Specifying the unboxability check on mutually recursive datatypes in OCaml

Simon Colin

March 20, 2018

Values in OCaml

Values that aren't primitives are made of a tag and the value

Can unbox single constructor single value types to only store them as their value

unboxed: example

```
type name = Name of string [@@unboxed]
```

Float arrays

Floats are two memory words, every other primitive is one

Float arrays and non float arrays have different field sizes

Floats vs. non-floats

```
# let array = Array.create_float 3;;
val array : float array = [|0.; 0.; 0.|]
# Array.set array 1 (Obj.magic 1);;
Segmentation fault (core dumped)
```

GADTs

OCaml has GADTs with which we can create a type that can contain values of any type

```
type printable =
  | Pair : 'a * ('a -> string) -> printable

type any =
  | Any : 'a -> any
```

Should we unbox such a type we would get segmentation faults

Rejected unboxed

```
# type ext = E : 'a -> ext [@@unboxed]
Error: This type cannot be unboxed because
       it might contain both float and non-float values.
       You should annotate it with [@@ocaml.boxed].
```

OCaml rightly rejects such types if we try to unbox them

However it also rejects types that should be unboxable

```
# type ('a, 'b) t =
  | R : 'a * int -> ('a, int) t
  | I : 'a u -> ('a, int) t
  and 'a u = U : ('a, _) t -> 'a u [@@unboxed];;
Error: This type cannot be unboxed because
       it might contain both float and non-float values.
       You should annotate it with [@@ocaml.boxed].
```

Modes

To decide whether types can be unboxed we used a typing system

Sep means that the type can either only contain floats or only non floats

Ind means that we don't assert anything about the type

This means that Sep $\subset$ Ind

The judgements

If all the types evaluate to the mode they are supposed to be the set of the definitions is well formed

If this is not the case there exists an offending definition that isn't of the right mode

Some rules

$$\frac{}{\Gamma \vdash \mathit{int} : m}$$

Base types are of all modes

$$\frac{\mathit{Def}; \Gamma \vdash A : \mathit{Sep} \qquad \mathit{Def}; \Gamma \vdash B : \mathit{Sep}}{\Gamma \vdash A \times B : \mathit{Sep}}$$

A pair of values cannot be a float so it is Sep

Parameterized types

$$\frac{(type(\alpha_i : m_i)^I t) \in Def \qquad \forall i \in I, \Gamma \vdash A_i : m.m_i}{Def; \Gamma \vdash (A_i)^I t : m}$$

We define a product of modes such that

$$m.m = m \qquad \text{Sep.Ind} = \text{Ind.Sep} = \text{Ind}$$

Existential types

$$\frac{Def; \Gamma, \alpha : \mathsf{Ind} \vdash T : m}{Def; \Gamma \vdash \exists \alpha, T : m}$$

An existentially quantified variable can never be Sep
Types featuring one can only be Sep only if that doesn't require the existential variable to be Sep

Fixpoint

If we apply the rules to work our way up we get a list of type variables and the modes that they should be

If a type doesn't evaluate to the mode we need it to be we know which type variables need to change mode

If we're trying to unbox an type that isn't unboxable we can raise an error