Formalizing Coq Modules in the MetaCoq project

XFC4101 Final Report

Yee-Jian Tan

March 25, 2023

For Rodolphe and Nicolas, who changed my life

Abstract

This is an abstract. I am formalizing Coq modules in MetaCoq, it is a missing piece of the already established MetaCoq project. In the second part, as a requirement for the mathematics requirement, I study some basic properties of lambda calculus, hopefully culminating in the decidability for conversion in real-world type systems such as the Martin-Lof Type Theory (MLTT).

Acknowledgements

I wouldn't come this far with the help of many people along the way. I would like to thank especially Nicolas Tabareau for having me in Nantes for the summer of 2022, which changed my life. Thank you professors Yang Yue and Martin Henz of the National University of Singapore for being my advisors for this project, even though it a rare one — externally proposed, cross-faculty, joint final year project.

I would also like to thank the people who inadvertently made this possible — Thomas Tan Chee Kun who gave me a foray into the field of Type Theory, then Rodolphe Lepigre who was an amazing mentor landing me into a summer of internship. Not forgetting the valuable friendship and advice I received in Gallinette team, for the people whom I interacted with: Pierre, Tomas, Yann, Arthur, Iwan, Kenji, Meven, Assia, Pierre-Marie, Hamza, Enzo, Yannick and everyone else that appeared during my stay in Gallinette. I truly loved the time there and can't wait to be back again.

Contents

Introduction 1

The Coq Proof Assistant is one of the most prominent proof assistants, with applications ranging from formalizing a mathematical theory with the Homotopy Type Theory (HoTT)'s univalent foundation of mathematics [**DBLP:journals/corr/BauerGLSSS16**], to a framework to verify a computational theory via the Iris proof mode for Concurrent Separation Logic [**jung2018iris**], to a practical large-scale verification project: the recent recipient of the 2021 ACM Software System Award — the CompCert compiler [1]. Although the Coq Proof Assistant is the frameworking making these projects possible, one might ask: "Who watches the watchers?" [2] Even though the theory behind Coq, the Polymorphic, Cumulative Calculus of Inductive Constructions is known be consistent and strongly normalizing (hence proof-checking is decidable), but the OCaml implementation of Coq is known to have an average of one critical bug per year which allows one to prove False statements in Coq.

DBLP:journals/corr/BauerGLSSS16

jung2018iris

1: `https://awards.acm.org/software-system`

2: From latin: *Quis Custodiet ipsos custodes?*

The MetaCoq project [3] is therefore started by the Gallinette Team in INRIA, to "formalize Coq in Coq" and acts as a platform to interact with Coq's terms directly, in a verified manner; an immediate application is to verify the correctness of the implementation of Coq. This also reduces the trust in the implementation of Coq to the correctness of the theories underlying Coq, moving from a "trusted code base" to a "trusted theory base". In 2020, the core language of Coq, minus a few features are already successfully verified in Coq [**coqcoqcorrect**]. However, there are still a few missing pieces not yet verified, among them the Module system of Coq. The Module system, although not part of the core calculus of Coq, is an important feature for Coq developers to develop in a modular fashion, providing massive abstraction and a suitable interface for reusing definitions and theorems.

3: `https://metacoq.github.io`

coqcoqcorrect

In this project, I aim to extend the MetaCoq formalization of Coq by formalizing the Module system of Coq in the MetaCoq project framework, by first understanding the implementation of Modules, and then providing a specification of the implementation of Coq modules, finally writing proofs to show the correctness of the current implementation. In particular, I will focus on the formalization of non-parametrized, plain modules.

In order to tackle this project, it is important to understand the theory behind the implementation of Coq; more explicitly, how Type Theory helps in theorem proving. I study under the supervision of Professor Yang Yue some results of Type Theory; alongside the formalization of Modules jointly supervised by Professor Nicolas Tabareau and Professor Martin Henz.

The first section of this report will give a brief mathematical overview of the relationship between Type Theory and Theorem Proving. Following that, I will give an introduction to the MetaCoq project, the language of Coq and Coq Modules, before describing some related works and finally a specification for Coq Modules which I will implement.

A Modular Global Environment 2

Appendix

Typing Rules for Coq Modules | A

Typing rules here are given by Coq: Typing Modules.

