"Typing-by-encoding"

A reductionistic approach to building type systems

François Pottier

Francois.Pottier@inria.fr

Overview

- What is "typing-by-encoding"?
- Encoding exceptions into sums (folklore).
- Encoding Java-like stack inspection into sets (with S. Smith & C. Skalka).
- Encoding information flow into labels (with S. Conchon).

"Typing-by-encoding"

Assumptions:

- A target calculus, with reduction $\rightarrow$ and administrative reduction $\rightarrow_\sim$.
- A source calculus, with reduction $\rightarrow$.
- A translation $[\![\cdot]\!]$ from the latter to the former.
- Soundness: $e \rightarrow e'$ implies $[\![e]\!] \rightarrow^\star \cdot \stackrel{\star}{\sim}\leftarrow [\![e']\!]$.
- Completeness: If e is stuck, then $[\![e]\!]$ goes wrong.

So far, everything is *untyped*.

A "type system factory"

Then, *any* type system for the target calculus which satisfies

- Subject Reduction: If $e : t$ and $e \longrightarrow f$, then $f : t$.
- Progress: No stuck expression is well-typed.
- Administrative Subject Expansion: If $f : t$ and $f \sim\!\!\leftarrow e$, then $e : t$.

gives rise to a type system for the source calculus, which satisfies Subject Reduction and Progress, defined by

$$e : t \qquad \Longleftrightarrow \qquad [\![e]\!] : t$$

(Core) ML

$$
\begin{array}{rrcl}
\text{Values} & v & ::= & \lambda x.e \\
\text{Expressions} & e & ::= & x \mid \lambda x.e \mid e\,e \mid \text{let } x = e \text{ in } e
\end{array}
$$

$$
\begin{array}{rcll}
(\lambda x.e)\,v & \rightarrow & e[v/x] & \\
\text{let } x = v \text{ in } e & \rightarrow & e[v/x] & \\
E[e] & \rightarrow & E[e'] & \text{when } e \rightarrow e'
\end{array}
$$

$$
E \quad ::= \quad [] \mid E\,e \mid v\,E \mid \text{let } x = E \text{ in } e
$$

ML+exceptions

$$
\begin{array}{rrcl}
\text{Values} & v & ::= & \ldots \mid \text{raise} \mid \text{try}\, e \\
\text{Expressions} & e & ::= & \ldots \mid \text{raise} \mid \text{try}\, e \\
\text{Answers} & a & ::= & v \mid \text{raise}\, v
\end{array}
$$

$$
\begin{array}{rcll}
U[\text{raise}\, v] & \rightarrow & \text{raise}\, v & \text{when } U \neq [] \\
\text{try}\, (\text{raise}\, v_1)\, v_2 & \rightarrow & v_2\, v_1 & \\
\text{try}\, v_1\, v_2 & \rightarrow & v_1 &
\end{array}
$$

$$
\begin{array}{rcl}
E & ::= & \ldots \mid \text{try}\, E\, v \\
U & ::= & [] \mid U\, e \mid v\, U \mid \text{let } x = U \text{ in } e
\end{array}
$$

ML+sums

$$
\begin{array}{rrcl}
\text{Values} & v & ::= & \ldots \mid \text{inl} \mid \text{inr} \mid \text{inl}\, v \mid \text{inr}\, v \mid \text{match} \mid \text{match}\, v \mid \text{match}\, v\, v \\
\text{Expressions} & e & ::= & \ldots \mid \text{inl} \mid \text{inr} \mid \text{match}
\end{array}
$$

$$
\begin{array}{rcl}
\text{match}\,(\text{inl}\, v)\, v_1\, v_2 & \rightarrow & v_1\, v \\
\text{match}\,(\text{inr}\, v)\, v_1\, v_2 & \rightarrow & v_2\, v
\end{array}
$$

