Inférence de flots d'information pour ML avec références et exceptions

Vincent Simonet

Séminaire LogiCal – Cristal – Moscova
Vendredi 2 février 2001

Inférence de flots d'information pour ML avec références et exceptions

Notre but est d'intégrer à un langage tel que CamlLight un système de types permettant d'analyser les flots d'information. On peut distinguer deux étapes principales :

1. Conception d'un système de type sur un langage du type mini-ML et preuve d'une propriété de non-interférence.
2. Écriture d'un algorithme d'inférence efficace et implantation effective.

Les extensions « simples » de ML (e.g. n-uplets, enregistrements, sommes) n'entrainent aucune difficulté. Deux extensions usuelles de ML sont intéressantes : références et exceptions.

Systèmes existants

Dennis Volpano et Geoffrey Smith (1997)
Système de type sur un langage impératif simple. Limité au premier ordre et à un nombre fini de références globales.

Nevin Heintze et Jon G. Riecke SLam Calculus (1997)
Lambda-calcul avec références et processus. Le typage des références n'est pas fin. Il n'est pas énoncé ni prouvé de propriété de sûreté.

Andrew C. Myers JFlow (1999)
Analyse de flots d'information sur Java. Le système est complexe et n'est pas prouvé.

Steve Zdancewic et Andrew C. Myers (2001)
Analyse sur un langage de bas niveau avec continuations.

ML avec références

$$v \; ::= k \mid x \mid m \mid \lambda x.e \mid \ell : v \qquad \text{(valeurs)}$$

$$e \; ::= v \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid e\ e \mid !\ e \mid e := e \mid \mathsf{ref}\ e \mid \ell : e \qquad \text{(expressions)}$$

$$\sigma \; ::= \{m_1 \mapsto v_1; \ldots; m_n \mapsto v_n\} \qquad \text{(états-mémoire)}$$

Les étiquettes ℓ appartiennent à un treillis $\mathcal{L}$. Elles représentent les niveaux de sécurité.

On ne définit pas de réduction pour le calcul étiqueté : étant donnée une expression e, on s'intéresse à la réduction de $\mathrm{strip}(e)$ (l'expression obtenue à partir de e en gommant les étiquettes).

Flots directs

$$\boxed{\begin{array}{l}\text{let } r = 0 \text{ in} \\ r := (H : 1); \\ !\ r\end{array}}$$

$$\boxed{\begin{array}{l}\text{let } r = 0 \text{ in} \\ \text{let } f = H : (\lambda xy.y) \text{ in} \\ r := f\ 1\ 2; \\ !\ r\end{array}}$$

Si une information secrète est stockée dans une référence r, le résultat du déréférencement de r doit également être considéré comme secret.

Flots indirects

$$\boxed{\begin{array}{l}\text{let } r = 0 \text{ in}\\ H : (r := 1);\\ !\ r\end{array}}$$

$$\boxed{\begin{array}{l}\text{let } r = \text{ref } 0 \text{ in}\\ \text{let } f = \lambda x.(r := x) \text{ in}\\ H : (f\ 1);\\ !\ r\end{array}}$$

Si une référence est modifiée dans un contexte (pc) secret, le contenu de la référence doit alors lui-même être considéré comme secret.

Lorsqu'une fonction est appliquée, il faut lui « passer » le niveau du pc au point d'appel.

Flots indirects (2)

```
let a = ref 0 in
let b = ref 0 in
let r = ref a in
H : (r := b);
(! r) := 1;
! a
```

```
let a = ref 0 in
let b = ref 0 in
let r = ref a in
H : ();
(! r) := 1;
! a
```

Approche par traduction

L'approche par traduction s'adapte difficilement :

- Il est nécessaire de suivre *en parallèle* deux réductions simultanément.
- Les axiomes sur la relation de typage deviennent difficiles à exprimer, notamment pour ce qui concerne le typage des références.

