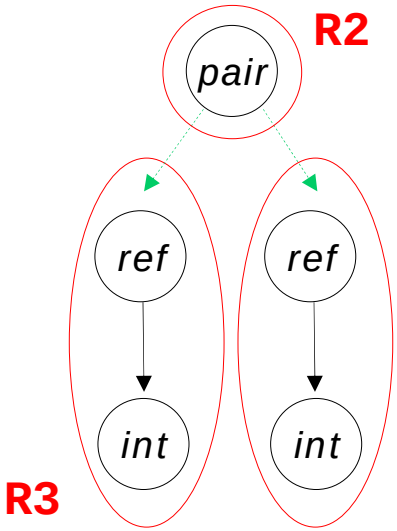
Ex : Paires de références

{R1 : ref int * ref int}



{R5 : ref [R6] * ref [R6]}
{R6 : ref int}





{R2 : ref [R3] * ref [R4]}
{R3 : ref int}
{R4 : ref int}

Langages source et cible

Langage source, impératif :

$$
\begin{array}{lcl}
v & := & x \mid n \mid 1 \mid \mathrm{inj}^{\mathrm{i}}\, v \mid (v_1, v_2) \mid (\lambda x.\, t) \mid p \mid l \\
p & := & \mathrm{proj}^{\mathrm{i}} \mid \mathrm{case} \mid \mathrm{ref} \mid \mathrm{get} \mid \mathrm{set} \\
t & := & v \mid (v\ t) \\
m & := & [l \mapsto v]
\end{array}
$$

primitive

location

(let x = t1 in t2) comme ((λx.t2) t1)

Langage cible, fonctionnel :

$$
\begin{array}{lcl}
v & := & x \mid n \mid 1 \mid \mathrm{inj}^{\mathrm{i}}\, v \mid (v_1, v_2) \mid (\lambda x.\, t) \mid p \mid h \mid i \\
p & := & \mathrm{proj}^{\mathrm{i}} \mid \mathrm{case} \mid \mathrm{map\_add} \mid \mathrm{map\_get} \mid \mathrm{map\_set} \\
t & := & v \mid (v\ t)
\end{array}
$$

map

clé

Capacités et types

Capacités :

$$C \quad := \quad \epsilon \mid \{\} \mid (C_1 \wedge C_2) \mid (\exists \rho . C) \mid \{\rho : \theta\} \mid \{\rho^\star : \theta\} \mid \{\rho_1 \multimap (\rho_2{}^\star : \theta)\}$$

Types des valeurs :

$$\begin{array}{lll} \tau & := & \mathrm{int} \mid \mathrm{unit} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \sigma_1 \to \sigma_2 \mid [\rho] \\ & & \mid \alpha \mid \forall\, \alpha . \tau \mid \forall\, \beta . \tau \mid \forall\, \rho . \tau \mid \forall\, \epsilon . \tau \end{array}$$

Types des termes :

$$\sigma \quad := \quad \tau \mid \exists \rho . \sigma \mid C \wedge \sigma$$

Types des régions :

$$\begin{array}{lll} \theta & := & \mathrm{int} \mid \mathrm{unit} \mid \theta_1 \times \theta_2 \mid \theta_1 + \theta_2 \mid \sigma_1 \to \sigma_2 \mid [\rho] \mid \mathrm{ref}\ \theta \mid \exists \rho . \theta \mid C \wedge \theta \\ & & \mid \alpha \mid \beta \mid \forall\, \alpha . \tau \mid \forall\, \beta . \tau \mid \forall\, \rho . \tau \mid \forall\, \epsilon . \tau \end{array}$$

Typage des valeurs et des termes

$$\textsc{var}\ \frac{(x:\tau)\ \in\ \Gamma}{\Gamma\ \vdash\ x\ :\ \tau} \qquad \textsc{fun}\ \frac{(\Gamma,\ x:\tau)\,;\,C\ \vdash\ t\ :\ \sigma \qquad \overline{\rho}\#\sigma,\Gamma}{\Gamma\ \vdash\ (\lambda x.\,t)\ :\ ((\exists\,\overline{\rho}.\,C\wedge\tau)\rightarrow\sigma)}$$

$$\textsc{int}\ \frac{}{\Gamma\ \vdash\ n\ :\ \mathrm{int}} \qquad \textsc{unit}\ \frac{}{\Gamma\ \vdash\ ()\ :\ \mathrm{unit}} \qquad \textsc{prim}\ \frac{(p:\tau)\ \in\ \mathsf{TP}}{\Gamma\ \vdash\ p\ :\ \tau}$$

$$\textsc{inj}\ \frac{\Gamma\ \vdash\ v\ :\ \tau_i}{\Gamma\ \vdash\ (\mathrm{inj}^{\mathrm{i}}\ v)\ :\ (\tau_1+\tau_2)} \qquad \textsc{pair}\ \frac{\Gamma\ \vdash\ v_1\ :\ \tau_1 \qquad \Gamma\ \vdash\ v_2\ :\ \tau_2}{\Gamma\ \vdash\ (v_1,v_2)\ :\ (\tau_1\times\tau_2)}$$

