Un système de types polymorphes

pour le langage BSML

Stage encadré par M Loulergue
au LACL de Parix XII

Projet **CARAML** :

- Bibliothèques de primitives parallèles
- Algorithmes en mémoire répartie
- Systèmes parallèles tolérants aux pannes
- Gestion dynamique des performances

Sujet de stage :

⋆ Conception d'un système de types
⋆ Sureté de typage
⋆ Création d'un algorithme de synthèse de types, implémentation
⋆ Extentions : types concrets, traits impératifs

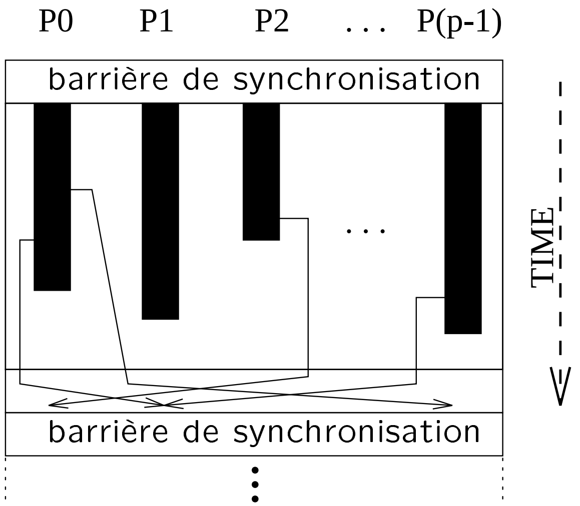
Plan

1. Modèle de parallélisme
2. BSML
3. Sytème de types
4. Sûreté
5. Algorithme d'inférence
6. Traits impératifs
7. Conclusion

Bulk Synchronous Parallelism (BSP)

Machine =

- ⋆ paires processeur-mémoire
- ⋆ réseau pour la communication
- ⋆ unité de synchronisation



$$\text{Temps}(s) = \max_{i:processeur} w_i^{(s)} + \max_{i:processeur} h_i^{(s)} * g + l$$

BSλ_P calcul

Syntaxe : $E ::= \bar{x} \mid E\ E \mid E\ e \mid \lambda\bar{x}.E \mid \lambda\dot{x}.E \mid \langle e, \ldots, e\rangle$
$\mid E\#E \mid E?E \mid (E \stackrel{e}{\rightarrow} E, E)$

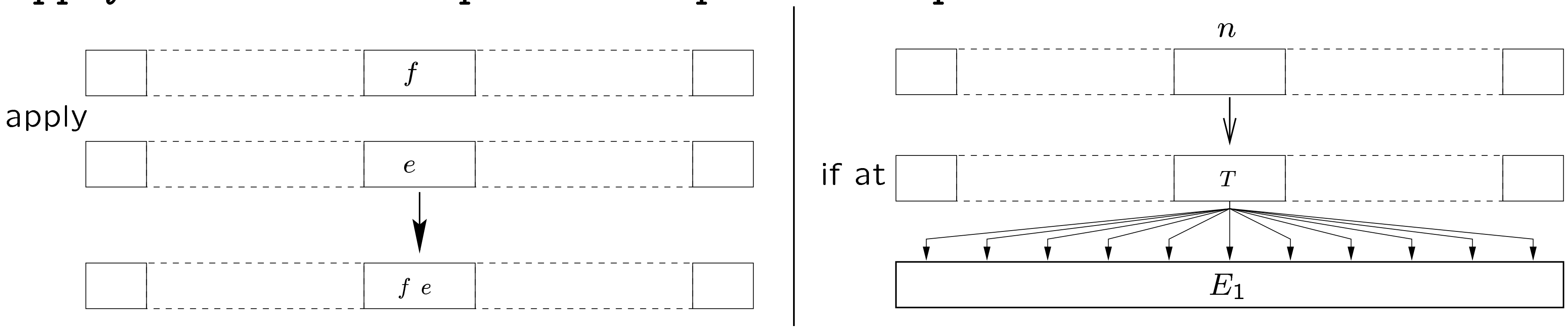
Règles :