Encoding exceptions into sums

$$
\begin{array}{rcl}
[\![x]\!] & = & \mathrm{inl}\, x \\
[\![\lambda x.e]\!] & = & \mathrm{inl}\, \lambda x.[\![e]\!] \\
[\![e_1\, e_2]\!] & = & \mathrm{match}\, [\![e_1]\!]\, (\lambda v_1.\mathrm{match}\, [\![e_2]\!]\, v_1\, \mathrm{inr})\, \mathrm{inr} \\
[\![\mathrm{raise}]\!] & = & \mathrm{inl}\, \mathrm{inr} \\
[\![\mathrm{try}\, e]\!] & = & \mathrm{inl}\, \lambda h.\mathrm{match}\, [\![e]\!]\, \mathrm{inl}\, h
\end{array}
$$

$$
\begin{array}{rcl}
[\![\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2]\!] & = & \mathrm{match}\, [\![e_1]\!]\, (\lambda x.[\![e_2]\!])\, \mathrm{inr} \\
[\![\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2]\!] & = & \mathrm{let}\ a = [\![e_1]\!]\ \mathrm{in} \\
 & & \mathrm{match}\, a\, (\lambda x.\mathrm{let}\ x = \mathrm{match}\, a\, (\lambda v.v)\, (\lambda v.\bot)\ \mathrm{in}\ [\![e_2]\!])\, \mathrm{inr}
\end{array}
$$

Typing ML+sums à la Hindley/Milner

$$\begin{aligned}
\text{inl} &: \forall\alpha\beta.\alpha \to \alpha + \beta \\
\text{inr} &: \forall\alpha\beta.\beta \to \alpha + \beta \\
\text{match} &: \forall\alpha\beta\gamma.\alpha + \beta \to (\alpha \to \gamma) \to (\beta \to \gamma) \to \gamma
\end{aligned}$$

Typing ML+exceptions, through the encoding

$$\frac{\Gamma(x) = \sigma \qquad \sigma \preccurlyeq t}{\Gamma \vdash x : t, \bot} \qquad \frac{\Gamma; x : t \vdash e : t', u}{\Gamma \vdash \lambda x.e : t \xrightarrow{u} t', \bot}$$

$$\frac{\Gamma \vdash e_1 : t_2 \xrightarrow{u} t, u \qquad \Gamma \vdash e_2 : t_2, u}{\Gamma \vdash e_1\, e_2 : t, u} \qquad \Gamma \vdash \text{raise} : t \xrightarrow{t} \bot, \bot$$

$$\frac{\Gamma \vdash e : t, u}{\Gamma \vdash \text{try}\, e : (u \xrightarrow{u'} t) \xrightarrow{u'} t, \bot} \qquad \frac{\Gamma \vdash e_1 : t, u \qquad \Gamma; x : \text{Gen}(t, \Gamma) \vdash e_2 : t', u}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : t', u}$$

Notation: t, u stands for $t + u$; $t \xrightarrow{u} t'$ stands for $t \to t' + u$; $\bot$ stands for a fresh meta-variable.

ML+Java-like stack inspection

Principals	$\mathcal{P}$
Resources	$\mathcal{R}$
Access rights matrix	$\mathcal{A} : \mathcal{P} \to 2^{\mathcal{R}}$

f	$::=$	$p.e$	Signed expression
e	$::=$	$x \mid \lambda x.f \mid e\,e \mid \text{let } x = e \text{ in } e$	"Signed ML"
	$\mid$	$r.e$	Enable privilege
	$\mid$	$r!e$	Check privilege
	$\mid$	f	

ML+Java-like stack inspection (cont'd)

$$\begin{array}{rrcl} \text{Evaluation contexts} & E & ::= & \ldots \mid r.E \mid p.E \\ \text{Stacks} & S & ::= & \epsilon \mid rS \mid pS \end{array}$$

If P stands for $\{p \in \mathcal{P}\,;\, r \in \mathcal{A}(p)\}$, then

$$S \vdash r \qquad \Longleftrightarrow \qquad S \in (\mathcal{P} \mid \mathcal{R})^\star P \mathcal{R}^\star r (P \mid \mathcal{R})^\star$$

$$\begin{array}{rcll} E[r!e] & \rightarrow & E[e] & \text{if } E \vdash r \\ r.v & \rightarrow & v & \\ p.v & \rightarrow & v & \end{array}$$