Nous envisageons donc une approche relativement différente en introduisant un « calcul partagé » et en donnant directement les règles de typage.

- Cette méthode est relativement simple et fait appel à des techniques connues : le cœur de la démonstration est une preuve de *Subject-reduction*.
- On perd cependant l'intuition offerte par la traduction pour « trouver » les bonnes règles de typage.

Calcul partagé

Le calcul partagé permet de considérer simultanément deux expressions et deux réductions en gérant le partage. On se fixe un ensemble clos L d'étiquettes considérées non secrètes.

Syntaxe

$$v \;::=\; k \mid x \mid m \mid \lambda x.e \mid \langle v \mid v \rangle$$

$$e \;::=\; v \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid e\ e \mid !\ e \mid e := e \mid \mathsf{ref}\ e \mid \langle e \mid e \rangle$$

On ne considère que les expressions où les $\langle \ldots \mid \ldots \rangle$ ne sont pas imbriqués.

Projections $\lfloor e \rfloor_1$ (resp. $\lfloor e \rfloor_2$) est l'expression obtenue à partir de e en ne conservant que les moitiés gauches (resp. droites) des crochets.

Codage Étant données deux expressions e_1 et e_2 du calcul étiqueté ayant même partie publique (i.e. $\lfloor e_1 \rfloor_L = \lfloor e_2 \rfloor_L$), on construit une expression du calcul partagé $[\![e_1 \mid e_2]\!]_L$.

Exemple de codage

let a = ref 0 in
let b = ref 0 in
let r = ref a in
$H : (r := b);$
$(!\ r) := 1;$
$!\ a$

$$e_1 \qquad e_2$$

$$\curlyvee$$

$$[\![e_1 \mid e_2]\!]$$

let a = ref 0 in
let b = ref 0 in
let r = ref a in
$H : ();$
$(!\ r) := 1;$
$!\ a$

let a = ref 0 in
let b = ref 0 in
let r = ref a in
$\langle r := b \mid () \rangle;$
$(!\ r) := 1;$
$!\ a$

Réduction dans le calcul partagé

$$
\begin{array}{rcll}
(\lambda x.e)\ v \ /\ \sigma & \xrightarrow{i} & e[x \Leftarrow v] \ /\ \sigma & (\beta_i) \\
\mathsf{let}\ x = v\ \mathsf{in}\ e \ /\ \sigma & \xrightarrow{i} & e[x \Leftarrow v] \ /\ \sigma & (\mathsf{let}_i) \\
\mathsf{ref}\ v \ /\ \sigma & \xrightarrow{i} & m \ /\ \sigma + \{m \mapsto \mathrm{new}_i(v)\} & (\mathsf{ref}_i) \\
m := v' \ /\ \sigma + \{m \mapsto v\} & \xrightarrow{i} & () \ /\sigma + \{m \mapsto \mathrm{update}_i(v, v')\} & (\mathsf{assign}_i) \\
!\ m \ /\ \sigma + \{m \mapsto v\} & \xrightarrow{i} & v \ /\ \sigma + \{m \mapsto v\} & (\mathsf{deref}_i)
\end{array}
$$

$$
\begin{array}{rcll}
\langle v_1 \mid v_2 \rangle\ \ v \ /\ \sigma & \xrightarrow{\varepsilon} & \langle v_1\ \lfloor v \rfloor_1 \mid v_2\ \lfloor v \rfloor_2 \rangle \ /\ \sigma & (\text{lift-app}) \\
\langle v_1 \mid v_2 \rangle := v \ /\ \sigma & \xrightarrow{\varepsilon} & \langle v_1 := \lfloor v \rfloor_1 \mid v_2 := \lfloor v \rfloor_2 \rangle \ /\ \sigma & (\text{lift-assign}) \\
op\ \ \langle v_1 \mid v_2 \rangle \ /\ \sigma & \xrightarrow{\varepsilon} & \langle op\ v_1 \mid op\ v_2 \rangle \ /\ \sigma & (\text{lift-op})
\end{array}
$$

