The Store

• primitive domains Location, Storable
• domains contain ⊥
• locations can be unallocated (unused)
• locations can be allocated but undefined
• store: a mapping from locations (refs) to values

Store = Location → (stored Storable + undefined + unused)

injection maps for tagged union:
sstored : Storable → (stored Storable + undefined + unused)
unde fined : (stored Storable + undefined + unused)
unused : (stored Storable + undefined + unused)
Update Operator

- \([a \mapsto b]\) is an operator that takes a function \(f\) to another function \(f[a \mapsto b]\). It is written postfix.

- **Definition**: Let \(f: X \to Y\) and let \(a, b\) be any values. The function \(f[a \mapsto b]\) :
  \(X \cup \{a\} \to Y \cup \{b\}\) is defined by:

  \[
  (f[a \mapsto b])(x) = \begin{cases} 
    b & \text{if } x = a \\
    f(x) & \text{if } x \neq a
  \end{cases}
  \]

- We can extend this notation to multiple successive changes as follows:

  \[
  f[a_1 \mapsto b_1, a_2 \mapsto b_2] = (f[a_1 \mapsto b_1])[a_2 \mapsto b_2]
  \]

- **Example**: Semantics of assignment. Suppose the identifier \(x\) is bound to the location \(l\). Then executing the assignment \(x := e\) has the effect of changing memory:

  \[
  \text{execute}[\{x := e\}]\ sto = sto[l \mapsto \text{eval}[\{e\}]\ sto]
  \]
Auxiliary Functions

empty-store : Store
allocate : Store → Store × Location
deallocate : Store × Location → Store
update : Store × Location × Storable → Store
fetch : Store × Location → Storable

empty-store = λ loc . unused

allocate sto =
  let loc = any-unused-location(sto) in
  ( sto[loc ↦ undefined] , loc )

deallocate(sto, loc) =
  sto[loc ↦ unused]

update(sto, loc, stble) =
  sto[loc ↦ stored stble]

fetch(sto, loc) =
  let stored-value(stored stble) = stble
  stored-value(undefined) = ⊥
  stored-value(unused) = ⊥
in
  stored-value(sto(loc))
Example

A simple language with expressions and assignment

Syntax

Command ::= Identifier := Expression
  | Command ; Command

Expression ::= Expression + Expression
  | Numeral
  | Identifier

Semantics

semantic function binds names to locations:
location : Identifier → Location
semantic map:
execute : Command → Store → Store
execute[I := E]sto =
  let int = evaluate[E]sto
  in
  update(sto, location I, int)

execute[C₁ ; C₂]sto =
  let sto’ = execute[C₁]sto
  in
  execute[C₂]sto’