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 λ□

N° d'ordre : 7567

Thèse de doctorat

présentée à

L'Université de Paris-Sud

U.F.R. Scientifique d'Orsay

par

PIERRE LETOUZEY

pour obtenir

le grade de docteur en sciences
de l'Université de Paris XI Orsay
spécialité : Informatique

Sujet :

Programmation fonctionnelle certifiée

L'extraction de programmes dans l'assistant Coq

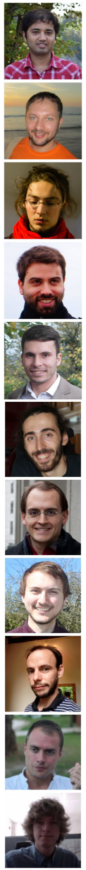
Soutenue le 9 juillet 2004 devant la commission d'examen composée de

M.	LEROY Xavier	président
M.	BERARDI Stefano	rapporteurs
M.	MONIN Jean-François	
Mme	BENZAKEN Véronique	examinateurs
M.	SCHWICHTENBERG Helmut	
Mme	PAULIN Christine	directeur

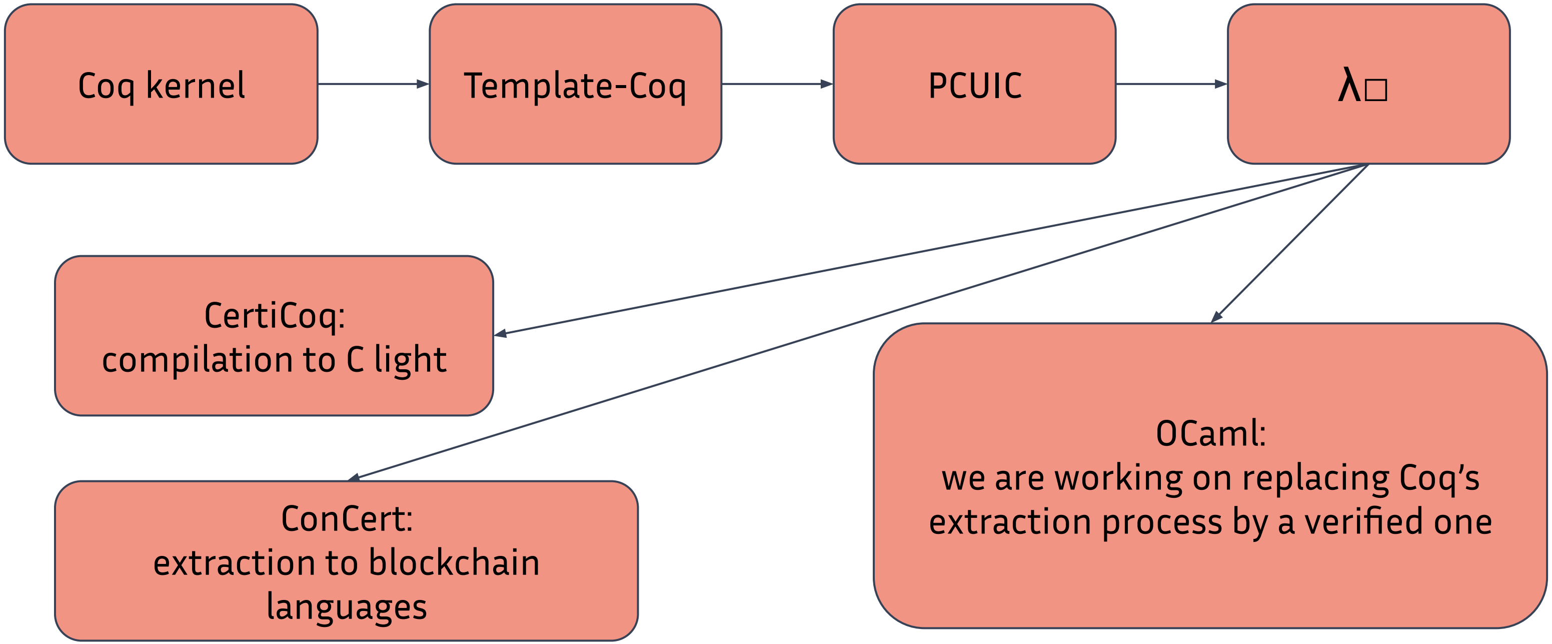
Inria

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



Subtle differences

Coq / lambda box

- structural fix
- higher-order constructors
- match on □
- de Bruijn
- fix / match