$$
i \in \{\varepsilon, 1, 2\} \qquad op \in \{!, \mathsf{ref}\}
$$

$$
\frac{e \ /\ \sigma \xrightarrow{i} e' \ /\ \sigma'}{E[e] \ /\ \sigma \xrightarrow{i} E[e'] \ /\ \sigma'} \ (\text{contexte}) \qquad \frac{e_i \ /\ \sigma \xrightarrow{i} e'_i \ /\ \sigma' \qquad e'_j = e_j}{\langle e_1 \mid e_2 \rangle \xrightarrow{\varepsilon} \langle e'_1 \mid e'_2 \rangle} \ (\text{parallel})
$$

Simulation et stabilité

Lemme [Simulation]

Si $e \,/\, \sigma \rightarrow e' \,/\, \sigma'$ alors $\begin{cases} \llbracket e \,/\, \sigma \rrbracket_1 \rightarrow^= \llbracket e' \,/\, \sigma' \rrbracket_1 \\ \llbracket e \,/\, \sigma \rrbracket_2 \rightarrow^= \llbracket e' \,/\, \sigma' \rrbracket_2 \end{cases}$

Si $\begin{cases} \llbracket e \,/\, \sigma \rrbracket_1 \rightarrow^* v_1 \\ \llbracket e \,/\, \sigma \rrbracket_2 \rightarrow^* v_2 \end{cases}$ alors il existe v tel que $\begin{cases} v_1 = \llbracket v \rrbracket_1 \\ v_2 = \llbracket v \rrbracket_2 \end{cases}$ et $e \,/\, \sigma \rightarrow^* v$.

Théorème [Stabilité]

Si $e \,/\, \sigma \rightarrow v \,/\, \tau$ et v ne comporte pas de crochets alors

$$\begin{cases} \llbracket e \rrbracket_1 \,/\, \llbracket \sigma \rrbracket_1 \rightarrow^* v \\ \llbracket e \rrbracket_2 \,/\, \llbracket \sigma \rrbracket_2 \rightarrow^* v \end{cases}$$

Exemple de réduction

	m_a	m_b	m_r
$\text{let } a = \text{ref } 0 \text{ in let } b = \text{ref } 0 \text{ in}$ $\text{let } r = \text{ref } a \text{ in}$ $\langle r := b \mid ()\rangle \,;\; (!\; r) := 1;\; !\; a$			
$\rightarrow^* \langle m_r := m_b \mid ()\rangle \,;\; (!\; m_r) := 1;\; !\; m_a$	0	0	m_a
$\rightarrow^* (!\; m_r) := 1;\; !\; m_a$	0	0	$\langle m_b \mid m_a\rangle$
$\rightarrow^* \langle m_b \mid m_a\rangle := 1;\; !\; m_a$			
$\rightarrow^* \langle m_b := 1 \mid m_a := 1\rangle \,;\; !\; m_a$			
$\rightarrow^* \langle () \mid m_a := 1\rangle \,;\; !\; m_a$	0	$\langle 1 \mid 0\rangle$	$\langle m_b \mid m_a\rangle$
$\rightarrow^* !\; m_a$	$\langle 0 \mid 1\rangle$	$\langle 1 \mid 0\rangle$	$\langle m_b \mid m_a\rangle$
$\rightarrow^* \langle 0 \mid 1\rangle$			

Relation de typage

Algèbre des types

$$
\begin{array}{rll}
t & ::= \mathsf{int} \mid \mathsf{unit} \mid (s \xrightarrow{pc} s') \mid (s\ \mathsf{ref}) & \text{(types)} \\
s & ::= t^{\ell} & \text{(types annotés)}
\end{array}
$$

On note $\ell_0 \lhd t^{\ell}$ si et seulement si $\ell_0 \leq \ell$.

