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 \textit{deep} equality
- \( a == b \): physical or \textit{shallow} equality

Code in equality.ml to the right uses both

What gets printed for the examples shown?

```ocaml
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

1. let a = 5 in (* box with 5 *)
2. let b = 5 in (* box with 5 *)
3. let c = a in (* box with copy a’s contents *)
4. printf "---ints---\n"
5. printf "a=b : %b\n" (a=b); (* a=b : true *)
6. printf "a==b : %b\n" (a==b); (* a==b : true *)
7. printf "a=c : %b\n" (a=c); (* a=c : true *)
8. printf "a==c : %b\n" (a==c); (* a==c : true *)

9. let x = ref 5 in (* pointer to box w/ 5 *)
10. let y = ref 5 in (* pointer to new box w/ 5 *)
11. let z = x in (* copy x’s pointer *)
12. printf "---int Refs---\n"
13. printf "x=y : %b\n" (x=y); (* x=y : true : same contents *)
14. printf "x==y : %b\n" (x==y); (* x==y : false : different locations*)
15. printf "x=z : %b\n" (x=z); (* x=z : true : same contents *)
16. 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

```scheme
; 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

<table>
<thead>
<tr>
<th>Equal</th>
<th>Unequal</th>
<th>Deep</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>&lt;&gt;</td>
<td>Deep</td>
<td>Shallow</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```ocaml
1 let s = "hi" in (* pointer to string of chars *)
2 let t = "hi" in (* pointer to different string of chars *)
3 let u = s in (* pointer to same place as s *)
4 printf "---Strings---\n";
5 printf "s=t : %b\n" (s=t); (* s=t : true : same contents *)
6 printf "s==t : %b\n" (s==t); (* s==t : false : different locations*)
7 printf "s=u : %b\n" (s=u); (* s=u : true : same contents *)
8 printf "s==u : %b\n" (s==u); (* s==u : true : same location *)

9 let f = {s="yo"; i=2} in (* pointer to new record *)
10 let g = {s="yo"; i=2} in (* pointer to new record *)
11 let h = f in (* pointer to existing record *)
12 printf "---Records---\n";
13 printf "f=g : %b\n" (f=g); (* f=g : true : same contents *)
14 printf "f==g : %b\n" (f==g); (* f==g : false : different locations*)
15 printf "f=h : %b\n" (f=h); (* f=h : true : same contents *)
16 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
Some Library functions distinguish between use of Deep/Shallow equality

- q suffix in function name indicates Shallow Equality is used

Examples from the List module

```ml
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

```ocaml
1 let x = "yikes" in
2 let y = "boo!" in
3 let z = "gulp" in

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

8 let d_yA = List.mem y listA in
9 let s_yA = List.memq y listA in
10
11 let d_yB = List.mem y listB in
12 let s_yB = List.memq y listB in
```
Answers: Deep / Shallow Differences

```ml
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 *)
```