1ML WITH SPECIAL EFFECTS

Andreas Rossberg

1ML WITH SPECIAL EFFECTS

featuring: type abstraction and monads

Andreas Rossberg

1ML

1ML

- Unifies ML core & module languages

1ML

- Unifies ML core & module languages
- A language with first-class modules *and* H/M polymorphism

1ML

- Unifies ML core & module languages
- A language with first-class modules *and* H/M polymorphism
- A syntax for F_ω that you actually want to program in

```
type MAP (Key : EQ) =
{
  type key = Key.t
  type map a
  empty 'a : map a
  lookup 'a : key → map a → opt a
  add 'a : key → a → map a → map a
}

Map (Key : EQ) :> MAP Key =
{
  type key = Key.t
  type map a = key → opt a
  empty = fun x ⇒ none
  lookup x m = m x
  add x y m = fun z ⇒ if Key.eq z x then some y else m z
}
```

$$
\begin{array}{l}
\mathsf{MAP} = \textbf{fun}\ (\mathsf{Key} : \mathsf{EQ}) \Rightarrow \textbf{type} \\
\{ \\
\quad \mathsf{key} : (= \textbf{type}\ \mathsf{Key.t}) \\
\quad \mathsf{map} : (\mathsf{a} : \textbf{type}) \to \textbf{type} \\
\quad \mathsf{empty} : \text{'}(\mathsf{a} : \textbf{type}) \to \mathsf{map\ a} \\
\quad \mathsf{lookup} : \text{'}(\mathsf{a} : \textbf{type}) \to \mathsf{key} \to \mathsf{map\ a} \to \mathsf{opt\ a} \\
\quad \mathsf{add} : \text{'}(\mathsf{a} : \textbf{type}) \to \mathsf{key} \to \mathsf{a} \to \mathsf{map\ a} \to \mathsf{map\ a} \\
\}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{Map}\ (\mathsf{Key} : \mathsf{EQ}) :> \mathsf{MAP\ Key} = \\
\{ \\
\quad \mathsf{key} = \textbf{type}\ \mathsf{Key.t} \\
\quad \mathsf{map\ a} = \textbf{type}\ (\mathsf{key} \to \mathsf{opt\ a}) \\
\quad \mathsf{empty} = \textbf{fun}\ \mathsf{x} \Rightarrow \mathsf{none} \\
\quad \mathsf{lookup\ x\ m} = \mathsf{m\ x} \\
\quad \mathsf{add\ x\ y\ m} = \textbf{fun}\ \mathsf{z} \Rightarrow \textbf{if}\ \mathsf{Key.eq\ z\ x}\ \textbf{then}\ \mathsf{some\ y}\ \textbf{else}\ \mathsf{m\ z} \\
\}
\end{array}
$$

Semantics

SEMANTICS

- Based on F-ing Modules [Rossberg, Russo, Dreyer 2010/14]

SEMANTICS

- Based on F-ing Modules [Rossberg, Russo, Dreyer 2010/14]
- Defined by elaboration into System F_ω

SEMANTICS

- Based on F-ing Modules [Rossberg, Russo, Dreyer 2010/14]
- Defined by elaboration into System F_ω
- Main task of elaboration: manage quantifiers for abstract types

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\textbf{type}\ t = \mathsf{int};\ f : \mathsf{int} \to t\}$$

$$\{t : (= \mathsf{int}), f : \mathsf{int} \to \mathsf{int}\}$$

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ t = \mathsf{int};\ f : \mathsf{int} \to t\}$$

$$\{t : (= \mathsf{int}), f : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ t;\ f : \mathsf{int} \to t\}$$

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ \mathsf{t};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\exists \alpha. \{\mathsf{t} : (= \alpha), \mathsf{f} : \mathsf{int} \to \alpha\}$$

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

$$\text{where } (= \tau) \ := \ (\tau \to \tau)$$

$$\{\textbf{type}\ \mathsf{t};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\exists\alpha.\ \{\mathsf{t} : (= \alpha), \mathsf{f} : \mathsf{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ \mathsf{t};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\exists\alpha.\ \{\mathsf{t} : (= \alpha), \mathsf{f} : \mathsf{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$(\mathsf{X} : \{\textbf{type}\ \mathsf{t}\}) \to \{\mathsf{f} : \mathsf{X.t} \to \mathsf{int}\}$$

$$\{\textbf{type}\ t = \text{int};\ f : \text{int} \to t\}$$

$$\{t : (= \text{int}), f : \text{int} \to \text{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ t;\ f : \text{int} \to t\}$$

$$\exists \alpha.\ \{t : (= \alpha), f : \text{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$(X : \{\textbf{type}\ t\}) \to \{f : X.t \to \text{int}\}$$

