Specifying the unboxability check on mutually recursive datatypes in OCaml

Simon Colin

March 21, 2018

Unboxed constructors 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

```
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

```
# let array = Array.make 3 0.0;;
val array : float array = [|0.; 0.; 0.|]
# Array.set array 1 (Obj.magic true);;
Segmentation fault (core dumped)
```

GADTs

Existentially quantified type variables.

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

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

Should we unbox any we would get segmentation faults

```
# let array = Array.make 2 (Any 0.0);;
val array : any array = [|Any <poly>; Any <poly>|]
# Array.set array 1 (Any true);;
Segmentation fault (core dumped)
```

Rejected unboxed

```
# type any = Any : 'a -> any [@@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 OCaml 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].
```

https://caml.inria.fr/mantis/view.php?id=7364

My project

Formalize the correctness condition for `[@@unboxed]`:
design a system of inference rules

Prototype implementation

Feburary 15th: visit to Damien Doligez (previous impl.)
– he liked the proposal.

(no compiler patch yet)

Modes

We propose a system of inference rules

$$Def; \Gamma \vdash A : m \qquad\qquad m \quad ::= \quad \mathsf{Sep} \mid \mathsf{Ind}$$

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

```
type ('a, 'b) t = 'a
```

$$\mathsf{type}\ (\alpha : \mathsf{Sep}, \beta : \mathsf{Ind})\ t\ =\ \alpha$$

Judgments

$$\vdash \mathit{Def} \qquad\qquad \mathit{Def}\,;\Gamma \vdash A : m$$

$$\begin{array}{lcl} m & ::= & \mathsf{Sep} \mid \mathsf{Ind} \\ A, B & ::= & \alpha \mid \mathsf{int} \mid A \to B \mid (A_i)^{i \in I}\, t \mid \exists \alpha.A \\ \mathit{Def} & ::= & \emptyset \mid \mathit{Def}\,, \mathsf{type}\ (\alpha_i : m_i)^{i \in I}\, t = A \end{array}$$

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

Base types are of all modes

$$\frac{}{\Gamma \vdash A \times B : m}$$

A pair of values cannot be a float so it is Sep (or Ind)

$$\frac{(\alpha : m) \in \Gamma}{\mathit{Def}; \Gamma \vdash \alpha : m}$$

The context Γ stores the modes of type variables (type parameters).

Existential types

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

Types featuring $\exists\alpha.$ can only be Sep if that doesn't require α to be Sep

$$Def\,;\Gamma \vdash \exists\alpha.\alpha : \mathsf{Ind} \qquad \cancel{Def\,;\Gamma \vdash \exists\alpha.\alpha : \mathsf{Sep}}$$

$$Def\,;\Gamma \vdash \exists\alpha.(\alpha \times (\alpha \to \mathsf{string})) : \mathsf{Sep}$$

Fixpoint

Def : how to guess the parameter modes?
$\Longrightarrow$ start with Ind everywhere

$$Def ; \Gamma \vdash A : m \qquad\qquad \vdash Def$$

We apply the rules to work our way up

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

$\Longrightarrow$ repeat until we reach fixpoint

Unboxing is allowed only if the type definition is Sep

Implementation

https://github.com/SimonColin/ocaml-unboxed-check-project

```
type tyvar = Var of string
type tyname = Name of string
type mode = Sep | Ind | Deepsep

type ty = Unit | Int| Float | ...

type def = Def of (tyvar * mode) list * tyname * ty * mode

val check_type :
  ty -> def list -> mode -> (tyvar * mode) list
val check_def :
  def -> def list -> mode -> (tyvar * mode) list
val check :
  def list -> def list
```

Not in this talk

- third mode `deepsep`
- GADT equations (not handled yet)

See the report.

Work done

The inference rules were worked out together with Gabriel Scherer.

The implementation, report and presentation were made by me with advice from Gabriel Scherer.

Thanks. Any question?

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

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