Jugements

$$pc, \Gamma \vdash e : s$$

Règles de typage : références

Ref
$$\frac{pc, \Gamma \vdash e : s \qquad pc \lhd s}{pc, \Gamma \vdash \mathsf{ref}\ e : (s\ \mathsf{ref})^{\ell}}$$

Deref
$$\frac{pc, \Gamma \vdash e : (s\ \mathsf{ref})^{\ell} \qquad s \leq s' \qquad \ell \lhd s'}{pc, \Gamma \vdash !\ e : s'}$$

Assign
$$\frac{pc, \Gamma \vdash e_1 : (s\ \mathsf{ref})^{\ell} \qquad pc, \Gamma \vdash e_2 : s \qquad pc \sqcup \ell \lhd s}{pc, \Gamma \vdash e_1 := e_2 : \mathsf{unit}^{\ell}}$$

Le contenu d'une référence a un niveau supérieur à égal

- au pc du point où la référence est créée,
- au pc de chaque point du programme où son contenu est susceptible d'être modifié.

Règles de typage : fonctions

Abs
$$\frac{pc', \Gamma[x \mapsto s] \vdash e : s'}{pc, \Gamma \vdash \lambda x.e : (s \xrightarrow{pc'} s')^{\ell}}$$

App
$$\frac{pc, \Gamma \vdash e_1 : (s \xrightarrow{pc \sqcup \ell} s')^{\ell} \qquad pc, \Gamma \vdash e_2 : s}{pc, \Gamma \vdash e_1\ e_2 : s'}$$

Le corps d'une abstraction est typé avec le niveau du contexte (pc') sous lequel la fonction est effectivement appliquée.

Règles Parallel et Label

Dans le calcul étiqueté

Label

$$\frac{pc \sqcup \ell, \Gamma \vdash e : s \qquad \ell \lhd s}{pc, \Gamma \vdash (\ell : e) : s}$$

Dans le calcul partagé

Parallel

$$\frac{pc \sqcup \ell, \Gamma \vdash_L e_1 : s \qquad pc \sqcup \ell, \Gamma \vdash_L e_2 : s \qquad \ell \notin L \qquad \ell \lhd s}{pc, \Gamma \vdash_L \langle e_1 \mid e_2 \rangle : s}$$

Subject-reduction et non-interférence

Théorème [Subject-reduction]
Si $pc, \emptyset \vdash_L e : s$ et $e \rightarrow^* e' \,/\, \sigma'$ alors il existe Γ tel que $pc, \Gamma \vdash_L e' \,/\, \sigma' : s$.

On en déduit immédiatement un résultat de non-interférence :

Corollaire [Non-interférence]
Si $e \rightarrow^* v$ et $pc, \emptyset \vdash_L e : \mathsf{int}^\ell$ avec $\ell \in L$ alors $\begin{cases} \llbracket e \rrbracket_1 \rightarrow^* v \\ \llbracket e \rrbracket_2 \rightarrow^* v \end{cases}$

Non-interférence dans le calcul étiqueté

Definition [$\trianglelefteq$]
On dit que e_1 « effectue plus d'effets de bord » que e_2 (et on note $e_1 \trianglelefteq_L e_2$) si on peut obtenir e_2 à partir de e_1 en remplaçant des valeurs closes situées sous des étiquettes secrètes (i.e. n'appartenant pas à L) par d'autres valeurs closes ayant au moins les mêmes types.

Théorème [Non-interférence]
Soient e_1 et e_2 deux expressions du calcul étiqueté.

Si $\begin{cases} pc, \Gamma \vdash e_1 : \mathsf{int}^\ell \\ \mathrm{strip}(e_i) \rightarrow^* v_i \\ e_1 \trianglelefteq_{\downarrow \ell} e_2 \end{cases}$ alors $v_1 = v_2$.