$$\forall \alpha.\ \{t : (= \alpha)\} \to \{f : \alpha \to \text{int}\}$$

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ \mathsf{t};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\exists\alpha.\ \{\mathsf{t} : (= \alpha), \mathsf{f} : \mathsf{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$(\mathsf{X} : \{\textbf{type}\ \mathsf{t}\}) \to \{\mathsf{f} : \mathsf{X.t} \to \mathsf{int}\}$$

$$\forall\alpha.\ \{\mathsf{t} : (= \alpha)\} \to \{\mathsf{f} : \alpha \to \mathsf{int}\}$$

$$(\mathsf{X} : \{\textbf{type}\ \mathsf{t};\ \mathsf{v} : \mathsf{t}\}) \to \{\textbf{type}\ \mathsf{u};\ \mathsf{f} : \mathsf{u} \to \mathsf{X.t}\}$$

$$\{\textbf{type}\ t = \mathsf{int};\ f : \mathsf{int} \to t\}$$

$$\{t : (= \mathsf{int}), f : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ t;\ f : \mathsf{int} \to t\}$$

$$\exists\alpha.\ \{t : (= \alpha), f : \mathsf{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$(X : \{\textbf{type}\ t\}) \to \{f : X.t \to \mathsf{int}\}$$

$$\forall\alpha.\ \{t : (= \alpha)\} \to \{f : \alpha \to \mathsf{int}\}$$

$$(X : \{\textbf{type}\ t;\ v : t\}) \to \{\textbf{type}\ u;\ f : u \to X.t\}$$

$$\forall\alpha.\ \{t : (= \alpha), v : \alpha\} \to \exists\beta.\ \{u : (= \beta), f : \beta \to \alpha\}$$

$$\{\textbf{type}\ \mathsf{t} = \mathsf{int};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\{\mathsf{t} : (= \mathsf{int}), \mathsf{f} : \mathsf{int} \to \mathsf{int}\}$$

where $(= \tau) \ := \ (\tau \to \tau)$

$$\{\textbf{type}\ \mathsf{t};\ \mathsf{f} : \mathsf{int} \to \mathsf{t}\}$$

$$\exists\alpha.\ \{\mathsf{t} : (= \alpha), \mathsf{f} : \mathsf{int} \to \alpha\}$$

[cf. Mitchell, Plotkin 1988]

$$(\mathsf{X} : \{\textbf{type}\ \mathsf{t}\}) \to \{\mathsf{f} : \mathsf{X.t} \to \mathsf{int}\}$$

$$\forall\alpha.\ \{\mathsf{t} : (= \alpha)\} \to \{\mathsf{f} : \alpha \to \mathsf{int}\}$$

$$(\mathsf{X} : \{\textbf{type}\ \mathsf{t};\ \mathsf{v} : \mathsf{t}\}) \to \{\textbf{type}\ \mathsf{u};\ \mathsf{f} : \mathsf{u} \to \mathsf{X.t}\}$$

$$\forall\alpha.\ \{\mathsf{t} : (= \alpha), \mathsf{v} : \alpha\} \to \exists\beta.\ \{\mathsf{u} : (= \beta), \mathsf{f} : \beta \to \alpha\}$$

[cf. Russo 1998]

$$\frac{\Gamma \vdash e : \quad \{x : \sigma, \overline{x' : \sigma'}\}}{\Gamma \vdash e.x : \quad \sigma}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\}}{\Gamma \vdash e.x : \qquad \sigma}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\}}{\Gamma \vdash e.x : \exists\overline{\alpha}.\,\sigma}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\, \{x : \sigma, \overline{x' : \sigma'}\}}{\Gamma \vdash e.x : \exists\overline{\alpha}.\, \sigma}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\, \sigma}{\Gamma \vdash x = e : \exists\overline{\alpha}.\, \{x : \sigma\}}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\, \{x : \sigma, \overline{x' : \sigma'}\}}{\Gamma \vdash e.x : \exists\overline{\alpha}.\, \sigma}$$

$$\frac{\Gamma \vdash e : \exists\overline{\alpha}.\, \sigma}{\Gamma \vdash x = e : \exists\overline{\alpha}.\, \{x : \sigma\}}$$

$$\frac{\begin{array}{c}\Gamma \vdash b_1 : \exists\overline{\alpha}_1.\, \{\overline{x_1 : \sigma_1}\} \\ \Gamma, \overline{\alpha}_1, \overline{x_1 : \sigma_1} \vdash b_2 : \exists\overline{\alpha}_2.\, \{\overline{x_2 : \sigma_2}\} \qquad \overline{x}_1 \not\cap \overline{x}_2\end{array}}{\Gamma \vdash b_1 \,;\, b_2 : \exists\overline{\alpha}_1\overline{\alpha}_2.\, \{\overline{x_1 : \sigma_1}, \overline{x_2 : \sigma_2}\}}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}{\Gamma \vdash x = e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma\})}$$