1. $(\lambda\dot{x}.e)e' \rightarrow e[\dot{x} \leftarrow e']$
2. $(\lambda\bar{x}.E)\ E' \rightarrow E[\bar{x} \leftarrow E']$
3. $(\lambda\dot{x}.E)\ e' \rightarrow E[\dot{x} \leftarrow e']$
4. *Apply*, $\langle t_0, \ldots, t_{p-1}\rangle\#\langle u_0, \ldots, u_{p-1}\rangle \rightarrow \langle t_0\ u_0, \ldots t_{p-1}\ u_{p-1}\rangle$
5. *Get*, $\langle t_0, \ldots, t_{p-1}\rangle?\langle n_0, \ldots, n_{p-1}\rangle \rightarrow \langle t_{n_0}, \ldots t_{n_{p-1}}\rangle$
6. *Ifat*, $(\langle t_0, \ldots, \overbrace{S}^{n}, \ldots, t_{p-1}\rangle \stackrel{n}{\rightarrow} E_1, E_2) \rightarrow T$

Le langage BSML

- `bsp_p : unit->int`, le nombre de processeurs
- `'a par`, type abstrait des vecteurs parallèles
- `mkpar : (int -> 'a) -> 'a par`

- `apply : ('a -> 'b) par -> 'a par -> 'b par`



- `ifat : (bool par) -> int -> 'a -> 'a -> 'a`

Suite BSML

- `put : (int -> 'a option) par -> (int -> 'a option) par`

Explication pour `put(mkpar(fun i` $\rightarrow f_i$ `))`

$\Rightarrow$ `put(` $\langle f_0, f_1, f_2 \rangle$ `)`

$\Rightarrow$

f_0 0	f_1 0	f_2 0
f_0 1	f_1 1	f_2 1
f_0 2	f_1 2	f_2 2

$\Rightarrow$

v_{00}	v_{10}	v_{20}
v_{01}	v_{11}	v_{21}
v_{02}	—	—

$\Rightarrow$

v_{00}	v_{01}	v_{02}
v_{10}	v_{11}	—
v_{20}	v_{21}	—

$\Rightarrow$ <

fun $0 \rightarrow v_{00} \mid 1 \rightarrow v_{10} \mid 2 \rightarrow v_{20} \mid \_ \rightarrow$ None

fun $0 \rightarrow v_{01} \mid 1 \rightarrow v_{11} \mid 2 \rightarrow v_{21} \mid \_ \rightarrow$ None

fun $0 \rightarrow v_{02} \mid \_ \rightarrow$ None

>

Mini-BSML

Mini-ML avec :

1. Opérateurs locaux: $+$, **fst**, **snd**, **fix**, **Ifthenelse**, **nc**, **isnc**
2. Construction des vecteurs parallèles énumérés: **mkpar**
3. Opérateurs globaux: **apply**, **put**, **ifat**

Valeurs : idem mini-ML mais avec vecteurs parallèles énumérés

Sémantique naturelle :

$$\frac{e_1 \triangleright (\textbf{fun}\ x \rightarrow e) \quad \forall i (e[x \leftarrow i] \triangleright v_i)}{\textbf{mkpar}\ e_1 \triangleright \langle v_0, ..., v_{p-1} \rangle}$$

$$\frac{e_1 \triangleright \langle \textbf{fun}\ x \rightarrow e'_0, \ldots, \textbf{fun}\ x \rightarrow e'_{p-1} \rangle \quad e_2 \triangleright \langle v'_0, \ldots, v'_{p-1} \rangle \quad \forall i (e'_i[x \leftarrow v'_i] \triangleright v_i)}{\textbf{apply}\ e_1\ e_2 \triangleright \langle v_0, \ldots, v_{p-1} \rangle}$$

