Projects in the MetaCoq ecosystem

Yannick Forster[1], Meven Lennon-Bertrand[2], Kenji Maillard[1], Pierre-Marie Pédrot[1], Kazuhiko Sakaguchi[1], Matthieu Sozeau[1], Nicolas Tabareau[1], and Théo Winterhalter[3]

[1] Gallinette Project-Team, Inria, Nantes, France
[2] CLASH group, Computer Laboratory, Cambridge, UK
[3] Deducteam Project-Team, Inria, Saclay, France

= En bref / In short

Laboratoire, institution / Laboratory, institution: Inria, Université de Nantes (LS2N).
Lieu du stage / Location: LS2N, UFR Sciences et Techniques, 2, rue de la Houssinière, Nantes.
Équipe d'accueil / Host team: Équipe-projet Inria Gallinette.
Contact: Yannick Forster (e-mail)
Encadrement / Supervision: The internship will take place in the Inria Gallinette team. It will be co-supervised, depending on the project, by a team of the mentioned authors. If an external supervisor is involved, a visit can be arranged and can likely be fully funded.
Mots clefs / Keywords: Type theory, Coq, type theory in type theory, type checking, extraction, meta-programming.
Langue de travail / Working language: English or French.

= Description, in English

Context

The MetaCoq project [2, 3] is a formalisation of Coq in Coq [6], which can be used as basis for meta-programming of tactics and commands, to prove meta-theoretic properties of the type theory of Coq such as subject reduction, and to verify programs crucial in the implementation of Coq such as type checking or extraction.

We offer various projects on MetaCoq, spanning research areas of programming language theory, type theory, interactive theorem proving, compiler correctness, meta-programming, and implementation of user interfaces. The below list is non-exhaustive and contains just possible general directions that we will gladly expand on upon request. Interested students are encouraged to contact Yannick Forster or any other mentioned author to discuss more details regarding the projects or possible projects not mentioned on the list below.

Objectives

- **Meta programming**
  - **Meta-programming interfaces for variables:** Meta-programming in MetaCoq currently requires dealing with de Bruijn indices, as in the OCaml implementation of Coq itself. Various other approaches to variables exist, among them HOAS (as used in Coq-Elpi), PHOAS, named, and well-scoped de Bruijn (i.e. having `term : nat -> Type`). We would like to investigate these techniques in the context of MetaCoq, and use the insights to develop a robust library for meta-programming with variables in MetaCoq.

  - **Univalent parametricity:** The (univalent) parametricity translation [5] allows automatic reasoning modulo equivalences. We want to provide a MetaCoq-based implementation of the translation, which would take as input various proved equivalences and then be able to transport definitions such as (recursive) functions and theorems along those equivalences.
  - **Generating graph relations for functions:** The Equations [4] plugin can generate the graph predicate of a function defined using Equations automatically. For manually defined functions, a user has to define the graph relation manually, which can be tedious, as observed e.g. here. We would like to implement such a translation using MetaCoq's meta-programming facilities.
  - **Support for nested inductive types:** Currently, the strict positivity check implemented in MetaCoq rejects all nested inductive types. We would like to implement the same check Coq performs in MetaCoq, to help making the requirements more transparent. As a next step, we would like to implement a translation of inductive types to their (strict) subterm relation, supporting nested inductive types as well.

- **Meta theory**
  - **The "quoting lemma" for various notions of reduction / evaluation:** A central result in the meta-theory of type theories, especially for syntactic models of type theory, is a "quoting lemma" that any reduction of a term t_1 to a term t_2 can be reified into an internal proof of inhabitation of the type modeling reduction. We would like to prove this result for increasingly larger subsystem of MetaCoq, and finally for MetaCoq's notions of reduction and weak call-by-value evaluation.
  - **Specification of the guard checker:** MetaCoq currently relies on an abstract guard condition to ensure termination of fixpoints, which is not implemented. We would like to design such a specification, and show that the assumptions MetaCoq currently makes about it, e.g. its interaction with substitution, hold. Next, an implementation should be given, and shown to correspond to this specification. This would serve as an accessible mathematical specification to Coq's guard checker. We could then work in future steps on showing that a program satisfying the guard condition can be translated to a theory with a weaker form of recursion [1].

- **Extraction**
  - **Optimisations:** Coq's extraction process to OCaml contains various optimisations that can be en- and disabled by the user. The optimisations are empirically correct, but no proofs exist. In this project, the students will choose one of the existing optimisations, re-implement them in MetaCoq, and prove their correctness.
  - **Primitive integers, floats, and arrays:** The operational theory of CIC specified in MetaCoq currently does not cover primitive integers, floats, and arrays. We would like to change the specification to include primitives, change the type checker to be able to type check terms with primitives, extend type and proof erasure, and finally extend extraction and its correctness proof accordingly.

- **Software / proof engineering**
  - **Automatic annotations of patterns:** A frequent nuisance in big Coq developments such as MetaCoq is automatically chosen names. We would like to work on tools and UI features to annotate existing proofs to become more robust with regard to names, starting with a modification of the `induction` tactic to print the names it automatically picked, and extending this to a tool annotating all uses of `induction`, `destruct`, `inversion`, etc with patterns.

References

[1] Eduardo Giménez. Structural recursive definitions in type theory. In *ICALP*, pages 397–408, 1998.

[2] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *Journal of Automated Reasoning*, 2020. URL: https://hal.inria.fr/hal-02167423, doi:10.1007/s10817-019-09540-0.

[3] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! Verification of type checking and erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, 2019. doi:10.1145/3371076.

[4] Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in Coq. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019. doi:10.1145/3341690.

[5] Nicolas Tabareau, Éric Tanter, and Matthieu Sozeau. The marriage of univalence and parametricity. *Journal of the ACM*, 68(1):1–44, February 2021. doi:10.1145/3429979.

[6] The Coq Development Team. The Coq proof assistant, September 2022. doi:10.5281/zenodo.7313584.