A.1 Overview

This appendix is divided into 5 parts. Section **??** will first define the necessary terminology. Then, sections **??** to **??** will define the typing rules of Coq Modules. They correspond to the 4 operations defined on modules:

1. **Formation Rules** (**??**) describes how to form modules.
2. **Evaluation Rules** (**??**) describes how to evaluate specific terms involving modules: with-expressions and path-expressions.
3. **Access Rules** (**??**) describes how to access contents of modules (i.e. definitions in modules) using pathnames.
4. **Typing Rules** (**??**) describes how a module is inductively typed.

A.1.1 Definitions

A structure is ... A module type is ... FIXME:

A.1.2 Judgement Rules

A.1.3 Variables

E represents environment, S represents structures, ...

A.2 Formation Rules

WF-Struct:

$$\frac{\mathrm{WF}(E, E')[]}{E[] \vdash \mathrm{WF}(\mathrm{Struct}\ E'\ \mathrm{End})}$$

A structure can be formed from the tail of a well-formed environment.

WF-Mod1

$$\frac{\mathrm{WF}(E)[] \qquad E[] \vdash \mathrm{WF}(S)}{\mathrm{WF}(E; \mathrm{Mod}(X : S))[]}$$

If structure S is well-formed under the well-formed environment E, then the environment containing the Module X of type S is well-formed.

WF-Mod2

$$\frac{E[] \vdash S_2 <: S_1 \qquad \mathrm{WF}(E)[] \qquad E[] \vdash \mathrm{WF}(S_1) \qquad E[] \vdash \mathrm{WF}(S_2)}{\mathrm{WF}(E; \mathrm{Mod}(X : S_1 := S_2))[]}$$

The previous also works if X is of type S_2, a subtype of S_1.

WF-Alias

$$\frac{\mathrm{WF}(E)[] \qquad E[] \vdash p : S}{\mathrm{WF}(E; \mathrm{ModA}(X == p))[]}$$

If the module pointed by p has type S in the global environment E, then we can create an aliased module X to p. This implies that $p \in E$.

WF-ModType

$$\frac{\mathrm{WF}(E)[] \qquad E[] \vdash \mathrm{WF}(S)}{\mathrm{WF}(E; \mathrm{ModType}(Y := S))[]}$$

A ModType is a named well-formed structure.

WF-Ind

$$\frac{\begin{array}{c}\mathrm{WF}(E; \mathrm{Ind}[r](\Gamma_I := \Gamma_C))[] \\ E[] \vdash p : \mathrm{Struct}\ e_1; \ldots; e_n; \mathrm{Ind}[r](\Gamma_I^{'} := \Gamma_C^{'}); \ldots\ \mathrm{End} \\ E[] \vdash \mathrm{Ind}[r](\Gamma_I^{'} := \Gamma_C^{'}) <: \mathrm{Ind}[r](\Gamma_I := \Gamma_C)\end{array}}{\mathrm{WF}(E; \mathrm{Ind}_p[r](\Gamma_I := \Gamma_C))[]}$$

An inductive definition strenghtened by p is well-formed if it is already well-formed in E and it is within p. This can be generalized with subtyping.

A.3 Evaluation Rules

A.3.1 Evaluating `with` expressions

WEval-With-Mod:

$$\frac{\begin{array}{c}E[] \vdash S \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{Mod}(X : S_1); e_{i+2}; \ldots; e_n\ \mathrm{End} \\ E; e_1; \ldots; e_i[] \vdash S_1 \longrightarrow \overline{S_1} \qquad E[] \vdash p : S_2 \qquad E; e_1; \ldots; e_i[] \vdash S_2 <: \overline{S_1}\end{array}}{\begin{array}{c}E[] \vdash S \text{ with } X := p \longrightarrow \\ \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{ModA}(X == p); e_{i+2}\{X/p\}; \ldots; e_n\{X/p\}\ \mathrm{End}\end{array}}$$

WEval-With-Mod-Rec:

$$\frac{\begin{array}{c}E[] \vdash S \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{Mod}(X : S_1); e_{i+2}; \ldots; e_n\ \mathrm{End} \\ E; e_1; \ldots; e_i[] \vdash S_1 \text{ with } p := p_1 \longrightarrow \overline{S_2}\end{array}}{\begin{array}{c}E[] \vdash S \text{ with } X_1.p := p_1 \longrightarrow \\ \mathrm{Struct}\ e_1; \ldots; e_i;\quad \mathrm{mod}\ X : \overline{S_2}; e_{i+2}\{X.p/p_1\}; \ldots; e_n\{X.p/p_1\}\ \mathrm{End}\end{array}}$$