$$\frac{f \triangleright \langle \textbf{fun}\ dst \rightarrow e'_0, \ldots, \textbf{fun}\ dst \rightarrow e'_{p-1} \rangle \quad \forall j \forall i (e'_j[dst \leftarrow\ i] \triangleright v^i_j)}{\textbf{put}\ f \triangleright \langle f'_0, \ldots, f'_{p-1} \rangle}$$

where $\forall i (f'_i = \textbf{fun}\ x \rightarrow \textbf{if}\ x = 0\ \textbf{then}\ v^i_0\ \textbf{else} \ldots \textbf{if}\ x = (p-1)\ \textbf{then}\ v^i_{p-1}\ \textbf{else nc()})$

Problèmes

- `let replicate = fun x->mkpar(fun i->x)`
  `in replicate(mkpar(fun i->5))`

○Caml donne le type: **(int par) par**

- `let fst' = (fun x -> fun y -> x) in ...`

1. `(fst 1 2)`
2. `(fst (mkpar (fun i -> i)) (mkpar (fun i -> i)))`
3. `(fst (mkpar (fun i -> i)) 1)`
4. `(fst 1 (mkpar (fun i -> i)))`

○Caml donne **int** pour la 4.

Un système de types

On utilise :

types $\tau ::= \kappa \mid \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 * \tau_2 \mid (\tau\ par)$

contraintes $C ::= \textbf{False} \mid \textbf{True} \mid \mathcal{L}(\alpha) \mid C_1 \wedge C_2 \mid C_1 \Rightarrow C_2$

Avec des **schémas de types** : $\sigma ::= \forall \alpha_1 ... \alpha_n . [\tau / C]$

Localité d'un type :

$$\frac{\forall \alpha \in \kappa \quad \mathcal{L}(\alpha)}{\textbf{True}}\ (Kappa) \qquad \frac{\forall \tau \quad \mathcal{L}(\tau\ par)}{\textbf{False}}\ (Par)$$

$$\frac{\mathcal{L}(\tau_1 \to \tau_2)}{\mathcal{L}(\tau_1) \wedge \mathcal{L}(\tau_2)}\ (Arrow) \qquad \frac{\mathcal{L}(\tau_1 * \tau_2)}{\mathcal{L}(\tau_1) \wedge \mathcal{L}(\tau_2)}\ (Couple)$$

Définitions

Construction des contraintes :

$$\frac{\tau \text{ atomique}}{\tau \rightsquigarrow \textbf{True}}(CAtom) \qquad \frac{\tau \rightsquigarrow C_1}{(\tau\ par) \rightsquigarrow \mathcal{L}(\tau) \wedge C_1}(CPar)$$

$$\frac{\tau_1 \rightsquigarrow C_1 \quad \tau_2 \rightsquigarrow C_2}{(\tau_1, \tau_2) \rightsquigarrow C_1 \wedge C_2}(CCouple)$$

$$\frac{\tau_1 \rightsquigarrow C_1 \qquad \tau_2 \rightsquigarrow C_2}{(\tau_1 \to \tau_2) \rightsquigarrow C_1 \wedge C_2 \wedge (\mathcal{L}(\tau_2) \Rightarrow \mathcal{L}(\tau_1))}(CArrow)$$

Substitution :

$$\varphi(\forall \alpha_1 \ldots \alpha_n.[\tau/C]) = \forall \alpha_1 \ldots \alpha_n.[\varphi(\tau)/\varphi(C) \bigwedge_{\beta_i \in Dom(\varphi) \cap \mathcal{F}(\tau/C)} C_{\varphi(\beta_i)}]$$

