Mike McClurg <mike.mcclurg@gmail.com>

[Caml-list] Obj.magic for polymorphic identifiers

Romain Bardou <romain@cryptosense.com> Tue, Apr 22, 2014 at 2:03 AM
Reply-To: Romain Bardou <romain@cryptosense.com>
To: Ocaml Mailing List <caml-list@inria.fr>

Hello,

I'm considering using Obj.magic and as the type-checker can no longer ensure type safety, I decided to come here for advice.

I want to implement the trick with GADTs where you test equality of unique identifiers, and if they match this adds an equality constraint on types. I want this code to be small and well abstracted in a module so that if this module is safe, then using this module cannot cause a seg fault.

Here is the signature of my module:

```
(******************************************************************************)
(** Polymorphic identifiers. *)

(** The type of identifiers associated to type ['a]. *)
type 'a t

(** Make a new, fresh identifier.
    This is the only way to obtain a value of type [t]. *)
val fresh: unit -> 'a t

(** Type constraint which is conditioned on identifier equality. *)
type (_, _) equal =
  | Equal: ('a, 'a) equal
  | Different: ('a, 'b) equal

(** Equality predicate. *)
val equal: 'a t -> 'b t -> ('a, 'b) equal

(** Convert an identifier to an integer.
    The integer is guaranteed to be unique for each call to {!fresh}. *)
val to_int: 'a t -> int
(******************************************************************************)
```

and here is the implementation:

```
(******************************************************************************)
type 'a t = int
```

```
let fresh =
  let counter = ref (-1) in
  fun () ->
    incr counter;
    !counter

type (_, _) equal =
  | Equal: ('a, 'a) equal
  | Different: ('a, 'b) equal

let equal (type a) (type b) (a: a t) (b: b t): (a, b) equal =
  if a = b then
    (Obj.magic (Equal: (a, a) equal): (a, b) equal)
  else
    Different

let to_int x =
  x
(******************************************************************************)
```

Finally, here is a test program:

```
(******************************************************************************)
open Polid

let () =

  let x = fresh () in
  let y = fresh () in

  let eq (type a) (type b) (t: a t) (u: b t) (a: a) (b: b) =
    match equal t u with
      | Equal ->
          if a = b then "true" else "false"
      | Different ->
          "false"
  in

  print_endline (eq x y 5 "salut"); (* false *)
  print_endline (eq x x 5 5); (* true *)
  print_endline (eq x x 5 9); (* false *)
  print_endline (eq y y "test" "salut"); (* false *)
  print_endline (eq y y "test" "test"); (* true *)
  print_endline (eq y x "salut" 5); (* false *)
  (* print_endline (eq x x 5 "salut"); (\* type error *\) *)
  (* print_endline (eq y y "salut" 5); (\* type error *\) *)

  ()
(******************************************************************************)
```

It relies heavily on the fact that "fresh ()" cannot be generalized as
'a t is abstract.

A typical use case is as follows: I have two heterogeneous association lists (using GADTs for existential types). As I iterate on one of those lists, I need to find the corresponding item in the other list. As I unpack the existential, the type-checker cannot prove that the existential types are equal, hence the need for a runtime check (a call to Polid.equal).

Can you find any reason why this would not be safe, or any better way to implement this?

Thank you,

--
Romain Bardou
--
Romain Bardou
Cryptosense
96bis boulevard Raspail, 75006 Paris, France
+33 (0)9 72 42 35 31

--