WEval-With-Def

This is the base case.

$$\frac{\begin{array}{c}E[] \vdash S \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; (c : T_1); e_{i+2}; \ldots; e_n\ \mathrm{End} \\ E; e_1; \ldots; e_i[] \vdash (c := t : T) <: (c : T_1)\end{array}}{E[] \vdash S \text{ with } c := t : T \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; (c := t : T); e_{i+2}; \ldots; e_n\ \mathrm{End}}$$

WEval-With-Def-Rec

$$\frac{\begin{array}{c}E[] \vdash S \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{Mod}(X_1 : S_1); e_{i+2}; \ldots; e_n\ \mathrm{End} \\ E; e_1; \ldots; e_i[] \vdash S_1 \text{ with } p := p_1 \longrightarrow \overline{S_2}\end{array}}{\begin{array}{c}E[] \vdash S \text{ with } X_1.p := t : T \longrightarrow \\ \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{Mod}(X : \overline{S_2}); e_{i+2}; \ldots; e_n\ \mathrm{End}\end{array}}$$

A.3.2 Evaluating paths ($p.X$)

WEval-Path-Mod1

$$\frac{\begin{array}{c}E[] \vdash p \longrightarrow \mathrm{Struct}\ e_1; \ldots; e_i; \mathrm{Mod}(X : S[:= S_1]); e_{i+2}; \ldots; e_n\ \mathrm{End} \\ E; e_1; \ldots; e_i[] \vdash S \longrightarrow \overline{S}\end{array}}{E[] \vdash p.X \longrightarrow \overline{S}}$$

WEval-Path-Mod2

$$\frac{\mathrm{WF}(E)[] \qquad \mathrm{Mod}(X : S[:= S_1]) \in E \qquad E[] \vdash S \rightarrow \overline{S}}{E[] \vdash X \rightarrow \overline{S}}$$

WEval-Path-Alias1

$$\frac{\begin{array}{c} E[] \vdash p \rightarrow \text{Struct } e_1; \dots; e_i; \mathrm{ModA}(X == p_1); e_{i+2}; \dots; e_n \text{ End} \\ E; e_1; \dots; e_i[] \vdash p_1 \rightarrow \overline{S} \end{array}}{E[] \vdash p.X \rightarrow \overline{S}}$$

WEval-Path-Alias2

$$\frac{\mathrm{WF}(E)[] \qquad \mathrm{ModA}(X == p_1) \in E \qquad E[] \vdash p_1 \rightarrow \overline{S}}{E[] \vdash X \rightarrow \overline{S}}$$

WEval-Path-Type1

$$\frac{\begin{array}{c} E[] \vdash p \rightarrow \text{Struct } e_1; \dots; e_i; \mathrm{ModType}(Y := S); e_{i+2}; \dots; e_n \text{ End} \\ E; e_1; \dots; e_i[] \vdash S \rightarrow \overline{S} \end{array}}{E[] \vdash p.Y \rightarrow \overline{S}}$$

WEval-Path-Type2

$$\frac{\mathrm{WF}(E)[] \qquad \mathrm{ModType}(Y := S) \in E \qquad E[] \vdash S \rightarrow \overline{S}}{E[] \vdash Y \rightarrow \overline{S}}$$

A.4 Access Rules

A.5 Typing Rules

MT-Eval

$$\frac{E[] \vdash p \rightarrow \overline{S}}{E[] \vdash p : \overline{S}}$$

MT-Str

$$\frac{E[] \vdash p \rightarrow \overline{S}}{E[] \vdash p : S/p}$$

Where S/p is the strengthening operation, defined on S where $S \rightarrow$ Struct $e_1; \dots; e_n$ End as:

$$\begin{aligned}
(c := t : T)/p &= (c := t : T) \\
(c : U)/p &= (c := p.c : U) \\
\mathrm{Mod}(X : S)/p &= \mathrm{ModA}(X == p.X) \\
\mathrm{ModA}(X == p')/p &= \mathrm{ModA}(X == p') \\
\mathrm{Ind}[r](\Gamma_I := \Gamma_C)/p &= \mathrm{Ind}_p[r](\Gamma_I := \Gamma_C) \\
\mathrm{Ind}_{p'}[r](\Gamma_I := \Gamma_C)/p &= \mathrm{Ind}_{p'}[r](\Gamma_I := \Gamma_C)
\end{aligned}$$

It strengthens the definition of inductives to equal to the current path only, solving the issue of equality of defined inductive types in modules.

