Formal type systems

- Constitute the precise, mathematical characterisation of informal type systems (such as occur in the manuals of most typed languages.)
- Basis for *type soundness* theorems: “any well-typed program cannot produce run-time errors (of some specified kind)”.
- Can decouple specification of typing aspects of a language from algorithmic concerns: the formal type system can define typing independently of particular implementations of type-checking algorithms.

Mini-ML expressions, M

$M ::=$	x	variable
$\mid$	$\mathtt{true}$	boolean values
$\mid$	$\mathtt{false}$	
$\mid$	$\mathtt{if}\ M\ \mathtt{then}\ M\ \mathtt{else}\ M$	conditional
$\mid$	$\lambda x(M)$	function abstraction
$\mid$	$M\ M$	function application
$\mid$	$\mathtt{let}\ x = M\ \mathtt{in}\ M$	local declaration
$\mid$	$\mathtt{nil}$	nil list
$\mid$	$M :: M$	list cons
$\mid$	$\mathtt{case}\ M\ \mathtt{of}\ \mathtt{nil} \Rightarrow M \mid x :: x \Rightarrow M$	case expression

ML types and expressions for mutable references

$$
\begin{array}{rcll}
\tau & ::= & \ldots & \\
 & | & unit & \text{unit type} \\
 & | & \tau\ ref & \text{reference type.} \\
 & & & \\
M & ::= & \ldots & \\
 & | & () & \text{unit value} \\
 & | & \texttt{ref}\ M & \text{reference creation} \\
 & | & !M & \text{dereference} \\
 & | & M := M & \text{assignment}
\end{array}
$$

Midi-ML's extra typing rules

(unit) $$\Gamma \vdash () : unit$$

(ref) $$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \texttt{ref}\ M : \tau\ ref}$$

(get) $$\frac{\Gamma \vdash M : \tau\ ref}{\Gamma \vdash\ !M : \tau}$$

(set) $$\frac{\Gamma \vdash M_1 : \tau\ ref \quad \Gamma \vdash M_2 : \tau}{\Gamma \vdash M_1 := M_2 : unit}$$

Example 3.1.1

The expression

$$\begin{array}{l} \mathtt{let}\, r = \mathtt{ref}\ \lambda x(x)\ \mathtt{in} \\ \quad \mathtt{let}\, u = (r := \lambda x'(\mathtt{ref}\ !x'))\ \mathtt{in} \\ \qquad (!r)() \end{array}$$

has type $unit$.

$: \forall\alpha\,((\alpha \to \alpha)\,\mathrm{ref})$

Example 3.1.1

The expression

$$\begin{array}{l} \mathtt{let}\ r = \mathtt{ref}\ \lambda x(x)\ \mathtt{in} \\ \quad \mathtt{let}\ u = (r := \lambda x'(\mathtt{ref}\ !x'))\ \mathtt{in} \\ \qquad (!r)() \end{array}$$

has type $unit$.

$: \forall\alpha\,((\alpha \to \alpha)\,ref) \triangleq \sigma$

Example 3.1.1

The expression

$$\begin{array}{l}
\texttt{let}\; r = \texttt{ref}\;\lambda x(x)\;\texttt{in} \\
\quad \texttt{let}\; u = (r := \lambda x'(\texttt{ref}\;!x'))\;\texttt{in} \\
\quad\quad (!r)()
\end{array}$$

$: \alpha'\,ref \to \alpha'\,ref$

has type $unit$.

$: \forall \alpha\,((\alpha \to \alpha)\,\mathit{ref}) \triangleq \sigma$

Example 3.1.1

The expression

$$
\begin{array}{l}
\mathtt{let}\; r = \mathtt{ref}\; \lambda x(x)\; \mathtt{in} \\
\quad \mathtt{let}\; u = (r := \lambda x'(\mathtt{ref}\; !x'))\; \mathtt{in} \\
\qquad (!r)()
\end{array}
$$

$: \alpha'\mathit{ref} \to \alpha'\mathit{ref}$

has type unit.

