# Report (Mini Project 1)

XIAO LIU

## 1. Objective: polymorphic mapping

A mapping is a collection of tuples like <key,val>;

that is, it maps a key to a value.

Implement the following polymorphic mapping in ML.

The current implementation of functions:

insert

lookup

remove

union

intersect

filter

return wrong results, but with correct type.

Replace "EmptyMap" with your implementation

### 2. Program

Original

|  |
| --- |
| signature KEY =  sig  type key;  val equal : key \* key -> bool  end  structure IntKey : KEY =  struct  type key = int  fun equal(e1,e2) = (e1=e2)  end  signature MAP =  sig  structure Key : KEY;  type key;  sharing type key = Key.key;  type 'value map;  val EmptyMap : 'value map;  val insert : (key \* 'value) -> 'value map -> 'value map;  val lookup : key -> 'value map -> 'value map;  val remove : key -> 'value map -> 'value map;  val union : (''map\*''map)list -> (''map\*''map) list -> (''map\*''map) list;  val intersect :(''map\*''map)list -> (''map\*''map) list -> (''map\*''map) list;  val filter : (key -> bool) -> 'value map -> 'value map  end |

Completed

|  |
| --- |
| signature KEY =  sig  type key;  val equal : key \* key -> bool  end  structure IntKey : KEY =  struct  type key = int  fun equal(e1,e2) = (e1=e2)  end  signature MAP =  sig  structure Key : KEY;  type key;  sharing type key = Key.key;  type 'value map;  val EmptyMap : 'value map;  val insert : (key \* 'value) -> 'value map -> 'value map;  val lookup : key -> 'value map -> 'value option;  val remove : key -> 'value map -> 'value map;  val union : 'value map -> 'value map -> 'value map;  val intersect : 'value map -> 'value map -> 'value map;  val filter : (key -> bool) -> 'value map -> 'value map  end  structure Map : MAP =  struct  structure Key = IntKey;  type key = Key.key;  type 'value map = (key \* 'value) list;  val EmptyMap = nil;  (\*Concatenate the two lists\*)  fun insert (k,v) m = [(k, v)] @ m  (\*Recursively compare the key with the first key in the Map\*)  fun lookup k m = case m of  [] => NONE  | (hdkey,hdvalue)::tl =>  if (Key.equal(hdkey, k)) then SOME hdvalue else lookup k tl  (\*Recursively compare the key with the first key in the map, if same, remove it\*)  fun remove k m = case m of  [] => nil  | (hdkey,hdvalue)::tl =>  if (Key.equal(hdkey, k)) then tl else (hdkey, hdvalue)::remove k tl    (\*Concatenate the two maps and remove the repeated tuples\*)  fun union m1 m2 = let fun search k m = case m of  [] => false  | (hdkey,hdvalue)::tl1 => if (Key.equal(hdkey, k)) then true else search k tl1  in  let fun delete m = case m of  [] => nil  | (hdkey, hdvalue)::tl2 => if (search hdkey tl2) then delete tl2 else (hdkey, hdvalue)::delete tl2  in  delete (m1 @ m2)  end  end  (\*Recursively search keys in m2 with the keys occur in m1\*)  fun intersect m1 m2 = let fun search k m = case m of  [] => nil  | (hdkey, hdvalue)::tl => if (Key.equal(hdkey, k)) then [(hdkey, hdvalue)] else search k tl  in  case m1 of  [] => nil  | (hdkey, hdvalue)::tl => (search hdkey m2) @ (intersect tl m2)  end  (\*Recursively apply the filter function with the first key\*)  fun filter f m = case m of  [] => nil  | (hdkey,hdvalue)::tl => if (f hdkey) then filter f tl else (hdkey, hdvalue)::filter f tl  end |

### 3. Output

|  |
| --- |
| [opening map.sml]  map.sml:28.25 Warning: calling polyEqual  signature KEY =  sig  type key  val equal : key \* key -> bool  end  structure IntKey : KEY  signature MAP =  sig  structure Key :  sig  type key  val equal : key \* key -> bool  end  type key  type 'a map  val EmptyMap : 'a map  val insert : key \* 'a -> 'a map -> 'a map  val lookup : key -> 'a map -> 'a option  val remove : key -> 'a map -> 'a map  val union : 'a map -> 'a map -> 'a map  val intersect : 'a map -> 'a map -> 'a map  val filter : (key -> bool) -> 'a map -> 'a map  sharing type Key.key = key  end  structure Map : MAP  val it = () : unit |

### 4. Test Case

Insert

|  |
| --- |
| - val s1= Map.insert (1,"Alice") [(2,"Bob"),(3,"Cat")];  val s1 = [(1,"Alice"),(2,"Bob"),(3,"Cat")] : string Map.map |

Lookup

|  |
| --- |
| - val s2= Map.lookup 3 [(1,"Alice"),(2,"Bob"),(3,"Cat")];  val s2 = SOME "Cat" : string option |

Remove

|  |
| --- |
| - val s3= Map.remove 1 [(1,"Alice"),(2,"Bob"),(3,"Cat")];  val s3 = [(2,"Bob"),(3,"Cat")] : string Map.map |

Intersect

|  |
| --- |
| - val s4= Map.intersect [(1,"Alice"),(2,"Bob"),(3,"Cat"),(4,"Jiang Ming"),(5,"Xiao Liu")] [(1,"Alice"),(2,"Bob"),(3,"Cat"),(4,"Ming Jiang"),(5,"Dongpeng Xu")];  val s4 = [(5,"Xiao Liu"),(4,"Jiang Ming"),(3,"Cat"),(2,"Bob"),(1,"Alice")]  : string Map.map |

Union

|  |
| --- |
| - val s5= Map.union [(1,"Alice"),(2,"Bob"),(3,"Cat")] [(4,"Jiang Ming"),(5,"Xiao Liu"),(6,"Dongpeng Xu")];  val s5 =  [(3,"Cat"),(6,"Dongpeng Xu"),(2,"Bob"),(5,"Xiao Liu"),(1,"Alice"),  (4,"Jiang Ming")] : string Map.map |

Filter

|  |
| --- |
| fun filter\_nonmember x = if (x<4) then true else false;  val filter\_nonmember = fn : int -> bool  - val GroupMember= Map.filter filter\_nonmember s5;  val GroupMember = [(6,"Dongpeng Xu"),(5,"Xiao Liu"),(4,"Jiang Ming")]  : string Map.map |