ML programs are typed

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

where

Type assignment relation $\Gamma \vdash e : ty$

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

\[
\begin{align*}
\Gamma & \vdash e_1 : ty_1 \\
\Gamma[x \mapsto ty_1] & \vdash e_2 : ty_2 \\
\hline
\Gamma & \vdash (\text{let } x = e_1 \text{ in } e_2) : ty_2
\end{align*}
\]

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

ML programs are typed

Programs of type $ty$: $\textbf{Prog}_{ty} \triangleq \{ e | \emptyset \vdash e : ty \}$

where

Type assignment relation

$$\Gamma \vdash e : ty$$

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

$$\Gamma \vdash e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin \text{dom}(\Gamma)$$

$$\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : ty_2$$

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

proof by induction on the derivation of $e, s \Rightarrow v, s'$
ML programs are typed

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

where

Type assignment relation $\Gamma \vdash e : ty$

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

$\Gamma \parallel e_1 : ty_1 \quad \Gamma[x \mapsto ty_1] \vdash e_2 : ty_2 \quad x \notin \text{dom}(\Gamma)$

$\Gamma \parallel (\text{let } x = e_1 \text{ in } e_2) : ty_2$

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

What about "progress"?
ML transition relation

\[(s, e) \rightarrow (s', e')\]

is inductively generated by rules following the structure of \(e\)—e.g. a simplification step

\[
(s, e_1) \rightarrow (s', e'_1)
\]

\[
\overline{(s, \text{let } x = e_1 \text{ in } e_2) \rightarrow (s', \text{let } x = e'_1 \text{ in } e_2)}
\]

a basic reduction

\[
u \text{ a canonical form}
\]

\[
(s, \text{let } x = v \text{ in } e) \rightarrow (s, e[v/x])
\]

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

Write \(\rightarrow^*\) for reflexive-transitive closure of \(\rightarrow\).

For example...
Recall (p.381):

\[ F \triangleq \]
\[
\text{let } a = \text{ref}()\text{in} \\
\text{let } b = \text{ref}()\text{in} \\
\text{fun } x \to \\
\text{if } x == a \text{ then } b \\
\text{else } a
\]

\[ G \triangleq \]
\[
\text{let } c = \text{ref}()\text{in} \\
\text{let } d = \text{ref}()\text{in} \\
\text{fun } y \to \\
\text{if } y == d \text{ then } d \\
\text{else } c
\]

For \( T \triangleq \text{fun } f \to \text{let } x = \text{ref}()\text{in} f(f\ x) == f\ x, \)

\( TF \) has value \( \text{false} \), whereas \( TG \) has value \( \text{true} \),

so \( F \not\sim_{ctx} G \).
\((\emptyset, \text{TF}) \rightarrow^* (s, Tv)\) where \(s \triangleq \{l_1 \mapsto (\cdot), l_2 \mapsto (\cdot)\}\)

\(v \triangleq \text{fun } x \rightarrow \text{if } x = l_1 \text{ then } l_2 \text{ else } l_1\)

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

\(\downarrow^*\)

\((s', v(vl_3) == vl_3)\)

where \(s' \triangleq \{l_1 \mapsto (\cdot), l_2 \mapsto (\cdot), l_3 \mapsto (\cdot)\}\)

\((s', vl_1 == vl_3)\)

\(\downarrow^*\)

\((s', l_2 == vl_3)\)

\(\downarrow^*\)

\((s', l_2 == l_1) \rightarrow^* (s', \text{false})\)
\[(\emptyset, T_G) \rightarrow^* (s, T_{v'}) \quad \text{where} \quad \left\{ \begin{array}{l}
 s \overset{\Delta}{=} \{ l_1 \mapsto (), l_2 \mapsto () \} \\
 v' \overset{\Delta}{=} \text{fun } x \rightarrow \text{if } x = = l_2 \text{ then } l_2 \text{ else } l_1
\end{array} \right.

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

\downarrow^*

(s', v(vl_3) = = vl_3)

\downarrow^*

(s', vl_1 = = vl_3)

\downarrow^*

(s', l_1 = = vl_3)

\downarrow^*

(s', l_1 = = l_1) \rightarrow^* (s', \text{true})
Theorem A.2 \( s,e \Rightarrow v,s' \iff (s,e) \rightarrow^*(s',v) \)

Proof via two lemmas:

1. \( s,e \Rightarrow v,s' \) implies \((s,e) \rightarrow^*(s',v)\) (by induction on derivation of \( s,e \Rightarrow v,s' \))

2. \((s,e) \rightarrow (s',e')\) implies \( \forall v,s'' \ (s',e' \Rightarrow v,s'' \) implies \( s,e \Rightarrow v,s''\)) (by induction on derivation of \( (s,e) \rightarrow (s',e')\))

Repeated use of 2 gives

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

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

\((s,e) \rightarrow^* (s',v) \) implies \( s,e \Rightarrow v,s' \)

\[\square\]
ML programs are typed

Programs of type $\text{ty}$: \[ \text{Prog}_\text{ty} \triangleq \{ e \mid \emptyset \vdash e : \text{ty} \} \]

where

Type assignment relation \[ \Gamma \vdash e : \text{ty} \]

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

\[ \Gamma \vdash e_1 : \text{ty}_1 \quad \Gamma[x \mapsto \text{ty}_1] \vdash e_2 : \text{ty}_2 \quad x \notin \text{dom}(\Gamma) \]

\[ \Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \text{ty}_2 \]

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

*What about “PROGRESS”?*
Progress

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

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

then either \( e \) is in canonical form

or \( (s, e) \rightarrow (s', e') \) holds for some \( s' \) and \( e' \).

(Proof by induction on the structure of \( e \).)