Aspects of a machine-checked intermediate language for extraction from Coq, in MetaCoq

Yannick Forster and Matthieu Sozeau, Gallinette Team, Nantes

Y. F. received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101024493.
Extraction in Coq

Coq’s Extraction turns 18 this year!

One of the central claims to fame of Coq works by first erasing types and proofs, obtaining a term in the untyped lambda-calculus $\lambda\Box$
The MetaCoq project

- a formalisation of Coq in Coq
  - confluence, validity, subject reduction
  - weak call-by-value standardisation (if \(t\) is of first-order inductive type and reduces to a value, then this value can be found with weak call-by-value evaluation)

- machine-checked programs regarding Coq:
  - a correct and complete type checker
  - an erasure procedure into an untyped version of Coq, removing proofs

- Vision: a fast kernel for daily use, a verified kernel for monthly use

- Future work:
  - eta, talk to Meven Lennon-Bertrand
  - SProp, talk to Yann Leray
  - modules, talk to Yee Jian Tan
  - template polymorphism
Towards verified extraction

Coq kernel → Template-Coq → PCUIC → $\lambda$□

CertiCoq: compilation to C light

ConCert: extraction to blockchain languages

OCaml: we are working on replacing Coq’s extraction process by a verified one
### Subtle differences

<table>
<thead>
<tr>
<th>Coq / lambda box</th>
<th>OCaml / C representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>● structural fix</td>
<td>● unary let rec</td>
</tr>
<tr>
<td>● higher-order constructors</td>
<td>● constructors are blocks</td>
</tr>
<tr>
<td>● match on □</td>
<td>● cannot match on functions</td>
</tr>
<tr>
<td>● de Bruijn</td>
<td>● named</td>
</tr>
<tr>
<td>● fix / match</td>
<td>● let rec / switch / proj</td>
</tr>
</tbody>
</table>

Uniform solutions necessary, to avoid doing work twice for different targets.
Aspects of a machine-checked intermediate language for extraction from Coq, in MetaCoq

Flags

Class

with

with

with

Section

Constr

Constr

(* Th. *)

Induc

(** Fix unfolding, with guard *)
| eval_fix f mfix idx argsv a av fn res :
  forall guarded : with_guarded_fix,
  eval f (mkApps (tFix mfix idx) argsv) ->
  eval a av ->
  cunfolds mfix idx = Some (#|argsv|, fn) ->
  eval (tApp (mkApps fn argsv) av) res ->
  eval (tApp f a) res

(** Fix stuck, with guard *)
| eval_fix_value f mfix idx argsv a av narg fn :
  forall guarded : with_guarded_fix,
  eval f (mkApps (tFix mfix idx) argsv) ->
  eval a av ->
  cunfolds mfix idx = Some (narg, fn) ->
  (#|argsv| < narg) ->
  eval (tApp f a) (tApp (mkApps (tFix mfix idx) argsv) av)

(** Fix unfolding, without guard *)
| eval_fix' f mfix idx a av fn res narg :
  forall unguarded : with_guarded_fix = false,
  eval f (tFix mfix idx) ->
  cunfolds mfix idx = Some (narg, fn) ->
  eval a av ->
  eval (tApp fn av) res ->
  eval (tApp f a) res
Economical proofs needed

1. “This theorem is true.”
2. “There is a proof of this theorem.”
3. “The proof of this theorem can be formalised.”
4. “The proof of this theorem can be formalised in less than a week.”

All problems are solvable by talking about observational congruence.

All problems are easy to solve for terms where all constructors and fixpoints are eta-expanded.

Both restructuring of proofs and proof engineering are central (missing UI features, missing nested induction principles: talk to Tomas Vallejos)
Aspects of a machine-checked intermediate language for extraction from Coq, in MetaCoq
Towards Verified Extraction from Coq to OCaml

Give a verified implementation of extraction

- formalise Coq in Coq
- formalise (a variant of) OCaml
- re-implement extraction
- verify it

https://metacoq.github.io

Joint work with Matthieu Sozeau, Pierre Giraud, Pierre-Marie Pedrót, and Nicolas Tabareau
Merci !

https://metacoq.github.io