ML+sets

$$\begin{array}{rrcl} \text{Values} & v & ::= & \ldots \mid R \mid {}_{\cdot r} \mid \vee_R \mid \wedge_R \\ \text{Expressions} & e & ::= & \ldots \mid R \mid {}_{\cdot r} \mid \vee_R \mid \wedge_R \end{array}$$

$$\begin{array}{rcll} R.r & \rightarrow & R & \text{if } r \in R \\ R_1 \vee R_2 & \rightarrow & R_1 \cup R_2 & \\ R_1 \wedge R_2 & \rightarrow & R_1 \cap R_2 & \end{array}$$

Encoding ML+stack inspection into ML+sets: security-passing style

$$
\begin{aligned}
[\![x]\!]_p &= x \\
[\![\lambda x.f]\!]_p &= \lambda x.\lambda s.[\![f]\!] \\
[\![e_1\, e_2]\!]_p &= [\![e_1]\!]_p\, [\![e_2]\!]_p\, s \\
[\![\text{let } x = e_1 \text{ in } e_2]\!]_p &= \text{let } x = [\![e_1]\!]_p \text{ in } [\![e_2]\!]_p \\
[\![r.e]\!]_p &= \text{let } s = s \vee (\{r\} \cap \mathcal{A}(p)) \text{ in } [\![e]\!]_p \\
[\![r!e]\!]_p &= \text{let } \_ = s.r \text{ in } [\![e]\!]_p \\
[\![f]\!]_p &= [\![f]\!] \\
[\![p.e]\!] &= \text{let } s = s \wedge \mathcal{A}(p) \text{ in } [\![e]\!]_p
\end{aligned}
$$

$$
(\!|\, e \,|\!) = [\![e]\!]_{p_0}[\varnothing/s]
$$

Typing ML+sets à la Hindley/Milner/Rémy

$$
\begin{array}{rcl}
\{r_1, \ldots, r_n\} & : & [\overrightarrow{r_i : \mathtt{Pre}}\,;\ \mathtt{Abs}] \\
._r & : & \forall\rho.[r : \mathtt{Pre}\,;\ \rho] \to [r : \mathtt{Pre}\,;\ \rho] \\
\vee_{\{r_1,\ldots,r_n\}} & : & \forall\rho\overrightarrow{\varphi_i}.[\overrightarrow{r_i : \varphi_i}\,;\ \rho] \to [\overrightarrow{r_i : \mathtt{Pre}}\,;\ \rho] \\
\wedge_{\{r_1,\ldots,r_n\}} & : & \forall\rho\overrightarrow{\varphi_i}.[\overrightarrow{r_i : \varphi_i}\,;\ \rho] \to [\overrightarrow{r_i : \varphi_i}\,;\ \mathtt{Abs}]
\end{array}
$$

Typing ML+stack inspection, through the encoding (excerpts)

$$\frac{s_2, (\Gamma; x : t_1) \vdash f : t_2}{p, s_1, \Gamma \vdash \lambda x.f : t_1 \xrightarrow{s_2} t_2} \qquad \frac{p, s, \Gamma \vdash e_1 : t_2 \xrightarrow{s} t \qquad p, s, \Gamma \vdash e_2 : t_2}{p, s, \Gamma \vdash e_1\, e_2 : t}$$

$$\frac{p, [r : \texttt{Pre}\,;\ rt], \Gamma \vdash e : t \qquad r \in \mathcal{A}(p)}{p, [r : ft\,;\ rt], \Gamma \vdash r.e : t} \qquad \frac{p, [r : \texttt{Pre}\,;\ rt], \Gamma \vdash e : t}{p, [r : \texttt{Pre}\,;\ rt], \Gamma \vdash r!e : t}$$

$$\frac{p_0, [\texttt{Abs}], \epsilon \vdash e : t}{e : t}$$

Notation: $t \xrightarrow{s} t'$ stands for $t \to s \to t'$.

