ML programs are typed

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

where

Type assignment relation

$\boxed{\Gamma \vdash e : ty}$

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

finite function from identifiers to types

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

[See A.4]

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

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

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). 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:

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

PRESERVATION

Theorem (Type ~~Soundness~~). 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) ⊆ dom(s) & loc(e') ⊆ 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):

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

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

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

$$(\emptyset, TF) \to^* (S, Tv) \quad \text{where} \begin{cases} S \triangleq \{l_1 \mapsto (), l_2 \mapsto ()\} \\ v \triangleq \text{fun } x \to \text{if } x == l_1 \\ \quad \text{then } l_2 \text{ else } l_1 \end{cases}$$

$$\downarrow^*$$

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

$$\downarrow^*$$

$$(S', v(v\,l_3) == v\,l_3) \quad \text{where } S' = \{l_1 \mapsto (), l_2 \mapsto (), l_3 \mapsto ()\}$$

$$\downarrow^*$$

$$(S', v\,l_1 == v\,l_3)$$

$$\downarrow^*$$

$$(S', l_2 == v\,l_3)$$

$$\downarrow^*$$

$$(S', l_2 == l_1) \longrightarrow^* (S', \text{false})$$

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

$$\downarrow^*$$

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

$$\downarrow^*$$

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

$$\downarrow^*$$

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

$$\downarrow^*$$

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

$$\downarrow^*$$

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

Relationship between evaluation and transition

$\boxed{\textbf{Theorem A.2} \quad s, e \Rightarrow v, s' \text{ iff } (s,e) \rightarrow^* (s', v)}$

Proof via two lemmas:

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

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

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

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

Repeated use of ② gives

$$(s,e) \rightarrow^* (s', e') \;\&\; s', e' \Rightarrow v, s'' \text{ implies } s, e \Rightarrow v, s''$$

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

$$(s,e) \rightarrow^* (s', v) \text{ implies } s, e \Rightarrow v, s' \qquad \square$$

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:

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

PRESERVATION

Theorem (Type ~~Soundness~~). 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.)