Programming Languages and Techniques (CIS120e)

Lecture 14
Oct 4, 2010

References

Digression: Options

An unfortunate situation...

```
let max_elt (l: int list) : int =
  fold max 0 l

;; assert_eq "max_elt" (max_elt [-1;-2;-3]) 0 (* ! *)
```

Options

A better way...

```
let max_el tl (l: int list) : int option =
  fold
    fun (x:int) (m: int option) ->
      begin match m with
        None -> max_x
        | Some y -> Some (max x y)
      end
    None
    l

;; assert_eq "max_el1" (max_el1 [-1;-2;-3]) (Some (-1))
```

```
let max_elt1 (l: int list) : int option =
  fold
    fun (x:int) (m: int option) ->
      begin match m with
        None -> Some x
        | Some y -> Some (max x y)
      end
    None
    l

;; assert_eq "max_el1" (max_el1 [-1;-2;-3]) (Some (-1))
```

Announcements

• Midterm 1 will be in class next Friday, October 15th
  – covers all material through next Wednesday’s lecture
  – next Thursday’s lab will be a review session
  – review problems will be available next Wednesday
  – exam will include at least one problem taken verbatim from a homework assignment
Another Example

let hd (l: 'a list) : 'a =
begin match l with
| [] -> failwith "hd of empty list"
| h::t -> h
end

let safe_hd (l: 'a list) : 'a option =
begin match l with
| [] -> None
| h::t -> Some h
end

One More Example

None is often a good initial value for a reference cell:

let m = ref None

let remember_max (x:int) : unit =
begin match !m with
| None ->
| Some y -> m := Some (max x y)
end

let max_so_far () : int =
begin match !m with
| None -> failwith "max_so_far called before remember_max"
| Some y -> y
end

Digression: OCaml vs. Java

- Java (and other imperative OO languages)...
  - every variable is mutable
  - any variable of “pointer type” (like queue node) can contain null
    - so we always have to think about the possibility of “null pointer
dereference”
  - variables of non-pointer types (like int) can never be null
- OCaml...
  - variables are immutable; ref cells are mutable
    - things are only mutable when we say they are
    - by looking at the type of something (and seeing whether it contains any
      refs), we can tell whether it may change out from under us
  - we can choose to wrap any value with an option, or not
    - again, we can tell from the type whether we need to think about the
      possibility that something may be “not there” (None)

Equality

Are r1 and r2 equal?
- They both point to cells containing the same value
- But the cells themselves are different (assigning into one
doesn’t change the other, for example)

OCaml provides two different equality tests:
- r1 = r2 -> true
- r1 == r2 -> false
  - “pointer equality” (same heap cell)
Shared State

Recall from last lecture:

```ocaml
type updown = {up: unit->int; down: unit->int}

let new_updown () : updown =
  let c = ref 0 in
  let u () : int =
    c := !c + 1;
    !c in
  let d () : int =
    c := !c - 1;
    !c in
  {up=u; down=d}

let ud = new_updown() in
assert_eq "ud1" (ud.up()) 1;
assert_eq "ud2" (ud.up()) 2;
assert_eq "ud3" (ud.down()) 1;
assert_eq "ud4" (ud.down())
```

Here, the up and down components of an updown share just a single reference cell.

More Complex Shared State

```ocaml
type 'a queue1 = 'a list ref

let create () : 'a queue1 =
  ref []

let enq (x: 'a) (q: 'a queue1) : unit =
  q := x :: !q

let deq (q: 'a queue1) : 'a =
  begin match List.rev !q with
  | [] -> failwith "deq"
  | h::t -> (q := List.rev t; h)
  end

let q = create () in
enq 1 q;
enq 2 q;
let x1 = deq q in
let x2 = deq q in
assert_eq "1" x1 1;
assert_eq "2" x2 2
```

Mutable Queues

- See lec14.ml...