

Extraction from Coq sort Type to Ocaml

Loïc Pottier*

*LEMME project, INRIA, 2004 route des Lucioles, B.P. 93, 06902 SOPHIA
ANTIPOLIS Cedex FRANCE, email: Loic.Pottier@sophia.inria.fr

Motivations

- Representation of mathematical structures needs the sort `Type`:
- Extraction of Coq works well on the sorts `Set` and `Prop`, not on `Type`.
- It may fail to produce good ML types (e.g. contrib Higman of Coq).

```
Structure Setoid : Type :=  
  {Carrier      : Set;  
   Equal        : Carrier -> Carrier -> Prop;  
   Prf_equiv    : (Equivalence Equal)}.
```

From Coq to Ocaml

Three ideas:

- types of Coq translated into terms of Ocaml (not types).
- forget if needed the type-checker of Ocaml (use `Obj.magic`).
- retract all Coq terms in sort Prop into a single constant Prop of Ocaml (C.Paulin).

Intermediate structure

Translation of Coq terms into untyped λ terms with recursion and pattern matching:

```
type _T =
  R of int          (* De Bruijn index          *)
| L of _T          (* abstraction            *)
| A of (_T array)  (* application            *)
| Cs of string     (* constant of the context, sorts *)
| Rec of int*(_T array) (* fixpoint definition    *)
| Ci of string*int*int*int (* inductive types and constructors,
                             with arities *)
| Case of _T * _T *(_T array) (* case construction      *)
;;
```

Products are translated into abstractions.

Retraction of sort Prop

- With extraction of Coq, terms of sort Prop are deleted. It works in Set/Prop context, because Prop and Set are disjoint.
- But fails in Type, because

$$\frac{t : Prop}{t : Type_i}$$

A term of sort Prop can occur in a context where it needs to be in Type.

⇒ We cannot wildly delete logical terms, just retract them:

a Coq term of sort Prop is translated in the `_T` term (Cs "Prop").

From λ T to Ocaml code

λ T	Ocaml
De Bruijn index $R(n)$	variable
abstraction $L(t)$	<code>(fun x -> t)</code>
application $A [f;t_1;\dots;t_n]$	<code>(f t1 ... tn)</code>
constants $C_s(s)$	<code>let s=def;;</code> if def exists.
fixpoint $Rec(k, [f_1;\dots;f_n])$	<code>let rec f1 = ... and f2 = ... and fn = ... in fk;;</code>
inductive types and constructors $C_i(t, -, -, -)$	constructors of a global Ocaml type (see example...)
$Case(a, t, [r_1;\dots;r_n])$	<code>match a with c1 -> r1 ... cn -> rn</code>

Ocaml type-checking

The produced code may be refused by the Ocaml type-checker. In that case, encapsulate all (suspect) terms by the Ocaml function `Obj.magic`. But we get less efficient code.

Example: euclidean division 30000 by 31 with `ARITH/Div.v`(DELL 350MHz, Windows 98):

standard extraction of Coq	1.1s
extraction without <code>Obj.magic</code>	1.3s
extraction with <code>Obj.magic</code>	3 s

Example: Higman lemma

On the Coq contrib `Higman.v`, the standard extraction of Coq fails:

```
Coq < Require Higman.
```

```
Coq < Write Caml File "higman" [Higman].
```

The axiom `False_rec` is translated into an exception.

Error during interpretation of command:

```
Write Caml File "higman" [Higman].
```

```
Error: Constant Minbad does not correspond to an ML type
```

With the presented method, without `Obj.magic` encapsulation, the Ocaml code (2572 lines) fails to type-check. But with it, it works well.

The main function `_Higman` takes a sequence of words $(w_i)_{i \in \mathbb{N}}$ using two characters, and return $i < j$ such that w_i is a sub-word of w_j .

Implementation: a Coq command.

```
Coq < Require FullExtraction.
```

Possibly rejected code (without `Obj.magic`):

```
Coq < Extract "<filename>" [<ident1> <ident2> ... ].
```

Accepted code (with `Obj.magic`):

```
Coq < Extract "<filename>" [<ident1> <ident2> ... ] exact.
```

Available at URL:

<http://www-sop.inria.fr/lemme/Loic.Pottier/Coq/extraction.html>.

Conclusion

Other experiments:

- Buchberger algorithm: the produced code is twice slower than the code produced by the standard extraction of Coq.
- use of the intermediate structure `_T` to produce C code and Java code (V.Prevesto).

Improvements:

- modularity and readability of the produced code.
- use the context of logical terms to delete them.
- use extraction as a fast normal form computation of Coq terms (at least non-functional terms of inductive types).

