Function Abstractions

- meaning of a defined function is a functional abstraction
- main semantic issue: how do free names (not parameters) get bound?

EXP$_1$ - Extended EXP

```
Expression ::=...
           | Identifier ( Actual-Parameter )

Declaration::=...
           | fun Identifier (Formal-Parameter)
                  = Expression

Formal-Parameter::=Identifier : Type-denoter

Actual-Parameter::=Expression
```

‘‘Actual-Argument’’ might be a better syntactic category name

- Language features
  - — purely functional
  - — no expression side-effects in EXP$_1$
  - — static binding of free names in declarations
  - — One-argument (unary) functions

EXP_1 **Semantics**

- Specify semantic domains
  - Functions take and return integers:
    Argument = Integer
    Function = Argument → Integer
  - now names can also denote functions:
    Bindable = *function* Function + *integer* Integer
- Specify Contextual Constraints
  - also called "Static Semantics"
  - a called function must be declared
  - actual and formal parameters must agree in type
- Specify semantic functions
  - an expression, given an env, yields a value
    evaluate : `Expression` → *(*Environ → Integer *)*
  - a declaration, given an env, yields a new env
    elaborate
    : `Declaration` → *(*Environ → Environ *)*

`Expression` Semantics

- Semantics of function *calls*:
    - find the function (entered into the environment at declaration time)
    - evaluate actual argument in environment of *call*
    - apply the found function to the actual's value
    - syntactic metavariables are *AP* : `Actual-Parameter` , *I* : `Identifier`

$$\begin{aligned} evaluate[\![\, I\,\mathbf{(}\,AP\,\mathbf{)}\,]\!]\ env\ =\ & \textbf{let}\ \ function\ func = find(env, I)\ \ \textbf{in}\\ & \textbf{let}\ \ arg = evaluate[\![AP]\!]\ env\ \ \textbf{in}\\ & func\ arg \end{aligned}$$

- *func* is a "function closure" (implemented as a code + environment pair)

`Expression` Semantics (cont'd)

- Semantics of function *definitions*:

  — constuct a function *abstraction* that
    - binds formal parm to λ-variable
    - evaluates body in current env overlain by formal binding
    - current (def.) env resolves free names (all ≠ *FP*)
    - (implemented by a *function closure*: code and environment pointers)

  — binds resulting abstraction to name *I*

  — syntactic metavariable *FP* : `Formal-Parameter`

$$
\begin{aligned}
&\mathit{elaborate}[\![\,\mathtt{fun}\ I\,\mathtt{(}\,FP\,\mathtt{)}\ =\ E\,]\!]\ \mathit{env}\ = \\
&\qquad\qquad \mathbf{let}\ \mathit{func}\ =\ \lambda x\ .\ \mathit{evaluate}[\![E]\!]\ (\mathit{env}[FP\ \mapsto\ x]) \\
&\qquad\qquad \mathbf{in} \\
&\qquad\qquad \mathit{bind}(I,\ \mathit{function\ func})
\end{aligned}
$$

— *func* : Argument → Integer

Argument = Integer

Example: Function Definition and Use

- an expression E in a surrounding environment $u[\mathtt{c} \mapsto 7]$:

$$\begin{aligned}
&\mathit{evaluate}\llbracket E \rrbracket\,(u[\mathtt{c} \mapsto 7]) \\
&= \mathit{evaluate}\llbracket \mathbf{let\ fun}\ \mathtt{Bump(n:}\ \mathbf{int}\ \mathtt{)} = \mathtt{n} + \mathtt{c}\ \mathbf{in} \\
&\qquad\qquad (\mathbf{let\ val}\ \mathtt{c} = \mathtt{3}\,\mathbf{in}\ \mathtt{6} * \mathtt{Bump(c))}\rrbracket\,(u[\mathtt{c} \mapsto 7]) \\
&= \mathit{evaluate}\llbracket \mathbf{let\ val}\ \mathtt{c} = \mathtt{3}\,\mathbf{in}\ \mathtt{6} * \mathtt{Bump(c)}\rrbracket\,(u[\mathtt{c} \mapsto 7, \mathtt{Bump} \mapsto f])
\end{aligned}$$

where

$f = \lambda a.\mathit{evaluate}\llbracket \mathtt{n} + \mathtt{c} \rrbracket\,(u[\mathtt{c} \mapsto 7, \mathtt{n} \mapsto a]) = \lambda a.a + 7$ — statically bound!

So unwinding the inner **let** expression we get:

$$\begin{aligned}
&\mathit{evaluate}\llbracket E \rrbracket\,(u[\mathtt{c} \mapsto 7]) \\
&= \mathit{evaluate}\llbracket \mathtt{6} * \mathtt{Bump(c)} \rrbracket\,(u[\mathtt{c} \mapsto 7, \mathtt{Bump} \mapsto f, \mathtt{c} \mapsto 3]) \\
&= \mathit{evaluate}\llbracket \mathtt{6} * \mathtt{Bump(c)} \rrbracket\,(u[\mathtt{Bump} \mapsto f, \mathtt{c} \mapsto 3]) \\
&= 6 \cdot f(3) = 6 \cdot ((\lambda a.a + 7)3) = 6 \cdot (3 + 7) = 60
\end{aligned}$$

- Questions:
  - — What happens if *dynamic* binding is used?
  - — What then is the *f* that is bound to `Bump`?

Static vs. Dynamic Binding

Static:

— Function = Argument → Value

— environment of call used only to look up function *name* and to evaluate *actual*

— declaration environment is frozen with function object (‘‘closed functional form’’, hence the term ‘‘closure’’)

Dynamic:

— Function = Environ → Argument → Value

— environment of call used to provide environment for *all* names (except *FP*)

— function *name* bound to an object that needs both an *env* and *arg*

$$
\begin{aligned}
&\mathit{evaluate}[\![I \,(\, AP \,)\,]\!]\ \mathit{env} = \\
&\qquad \textbf{let}\ \ \mathit{function\ func} = \mathit{find(env, I)}\ \ \textbf{in} \\
&\qquad \textbf{let}\ \ \mathit{arg} = \mathit{evaluate}[\![AP]\!]\ \mathit{env}\ \ \textbf{in} \\
&\qquad \mathit{func\ env\ arg} \quad \text{— note the } \mathit{env} \text{ argument}
\end{aligned}
$$

Static vs. Dynamic Binding (cont'd)

$$
\begin{aligned}
&\mathit{elaborate}[\![\texttt{fun}\ I\,\texttt{(}FP\texttt{)} = E]\!]\ \mathit{env} = \\
&\qquad \textbf{let}\ \mathit{func} = \lambda\rho\,.\,\lambda x\,.\,\mathit{evaluate}[\![E]\!]\,(\rho[FP \mapsto x]) \\
&\qquad\ \textbf{in}\quad \text{— note the } \lambda\rho \text{ abstraction} \\
&\qquad \mathit{bind}(I,\ \mathit{function}\ \mathit{func})
\end{aligned}
$$

- Re-compute the Example above assuming dynamic binding?
- What is the function *f*?