$$\textsc{val}\ \frac{\Gamma\ \vdash\ v\ :\ \tau}{\Gamma\,;\,\{\}\ \vdash\ v\ :\ \tau} \qquad \textsc{app}\ \frac{\Gamma\ \vdash\ v\ :\ (\sigma_1\rightarrow\sigma_2) \qquad \Gamma\,;\,C\ \vdash\ t\ :\ \sigma_1}{\Gamma\,;\,C\ \vdash\ (v\ t)\ :\ \sigma_2}$$

$$\exists\rho\text{-}\textsc{elim}\ \frac{\Gamma\,;\,C\ \vdash\ t\ :\ \sigma \qquad \rho\#\Gamma,\sigma}{\Gamma\,;\,(\exists\rho.C)\ \vdash\ t\ :\ \sigma} \qquad \textsc{frame}\ \frac{\Gamma\,;\,C_2\ \vdash\ t\ :\ \sigma}{\Gamma\,;\,(C_1\wedge C_2)\ \vdash\ t\ :\ (C_1\wedge\sigma)}$$

$$\textsc{sub-left}\ \frac{\Gamma\,;\,C_2\ \vdash\ t\ :\ \sigma \qquad C_1\ \leq\ C_2}{\Gamma\,;\,C_1\ \vdash\ t\ :\ \sigma} \qquad \textsc{sub-right}\ \frac{\Gamma\,;\,C\ \vdash\ t\ :\ \sigma_1 \qquad \sigma_1\ \leq\ \sigma_2}{\Gamma\,;\,C\ \vdash\ t\ :\ \sigma_2}$$

Règles de sous-typage (1)

$$
\begin{array}{lcll}
\text{FREE} & : & C & \leq & \{\} \\
\text{NEW-RGN} & : & \{\} & \leq & \exists\rho.\{\rho^\star : \theta\} \\
\text{UNFOCUS} & : & \{\rho_2 \multimap (\rho^\star : \theta)\}\{\rho_2 : \theta\} & \leq & \{\rho^\star : \theta\} \\
 & & & & \\
\text{PACK-RGN} & : & \{\rho_2 : \exists\rho.\theta\} & \equiv & \exists\rho.\{\rho_2 : \theta\} \\
\text{PACK-CAP} & : & \{\rho_2 : C \wedge \theta\} & \equiv & C \wedge \{\rho_2 : \theta\} \\
 & & & & \\
\text{OPEN-REF} & : & \{\rho_1 : \text{ref } \theta\} & \equiv & \exists\rho_2.\{\rho_1 : \text{ref } [\rho_2]\} \wedge \{\rho_2 : \theta\} \\
\text{OPEN-PROD} & : & \{\rho : (\theta_1 \times \theta_2)\} & \equiv & \exists\rho_1.\exists\rho_2.\{\rho : ([\rho_1] \times [\rho_2])\} \wedge \{\rho_1 : \theta_1\} \wedge \{\rho_2 : \theta_2\} \\
\text{OPEN-SUM} & : & \{\rho_2 : (\theta_1 + \theta_2)\} & \equiv & \exists y_1.\exists y_2.\{\rho_2 : ([y_1] + [y_2])\} \wedge \{{y_1}^\star : \theta_1\} \wedge \{{y_2}^\star : \theta_2\}
\end{array}
$$

$$
\begin{array}{lcll}
\text{FUNC} & : & \tau & \equiv & \exists\rho_2.\{\rho_2 : \tau\}[\rho_2] \\
\text{ADOPT} & : & \{\rho^\star : \theta\}\{\rho_2 : \theta\}[\rho_2] & \leq & \{\rho^\star : \theta\}[\rho] \\
\text{FOCUS} & : & \{\rho^\star : \theta\}[\rho] & \leq & \exists\rho_2.\{\rho_2 \multimap (\rho^\star : \theta)\}\{\rho_2 : \theta\}[\rho_2]
\end{array}
$$

Règles de sous-typage (2)

$$
\begin{array}{lcll}
\textsc{neutral} & : & \{\} \wedge C & \equiv & C \\
\textsc{assoc} & : & (C_1 \wedge C_2) \wedge C_3 & \equiv & C_1 \wedge (C_2 \wedge C_3) \\
\textsc{comm} & : & C_1 \wedge C_2 & \equiv & C_2 \wedge C_1 \\
\textsc{exists} & : & [\rho \to \rho']\, C & \leq & \exists \rho.C \\
\textsc{extrude} & : & (\exists \rho.C_1) \wedge C_2 & \equiv & \exists \rho.(C_1 \wedge C_2)
\end{array}
$$

$$
\begin{array}{lcll}
\textsc{swap-cap-}\exists\rho & : & C \wedge \exists \rho.\sigma & \equiv & \exists \rho.C \wedge \sigma \\
\textsc{swap-}\exists\rho\textsc{-}\exists\rho & : & \exists \rho_1.\exists \rho_2.\sigma & \equiv & \exists \rho_2.\exists \rho_1.\sigma \\
\textsc{factorize} & : & C_1 \wedge (C_2 \wedge \sigma) & \equiv & (C_1 \wedge C_2) \wedge \sigma \\
\textsc{extrude} & : & (\exists \rho.C) \wedge \sigma & \equiv & \exists \rho.(C \wedge \sigma) \\
\textsc{neutral} & : & \{\} \wedge \sigma & \equiv & \sigma \\
\textsc{fresh} & : & \exists \rho.\sigma & \equiv & \sigma \\
\textsc{exists} & : & [\rho \to \rho']\, \sigma & \leq & \exists \rho.\sigma
\end{array}
$$