$$\frac{\begin{array}{c}\Gamma \vdash b_1 : \mathrm{M}_{\overline{\kappa}_1}(\lambda\overline{\alpha}_1.\,\{\overline{x_1 : \sigma_1}\}) \\ \Gamma, \overline{\alpha}_1, \overline{x_1 : \sigma_1} \vdash b_2 : \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_2.\,\{\overline{x_2 : \sigma_2}\}) \qquad \overline{x}_1 \not\cap \overline{x}_2\end{array}}{\Gamma \vdash b_1\,;b_2 : \mathrm{M}_{\overline{\kappa}_1} \cdot \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_1\overline{\alpha}_2.\,\{\overline{x_1 : \sigma_1}, \overline{x_2 : \sigma_2}\})}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

$$\text{where } \mathrm{M}_{\overline{\kappa}} = \lambda c : (\overline{\kappa} \to \Omega).\ \exists\overline{\alpha{:}\kappa}.\ c\,\overline{\alpha}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}{\Gamma \vdash x = e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma\})}$$

$$\frac{\begin{array}{c}\Gamma \vdash b_1 : \mathrm{M}_{\overline{\kappa}_1}(\lambda\overline{\alpha}_1.\,\{\overline{x_1 : \sigma_1}\}) \\ \Gamma, \overline{\alpha}_1, \overline{x_1 : \sigma_1} \vdash b_2 : \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_2.\,\{\overline{x_2 : \sigma_2}\}) \qquad \overline{x}_1 \not\cap \overline{x}_2\end{array}}{\Gamma \vdash b_1\,;b_2 : \mathrm{M}_{\overline{\kappa}_1}\cdot\mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_1\overline{\alpha}_2.\,\{\overline{x_1 : \sigma_1}, \overline{x_2 : \sigma_2}\})}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

where $\mathrm{M}_{\overline{\kappa}} = \lambda c : (\overline{\kappa} \to \Omega).\ \exists\overline{\alpha{:}\kappa}.\ c\,\overline{\alpha}$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}{\Gamma \vdash x = e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma\})}$$

$$\frac{\begin{array}{c}\Gamma \vdash b_1 : \mathrm{M}_{\overline{\kappa}_1}(\lambda\overline{\alpha}_1.\,\{\overline{x_1 : \sigma_1}\}) \\ \Gamma, \overline{\alpha}_1, \overline{x_1 : \sigma_1} \vdash b_2 : \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_2.\,\{\overline{x_2 : \sigma_2}\}) \qquad \overline{x}_1 \not\cap \overline{x}_2\end{array}}{\Gamma \vdash b_1 ; b_2 : \mathrm{M}_{\overline{\kappa}_1} \cdot \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_1\overline{\alpha}_2.\,\{\overline{x_1 : \sigma_1}, \overline{x_2 : \sigma_2}\})}$$

where $\mathrm{M}_{\overline{\kappa}_1} \cdot \mathrm{M}_{\overline{\kappa}_2} = \mathrm{M}_{\overline{\kappa}_1\overline{\kappa}_2}$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

$$\text{where } \mathrm{M}_{\overline{\kappa}} = \lambda c : (\overline{\kappa} \to \Omega).\ \exists\overline{\alpha{:}\kappa}.\ c\,\overline{\alpha}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}{\Gamma \vdash x = e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma\})}$$

$$\frac{\Gamma \vdash b_1 : \mathrm{M}_{\overline{\kappa}_1}(\lambda\overline{\alpha}_1.\,\{\overline{x_1 : \sigma_1}\}) \qquad \Gamma, \overline{\alpha}_1, \overline{x_1 : \sigma_1} \vdash b_2 : \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_2.\,\{\overline{x_2 : \sigma_2}\}) \qquad \overline{x}_1 \not\cap \overline{x}_2}{\Gamma \vdash b_1\,;b_2 : \mathrm{M}_{\overline{\kappa}_1} \cdot \mathrm{M}_{\overline{\kappa}_2}(\lambda\overline{\alpha}_1\overline{\alpha}_2.\,\{\overline{x_1 : \sigma_1}, \overline{x_2 : \sigma_2}\})}$$

$$\text{where } \mathrm{M}_{\overline{\kappa}_1} \cdot \mathrm{M}_{\overline{\kappa}_2} = \mathrm{M}_{\overline{\kappa}_1\overline{\kappa}_2}$$

