Contents

1 Module `Hweak` : The type of the elements stored in the table.

```
module type S =
  sig
```

`type key`

The type of the elements stored in the table.

`type 'a t`

The type of weak hash tables from type `key` to type `'a`. if etheir the key or the data of a binding is freed by the GC, the binding is silently droped

`val create : int -> 'a t`

`create n` creates a new empty weak hash table, of initial size `n`. The table will grow as needed.

`val clear : 'a t -> unit`

Remove all elements from the table.

`val add : 'a t -> key -> 'a -> unit`

`add tbl key x` adds a binding of `k` to `x` in table `t`. Previous binding for `x` are not removed, and which binding will be find by next `find` (or `merge`) is unspecified

`val replace : 'a t -> key -> 'a -> unit`

`replace tbl key x` replace the current binding of `key` in table `t` by a binding from `key` to `x`. If there was no such binding a new one is still created. This new binding will be the one find by next find (and merge)

`val remove : 'a t -> key -> unit`

`remove tbl x` removes the current binding of `x` in `tbl`. if there is another binding of `x` in `tbl` then it became the current one. It does nothing if x is not bound in `tbl`

`val merge : 'a t -> key -> 'a -> 'a`

`merge tbl key x` returns the current binding of `k` in `t` if any, or else adds a bindding of `k` to `x` in the table and return `x`.

```
val find : 'a t -> key -> 'a
```

`find tbl key` returns the current binding of `k` in `t` if any, otherwise raise Not_found

```
val find_all : 'a t -> key -> 'a list
```

`find tbl key` returns the current binding of `k` in `t` if any, otherwise raise Not_found

```
val mem : 'a t -> key -> bool
```

`mem tbl x` checks if `x` is bound in `tbl`.

```
val iter : (key -> 'a -> unit) -> 'a t -> unit
```

`iter f tbl` applies `f` to all bindings in table `tbl`. `f` receives the key as first argument, and the associated value as second argument. The order in which the bindings are passed to `f` is unspecified. Each binding is presented exactly once to `f`.

```
val fold : (key -> 'a -> 'b -> 'b) -> 'a t -> 'b -> 'b
```

`fold f tbl init` computes `(f kN dN ... (f k1 d1 init)...)`, where `k1 ... kN` are the keys of all bindings in `tbl`, and `d1 ... dN` are the associated values. The order in which the bindings are passed to `f` is unspecified. Each binding is presented exactly once to `f`.

```
val count : 'a t -> int
val stats : 'a t -> int * int * int * int * int * int
```

some statistic function

```
  end

module Make :
   functor (H : Hashtbl.HashedType) -> S  with type key = H.t
```

2 Module `Weak_memo` : The memo class provides an easy way to remember the real class of an object.

This module contain the `Weak_memo.c` class that contain object that can be recover latter from the same object with a different type. An object that is into a `Weak_memo.c` object is not prevented from recolection by the Gc.

```
class < .. > c : int ->
  object
```

```
    method add : (< .. > as 'a) -> unit
```

Add an object to the memo. This won't prevent this object to be recollected by the Gc. Do nothing if the object is already there.

`method find : 'b. (< .. > as 'b) -> 'a`

Find if the object is in the memo. If yes then return it (with the good type). Otherwise raise `Not_found`

`method mem : 'c. (< .. > as 'c) -> bool`

Return true if there is an object in the memo that is equal to the object given as argument. Return false otherwise.

`method remove : 'd. (< .. > as 'd) -> unit`

Remove an object from the memo. If it is not there, do nothing

`method clear : unit -> unit`

Empty the memo object.

`method count : int`

Return the number of object in the memo.

`end`

The memo class.

`new Weak_memo.c size` create a new memo object with an original size of `size`. It will grow as needed.