Règles de typage

$$\frac{[\tau/C] \leq E(x)}{E \vdash x : [\tau/C]}(Var) \qquad \frac{[\tau/C] \leq TC(c)}{E \vdash c : [\tau/C]}(Const) \qquad \frac{[\tau/C] \leq TC(op)}{E \vdash op : [\tau/C]}(Op)$$

$$\frac{E + \{x : [\tau_1/C_{\tau_1}]\} \vdash e : [\tau_2/C_2]}{E \vdash (\textbf{fun } x \rightarrow e) : [\tau_1 \rightarrow \tau_2/C_{(\tau_1 \rightarrow \tau_2)} \wedge C_2]}(Fun)$$

$$\frac{E \vdash e_1 : [\tau' \rightarrow \tau/C_1] \qquad E \vdash e_2 : [\tau'/C_2]}{E \vdash (e_1\ e_2) : [\tau/C_1 \wedge C_2]}(App)$$

$$\frac{E \vdash e_1 : [\tau_1/C_1] \qquad E + \{x : Gen([\tau_1/C_1], E)\} \vdash e_2 : [\tau_2/C_2]}{E \vdash \textbf{let } x = e_1 \textbf{ in } e_2 : [\tau_2/C_1 \wedge C_2 \wedge \mathcal{L}(\tau_2) \Rightarrow \mathcal{L}(\tau_1)]}(Let)$$

$$\frac{E \vdash e_1 : [\tau_1/C_1] \qquad E \vdash e_2 : [\tau_2/C_2]}{E \vdash (e_1, e_2) : [\tau_1 * \tau_2/C_1 \wedge C_2]}(Couple)$$

Exemples

$$
\begin{array}{lcl}
TC(\textbf{fst}) & = & \forall\alpha\beta.[(\alpha * \beta) \rightarrow \alpha / \mathcal{L}(\alpha) \Rightarrow \mathcal{L}(\beta)] \\
TC(\textbf{mkpar}) & = & \forall\alpha.[(int \rightarrow \alpha) \rightarrow (\alpha\ par) / \mathcal{L}(\alpha)] \\
TC(\textbf{apply}) & = & \forall\alpha\beta.[((\alpha \rightarrow \beta)\ par * (\alpha\ par)) \rightarrow (\beta\ par) / \mathcal{L}(\alpha) \wedge \mathcal{L}(\beta)] \\
TC(\textbf{ifat}) & = & \forall\alpha.[(int * (bool\ par) * \alpha * \alpha) \rightarrow \alpha / \mathcal{L}(\alpha) \Rightarrow \textbf{False}] \\
TC(\textbf{put}) & = & \forall\alpha.[(int \rightarrow \alpha)\ par \rightarrow (int \rightarrow \alpha)\ par / \mathcal{L}(\alpha)] \\
TC(\textbf{isnc}) & = & \forall\alpha.[\alpha \rightarrow bool / \mathcal{L}(\alpha)]
\end{array}
$$

`replicate`:$\forall\alpha.[\alpha \rightarrow (\alpha\ par) / \mathcal{L}(\alpha)]$

et $\vdash$ (**mkpar** (**fun** $i \rightarrow$ 5)) : $(int\ par)$ donc $Solve(\mathcal{L}(int\ par)) =$ **False**.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{int \leq int}{\{b : int\ par, a : int\} \vdash a : int}
}{\{b : int\ par\} \vdash \textbf{fun}\ b \rightarrow a : [(int\ par) \rightarrow int / C_{(int\ par) \rightarrow int}]}
}{\vdash \textbf{fun}\ a \rightarrow \textbf{fun}\ b \rightarrow a : ?}
\quad
\dfrac{int \leq TC(1)}{\vdash 1 : int}
}{\vdash (\textbf{fun}\ a \rightarrow \textbf{fun}\ b \rightarrow a)\ 1 : ?}
}{}
\quad
\dfrac{}{\vdash \ldots}
}{\vdash ((\textbf{fun}\ a \rightarrow \textbf{fun}\ b \rightarrow a)\ 1)(mkpar\ (\textbf{fun}\ i \rightarrow 5)) : ?}
$$