[cf. productoids, Tate 2013; polymonads Hicks, Bierman, Guts, Leijen, Swamy, 2014]

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\,\sigma)}$$

$$e.x \quad \rightsquigarrow \quad \textsf{unpack}\ \langle\overline{\alpha}, y\rangle = e\ \textsf{in pack}\ \langle\overline{\alpha}, y.x\rangle$$

$$\frac{\Gamma \vdash e : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\, \{x : \sigma, \overline{x' : \sigma'}\})}{\Gamma \vdash e.x : \mathrm{M}_{\overline{\kappa}}(\lambda\overline{\alpha}.\, \sigma)}$$

$$e.x \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\, y.x)\ e$$

$$e.x \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\, y.x)\ e$$

$$e.x \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\ y.x)\ e$$

$$x = e \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\{x = y\})\ e$$

$$e.x \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\, y.x)\ e$$

$$x = e \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\{x = y\})\ e$$

$$b_1\,;b_2 \quad \rightsquigarrow$$

$$e.x \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\, y.x)\ e$$

$$x = e \quad \rightsquigarrow \quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\{x = y\})\ e$$

$$\begin{aligned} b_1\,;b_2 \quad \rightsquigarrow \quad & \mathsf{join}_{M_{\overline{\kappa}_1}\, M_{\overline{\kappa}_2}}(\mathsf{map}_{M_{\overline{\kappa}_1}}\ (\lambda\overline{\alpha}_1.\lambda y_1. \\ & \qquad \mathsf{map}_{M_{\overline{\kappa}_2}}\ (\lambda\overline{\alpha}_2.\lambda y_2.\, y_1 \# y_2) \\ & \qquad\quad (\mathsf{let}\ \overline{x_1 = y_1.x_1}\ \mathsf{in}\ b_2)) \\ & \qquad b_1) \end{aligned}$$

$$e.x \quad\leadsto\quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\ y.x)\ e$$

$$x = e \quad\leadsto\quad \mathsf{map}_{M_{\overline{\kappa}}}\ (\lambda\overline{\alpha}.\lambda y.\{x = y\})\ e$$

$$\begin{aligned} b_1\,;b_2 \quad\leadsto\quad & [\ y_1\#y_2 \mid y_1 \leftarrow b_1, \\ & \qquad\qquad y_2 \leftarrow \mathsf{let}\ \overline{x_1 = y_1.x_1}\ \mathsf{in}\ b_2] \end{aligned}$$

Generativity

$$(\mathsf{x} : t_1) \rightarrow t_2$$

Generativity

$$(\mathsf{x} : t_1) \rightarrow t_2$$

$$\forall \overline{\alpha}_1 .\, \sigma_1 \rightarrow \exists \overline{\alpha}_2 .\, \sigma_2$$

GENERATIVITY

$$(\mathsf{x} : t_1) \to t_2$$

$$\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$$

$$\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$$

GENERATIVITY

$$A = \mathsf{Map\ Int}$$
$$B = \mathsf{Map\ Int}$$

GENERATIVITY

```
A = Map Int
B = Map Int

m    = A.add 1 "foo" (A.add 3 "bar" A.empty)
test = B.lookup 2 m
```

Generativity

```
A = Map Int
B = Map Int

m    = A.add 1 "foo" (A.add 3 "bar" A.empty)
test = B.lookup 2 m
```

GENERATIVITY

$$(\mathsf{x} : t_1) \to t_2$$

$$\forall \overline{\alpha}_1 .\, \sigma_1 \to \exists \overline{\alpha}_2 .\, \sigma_2$$

$$\forall \overline{\alpha}_1 .\, \sigma_1 \to M(\lambda \overline{\alpha}_2 .\, \sigma_2)$$

Generativity

$$(\mathsf{x} : t_1) \to t_2$$

$\forall \overline{\alpha}_1 . \, \sigma_1 \to \exists \overline{\alpha}_2 . \, \sigma_2$ "generative" functor (SML)

$$\forall \overline{\alpha}_1 . \, \sigma_1 \to M(\lambda \overline{\alpha}_2 . \, \sigma_2)$$

GENERATIVITY

$$(\mathsf{x} : t_1) \to t_2$$

$\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$ "generative" functor (SML)

$\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$

$\exists\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2$ "applicative" functor (Ocaml)

Generativity

$$(\mathsf{x} : t_1) \to t_2$$

$\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$ "generative" functor (SML)

$\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$

$\exists\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2$ "applicative" functor (Ocaml)

$M(\lambda\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2)$

```
type SYMBOL =
{
  type symbol
  insert : string → symbol
  lookup : symbol → string
  eq : symbol → symbol → bool
}

Symbol () :> SYMBOL =
{
  type symbol = int
  table = ref []
  insert s = table := s :: !table; length !table
  lookup n = nth !table (length !table - n)
  eq x y = (x == y)
}
```

[cf. Ahmed, Dreyer, Rossberg 2009]

GENERATIVITY

- We can hoist the abstraction effect out of a function, but only if it contains no other effects.