$\sigma > (\alpha'\mathit{ref} \to \alpha'\mathit{ref})\,\mathit{ref}$

$: \forall \alpha ((\alpha \to \alpha)\, ref) \triangleq \sigma$

Example 3.1.1

The expression

$$
\begin{array}{l}
\mathtt{let}\; r = \mathtt{ref}\; \lambda x(x)\; \mathtt{in} \\
\quad \mathtt{let}\; u = (r := \lambda x'(\mathtt{ref}\, !x'))\; \mathtt{in} \\
\quad\quad (!r)()
\end{array}
$$

$: \alpha'\, ref \to \alpha'\, ref$

has type $unit$.

$\sigma > (\alpha'\, ref \to \alpha'\, ref)\, ref$

$\sigma > (unit \to unit)\, ref$

Midi-ML transition system (p34)

$$\boxed{\begin{array}{l} \langle M, s\rangle \to \langle M', s'\rangle \\ \langle M, s\rangle \to \mathrm{FAIL} \end{array}}$$

inductively defined on Slide 34 + Fig. 4

M, M' range over Midi-ML expressions

s, s' range over states = finite functions $\{x_1 \mapsto V_1, \ldots, x_n \mapsto V_n\}$ mapping variables to values

$$\boxed{V ::= x \mid \lambda x(M) \mid () \mid \mathrm{true} \mid \mathrm{false} \mid \mathrm{nil} \mid V :: V}$$

Midi-ML transition system (p34)

$$\boxed{\begin{array}{l} \langle M, s \rangle \to \langle M', s' \rangle \\ \langle M, s \rangle \to \text{FAIL} \end{array}}$$

M, M' range over Midi-ML expressions

s, s' range over states = finite functions $\{x_1 \mapsto V_1, \ldots, x_n \mapsto V_n\}$ mapping variables to values

insist that free vars of $M \subseteq \{x_1, \ldots, x_n\}$

Midi-ML transitions involving references

$$\langle !x, s\rangle \to \langle s(x), s\rangle \quad \text{if } x \in dom(s)$$

$$\langle !V, s\rangle \to FAIL \quad \text{if } V \text{ not a variable}$$

$$\langle x := V', s\rangle \to \langle (), s[x \mapsto V']\rangle$$

$$\langle V := V', s\rangle \to FAIL \quad \text{if } V \text{ not a variable}$$

$$\langle \mathtt{ref}\ V, s\rangle \to \langle x, s[x \mapsto V]\rangle \quad \text{if } x \notin dom(s)$$

where V ranges over *values*:

$$V ::= x \mid \lambda x(M) \mid () \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{nil} \mid V :: V$$

$$\left\langle \begin{array}{l} \mathbf{let}\ r = \mathbf{ref}\ \lambda x(x)\ \mathbf{in} \\ \mathbf{let}\ u = (r := \lambda x'(\mathbf{ref}\ !x'))\ \mathbf{in} \\ (!r)() \end{array} ,\ \{\} \right\rangle \longrightarrow^{*}$$

$$\left\langle \begin{array}{l} \mathbf{let}\ u = (r := \lambda x'(\mathbf{ref}\ !x'))\ \mathbf{in} \\ (!r)() \end{array} ,\ \{ r \mapsto \lambda x(x) \} \right\rangle$$

$$\left\langle \begin{array}{l} \text{let } r = \text{ref } \lambda x(x) \text{ in} \\ \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array} , \{\} \right\rangle \longrightarrow^*$$

$$\left\langle \begin{array}{l} \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array} , \{ r \mapsto \lambda x(x) \} \right\rangle \longrightarrow^*$$

$$\langle (!r)() , \{ r \mapsto \lambda x'(\text{ref } !x') \} \rangle$$

$$\left\langle \begin{array}{l} \text{let } r = \text{ref } \lambda x(x) \text{ in} \\ \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array} , \{\} \right\rangle \longrightarrow^{*}$$

$$\left\langle \begin{array}{l} \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)(\,) \end{array} , \{ r \mapsto \lambda x(x) \} \right\rangle \longrightarrow^{*}$$

$$\langle (!r)() , \{ r \mapsto \lambda x'(\text{ref } !x') \} \rangle \longrightarrow$$

