CSCI 2041: Deep and Shallow Equality

Chris Kauffman

Last Updated:
Thu Oct 18 22:42:18 CDT 2018

Logistics

Goals Today

- Finish Higher-order Funcs
- Deep/Shallow Equality

Later this Week

- Wed: Scope and Functions
- Fri: Curried Funcs/Return Vals

Next Week

- Mon: Review
- Wed: **Exam 2**
- Fri: Lecture

Exam 1

Regrade requests via Gradescope, Due Mon 10/15

Assignment 3 `multimanager`

- Manage multiple lists
- Records to track lists/undo
- `option` to deal with editing
- Higher-order funcs for easy bulk operations
- Due Mon 10/22
- Test cases over the weekend

Exercise: Deep and Shallow Equality

- Some folks have noticed OCaml has two means of comparing values
  - `a = b` : structural or **deep** equality
  - `a == b` : physical or **shallow** equality
- Code in `equality.ml` to the right uses both
- What gets printed for the examples shown?

```
let a = 5 in
let b = 5 in
let c = a in
printf "---ints---\n";
printf "a=b  : %b\n" (a=b);
printf "a==b : %b\n" (a==b);
printf "a=c  : %b\n" (a=c);
printf "a==c : %b\n" (a==c);

let x = ref 5 in
let y = ref 5 in
let z = x in
printf "---int Refs---\n";
printf "x=y  : %b\n" (x=y);
printf "x==y : %b\n" (x==y);
printf "x=z  : %b\n" (x=z);
printf "x==z : %b\n" (x==z);
```

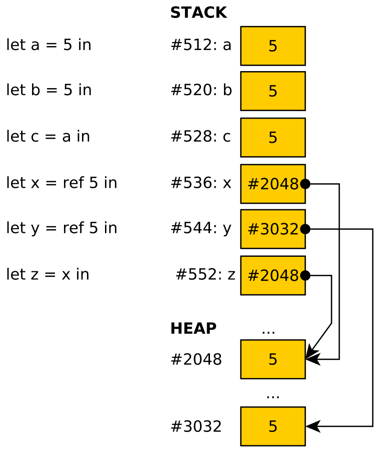
Answers: Deep and Shallow Equality

```
let a = 5 in                      (* box with 5  *)
let b = 5 in                      (* box with 5 *)
let c = a in                      (* box with copy a's contents *)
printf "---ints---\n";
printf "a=b  : %b\n" (a=b);       (* a=b  : true *)
printf "a==b : %b\n" (a==b);      (* a==b : true *)
printf "a=c  : %b\n" (a=c);       (* a=c  : true *)
printf "a==c : %b\n" (a==c);      (* a==c : true *)

let x = ref 5 in                  (* pointer to box w/ 5 *)
let y = ref 5 in                  (* pointer to new box w/ 5 *)
let z = x in                      (* copy x's pointer *)
printf "---int Refs---\n";
printf "x=y  : %b\n" (x=y);       (* x=y  : true  : same contents *)
printf "x==y : %b\n" (x==y);      (* x==y : false : different locations*)
printf "x=z  : %b\n" (x=z);       (* x=z  : true  : same contents *)
printf "x==z : %b\n" (x==z);      (* x==z : true  : same location *)
```

Answers: Deep and Shallow Equality

- Deep equality checks entire structure for corresponding equal values
- Shallow equality checks only if memory box contains the same value
- Pointers are stored as integers (notated in figure as #2048)
- Both work the same for boxed values like `int`
- Return different answers for unboxed values like `refs`



Deep vs Shallow Equality is in Every Language

C/C++

- `==` and `!=` operators compare single blocks of memory, mostly shallow equality
- Typically write an equality function to compare deep/recursive data

Java

- `==` and `!=` identical to C
- `a.equals(b)`: create methods to define meaning of deep equality for a class

Scheme

- `equal?` : deep equality
- `eq?` : shallow equality

```
; create two distinct lists, same elems
guile-scheme> (define x (list 1 2 3))
guile-scheme> (define y (list 1 2 3))

; check deep and shallow equality
guile-scheme> (equal? x y)    ; deep
#t                            ; true
guile-scheme> (eq? x y)       ; shallow
#f                            ; false
```