GENERATIVITY IN 1ML

$$(x : T_1) \leadsto T_2$$

impure function

$$(x : T_1) \rightarrow T_2$$

pure function

Generativity in 1ML

$(x : T_1) \rightsquigarrow T_2$ $\quad\leadsto\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$

impure function $\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$

$(x : T_1) \to T_2$ $\quad\leadsto\quad$ $\exists\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2$

pure function $\quad$ $M(\lambda\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2)$

EFFECTS

$$\eta ::= \mathsf{P} \mid \mathsf{I}$$

EFFECTS

$$\eta ::= \mathsf{P} \mid \mathsf{I}$$

$$\Gamma \vdash e :_{\eta} \exists \overline{\alpha}.\, \sigma$$

Effects

$$\eta ::= \mathsf{P} \mid \mathsf{I}$$

$$\Gamma \vdash e :_{\eta} \exists \overline{\alpha}.\, \sigma$$

$$\Gamma \vdash e :_{\eta} M(\lambda \overline{\alpha}.\, \sigma)$$

EFFECTS

$$\eta ::= \mathsf{P} \mid \mathsf{I}$$

$$\Gamma \vdash e :_\eta \exists \overline{\alpha}.\, \sigma$$

$$\Gamma \vdash e :_\eta M(\lambda \overline{\alpha}.\, \sigma)$$

$$\mathsf{P} \leq \mathsf{I}$$

EFFECT POLYMORPHISM

Why?

WHY?

- In 1ML, functions = functors, so *all* functions are effect-typed

WHY?

- In 1ML, functions = functors, so *all* functions are effect-typed
- Natural desire to refine the effect algebra

Why?

- In 1ML, functions = functors, so *all* functions are effect-typed
- Natural desire to refine the effect algebra
- ...but many notions of effect don't scale without effect polymorphism

WHY?

- In 1ML, functions = functors, so *all* functions are effect-typed
- Natural desire to refine the effect algebra
- ...but many notions of effect don't scale without effect polymorphism
- Plus, here it implies something new: generativity polymorphism

WHY?

- In 1ML, functions = functors, so *all* functions are effect-typed
- Natural desire to refine the effect algebra
- ...but many notions of effect don't scale without effect polymorphism
- Plus, here it implies something new: generativity polymorphism
- ...and that allows us to recover MacQueens notion of "true higher-order" modules [MacQueen, Tofte 1994]

EXTENDING THE LANGUAGE

$$(\text{types}) \quad t \ ::= \ \ldots \ | \ \textbf{effect} \ | \ (x : t) \rightarrow f / t$$

$$(\text{effects}) \ f \ ::= \textbf{pure} \ | \ \textbf{impure} \ | \ x \ | \ f , f$$

Extending the Language

$$(\text{types}) \quad t \ ::= \ \ldots \mid \textbf{effect} \mid (x : t) \rightarrow f/t$$

$$(\text{effects}) \ f \ ::= \ \textbf{pure} \mid \textbf{impure} \mid x \mid f, f$$

$$\begin{aligned} (x : t_1) \rightarrow t_2 \quad &:= \quad (x : t_1) \rightarrow \mathsf{pure}/t_1 \\ (x : t_1) \rightsquigarrow t_2 \quad &:= \quad (x : t_1) \rightarrow \mathsf{impure}/t_2 \end{aligned}$$

$$\mathsf{map} : (\mathsf{a} : \textbf{type}) \to (\mathsf{b} : \textbf{type}) \to (\mathsf{e} : \textbf{effect}) \to (\mathsf{a} \to \mathsf{e}/\mathsf{b}) \to \mathsf{list\ a} \to \mathsf{e}/\mathsf{list\ b}$$

$$\text{map} : (\text{a} : \textbf{type}) \rightarrow (\text{b} : \textbf{type}) \rightarrow (\text{e} : \textbf{effect}) \rightarrow (\text{a} \rightarrow \text{e/b}) \rightarrow \text{list a} \rightarrow \text{e/list b}$$

