Parameter Denotations

- What does a parameter *mean*?
  - An ‘‘actual parameter’’ given *env sto*, yields an Argument
  - Meaning of a formal parameter depends on the ‘‘parameter passing’’ mechanism
- Kinds of Parameter Mechanisms (partial taxonomy)
  - *Definitional: FP* bound to *actual* at call time (no allocation)
    - Constant parameters
    - Variable parameters
    - Procedural parameters
    - Functional parameters
  - *Copy Mechanisms: FP* is an allocated local variable *initialized* at call time
    - Value (copy-in) parameters
    - Result (copy-out) parameters
    - Copy-in/Copy-out parameters

$\mathbf{IMP_{2d}}$ = $\mathbf{IMP_1}$ + Definitional Parameters

- Features
  - — argument can be first-class value or reference (Location , *l*-value).
  - — latter are often called **`var`** parameters (*implemented* by copying references—this is *call-by-reference*)
  - — **`const`** parameters are *not* call-by-value: nothing is copied, *no local variable exists*. Equivalent to a **`const`** declaration at start of subroutine body.
  - — expressions still only evaluate to first-class values (no *l*-valued expressions in this language)
- Syntax of $\mathbf{IMP_{2d}}$

```
Formal-Parameter::=
          const Identifier : Type-denoter
        | var Identifier : Type-denoter

Actual-Parameter::=Expression
        | var Identifier
```

$\mathbf{IMP_{2d}}$ Semantics

- Specify semantic domains
  Value $=$ $\cdots$
  Argument $=$ *value* Value $+$ *variable* Location
  Function $=$ $\cdots$
  Procedure $=$ $\cdots$
  Bindable $=$ *value* Value $+$ *variable* Location
  $+$ *function* Function $+$ *procedure* Procedure
  — superset of Argument

- Specify Contextual Constraints

  — **`proc`** p**`(var`** x **`:`** $\cdots$ **`)`** declaration $\Rightarrow$
  p**`(var`** I**`)`** call

  — **`proc`** p**`(const`** c **`:`** $\cdots$ **`)`** declaration $\Rightarrow$ p**`(`**E**`)`**
  call

- Specify semantic functions

  — as for $\mathbf{IMP_1}$ with edits as follows

Semantic Function for `Actual-Parameter`

- `Actual-Parameter` like an `Expression`, but can yield Value (*r*-value) or Location (*l*-value)

  Argument = *value* Value + *variable* Location

  give−argument : `Actual-Parameter` → (Environ → Store → Argument)

  $give\text{-}argument[\![\textbf{var}\ I]\!]\ env\ sto\ =$
  let *variable loc = find(env, I)* **in**
  variable loc

  $give\text{-}argument[\![E]\!]\ env\ sto\ =$
  $value(evaluate[\![E]\!]\ env\ sto\ )$
  — tag Value as Argument

`Expression` Semantics

$evaluate[\![I\,(\,AP\,)\,]\!]\ env\ sto\ =$
let *function func = find(env, I)* **in**
let $arg = give\text{-}argument[\![AP]\!]\ env\ sto$ **in**
func arg sto

`Command` Semantics

$execute[\![I\,(\,AP\,)\,]\!]\ env\ sto\ =$
let *procedure proc = find(env, I)* **in**
let $arg = give\text{-}argument[\![AP]\!]\ env\ sto$ **in**
proc arg sto

Semantic Function for `Formal-Parameter`

- What does a definitional *FP* mean?
- Formal Parameters denote *(*Argument $\rightarrow$ Environ *)* maps
  - they take an argument and return a *name binding*
  - semantic rule for **`func`**/**`proc`** declaration overlays this elementary binding on the environment of definition (see below)

bind−parameter : `Formal-Parameter` $\rightarrow$
*(*Argument $\rightarrow$ Environ *)*

bind−parameter⟦**`var`** *I* **`:`** *T*⟧ *(variable loc)* =
bind(I, variable loc)

bind−parameter⟦**`const`** *I* **`:`** *T*⟧ *(value val)* =
bind(I, value val)

Declaration Semantics

- *function* declaration: generalizes prior approach as follows:

  EXP$_1$:

  let $func = \lambda x\,.\,\lambda\sigma.evaluate[\![E]\!]$
  $overlay([FP \mapsto x], env)\ \sigma$

  EXP$_{2d}$:

  let $func = \lambda x\,.\,\lambda\sigma\,.\,evaluate[\![E]\!]$
  $overlay(bind\text{-}parameter[\![FP]\!]\ x, env)\ \sigma$

  $elaborate[\![\texttt{func}\ I(FP) \sim E]\!]\ env\ sto =$
  let $func = \lambda x\,.\,\lambda\sigma\,.\,evaluate[\![E]\!]$
  $overlay(bind\text{-}parameter[\![FP]\!]\ x, env)\ \sigma$
  in
  $(\ bind(I, function\ func),\ sto\ )$

- *procedure* declaration analogous

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