ML avec exceptions

Version étiquetée

$$v ::= k \mid x \mid \lambda x.e \mid \ell : v \qquad \text{(valeurs)}$$

$$e ::= v \mid \text{let } x = e \text{ in } e \mid e\ e \mid \text{raise } e \mid \text{try } e \text{ with } x \rhd e \mid \ell : e \qquad \text{(expressions)}$$

$$a ::= v \mid \text{raise } v \mid \ell : a \qquad \text{(réponses)}$$

Version partagée

$$v ::= k \mid x \mid m \mid \lambda x.e \mid \langle v \mid v \rangle \qquad \text{(valeurs)}$$

$$e ::= v \mid \text{let } x = e \text{ in } e \mid e\ e \mid \text{raise } e \mid \text{try } e \text{ with } x \rhd e \mid \langle e \mid e \rangle \qquad \text{(expressions)}$$

$$a ::= v \mid \text{raise } v \mid \langle a \mid a \rangle \qquad \text{(réponses)}$$

Traduction du langage à exceptions dans un langage avec sommes

Une approche permettant d'obtenir des règles de typage est de coder les expressions de *ML avec exceptions* dans *ML avec sommes*. Cependant, le système ainsi obtenu n'est pas assez fin.

Cas de l'application

$$[\![e_1\ e_2]\!] = \text{match } [\![e_1]\!] \text{ with } (\lambda.v_1 \text{ match } [\![e_2]\!] \text{ with } v_1\ \ \mathrm{inj}_2\,)\ \ \mathrm{inj}_2$$

App

$$\frac{pc, \Gamma \vdash [\![e_1]\!] : ((s \xrightarrow{pc'} (s' + s_0)^{\ell'})^{\ell} + s_0)^{\ell_1} \qquad pc, \Gamma \vdash [\![e_2]\!] : (s + s_0)^{\ell_2}}{pc, \Gamma \vdash [\![e_1\ e_2]\!] : (s' + s_0)^{\ell_1 \sqcup \ell_2 \sqcup \ell \sqcup \ell'}}$$

$$pc' = pc \sqcup \ell \sqcup \ell_1 \sqcup \ell_2$$

Typage avec alternatives

Algèbre des types

$$
\begin{array}{lll}
a & ::= \mathsf{Abs} \mid \mathsf{Pre}^{\ell}\ s & \text{(alternatives)} \\
t & ::= \mathsf{int} \mid \mathsf{unit} \mid (s \xrightarrow{pc\ [a]} s') \mid (s\ \mathsf{ref}) & \text{(types)} \\
s & ::= t^{\ell} & \text{(types annotés)}
\end{array}
$$

$\mu(\mathsf{Abs}) = \bot$ et $\mu(\mathsf{Pre}^{\ell}\ s) = \ell$

Exemple de l'application

$$
\textsf{App}\quad \frac{pc, \Gamma \vdash e_1 : (s \xrightarrow{pc'\ [a]} s')^{\ell}\ [a_1] \qquad pc \sqcup \mu(a_1), \Gamma \vdash e_2 : s\ [a_2] \qquad \ell \lhd s'}{pc, \Gamma \vdash e_1\ e_2 : s\ [a_1 \sqcup a_2 \sqcup a]}
$$

$$
pc' = pc \sqcup \ell \sqcup \mu(a_1) \sqcup \mu(a_2)
$$

Exceptions en CamlLight

Le modèle précédent reste trop limité pour traiter convenablement les exceptions de CamlLight.

On dispose de plusieurs sortes d'exceptions Le langage permet de rattraper une exception particulière ou toutes les exceptions simultanément. On souhaite pouvoir distinguer un niveau pour chaque exception disponible. Il suffit d'utiliser une rangée d'alternatives.

On souhaite traiter finement deux constructions

$$\text{try} \ \ldots \ \text{with } x \triangleright (\ldots; \ \text{raise } x)$$

$$\text{try} \ \ldots \ \text{finally} \ \ldots$$