Sémantique à petits pas et Sûreté

Evaluation :

$$e_0 \rightharpoonup e_1 \rightharpoonup e_2 \rightharpoonup \ldots \rightharpoonup v$$

$$\mathbf{mkpar}(\mathbf{fun}\ x \to\ e) \overset{\varepsilon}{\rightharpoonup} \langle e[x \leftarrow 0], \ldots, e[x \leftarrow (p-1)]\rangle \quad (\delta_{mkpar})$$

$$\frac{e \overset{\varepsilon}{\rightharpoonup} e'}{\Gamma(e) \rightharpoonup \Gamma(e')}$$

Contexte : $\Gamma \ ::= \ [] \ | \ \ldots \ | \ \langle e, \ldots, \overbrace{\Gamma}^{i}, e, \ldots, e\rangle \ \ldots$

Moins typable : $e_1 \sqsubseteq e_2$ si $E \vdash e_1 : [\tau/C]$ alors $\exists C'$ tels que $\forall \varphi \in \phi_C$ alors $\varphi \models C'$ et $E \vdash e_2 : [\tau/C']$.

$\star$ si $e \overset{*}{\rightharpoonup} e'$ alors $e \sqsubseteq e'$

$\star$ si $\emptyset \vdash e : [\tau/C]$ et e en forme normale alors c'est une valeur.

$\star$ si $E \vdash e : [\tau/C]$ alors $\exists\ e_\lambda \in BS\lambda_p$ tel que $e\mathcal{T}e_\lambda$.

Algorithme d'inférence, exemple

Comme pour mini-ML : instance trivialle, unification de types
Mais ajoût de la résolution des contraintes.

```
let replicate = fun x -> mkpar(fun i -> x)
   in replicate(replicate 2);;
```

```
OP: mkpar : [((int -> a3) -> (a3 par)) | L(a3)]
VAR: x : a1
FUN: (fun i -> x) : [(a4 -> a1) | (L(a1) => L(a4))]
APP: (mkpar (fun i -> x)) : [(a1 par) | L(a1)]
FUN: (fun x -> (mkpar (fun i -> x))) : [(a1 -> (a1 par)) | L(a1)]
VAR: replicate : [(a6 -> (a6 par)) | L(a6)]
VAR: replicate : [(a7 -> (a7 par)) | L(a7)]
APP: (replicate 2) : (int par)
A constraint is solved to false
```

Traits impératifs

Mémoires : $S = [s_0, \ldots, s_{p-1}]$
Mémoires vides : $\emptyset_{\bowtie} = \{\emptyset_0, \ldots, \emptyset_{p-1}\}$

Evaluation :

$$e/S \rightharpoonup e'/S'$$
$$e_0/S_0 \rightharpoonup e_1/S_1 \rightharpoonup e_2/S_2 \rightharpoonup \ldots \rightharpoonup v/S$$

2 types de réduction :

1. un processeur : $e/s_i \overset{i}{\rightharpoonup} e'/s'_i$
2. le réseau : $e/S \overset{\bowtie}{\rightharpoonup} e'/S'$

Réduction locale :

$$(\textbf{fun}\ x \to e)\ v\ /\ s_i \quad \underset{i}{\overset{\varepsilon}{\rightharpoonup}} \quad e[x \leftarrow v]\ /\ s_i \quad (\beta_{fun_i})$$
$$(\textbf{let}\ x = v\ \textbf{in}\ e)\ /\ s_i \quad \underset{i}{\overset{\varepsilon}{\rightharpoonup}} \quad e[x \leftarrow v]\ /\ s_i \quad (\beta_{let_i})$$

Réduction globale :

$$
\begin{array}{llll}
(\textbf{fun } x \to e)\ v\ /\ S & \overset{\varepsilon}{\underset{i}{\rightharpoonup}} & e[x \leftarrow v]\ /\ S & (\beta_{fun\bowtie}) \\
(\textbf{let } x = v \textbf{ in } e)\ /\ S & \overset{\varepsilon}{\underset{i}{\rightharpoonup}} & e[x \leftarrow v]\ /\ S & (\beta_{let\bowtie})
\end{array}
$$

