Specifying the unboxability check on mutually recursive datatypes in OCaml

Simon Colin, Rodolphe Lepigre, Gabriel Scherer

January 28, 2019



Unboxed constructors in OCaml

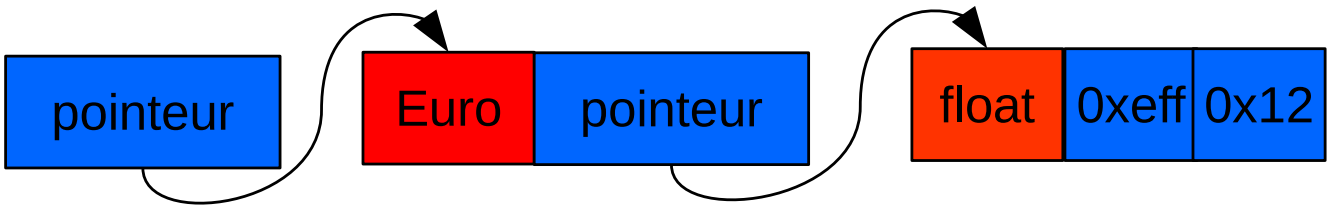
OCaml values are **words**, that are either

- an immediate value (`2` , `true` , `[]` , `None` ...)
- a pointer to a **block** on the heap

where a **block** contains:

- a **header**
- some arguments which are **words**

```
type euro = Euro of float
```



Unboxed constructors in OCaml

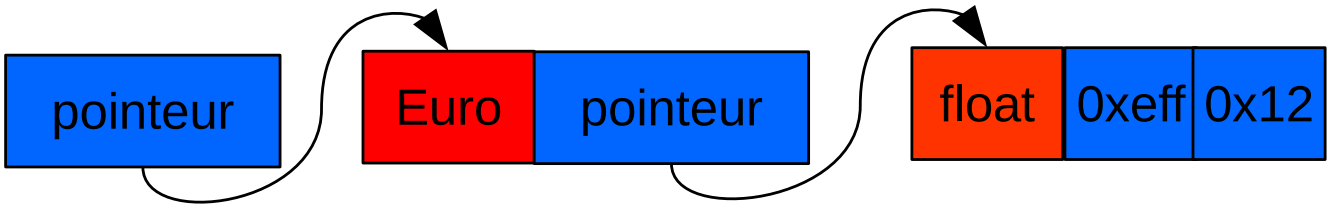
OCaml values are **words**, that are either

- an immediate value (`2` , `true` , `[]` , `None` ...)
- a pointer to a **block** on the heap

where a **block** contains:

- a **header**
- some arguments which are **words**

```
type euro = Euro of float
```




```
type euro = Euro of float [@@unboxed]
```

Float arrays

Generic OCaml arrays: each value is a word.

Float array optimization:

- remove pointer and header
- store the two words inline

Dynamic choice at array-creation time.

```
# 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, 'kind) tree =
  | Root : { value : 'a; rank : int } -> ('a, root) tree
  | Inner : { parent : 'a node } -> ('a, inner) tree

and 'a node =
  Node : ('a, _) tree -> 'a node [@@ocaml.unboxed]
```

(Markus Mottl, https://caml.inria.fr/mantis/view.php?id=7364)

Modes

We propose a system of inference rules

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

`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

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

Mode signature: $\texttt{type}\ (\alpha : \texttt{Sep}, \beta : \texttt{Ind})\ t\ =\ \alpha$

Judgments

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

$$
\begin{array}{lcl}
m & ::= & \texttt{Sep} \mid \texttt{Ind} \\
A, B & ::= & \alpha \mid \mathsf{int} \mid A \to B \mid (A_i)^{i \in I}\, t \mid \forall \alpha.A \mid \exists \alpha.A \mid B \restriction (\alpha = A) \\
\Gamma & ::= & \varnothing \mid \alpha : m \\
\Sigma & ::= & \varnothing \mid \Sigma, \texttt{type}\ (\alpha_i : m_i)^{i \in I}\, t = A
\end{array}
$$

Some inference rules

$$\frac{}{\Gamma \vdash \mathsf{int} : m} \qquad \frac{}{\Gamma \vdash \mathsf{bool} : m} \qquad \frac{}{\Gamma \vdash \mathsf{float} : m}$$

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`)

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

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

Inference rule for parametrized types

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

We define a product of modes such that

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

Existential types

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

Types featuring $\exists\alpha.$ can only be `Sep` if that does not require α to be `Sep`

$$\Sigma; \Gamma \vdash \exists \alpha.\alpha : \texttt{Ind} \qquad \cancel{\Sigma; \Gamma \vdash \exists \alpha.\alpha : \texttt{Sep}}$$

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

Checking a whole block

$$\frac{\Sigma = (\texttt{type}\ (\alpha_i : m_i)^{i \in I_k}\, t_k = A_k)^{k \in K} \qquad \forall k \in K,\ \Sigma; \varnothing \vdash A_k : \texttt{Sep}}{\vdash \Sigma}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\frac{}{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}$$
$$\frac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind}, ? \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, ? \vdash \beta : \texttt{Sep}}}{\dfrac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind}, ? \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, ? \vdash \beta : \texttt{Sep}}}{\dfrac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, ? \vdash \beta : \texttt{Sep}}}{\dfrac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, ? \vdash \beta : \texttt{Sep}}}{\dfrac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, \beta : \texttt{Sep} \vdash \beta : \texttt{Sep}}}{\dfrac{?, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, \beta : \texttt{Sep} \vdash \beta : \texttt{Sep}}}{\dfrac{\beta : \texttt{Sep}, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; ? \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, \beta : \texttt{Sep} \vdash \beta : \texttt{Sep}}}{\dfrac{\beta : \texttt{Sep}, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; \beta : \texttt{Sep} \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Right-to-left algorithm

