The extraction in Coq

Pierre Letouzey

Coq Implementors Workshop – 30/5/2016

Extraction in a nutshell

In the extracted code, we discard :

- any proof `p : ... : Prop`
- anything which is a Coq type `t : sort`
  - in fact, even type scheme `ts : forall ..., sort`

Justification :

- A match on a proof stays in a proof
- No match on a type or type scheme

Sources

- The directory : `plugins/extraction/`
- The core : `extraction.ml` (+ `miniml.mli`)
- The entry : `extract_env.ml`
- The ugly : `common.ml`
- And of course : `ocaml.ml`, `haskell.ml`, `scheme.ml`

Issues

- This classification isn't definitive
  - `x:X:Type` kept, could become a proof or type "later".
- "Mixed" terms : `fun b => if b then nat else True`
- Maintain execution order
  - What about a partial application waiting a precondition ?

Hence we keep the code structure (at first), and replace discarded code by an arbitrary constant `__`

More issues

Actually, some proofs may be analysed in programs

- `False_rect`
- `Eq_rect`
- `Acc_rect`

Demo

- `extraction_cornercases.v`

Cumulativity made explicit ?

- `explicit_cumul.v`

More Difficulties

- The extraction of types : best effort
- Renaming (see wiki)
- "Optimizations"
- Be vigilant to changes in Coq
  - Singletons with no content
  - Modules
  - Native projections
  - Universe polymorphism ?

TODO

- Certified extraction
- Guaranteed interaction with external code (imperative?)
- Deeper transformations (monads into "try" or imperative)
- New target languages : Scala, F#, ...
- An "Extraction Compute" command
- A better "Extraction Implicit"
- Take advantage of novelties in the target languages