ML+dependency

$$\begin{array}{rll} \text{Expressions} & e ::= \ldots \mid l : e & (l \in \mathcal{L}) \\ \text{Evaluation contexts} & \text{arbitrary} & \end{array}$$

$$(l : e_1)\, e_2 \quad \rightarrow \quad l : (e_1\, e_2)$$

Stability: If $e \rightarrow^\star f$ and $\lfloor f \rfloor_L = f$, then $\lfloor e \rfloor_L \rightarrow^\star f$.

ML+pairs+labels

$$\begin{array}{rl} \text{Expressions} & e ::= \ldots \mid \langle e, e \rangle \mid \text{fst} \mid \text{snd} \mid l \mid @ \quad (l \in \mathcal{L}) \\ \text{Evaluation contexts} & \text{arbitrary} \end{array}$$

$$\begin{array}{rcl} \text{fst}\ \langle e_1, e_2 \rangle & \rightarrow & e_1 \\ \text{snd}\ \langle e_1, e_2 \rangle & \rightarrow & e_2 \\ l \mathbin{@} m & \rightarrow & l \sqcup_{\mathcal{L}} m \\ (e_1 \mathbin{@} e_2) \mathbin{@} e_3 & \rightarrow_{@} & e_1 \mathbin{@} (e_2 \mathbin{@} e_3) \\ \epsilon \mathbin{@} e & \rightarrow_{@} & e \end{array}$$

Encoding dependencies into labels

$$
\begin{array}{rcl}
[\![x]\!] & = & \langle \mathrm{fst}\ x, \mathrm{snd}\ x \rangle \\
[\![\lambda x.e]\!] & = & \langle \lambda x.[\![e]\!], \bot_{\mathcal{L}} \rangle \\
[\![e_1\ e_2]\!] & = & \mathrm{letp}\ \langle x, t \rangle = [\![e_1]\!]\ \mathrm{in} \\
 & & \mathrm{letp}\ \langle y, u \rangle = x\ [\![e_2]\!]\ \mathrm{in} \\
 & & \langle y, t \mathbin{@} u \rangle \\
[\![\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2]\!] & = & \mathrm{let}\ x = [\![e_1]\!]\ \mathrm{in}\ [\![e_2]\!] \\
[\![l : e]\!] & = & \mathrm{letp}\ \langle x, t \rangle = [\![e]\!]\ \mathrm{in} \\
 & & \langle x, l \mathbin{@} t \rangle
\end{array}
$$

The "type system factory"

Equip ML+pairs+labels with a compositional type system such that

- $\rightarrow$ and $\rightarrow_@$ preserve types;
- labels may be viewed as types, and $l : m$ implies $l \leq_{\mathcal{L}} m$.

This defines a type system for ML+dependency which enjoys Subject Reduction and Non-Interference:

$$\text{If } e : \texttt{int} \times l \text{ and } e \rightarrow^\star v, \text{ where } v \text{ is a value, then } \lfloor e \rfloor_{\downarrow l} \rightarrow^\star v.$$

```
let fix ff =
  (fun f x -> ff (f f) x) (fun f x -> ff (f f) x)

let exists = fix (
  fun exists predicate list ->
    match list with
      Nil ->
        false
    | Cons (element, rest) ->
        if predicate element then
          true
        else
          exists predicate rest
)
```

Equip ML+pairs+labels with a subtyping-constraint-based type system.

Then, the (inferred) type of *exists* is

$$\forall\alpha\psi\zeta \mid \{\zeta \leq [\,\texttt{Nil} \mid \texttt{Cons}\ \mathit{of}\ (\alpha^{\varphi} \times \zeta)^{\psi}\,]^{\psi}\}.$$

$$(\alpha^{\varphi} \to \texttt{bool}^{\psi})^{\psi} \to \zeta \to \texttt{bool}^{\psi}$$

```
let users =
  Cons({ login = "Pam"; pw = Sys : "7nuggets" },
  Cons({ login = "Sam"; pw = Sys : "" },
  Nil))

let query1 =
  exists (fun r →
    r.login = Priv : "Monica"
  ) users

let query2 =
  exists (fun r →
    r.pw = ""
  ) users
```