Judgment: $\Sigma; \Gamma \vdash A : m$
Algorithm: $\Sigma, A, m \Longrightarrow \Gamma$

Suppose Σ is $\texttt{type}\ (\alpha : \texttt{Ind}, \beta : \texttt{Sep})t = \beta$.
Question: mode for β in $\texttt{type}\ (\beta)u = \exists\alpha.(\alpha, \beta)t$

$$\dfrac{\dfrac{}{\alpha : \texttt{Ind} \vdash \alpha : \texttt{Ind}} \qquad \dfrac{}{\alpha : \texttt{Ind}, \beta : \texttt{Sep} \vdash \beta : \texttt{Sep}}}{\dfrac{\beta : \texttt{Sep}, \alpha : \texttt{Ind} \vdash (\alpha, \beta)t : \texttt{Sep}}{\Sigma; \beta : \texttt{Sep} \vdash \exists\alpha.(\alpha, \beta)t : \texttt{Sep}}}$$

Answer: $\texttt{type}\ (\beta : \texttt{Sep})u$

Mutually-recursive types: fixpoint

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

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

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$\Sigma_0 \quad = \quad (\beta : \texttt{Ind})\ u, (\alpha : \texttt{Ind}, \beta : \texttt{Ind})\ t$

$$(\beta : \texttt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \texttt{Sep} \quad \Rightarrow \quad \text{ok}$$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$\Sigma_1 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$\Sigma_1 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$$\Sigma_1 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t$$

$$(\beta : \mathtt{Ind})\ u = \exists \alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

Fixpoint: example

```
type 'b u = exists 'a. ('a, 'b) t
and ('a, 'b) t = 'b
```

$$\Sigma_0 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t$$

$$(\beta : \mathtt{Ind})\ u = \exists\alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Ind})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$$\Sigma_1 \quad = \quad (\beta : \mathtt{Ind})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t$$

$$(\beta : \mathtt{Ind})\ u = \exists\alpha.(\alpha, \beta)\ t : \mathtt{Sep} \quad \Rightarrow \quad \beta : \mathtt{Sep}$$

$$(\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t = \beta : \mathtt{Sep} \quad \Rightarrow \quad \mathsf{ok}$$

$$\Sigma_2 \quad = \quad (\beta : \mathtt{Sep})\ u, (\alpha : \mathtt{Ind}, \beta : \mathtt{Sep})\ t$$

Conclusion

- Analysis problem presented as a type system
- From rules to algorithm: right-to-left reading

Proposed implementation: (reviews welcome!)
https://github.com/ocaml/ocaml/pull/2188

Not in this talk (see paper):

- third mode `Deepsep`
- GADT equations
- cyclic types
- principality issues

Thanks. Any question?

Deepsep

```
type safe_or_not = Any : ('b * int) t -> safe_or_not
```

Deepsep

```
type safe_or_not = Any : ('b * int) t -> safe_or_not


type _ t = Danger : 'b -> ('b * int) t [@@unboxed]
```

Deepsep

```
type safe_or_not = Any : ('b * int) t -> safe_or_not
```

```
type _ t = Danger : 'b -> ('b * int) t [@@unboxed]
```

(A : Deepsep): all sub-components of A must be Sep.

$$\frac{\Gamma \vdash A : m.\mathtt{Ind} \qquad \Gamma \vdash B : m.\mathtt{Ind}}{\Gamma \vdash A \times B : m}$$

$$\mathtt{Deepsep}.m = \mathtt{Deepsep}$$

GADT equations

$$\dfrac{\dfrac{\Gamma, \alpha : \texttt{Deepsep} \vdash B : \texttt{Sep}}{\Gamma, \alpha : \texttt{Ind} \vdash B \restriction (\alpha = \mathsf{int}) : \texttt{Sep}}}{\Gamma \vdash \exists \alpha.\ B \restriction (\alpha = \mathsf{int}) : \texttt{Sep}}$$

GADT equations

$$\frac{\Gamma, \alpha : \texttt{Deepsep} \vdash B : \texttt{Sep}}{\dfrac{\Gamma, \alpha : \texttt{Ind} \vdash B \restriction (\alpha = \mathsf{int}) : \texttt{Sep}}{\Gamma \vdash \exists \alpha.\ B \restriction (\alpha = \mathsf{int}) : \texttt{Sep}}}$$

$$\frac{\forall \Gamma' \geqslant \Gamma, \qquad \Gamma' \vdash (A_1 = A_2) \implies \Gamma' \vdash B : m}{\Sigma; \Gamma \vdash B \restriction (A_1 = A_2) : b}$$

$$\frac{\forall m, \quad \Gamma' \vdash A_1 : m \iff \Gamma' \vdash A_2 : m}{\Gamma' \vdash (A_1 = A_2)}$$

Non-principality

```
type (_, _) eq =
| Refl : ('a, 'a) eq

type (_, _) weird_eq =
| Weird_refl : 'a -> ('a, 'a) weird_eq
[@@unboxed]
```

$(\alpha : \texttt{Ind}, \beta : \texttt{Sep})$ weird_eq $\qquad$ $(\alpha : \texttt{Sep}, \beta : \texttt{Ind})$ weird_eq