

 SAMFYB include new lecture

e182040 3 hours ago

1 contributor

115 lines (89 sloc) 2.38 KB

Lecture 23 - Imperative Programming

17 April 2018

"Cells" & Typing of Cells

How do we represent a value `10` contained in a cell?

```
10 : int ref

(* initialize *) val c = ref 10
(* ref e : t ref iff e : t *)

(* read *) val b = !c
(* !e : t iff e : t ref *)

(* write *) val _ = c := 5
(* (e1 := e2) : unit
 * iff e1 : t ref, e2 : t
 *)
```

Box Reference

```
val d = c (* d refers to the exact same box as c *)

val _ = d := 17
val 17 = !c
```

Imperative Programming in SML

```
(* deposit : int ref -> int -> unit *)
fun deposit (a : int ref) (d : int) : unit = a := !a + d

fun withdraw (a : int ref) (w : int) : unit = a := !a - w

val account = ref 200
(deposit account 100; withdraw account 50)
```

```

(* (e1; e2; ...; en) : t iff en : t
   * - reduces to vn iff all ei reduces to some value AND en => vn
   *)

```

Parallelism of Imperative Operations

	Persistent Data	Ephemeral Data
Parallelism	FP	Concurrency
Sequential	w. even "benign mutation"	O.K.

Examples

```

type graph = int -> int list
val G : graph = fn 1 => [2, 3]
                  | 2 => [1, 3]
                  | 3 => [4]
                  | _ => []
(* This is a directed graph. *)

(* reachable : graph -> (int * int) -> bool
   * reachable g (x, y) returns true iff y is reachable from x via edges in g
   *)
(* Recall: *)
List.exists : ('a -> bool) -> 'a list -> bool
List.all : (* --- *)

(* mem : int -> int list -> bool
   * mem x L returns true iff x in L
   *)
fun mem x = List.exists (fn y => x = y)

(* A WRONG implementation *)
fun reachable g (x, y) =
  let (* dfs : int -> bool *)
    fun dfs n = (n = y) orelse (List.exists dfs (g n))
  in
    dfs x
  end

(* evaluation trace
   * reachable G (1, 4)
   * dfs 1
   * 1 <> 4
   * List.exists dfs [2, 3]
   * dfs 2
   * 2 <> 4
   * List.exists dfs [1, 3]
   * forever forever ...
   *)

(* We can solve this problem by keeping track of where we've visited! *)
fun reachable g (x, y) =

```

```
let
  val visited : int list ref = ref nil (* initialize *)
(* side note : value restriction prevents polymorphic type at top level *)
  fun dfs n = (n = y) orelse
    (not (mem n !visited) andalso
      (visited := n::(!visited) ; List.exists dfs (g n)))
in
  dfs x
end
(* This is benign mutation. There is no mutation percolates outside. *)
```