Opérateurs d'allocations globaux :

$$
\begin{array}{lll}
\textbf{ref}(v)\ /\ S & \overset{\varepsilon}{\underset{\bowtie}{\rightharpoonup}} & \ell_{(i)}\ /\ S' \text{ ou } S' = [s_0 + \{\ell_0 \leftarrow v\}, \ldots, s_{p-1} + \{\ell_{p-1} \leftarrow v\}] \\
\textbf{!}(\ell_{(i)})\ /\ S & \overset{\varepsilon}{\underset{\bowtie}{\rightharpoonup}} & \forall i\ s_i(\ell_i)\ /\ S \text{ si } \ell_i \in Dom(s_i) \\
\textbf{:=}(\ell_{(i)}, v)\ /\ S & \overset{\varepsilon}{\underset{\bowtie}{\rightharpoonup}} & ()\ /\ S' \quad \text{ou } S' = [s_0 + \{\ell_0 \leftarrow v\}, \ldots, s_{p-1} + \{\ell_{p-1} \leftarrow v\}] \\
\ell_{(i)}\ /\ s_j & \overset{\varepsilon}{\underset{j}{\rightharpoonup}} & \ell_j\ /\ s_j
\end{array}
$$

2 types de contextes :

1. $\Gamma_{\bowtie} = \Gamma$ sans les cas vecteurs parallèles
2. $\Gamma_l^i = \ldots \mid \langle e, \ldots, \overbrace{\Gamma}^{i}, e, \ldots, e \rangle$

$$
\frac{e\ /\ s_i \overset{\varepsilon}{\underset{i}{\rightharpoonup}} e'\ /\ s'_i}{\Gamma_l^i(e)\ /\ s_i \rightharpoonup \Gamma_l^i(e')\ /\ s'_i}
\qquad \Bigg| \qquad
\frac{e\ /\ S \overset{\varepsilon}{\underset{\bowtie}{\rightharpoonup}} e'\ /\ S'}{\Gamma_{\bowtie}(e)\ /\ S \rightharpoonup \Gamma_{\bowtie}(e')\ /\ S'}
$$

Exemple et problèmes

$$
\begin{array}{lll}
 & \textbf{let } a = \textbf{ref}(0) \textbf{ in mkpar}(\textbf{fun } j \to a := j) & / \ \emptyset_{\bowtie} \\
\rightharpoonup & \textbf{let } a = \ell_{(i)} \textbf{ in mkpar}(\textbf{fun } i \to a := j) & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 0\}\} \\
\rightharpoonup & \textbf{mkpar}(\textbf{fun } j \to \ell_{(i)} := i) & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 0\}\} \\
\rightharpoonup & \langle \ell_{(i)} := 0, \ell_{(i)} := 1 \rangle & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 0\}\} \\
\rightharpoonup & \langle \ell_{(i)} := 0, \ell_1 := 1 \rangle & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 0\}\} \\
\rightharpoonup & \langle \ell_{(i)} := 0, () \rangle & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 1\}\} \\
\rightharpoonup & \ldots & \\
\rightharpoonup & \langle (), () \rangle & / \ \{\{\ell_0 \leftarrow 0\}, \{\ell_1 \leftarrow 1\}\}
\end{array}
$$

Problème : si un **if at** suit... problème de synchronisation...

Autre problème : le put.

2 solutions :

1. empêcher l'envoi d'allocations locales lors du typage
2. changer la sémantique (créer de nouvelles allocations...)

Conclusion

Fait :

1. Plus de distinctions syntaxiques, cohérent avec le BSλ_p-calcul
2. Sémantique proche de l'évaluation distribuée, sûreté
3. Algorithme d'inférence de types
4. Mini-BSML le plus expressif
5. Ajoût de traits impératifs

A faire :

1. Ajoût d'un système contre les effets de bord
2. Filtrage étendu et exceptions
3. BSML complet pour le calcul par grille
4. Interaction avec le $join$-calcul