OCaml / C representation

- unary let rec
- constructors are blocks
- cannot match on functions
- named
- let rec / switch / proj

uniform solutions necessary,

to avoid doing work twice for different targets

Flags

```
(** 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 ->
    cunfold_fix 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 ->
    cunfold_fix 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) ->
  cunfold_fix mfix idx = Some (narg, fn) ->
  eval a av ->
  eval (tApp fn av) res ->
  eval (tApp f a) res
```

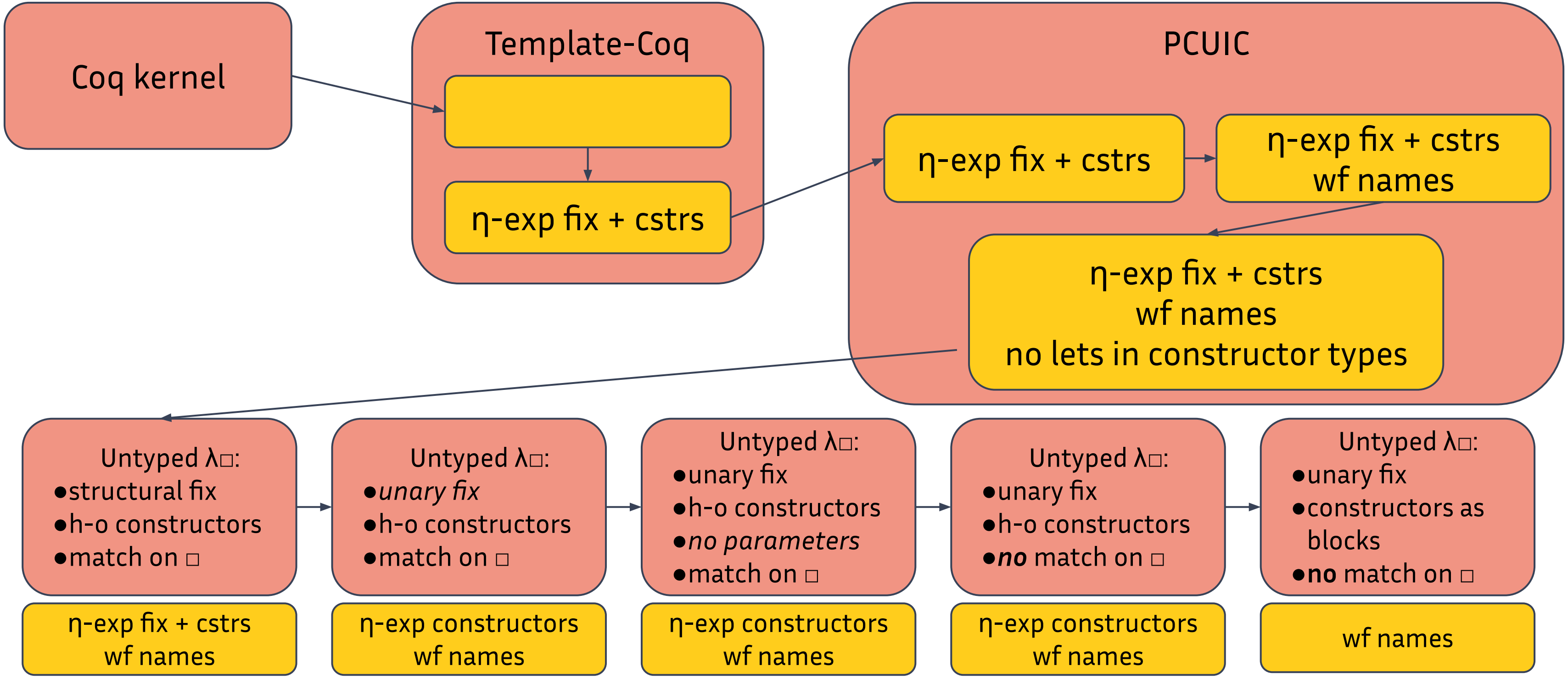
Economical proofs needed

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)


Coq kernel
Template-Coq
η-exp fix + cstrs
PCUIC
η-exp fix + cstrs
η-exp fix + cstrs
wf names
η-exp fix + cstrs
wf names
no lets in constructor types
Untyped λ□:
•structural fix
•h-o constructors
•match on □
η-exp fix + cstrs
wf names
Untyped λ□:
•*unary fix*
•h-o constructors
•match on □
η-exp constructors
wf names
Untyped λ□:
•unary fix
•h-o constructors
•*no parameters*
•match on □
η-exp constructors
wf names
Untyped λ□:
•unary fix
•h-o constructors
•***no*** match on □
η-exp constructors
wf names
Untyped λ□:
•unary fix
•constructors as blocks
•**no** match on □
wf names

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

Inria