ML programs are typed

Programs of type ty: $\boxed{\mathbf{Prog}_{ty}} \triangleq \{ e \mid \emptyset \vdash e : ty \}$

where

Type assignment relation $\boxed{\Gamma \vdash e : ty}$

Γ = typing context — finite function from identifiers to types

e = expression to be typed

ty = type

is inductively generated by axioms and rules *following the structure* of e, for example: [See A.4]

$$\frac{\Gamma \vdash e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin dom(\Gamma)}{\Gamma \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2) : ty_2}$$

Theorem (Type Soundness). If $e, s \Rightarrow v, s'$ and $e \in \mathbf{Prog}_{ty}$, then $v \in \mathbf{Prog}_{ty}$.

ML programs are typed

Programs of type ty: $\boxed{\mathbf{Prog}_{ty}} \triangleq \{ e \mid \emptyset \vdash e : ty \}$

where

Type assignment relation $\boxed{\Gamma \vdash e : ty}$

Γ = typing context

e = expression to be typed

ty = type

is inductively generated by axioms and rules *following the structure* of e, for example:

$$\frac{\Gamma \vdash e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin dom(\Gamma)}{\Gamma \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2) : ty_2}$$

Theorem (Type Soundness). If $e, s \Rightarrow v, s'$ and $e \in \mathbf{Prog}_{ty}$, then $v \in \mathbf{Prog}_{ty}$.

proof by induction on the derivation of $e, s \Rightarrow v, s'$

ML programs are typed

Programs of type ty: $\boxed{\mathbf{Prog}_{ty}} \triangleq \{ e \mid \emptyset \vdash e : ty \}$

where

Type assignment relation $\boxed{\Gamma \vdash e : ty}$ $\begin{cases} \Gamma & = \text{typing context} \\ e & = \text{expression to be typed} \\ ty & = \text{type} \end{cases}$

is inductively generated by axioms and rules *following the structure* of e, for example:

$$\frac{\Gamma \vdash e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin dom(\Gamma)}{\Gamma \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2) : ty_2}$$

Theorem (Type ~~Soundness~~ PRESERVATION). If $e, s \Rightarrow v, s'$ and $e \in \mathbf{Prog}_{ty}$, then $v \in \mathbf{Prog}_{ty}$.

What about "PROGRESS" ?

ML transition relation $\boxed{(s\ ,\ e) \to (s'\ ,\ e')}$ where $loc(e) \subseteq dom(s)$ & $loc(e') \subseteq dom(s')$

is inductively generated by rules following the structure of e—e.g.

a simplification step

$$\frac{(s\ ,\ e_1) \to (s'\ ,\ e_1')}{(s\ ,\ \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2) \to (s'\ ,\ \mathtt{let}\ x = e_1'\ \mathtt{in}\ e_2)}$$

a basic reduction

$$\frac{v \text{ a canonical form}}{(s\ ,\ \mathtt{let}\ x = v\ \mathtt{in}\ e) \to (s\ ,\ e[v/x])}$$

(see Sect. A.5 for the full definition).

Write $\to^*$ for reflexive-transitive closure of $\to$.

For example...

Recall (p381):

$F \triangleq$
let a = ref() in
let b = ref() in
fun $x \to$
if $x == a$ then b
else a

$G \triangleq$
let c = ref() in
let d = ref() in
fun $y \to$
if $y == d$ then d
else c

For $T \triangleq$ fun $f \to$ let x = ref() in $f(f\,x) == f\,x$,
$T\,F$ has value false, whereas $T\,G$ has value true,
so $F \not\cong_{\mathrm{ctx}} G$.

$(\emptyset, TF) \to^* (s, Tv)$ where $\begin{cases} s \triangleq \{\ell_1 \mapsto (), \ell_2 \mapsto ()\} \\ v \triangleq \text{fun } x \to \text{if } x == \ell_1 \text{ then } \ell_2 \text{ else } \ell_1 \end{cases}$

$\downarrow^*$

$(s, \text{let } x = \text{ref}() \text{ in } v(v\,x) == v\,x)$

$\downarrow^*$

$(s', v(v\,\ell_3) == v\,\ell_3)$ where $s' = \{\ell_1 \mapsto (), \ell_2 \mapsto (), \ell_3 \mapsto ()\}$

$\downarrow^*$

$(s', v\,\ell_1 == v\,\ell_3)$

$\downarrow^*$

$(s', \ell_2 == v\,\ell_3)$

$\downarrow^*$

$(s', \ell_2 == \ell_1) \longrightarrow^* (s', \text{false})$

$(\emptyset, T\,G) \to^* (s, T\,v')$ where $\begin{cases} s \triangleq \{\ell_1 \mapsto (), \ell_2 \mapsto ()\} \\ v' \triangleq \text{fun } x \to \text{if } x == \ell_2 \text{ then } \ell_2 \text{ else } \ell_1 \end{cases}$

$$\downarrow^*$$

$(s, \text{let } x = \text{ref}() \text{ in } v(v\,x) == v\,x)$

$$\downarrow^*$$

$(s', v(v\,\ell_3) == v\,\ell_3)$ where $s' = \{\ell_1 \mapsto (), \ell_2 \mapsto (), \ell_3 \mapsto ()\}$

$$\downarrow^*$$

$(s', v\,\ell_1 == v\,\ell_3)$

$$\downarrow^*$$

$(s', \ell_1 == v\,\ell_3)$

$$\downarrow^*$$

$(s', \ell_1 == \ell_1) \longrightarrow^* (s', \text{true})$

Relationship between evaluation and transition

Theorem A.2 $\quad s, e \Rightarrow v, s'$ iff $(s, e) \to^* (s', v)$

<u>Proof</u> via two lemmas:

① $\boxed{s, e \Rightarrow v, s' \text{ implies } (s, e) \to^* (s', v)}$

(by induction on derivation of $s, e \Rightarrow v, s'$)

② $\boxed{(s, e) \to (s', e') \text{ implies } \forall v, s'' \, (s', e' \Rightarrow v, s'' \text{ implies } s, e \Rightarrow v, s'')}$

(by induction on derivation of $(s, e) \to (s', e')$)

Repeated use of ② gives

$(s, e) \to^* (s', e')$ & $s', e' \Rightarrow v, s''$ implies $s, e \Rightarrow v, s''$

So since $s', v \Rightarrow v, s'$, get converse of ①:

$(s, e) \to^* (s', v)$ implies $s, e \Rightarrow v, s'$ $\quad \square$

ML programs are typed

Programs of type ty: $\boxed{\mathbf{Prog}_{ty}} \triangleq \{\, e \mid \emptyset \vdash e : ty \,\}$

where

Type assignment relation $\boxed{\Gamma \vdash e : ty}$
$\begin{cases} \Gamma & = \text{typing context} \\ e & = \text{expression to be typed} \\ ty & = \text{type} \end{cases}$

is inductively generated by axioms and rules *following the structure* of e, for example:

$$\frac{\Gamma \vdash e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin dom(\Gamma)}{\Gamma \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2) : ty_2}$$

Theorem (Type ~~Soundness~~ PRESERVATION). If $e, s \Rightarrow v, s'$ and $e \in \mathbf{Prog}_{ty}$, then $v \in \mathbf{Prog}_{ty}$.

What about "PROGRESS" ?

Progress

Evaluation of well-typed programs does not get stuck, in the sense that

if $e \in \mathrm{Prog}_{ty}$ and $loc(e) \subseteq dom(s)$

then <u>either</u> e is in canonical form

<u>or</u> $(s,e) \to (s',e')$ holds for some s' & e'.

(Proof by induction on the structure of e.)