Procedure Abstractions

- a procedure value is a map from arguments to a store transformation
- as before: how do free names (not parameters) get bound?

Syntax of **IMP** $_1$

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

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

Declaration::=...
         | func Identifier (Formal-Parameter)
              ~ Expression
         | proc Identifier (Formal-Parameter)
              ~ Command

Formal-Parameter::=
                 const Identifier : Type-denoter

Actual-Parameter::=Expression
```

- Language features
  - imperative
  - no expression or *function* call side-effects in IMP_1
  - static binding of free names in declarations
  - single-argument (unary) functions and procedures
  - arguments are "call-by-constant"
    - formals are initialized constants in the subroutine body
    - *no variables* allocated $\Rightarrow$ *not* call-by-value

$\mathbf{IMP_1}$ Semantics

- Specify semantic domains

  Value = *truth−value* Boolean + *integer* Integer
  —first class values

  *(*Value ≡ Expressible *)*

  Argument = Value

  Function = Argument → Store → Value

  Procedure = Argument → Store → Store

  — no Environ argument ⇒ static binding

  — requires Store argument to look up variable contents

  Bindable = *value* Value + *variable* Location
  + *function* Function + *procedure* Procedure

  — names can denote a constant, a variable (≡ reference), a function, or a procedure

- Specify Contextual Constraints

  — actual/formal agreement in number and type

  — prior declaration of called subroutine

- Specify semantic functions

  all signatures exactly as in $\mathbf{IMP_0}$

  evaluate : `Expression` → *(*Environ → Store → Value *)*
  execute : `Command` → *(*Environ → Store → Store *)*

  elaborate : `Declaration` →

Expression Semantics

- Semantics of *function* calls:
  - *env* and *sto* instead of just *env*
  - function *func* needs both *arg* and *sto*
  - otherwise like $\mathbf{EXP_1}$
  - result is a Value

evaluate $[\![I\ (\ AP\ )]\!]$ *env sto* =
let *function func = find(env, I)* **in**
let *arg = evaluate* $[\![AP]\!]$ *env sto* **in**
func arg sto

  - Note *func :* Argument $\rightarrow$ Store $\rightarrow$ Value

Command Semantics

- Semantics of *procedure* calls:
  - — exactly analogous to function calls
  - — result is an altered Store

$$execute[\![I\ (\ AP\ )\ ]\!]\ env\ sto\ =$$
$$\textbf{let}\ \ procedure\ proc = find(env, I)\ \ \textbf{in}$$
$$\textbf{let}\ \ arg = evaluate[\![AP]\!]\ env\ sto\ \ \textbf{in}$$
$$proc\ arg\ sto$$

  - — Note *proc :* Argument $\rightarrow$ Store $\rightarrow$ Store

`Declaration` Semantics

- Semantics of *function* declaration: analogous to **EXP 1**
  - — constucts a function *abstraction* that
    - binds formal parm to λ-variable
    - evaluates body in *env* of definition, overlain by formal binding
    - definition *env* resolves free names (all names except *FP*)
  - — binds resulting abstraction to name *I*
- Differences from functional language **EXP 1**
  - — *sto* is an argument to *elaborate* (for variable allocation)
    - note *sto* is unaltered by function declaration; in effect this is a constant declaration
  - — pair of *(env, sto)* returned
  - — *func* abstraction requires both λ*x*, λσ— an *arg* and *sto* at call time

elaborate⟦ **func** *I* **(** *FP* **)** ~ *E*⟧ *env sto* =
let *func* = λ*x* . λσ . *evaluate*⟦*E*⟧ (*env*[*FP* ↦ *x*]) σ
in
(*bind*(*I*, **function** *func*), *sto*)

`Declaration` Semantics (cont'd)

- Semantics of *procedure* declaration
  - — exactly analogous to above
  - — also a "constant" declaration (store unaltered by declaration)

$$elaborate[\![\textbf{proc}\ I(FP) \sim C]\!]\ env\ sto =$$
$$\textbf{let}\ proc = \lambda x\ .\ \lambda\sigma\ .\ execute[\![C]\!]\ (env[FP \mapsto x])\ \sigma$$
$$\textbf{in}$$
$$(\ bind(I, procedure\ proc),\ sto\ )$$

- Procedures are like commands; functions are like expressions:

  proc : Argument $\rightarrow$ Store $\rightarrow$ Store
  - — ''parameterized command'' with encapsulated environment

  func : Argument $\rightarrow$ Store $\rightarrow$ Value
  - — ''parameterized expression'' with encapsulated environment