The Environment

- syntactic domain `Identifier`
- semantic domain Bindable $\equiv$ Denotable of "nameable values"
- Bindable contains $\perp$
- identifiers can be *unbound*
- environments exist even if no notion of store or assignment
- *env*: a mapping from identifiers (names) to values

Environ $=$ `Identifier` $\rightarrow$ (*bound* Bindable $+$ *unbound*)

injection maps for tagged union:
bound : Bindable $\rightarrow$ (*bound* Bindable $+$ *unbound*)
unbound : (*bound* Bindable $+$ *unbound*)

Auxiliary Functions

$empty-environ$: Environ
$bind$: Identifier $\times$ Bindable $\rightarrow$ Environ
$overlay$: Environ $\times$ Environ $\rightarrow$ Environ
$find$: Environ $\times$ Identifier $\rightarrow$ Bindable

$$empty-environ = \lambda I \,.\, unbound$$

$$bind\ (I, bdble) = \lambda J \,.\, \textbf{if}\ I = J\ \textbf{then}\ bound\ bdble\ \textbf{else}\ unbound$$

$$overlay(env_{new}, env) = \lambda I \,.\, \textbf{if}\ env_{new}(I) \neq unbound\ \textbf{then}\ env_{new}(I)\ \textbf{else}\ env(I)$$

— second argument *env* overridden by first argument env_{new}

$$find(env, I) = \textbf{let}\ bound-value(bound\ bdble) = bdble,\ bound-value(unbound) = \perp\ \textbf{in}\ bound-value(env(I))$$

Abstract Grammar - EXP

```
    Program::=Expression

 Expression::=Numeral
             | Expression + Expression
               ...
             | Identifier
             | let Declaration in Expression

Declaration::=val Identifier = Expression
```

- simple expression language EXP (Watt)

EXP Semantics

- Specify semantic domains
  - — only integers are denotable in EXP
  - — Bindable = Integer
- Specify semantic functions
  - — expressions are *evaluated* in an environment
  - — produce an *expressible value*

  evaluate : `Expression` $\rightarrow$ *(*Environ $\rightarrow$ Integer *)*
  - — syntactic metavariables are N *:* `Numeral`, E *:* `Expression`, I *:* `Identifier`

$$evaluate[\![N]\!]\ env = valuation\ N$$

$$evaluate[\![E_1 + E_2]\!]\ env = evaluate[\![E_1]\!]\ env + evaluate[\![E_2]\!]\ env$$

$$evaluate[\![E_1 * E_2]\!]\ env = evaluate[\![E_1]\!]\ env \cdot evaluate[\![E_2]\!]\ env$$

EXP Semantics (cont'd)

$evaluate[\![I]\!]\ env\ =$
$\quad find(env, I)$

$evaluate[\![\texttt{let}\ D\ \texttt{in}\ E]\!]\ env\ =$
$\quad \textbf{let}\ \ env_1\ =\ elaborate[\![D]\!]\ env$
$\quad \textbf{in}$
$\quad evaluate[\![E]\!]\ (overlay(env_1, env))$

EXP Semantics (cont'd)

— declarations are *elaborated* in an environment

— produce another environment

elaborate :
`Declaration` $\rightarrow$ *(*Environ $\rightarrow$ Environ *)*

— only one kind of declaration: "const"

— syntactic metavariables are *E :* `Expression`, *I :* `Identifier`, *D :* `Declaration`.

elaborate⟦**`val`** *I* **`=`** *E*⟧ *env* =
bind (I, evaluate⟦*E*⟧ *env)*

Example

- evaluate the expression
  let val x=3**in** (**let val** x=x*2**in** x * x) * x

  in the surrounding environment e_1:

  evaluate⟦**let val** x=3**in** (**let val** x=x*2**in** x * x) * x⟧ e_1

  = **let** e_2 = *elaborate*⟦**val** x=3⟧ e_1
  in*evaluate*⟦(**let val** x=x*2**in** x * x) * x⟧ *(overlay*(e_2, e_1))

  = **let** e_2 = *bind* (x, *evaluate*⟦3⟧ e_1)
  in*evaluate*⟦(**let val** x=x*2**in** x * x) * x⟧ *(overlay*(e_2, e_1))
  . . .

  = **let** e_2 = *bind* (x, *3*)
  in*evaluate*⟦(**let val** x=x*2**in** x * x) * x⟧ *(overlay*(e_2, e_1))

  = *evaluate*⟦(**let val** x=x*2**in** x * x) * x⟧
  (overlay(e_1[x ↦ *3*], e_1))

  = *evaluate*⟦(**let val** x=x*2**in** x * x) * x⟧ e_1[x ↦ *3*]

  = *evaluate*⟦**let val** x=x*2**in** x * x⟧ e_1[x ↦ *3*]
  · *evaluate*⟦x⟧ e_1[x ↦ *3*]

  . . .
  = *evaluate*⟦x * x⟧ e_1[x ↦ *6*]·*find*(e_1[x ↦ *3*], x)
  . . .
  = *36·3 = 108*