$$\begin{aligned}
&\text{compose } (\text{a} : \textbf{type})\ (\text{b} : \textbf{type})\ (\text{c} : \textbf{type})\ (\text{e1} : \textbf{effect})\ (\text{e2} : \textbf{effect}) \\
&\qquad\qquad (\text{f} : \text{b} \rightarrow \text{e2/c})\ (\text{g} : \text{a} \rightarrow \text{e1/b})\ (\text{x} : \text{a}) = \\
&\quad \text{f (g x)}
\end{aligned}$$

$$\text{map} : (\text{a} : \textbf{type}) \to (\text{b} : \textbf{type}) \to (\text{e} : \textbf{effect}) \to (\text{a} \to \text{e/b}) \to \text{list a} \to \text{e/list b}$$

$$\begin{aligned}&\text{compose } (\text{a} : \textbf{type})\ (\text{b} : \textbf{type})\ (\text{c} : \textbf{type})\ (\text{e1} : \textbf{effect})\ (\text{e2} : \textbf{effect})\\ &\qquad (\text{f} : \text{b} \to \text{e2/c})\ (\text{g} : \text{a} \to \text{e1/b})\ (\text{x} : \text{a}) =\\ &\quad \text{f (g x)}\end{aligned}$$

$$\begin{aligned}&\text{compose} : (\text{a} : \textbf{type}) \to (\text{b} : \textbf{type}) \to (\text{c} : \textbf{type}) \to (\text{e1} : \textbf{effect}) \to (\text{e2} : \textbf{effect}) \to\\ &\qquad (\text{b} \to \text{e1/c}) \to (\text{a} \to \text{e2/b}) \to (\text{a} \to \text{(e1,e2)/c})\end{aligned}$$

$\text{map} : (a : \textbf{type}) \rightarrow (b : \textbf{type}) \rightarrow (e : \textbf{effect}) \rightarrow (a \rightarrow e/b) \rightarrow \text{list } a \rightarrow e/\text{list } b$

$\text{compose } (a : \textbf{type})\ (b : \textbf{type})\ (c : \textbf{type})\ (e1 : \textbf{effect})\ (e2 : \textbf{effect})$
$\quad (f : b \rightarrow e2/c)\ (g : a \rightarrow e1/b)\ (x : a) =$
$\quad f\ (g\ x)$

$\text{compose} : (a : \textbf{type}) \rightarrow (b : \textbf{type}) \rightarrow (c : \textbf{type}) \rightarrow (e1 : \textbf{effect}) \rightarrow (e2 : \textbf{effect}) \rightarrow$
$\quad (b \rightarrow e1/c) \rightarrow (a \rightarrow e2/b) \rightarrow (a \rightarrow (e1,e2)/c)$

$\text{compose } f\ g\ x = f\ (g\ x)$

map : (a : **type**) → (b : **type**) → (e : **effect**) → (a → e/b) → list a → e/list b

compose (a : **type**) (b : **type**) (c : **type**) (e1 : **effect**) (e2 : **effect**)
(f : b → e2/c) (g : a → e1/b) (x : a) =
f (g x)

compose : (a : **type**) → (b : **type**) → (c : **type**) → (e1 : **effect**) → (e2 : **effect**) →
(b → e1/c) → (a → e2/b) → (a → (e1,e2)/c)

compose f g x = f (g x)

compose 'a 'b 'c 'e1 'e2 : (b → e1/c) → (a → e1/b) → (a → (e1,e2)/c)

Generativity Polymorphism

Generativity Polymorphism

$$\begin{aligned}
&\text{Const} = \textbf{fun}\ (a : \textbf{type}) \Rightarrow \text{int} \quad ;;\ :\ (a : \textbf{type}) \rightarrow (= \text{int})\\
&\text{Id} = \textbf{fun}\ (a : \textbf{type}) \Rightarrow a \quad ;;\ :\ (a : \textbf{type}) \rightarrow (= a)\\
&\text{Appl} = \text{Id} :> \textbf{type} \rightarrow \textbf{type}\\
&\text{Gen} = \text{Id} :> \textbf{type} \rightsquigarrow \textbf{type}
\end{aligned}$$

Generativity Polymorphism

$$\begin{aligned}
&\text{Const} = \textbf{fun}\ (\text{a} : \textbf{type}) \Rightarrow \text{int} \quad ;; : (\text{a} : \textbf{type}) \rightarrow (= \text{int}) \\
&\text{Id} \quad\;\; = \textbf{fun}\ (\text{a} : \textbf{type}) \Rightarrow \text{a} \quad\;\; ;; : (\text{a} : \textbf{type}) \rightarrow (= \text{a}) \\
&\text{Appl} \;\; = \text{Id} :> \textbf{type} \rightarrow \textbf{type} \\
&\text{Gen} \quad = \text{Id} :> \textbf{type} \leadsto \textbf{type}
\end{aligned}$$

$$\text{Apply}\ (\text{F} : \textbf{type} \rightarrow \textbf{type})\ (\text{a} : \textbf{type}) = \text{F a}$$

