30 April 20

Types for program modules.

1986 MacQueen: Using Dep Types to Express Modular Structure.
+ Moggi, Mitchell, Lillibridge, Stone, Petersen, Cruy, H.
+ Dreyer, Rossberg, Russo, Elsmar

Last time:

- isolated dep kinds derived from singleton kinds
- express type definitions express sharing specs by choice of representative

\[ t = t' = t'' \]
\[ t' :: T \]
\[ t :: s(t') \]
\[ t'' :: s(t'') \]
Key idea

Role of effects in module type systems.

* distinguish between module values and module computations

* "types rule" — all cross-module interaction is via their types = signatures

* modules values have fully explicit signatures — via self-application

\[
\text{\textbf{u} \quad \text{sig type } t \quad \ldots \quad \text{end}}
\]

\[
\text{\textbf{v} \quad \text{sig type } t = \text{v.t} \quad \ldots \quad \text{end}}
\]

\[
\text{fully transparent skeleton!}
\]
Reminder

Modal effects ( lax modality )

Expressions / values

M: commands / comp's encapsulation
e ::= \text{comp}\ (m)\quad\text{(comp)}

M ::= \text{ret}\ (e)

1 \text{ bind}\ (e_1; x. M_2)

↑ \text{true } \text{comp} (c)

OA ≡ Δ Δ A (1)

↑ persistence \{GC\}

fat state

Hierarchy + families are forms of dependency.
Structure of Types for Modules

"static dependency" so as to maintain the phase separation!

\[
\begin{align*}
\sigma &::= \\
&\quad [K] \quad \text{kind} \\
&\quad \{2\} \quad \text{the} \\
&\quad \{\sigma\} \quad \text{class} \\
&\quad \{x\} \quad \text{var} \\
&\quad \cdot \quad \text{constructor variable!} \\
\end{align*}
\]

module consisting a single constructor (i.e. a static module)

module consisting a single value (i.e. a dyn module)

encapsulated module comp.

hierarchy (static dep.)

families (static dep.)
Capsules = encapsulated module comp's (cf. comp (e) has comp^w (m))

\[ \text{comp} [0] (M) : \{ o \} \]
\[ \text{when } M : \sigma \]

comp. of a module of signature \( \sigma \),

M : \( \sigma \) "sealing"

\[ \text{bind [o2]} (V_i; X.u. M_2) = \text{continuation} \]

"twinning"
it is fresh! by α-conv!
uniquely associated with
the exact capsule in the
core arg of bind!
Notice

1. Must bind a capsule before using it.

```
structure S :> \Sigma = M
structure s : t + s : t...
structure s = (M :> \Sigma)
```

2. Get abstract for E's

abstract e is t with x :\Sigma 

package (M :> \Sigma) client
“Sealing/ascrption is a pro forma effect”!

\[ M : > \sigma \quad \text{sealed w/} \sigma \]

\[ \{ \sigma \} \]

Could be a value!

\[ \rightarrow \text{maximizes modularity} \]

b/c \( V : > \sigma \) can later become \( M : > \sigma \), and the client is none the wiser!
$p ::= x_1 p_1 \mid p_2 \{ \text{expr} \} \rightarrow 1 \ p(v)$

$X.A.B.C \rightarrow \overline{\text{static part}}$

$st \ (X.A.B.C) \rightarrow \overline{\text{static part}}$

$X.A.B.C \rightarrow \overline{\text{dynamic part}}$

Paths are generalized variables!

"NEUTRAL" vs "NORMAL"
$$V ::= P$$

- path
- "open" value

$$[c]$$
- constructor
- value

$$[v]$$
- capsule

$$M : > \sigma$$
- value

$$\langle v_1, v_2 \rangle$$
- value!

$$M ::= \text{let } v \mid \text{bind } (v_1; x . M)$$
- \text{bind of a core lang comp inside a module}
\[ \Delta + \sigma_i < \sigma_j, \quad \forall i, j \]

- derived from sub-inding

\[ [\Delta + [K_i]] < : [K_j] \]

- if \( \Delta + K_1 < : K_2 \)

(forget sharing /clefs)

\[ \text{Related:} \]

- singleton signatures

  (cf \( S_{K(c)} \))

\[ S_{S_0} (V) \]

- has a def. statt.

\[ S_{[K]} (V) = [S_{K} (st(V))] \]
\[ \Gamma \vdash P : u : \sigma_1 \times \rho_2 \]
\[ \rightarrow s^+(P \cdot 1) = \sigma_1 \]
\[ \Gamma \vdash P \cdot 1 : \sigma, \quad \neg \]
\[ \Gamma \vdash P \cdot 2 : \sigma \]
\[ \Gamma \vdash \text{Propagate} (\text{substitute the static part of } P \text{ into signature } \rho_2). \]
\[ \rightarrow \text{phase separation!} \]
\[ \Gamma \vdash P : u : \sigma_1 \rightarrow \sigma_2 \]
\[ \Gamma \vdash \nu_1 : \sigma_1 \]
\[ \Gamma \vdash \nu_1 \cdot \nu : [s^+(\nu_1) \cdot \nu] \sigma_2 \]
Moral
Dependent types express modular structure!

1. Dep kinds (no exts)
   sub-kind bug
   high singleton s

2. (statically) dep sigs
   phase separation / effects
   i. abstraction is done properly
   of CoqType mechanization
   of safety of Coq
   Twelf