A.5.1 Subtyping rules

MSub-Str

$$\frac{\begin{array}{c}E; e_1; \ldots; e_n[] \vdash e_{\sigma(i)} <: e'_i \text{ for } i = 1, \ldots, m \\ \sigma : \{1, \ldots, m\} \to \{1, \ldots, n\} \text{ injective}\end{array}}{E[] \vdash \text{Struct } e_1; \ldots; e_n \text{ End} <: \text{Struct } e'_1; \ldots; e'_m \text{ End}}$$

Mod-Mod:

$$\frac{E[] \vdash S_1 <: S_2}{E[] \vdash \text{Mod}(X : S_1) <: \text{Mod}(X : S_2)}$$

Alias-Mod:

$$\frac{E[] \vdash p : S_1 E[] \vdash S_1 <: S_2}{E[] \vdash \text{ModA}(X == p) <: \text{Mod}(X : S_2)}$$

Mod-Alias:

$$\frac{E[] \vdash p : S_2 E[] \vdash S_1 <: S_2 E[] \vdash X =_{\beta\delta\iota\zeta\eta} p}{E[] \vdash \text{Mod}(X : S_1) <: \text{ModA}(X == p)}$$

Alias-Alias

$$\frac{E[] \vdash p_1 =_{\beta\delta\iota\zeta\eta} p_2}{E[] \vdash \text{ModA}(X == p_1) <: \text{ModA}(X == p_2)}$$

Modtype-Modtype

$$\frac{E[] \vdash S_1 <: S_2 E[] \vdash S_2 <: S_1}{E[] \vdash \text{ModType}(Y := S_1) <: \text{ModType}(Y := S_2)}$$

Structure element subtyping rules

Assum-Assum

$$\frac{E[] \vdash T_1 \leq_{\beta\delta\iota\zeta\eta} T_2}{E[] \vdash (c : T_1) <: (c : T_2)}$$

Def-Assum

$$\frac{E[] \vdash T_1 \leq_{\beta\delta\iota\zeta\eta} T_2}{E[] \vdash (c := t : T_1) <: (c : T_2)}$$

Assum-Def

$$\frac{E[] \vdash T_1 \leq_{\beta\delta\iota\zeta\eta} T_2 \qquad E[] \vdash c =_{\beta\delta\iota\zeta\eta} t_2}{E[] \vdash (c : T_1) <: (c := t_2 : T_2)}$$

Def-Def

$$\frac{E[] \vdash T_1 \leq_{\beta\delta\iota\zeta\eta} T_2 \qquad E[] \vdash t_1 =_{\beta\delta\iota\zeta\eta} t_2}{E[] \vdash (c := t_1 : T_1) <: (c := t_2 : T_2)}$$

Ind-Ind

$$\frac{E[] \vdash \Gamma_I =_{\beta\delta\iota\zeta\eta} \Gamma'_I \qquad E[\Gamma_I] \vdash \Gamma_C =_{\beta\delta\iota\zeta\eta} \Gamma'_C}{\text{Ind}[r](\Gamma_I := \Gamma_C) <: \text{Ind}[r](\Gamma'_I := \Gamma'_C)}$$

Indp-Ind

$$\frac{E[] \vdash \Gamma_I =_{\beta\delta\iota\zeta\eta} \Gamma'_I \qquad E[\Gamma_I] \vdash \Gamma_C =_{\beta\delta\iota\zeta\eta} \Gamma'_C}{\mathrm{Ind}_p[r](\Gamma_I := \Gamma_C) <: \mathrm{Ind}[r](\Gamma'_I := \Gamma'_C)}$$

Indp-Indp

$$\frac{E[] \vdash \Gamma_I =_{\beta\delta\iota\zeta\eta} \Gamma'_I \qquad E[\Gamma_I] \vdash \Gamma_C =_{\beta\delta\iota\zeta\eta} \Gamma'_C \qquad E[] \vdash p = \beta\delta\iota\zeta\eta p'}{\mathrm{Ind}_p[r](\Gamma_I := \Gamma_C) <: \mathrm{Ind}_{p'}[r](\Gamma'_I := \Gamma'_C)}$$