GENERATIVITY POLYMORPHISM

$$\begin{aligned}
\mathsf{Const} &= \textbf{fun}\ (\mathsf{a} : \textbf{type}) \Rightarrow \mathsf{int} \quad ;;\ :\ (\mathsf{a} : \textbf{type}) \to (= \mathsf{int}) \\
\mathsf{Id} &= \textbf{fun}\ (\mathsf{a} : \textbf{type}) \Rightarrow \mathsf{a} \quad ;;\ :\ (\mathsf{a} : \textbf{type}) \to (= \mathsf{a}) \\
\mathsf{Appl} &= \mathsf{Id} :> \textbf{type} \to \textbf{type} \\
\mathsf{Gen} &= \mathsf{Id} :> \textbf{type} \leadsto \textbf{type}
\end{aligned}$$

$$\mathsf{Apply}\ (\mathsf{F} : \textbf{type} \to \textbf{type})\ (\mathsf{a} : \textbf{type}) = \mathsf{F}\ \mathsf{a}$$

[cf. Kuan, MacQueen, 2009]

Generativity Polymorphism

$$
\begin{aligned}
\text{Const} &= \textbf{fun}\ (\text{a} : \textbf{type}) \Rightarrow \text{int} \quad ;; : (\text{a} : \textbf{type}) \to (= \text{int}) \\
\text{Id} &= \textbf{fun}\ (\text{a} : \textbf{type}) \Rightarrow \text{a} \quad ;; : (\text{a} : \textbf{type}) \to (= \text{a}) \\
\text{Appl} &= \text{Id} :> \textbf{type} \to \textbf{type} \\
\text{Gen} &= \text{Id} :> \textbf{type} \leadsto \textbf{type}
\end{aligned}
$$

$$
\text{Apply}\ (\text{e} : \textbf{effect})\ (\text{F} : \textbf{type} \to \text{e}/\textbf{type})\ (\text{a} : \textbf{type}) = \text{F a}
$$

[cf. Kuan, MacQueen, 2009]

Generativity Polymorphism

$$
\begin{aligned}
\mathsf{Const} &= \textbf{fun}\ (\mathsf{a} : \textbf{type}) \Rightarrow \mathsf{int} & ;;\ : (\mathsf{a} : \textbf{type}) \to (= \mathsf{int}) \\
\mathsf{Id} &= \textbf{fun}\ (\mathsf{a} : \textbf{type}) \Rightarrow \mathsf{a} & ;;\ : (\mathsf{a} : \textbf{type}) \to (= \mathsf{a}) \\
\mathsf{Appl} &= \mathsf{Id} :> \textbf{type} \to \textbf{type} \\
\mathsf{Gen} &= \mathsf{Id} :> \textbf{type} \leadsto \textbf{type}
\end{aligned}
$$

$$
\mathsf{Apply}\ (\mathsf{e} : \textbf{effect})\ (\mathsf{F} : \textbf{type} \to \mathsf{e}/\textbf{type})\ (\mathsf{a} : \textbf{type}) = \mathsf{F}\ \mathsf{a}
$$

[cf. Kuan, MacQueen, 2009]

$$
\begin{aligned}
\mathsf{t} &= \mathsf{Apply}\ \textbf{pure}\ \mathsf{Const}\ \mathsf{bool} & ;;\ = \mathsf{int} \\
\mathsf{u} &= \mathsf{Apply}\ \textbf{pure}\ \mathsf{Id}\ \mathsf{bool} & ;;\ = \mathsf{bool} \\
\mathsf{v} &= \mathsf{Apply}\ \textbf{pure}\ \mathsf{Appl}\ \mathsf{bool} & ;;\ = \mathsf{Apply}\ \textbf{pure}\ \mathsf{Appl}\ \mathsf{bool} \\
\mathsf{w} &= \mathsf{Apply}\ \textbf{impure}\ \mathsf{Gen}\ \mathsf{bool} & ;;\ \textit{fresh}
\end{aligned}
$$

Polymorphic Generativity

$(x:T_1) \rightsquigarrow T_2$ $\quad\leadsto\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$

impure function $\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$

$(x:T_1) \to T_2$ $\quad\leadsto\quad$ $\exists\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2$

pure function $\quad$ $M(\lambda\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2)$

Polymorphic Generativity

$(x : T_1) \rightsquigarrow T_2$ $\quad\leadsto\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to \exists\overline{\alpha}_2.\,\sigma_2$

impure function $\quad$ $\forall\overline{\alpha}_1.\,\sigma_1 \to M(\lambda\overline{\alpha}_2.\,\sigma_2)$