Python

Like Scheme, different op names

- `x == y` : deep equality
- `x is y` : shallow equality

Convenient Deep Equality in OCaml

Equal	Unequal	
=	<>	Deep
==	!=	Shallow

- Data defined via standard mechanisms in OCaml gets automatically has deep equality defined for it
- Arrays, Strings, Tuples, Records, Algebraic, all "just work"

```
let s = "hi" in                (* pointer to string of chars *)
let t = "hi" in                (* pointer to different string of chars *)
let u = s in                   (* pointer to same place as s *)
printf "---Strings---\n";
printf "s=t  : %b\n" (s=t);    (* s=t  : true  : same contents *)
printf "s==t : %b\n" (s==t);   (* s==t : false : different locations*)
printf "s=u  : %b\n" (s=u);    (* s=u  : true  : same contents *)
printf "s==u : %b\n" (s==u);   (* s==u : true  : same location *)

let f = {s="yo"; i=2} in       (* pointer to new record *)
let g = {s="yo"; i=2} in       (* pointer to new record *)
let h = f in                   (* pointer to existing record *)
printf "---Records---\n";
printf "f=g  : %b\n" (f=g);    (* f=g  : true  : same contents *)
printf "f==g : %b\n" (f==g);   (* f==g : false : different locations*)
printf "f=h  : %b\n" (f=h);    (* f=h  : true  : same contents *)
printf "f==h : %b\n" (f==h);   (* f==h : true  : same location *)
```

Choosing Deep vs Shallow Equality

- Generally use Deep equality, usually what is "intended"

  Are these two things equal to one another?

- Keep in mind Deep equality may visit whole data structure
  - All chars of a `string`
  - All elements of a `list` or `array`
  - All fields of a record, etc.
- $O(N)$ operation where N is the size of the data structure
- This may have performance implications:
- In some special cases, may be reasonable to use Shallow equality which is an $O(1)$ operation

Library Functions and Deep/Shallow Equality

- Some Library function distinguish between use of Deep/Shallow equality
- q suffix in function name indicates Shallow Equality is used
- Examples from the `List` module

```
val mem : 'a -> 'a list -> bool
  'mem elem list' is true if and only if elem is equal to an element
  of list.

val memq : 'a -> 'a list -> bool
  Same as List.mem, but uses physical (shallow) equality instead of
  structural (deep) equality to compare list elements.

val assoc : 'a -> ('a * 'b) list -> 'b
  'assoc key alist' returns the value associated with key in the list
  of pairs alist.

val assq : 'a -> ('a * 'b) list -> 'b
  Same as List.assoc, but uses physical (shallow) equality instead
  of structural (deep) equality to compare keys.
```

Exercise: Deep / Shallow Differences

- Code below searches a list for an element using
  - `mem` : deep equality
  - `memq` : shallow equality
- Determine values for results of searches
- Draw a picture of `x,y,z, listA, listB` to justify answers

```
let x = "yikes" in
let y = "boo!"  in
let z = "gulp"  in

let listA = [      x;      y;      z] in
let listB = ["yikes"; "boo!"; "gulp"] in

let d_yA = List.mem  y listA in
let s_yA = List.memq y listA in

let d_yB = List.mem  y listB in
let s_yB = List.memq y listB in
```

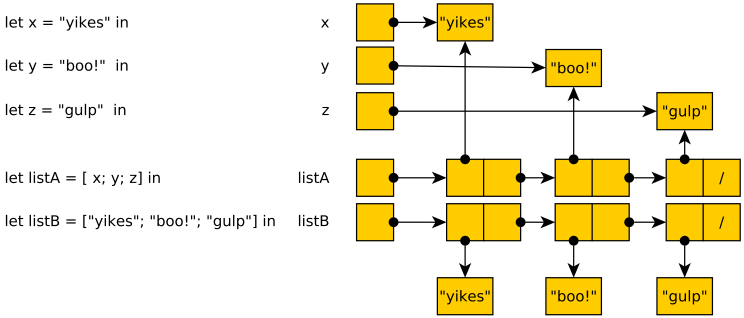
Answers: Deep / Shallow Differences

```
let x = "yikes" in
let y = "boo!"  in
let z = "gulp"  in

let listA = [      x;       y;      z] in
let listB = ["yikes"; "boo!"; "gulp"] in

let d_yA = List.mem  y listA in   (* deep equals: true *)
let s_yA = List.memq y listA in   (* shallow equals: true *)

let d_yB = List.mem  y listB in   (* deep equals : true *)
let s_yB = List.memq y listB in   (* shallow equals: false *)
```


let x = "yikes" in
let y = "boo!" in
let z = "gulp" in
let listA = [x; y; z] in
let listB = ["yikes"; "boo!"; "gulp"] in
x
y
z
listA
listB
"yikes"
"boo!"
"gulp"
/
/
"yikes"
"boo!"
"gulp"