$$\langle (\lambda x'(\text{ref } !x'))() , \{ r \mapsto \lambda x'(\text{ref } !x') \} \rangle$$

$$\left\langle \begin{array}{l} \text{let } r = \text{ref } \lambda x(x) \text{ in} \\ \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array}, \{\} \right\rangle \longrightarrow^{*}$$

$$\left\langle \begin{array}{l} \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array}, \{r \mapsto \lambda x(x)\} \right\rangle \longrightarrow^{*}$$

$$\langle (!r)(), \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle \longrightarrow$$

$$\langle {\color{red}(\lambda x'(\text{ref } !x'))()}, \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle \longrightarrow$$

$$\langle \text{ref } !(), \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle$$

$$\left\langle \begin{array}{l} \text{let } r = \text{ref } \lambda x(x) \text{ in} \\ \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array}, \{\} \right\rangle \longrightarrow^*$$

$$\left\langle \begin{array}{l} \text{let } u = (r := \lambda x'(\text{ref } !x')) \text{ in} \\ (!r)() \end{array}, \{r \mapsto \lambda x(x)\} \right\rangle \longrightarrow^*$$

$$\langle (!r)(), \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle \longrightarrow$$

$$\langle (\lambda x'(\text{ref } !x'))(), \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle \longrightarrow$$

$$\langle \text{ref } !(), \{r \mapsto \lambda x'(\text{ref } !x')\} \rangle \longrightarrow$$

FAIL

Value-restricted typing rule for `let`-expressions

$$(\mathbf{letv}) \quad \frac{\Gamma \vdash M_1 : \tau_1 \qquad \Gamma, x : \forall\, A\,(\tau_1) \vdash M_2 : \tau_2}{\Gamma \vdash \mathtt{let}\, x = M_1\, \mathtt{in}\, M_2 : \tau_2}\ (\dagger)$$

$(\dagger)$ provided $x \notin dom(\Gamma)$ and

$$A = \begin{cases} \{\,\} & \text{if } M_1 \text{ is not a value} \\ ftv(\tau_1) - ftv(\Gamma) & \text{if } M_1 \text{ is a value} \end{cases}$$

(Recall that values are given by
$V ::= x \mid \lambda x(M) \mid () \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{nil} \mid V :: V$.)

With (letv) rule this gets type scheme

$\sigma' \triangleq \forall \{\,\}((\alpha \to \alpha)\,ref)$

Example 3.1.1

The expression

$$\begin{array}{l} \texttt{let}\; r = \texttt{ref}\; \lambda x(x)\; \texttt{in} \\ \quad \texttt{let}\; u = (r := \lambda x'(\texttt{ref}\; !x'))\; \texttt{in} \\ \qquad (!r)() \end{array}$$

$(\alpha'\,ref \to \alpha'\,ref)\,ref \not\prec \sigma'$

has type $unit$.

$(unit \to unit)\,ref \not\prec \sigma'$

Type soundness for Midi-ML with the value restriction

For any closed Midi-ML expression M, if there is some type scheme σ for which

$$\vdash M : \sigma$$

is provable in the value-restricted type system (axioms and rules on Slides 16–18, 32 and 35), then *evaluation of M does not fail*, i.e. there is no sequence of transitions of the form

$$\langle M, \{\,\}\rangle \longrightarrow \cdots \longrightarrow FAIL$$

for the transition system $\longrightarrow$ defined in Figure 4 (where $\{\,\}$ denotes the empty state).

N.B. with (letv) rule, some Mini-ML expressions that were typeable become untypeable in Midi-ML, eg.

$$\text{let } f = (\lambda x(x))(\lambda y(y)) \text{ in } (f\ \text{true}) :: (f\ \text{nil})$$

(Can often use η-expansion or β-reduction to get around the problem.)

1990-style SML

$$\lambda x(x) : \forall \alpha\,(\alpha \to \alpha)$$

applicative type variable

$$\lambda x(\mathrm{ref}\,x) : \forall \_\alpha\,(\_\alpha \to \_\alpha\;\mathrm{ref})$$

imperative type variable

+ a restricted let typing rule (p36)