$(x : T_1) \to T_2$ $\quad\leadsto\quad$ $\exists\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2$

pure function $\quad$ $M(\lambda\overline{\alpha}_2.\,\forall\overline{\alpha}_1.\,\sigma_1 \to \sigma_2)$

$(x : T_1) \to f/T_2$

effect-polymorphic function

Polymorphic Generativity

$(x : T_1) \rightsquigarrow T_2 \qquad \leadsto \qquad \forall \overline{\alpha}_1 . \sigma_1 \to \exists \overline{\alpha}_2 . \sigma_2$

impure function $\qquad \forall \overline{\alpha}_1 . \sigma_1 \to M(\lambda \overline{\alpha}_2 . \sigma_2)$

$(x : T_1) \to T_2 \qquad \leadsto \qquad \exists \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to \sigma_2$

pure function $\qquad M(\lambda \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to \sigma_2)$

$(x : T_1) \to f / T_2 \qquad \leadsto \qquad \exists \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to (\sigma_2 + \exists \overline{\alpha}_2 . \sigma_2)$

effect-polymorphic function

Polymorphic Generativity

$(x : T_1) \leadsto T_2 \qquad \rightsquigarrow \qquad \forall \overline{\alpha}_1 . \sigma_1 \to \exists \overline{\alpha}_2 . \sigma_2$

impure function $\qquad \forall \overline{\alpha}_1 . \sigma_1 \to M(\lambda \overline{\alpha}_2 . \sigma_2)$

$(x : T_1) \to T_2 \qquad \rightsquigarrow \qquad \exists \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to \sigma_2$

pure function $\qquad M(\lambda \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to \sigma_2)$

$(x : T_1) \to f / T_2 \qquad \rightsquigarrow \qquad \exists \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to (\sigma_2 + \exists \overline{\alpha}_2 . \sigma_2)$

effect-polymorphic function $\qquad M(\lambda \overline{\alpha}_2 . \forall \overline{\alpha}_1 . \sigma_1 \to M'(\lambda \overline{\alpha}_2 . \sigma_2))$

The "Poly-Generativity Polymonad"

The "Poly-Generativity Polymonad"

$$M ::= \lambda c : (\overline{\kappa} \to \Omega).\Big(\sum_i \exists \overline{\alpha}_i.\, c\, \overline{\sigma}_i \Big)$$

THE "POLY-GENERATIVITY POLYMONAD"

$$M ::= \lambda c : (\overline{\kappa} \to \Omega).\left(\sum_i \exists \overline{\alpha}_i .\, c\, \overline{\sigma}_i \right)$$

$$\left(\lambda c. \sum_i \exists \overline{\alpha}_i .\, c\, \overline{\tau}_i \right) \cdot \left(\lambda c. \sum_j \exists \overline{\beta}_j .\, c\, \overline{\sigma}_j \right) \quad = \quad \lambda c. \sum_i \sum_j \exists \overline{\alpha}_i \overline{\beta}_j .\, c\, \overline{\tau}_i\, \overline{\sigma}_j$$

THE "POLY-GENERATIVITY POLYMONAD"

$$M ::= \lambda c : (\overline{\kappa} \to \Omega).\left(\sum_i \exists \overline{\alpha}_i . \, c \, \overline{\sigma}_i \right)$$

$\overline{\eta}$

$$\left(\lambda c. \sum_i \exists \overline{\alpha}_i . \, c \, \overline{\tau}_i \right) \cdot \left(\lambda c. \sum_j \exists \overline{\beta}_j . \, c \, \overline{\sigma}_j \right) \quad = \quad \lambda c. \sum_i \sum_j \exists \overline{\alpha}_i \overline{\beta}_j . \, c \, \overline{\tau}_i \, \overline{\sigma}_j$$

SUMMARY

SUMMARY

- Type abstraction can be viewed as an effect

Summary

- Type abstraction can be viewed as an effect
- Elaborating it into existential types forms a (poly)monad

SUMMARY

- Type abstraction can be viewed as an effect
- Elaborating it into existential types forms a (poly)monad
- An effect system allows multiple modes of type generativity

SUMMARY

- Type abstraction can be viewed as an effect
- Elaborating it into existential types forms a (poly)monad
- An effect system allows multiple modes of type generativity
- Effect polymorphism gives rise to generativity polymorphism

Summary

- Type abstraction can be viewed as an effect
- Elaborating it into existential types forms a (poly)monad
- An effect system allows multiple modes of type generativity
- Effect polymorphism gives rise to generativity polymorphism
- Monads are a great abstraction, even in places you don't expect!

THANK YOU.