```
// Syntactic domains (example, adjust as needed)

type ide = string

type boolean =
    | True
    | False

type exp =
    | Eint of int
    | Eplus of (exp * exp)
    | Eminus of (exp * exp)
    | Eide of ide
    | Ebool of boolean
    | Eeql of (exp * exp)
    | Eleq of (exp * exp)
    | Enot of exp
    | Eand of (exp * exp)
    | Eor of (exp * exp)
    | Eifthenelse of (exp * exp * exp)
    | Eapp of exp * exp
    | Efun of ide * exp
    | Elet of (ide * exp * exp)

type com =
    | Cassign of ide * exp
    | Cvar of ide * exp
    | Cconst of ide * exp
    | Cifthenelse of exp * pseq * pseq
    | Cwhile of exp *pseq
    | CdoNTimes of exp * pseq

and pseq =
    | Pseq of com * pseq
    | Pend

type prog = Prog of pseq * exp

// Error handling (example, adjust as needed)

let unbound_identifier_error ide =
    failwith (sprintf "unbound identifier %s" ide)

let negative_natural_number_error () =
    failwith "natural numbers must be positive or zero"

let type_error () = failwith "type error"

let memory_error () =
    failwith "access to a location that is not available"
```

```
let not_a_location_error i =
    failwith (sprintf "not a location: %s" i)

// Semantic domains (example, adjust as needed)

type eval =
    | Int of int
    | Bool of bool
    | Fun of (ide * env * exp)

and loc = int

and mval = eval

and store = int * (loc -> mval) // The first element is the first "empty" location

and dval =
    | E of eval
    | L of loc

and env = ide -> dval

let empty_store = (0, (fun l -> memory_error ()))

let apply_store st l = (snd st) l

let allocate: store -> loc * store =
    fun st ->
        let l = fst st in
        let l1 = l + 1 in
        let st1 = (l1, snd st) in
        (l, st1)

let update: store -> loc -> mval -> store =
    fun st l mv ->
        match st with
        | (maxloc, fn) -> let fn1 l1 = if l = l1 then mv else fn l1 in (maxloc,
fn1)

let empty_env = fun v -> unbound_identifier_error v

let bind e v r = fun v1 -> if v1 = v then r else e v1

let apply_env e v = e v
```