Formalizing Coq Modules in the MetaCoq project

IMPRS-TRUST Application Talk

Yee-Jian Tan

February 1, 2023

Contents

Background

Background

- May-Jul 2022 (after 4th year undergraduate), interned at the Gallinette team in Nantes, France, advised by Nicolas Tabareau.
- Chosen among 3 possible projects: Modules, Parametricity Plugin, Eta-Reduction
- From Aug 2022 onwards: bachelor's thesis in the National University of Singapore: co-advised by Nicolas Tabareau (INRIA), Martin Henz (NUS Computing) and Yue Yang (NUS Mathematics).

Motivation

Motivation

- Coq is a proof assistant based on Type Theory. In particular, the **P**olymorphic, **Cu**mulative Calculus of (co-)**I**nductive **C**onstructions (PCUIC).
- We trust the theory to be correct: PCUIC is proven strongly normalizing, reduction and conversion are shown to be decidable…

Motivation

- Coq is a proof assistant based on Type Theory. In particular, the **P**olymorphic, **Cu**mulative Calculus of (co-)**I**nductive **C**onstructions (PCUIC).
- We trust the theory to be correct: PCUIC is proven strongly normalizing, reduction and conversion are shown to be decidable…
- But when we use Coq, we use the OCaml implementation.

Motivation

- Coq is a proof assistant based on Type Theory. In particular, the **P**olymorphic, **Cu**mulative Calculus of (co-)**I**nductive **C**onstructions (PCUIC).
- We trust the theory to be correct: PCUIC is proven strongly normalizing, reduction and conversion are shown to be decidable...
- But when we use Coq, we use the OCaml implementation.
- Is this implementation also reliable? Correct?

Bugs

- On average, one critical bug a year!
- Critical meaning possibly threatening the correctness of proved theorems - proving False in Coq!

Bugs

- On average, one critical bug a year!
- Critical meaning possibly threatening the correctness of proved theorems - proving False in Coq!
- MetaCoq project in 2018 as a metaprogramming platform for Coq, building on existing systems such as TemplateCoq (2014).
- Bonus: formalize the theory of Coq, PCUIC, using MetaCoq.

Bugs

- On average, one critical bug a year!
- Critical meaning possibly threatening the correctness of proved theorems - proving False in Coq!
- MetaCoq project in 2018 as a metaprogramming platform for Coq, building on existing systems such as TemplateCoq (2014).
- Bonus: formalize the theory of Coq, PCUIC, using MetaCoq.
- Slogan: the Coq implementation you run every week/month.

What is the theory of Coq?

The type theory behind the Coq proof assistant is PCUIC:

- (Universe) **P**olymorphic
- **Cu**mulative
- Calculus of (co-)**I**nductive **C**onstructions (CoC with Inductive Types)
- Calculus of Constructions (Coquand and Huet, 1988)

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code
- Coq - **TemplateCoq** - PCUIC - Machine Code

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code
- Coq - **TemplateCoq** - PCUIC - Machine Code
- Coq - TemplateCoq - PCUIC - **Checker & Erasure** - Machine Code

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code
- Coq - **TemplateCoq** - PCUIC - Machine Code
- Coq - TemplateCoq - PCUIC - **Checker & Erasure** - Machine Code
- Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq (2020)

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code
- Coq - **TemplateCoq** - PCUIC - Machine Code
- Coq - TemplateCoq - PCUIC - **Checker & Erasure** - Machine Code
- Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq (2020)
- Caveats:

MetaCoq - An Overview

- From high level to low level:
- Coq - PCUIC - Machine Code
- Coq - **TemplateCoq** - PCUIC - Machine Code
- Coq - TemplateCoq - PCUIC - **Checker & Erasure** - Machine Code
- Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq (2020)
- Caveats:
    1. Eta-expansion
    2. Template Polymorphism
    3. SProps (Proof-irrelevant Propositions)
    4. **Modules**

Coq Modules

Example - Definitions

Modules as “collections of definitions”.

```
Inductive nat :=
| O
| S : nat -> nat.

Fixpoint plus (n m: nat) :=
    match n with
    | S n' => S (plus n' m)
    | O => m
    end.
```

Example - Modules

"Packaging" definitions into a Module (Type).

```
(* A magma is a set with a binary (closed) operation. *)
Module Type Magma.
    Parameter T: Set.
    Parameter op: T -> T -> T.
End Magma.


Module Nat: Magma.
    Definition T := nat.
    Definition op := plus.
End Nat.
```

Example - Aliasing

Modules can be aliased for ease of reference.

```
Module Type M := Magma.
Module MyNat: M := Nat.
```

Example - Functors

Higher-order modules - Functors.

```
(* A functor transforming a magma into another magma. *)
Module DoubleMagma (M: Magma): Magma.
    Definition T := M.T.
    Definition op x y := M.op (M.op x y) (M.op x y).
End DoubleMagma.

Module NatWithDoublePlus := DoubleMagma Nat.
```

Towards a Specification of Plain Modules

- There are many contention/issues regarding the semantics of functors: OpenIssuesWithModules

Towards a Specification of Plain Modules

- There are many contention/issues regarding the semantics of functors: OpenIssuesWithModules
- I only work on **plain** (non-parametrized) modules and module types, aliasing.

Towards a Specification of Plain Modules

- There are many contention/issues regarding the semantics of functors: OpenIssuesWithModules
- I only work on **plain** (non-parametrized) modules and module types, aliasing.
- 2nd class object; cannot pass around as a term
- Restricted range of operations
    1. Declaration/Definition (including aliasing)
    2. Using modules: access definitions in it using path names
    3. Refining a module: create new module by replacing entries of existing

Representation of Global Environment and Modules

- Before: global environment is a list of name-definition pairs for constants and inductive types.

[1] Slightly different from named function application for lambda calculus with constants: plus 2 $3 \rightarrow_{\delta} 5$

Representation of Global Environment and Modules

- Before: global environment is a list of name-definition pairs for constants and inductive types.
- After: Modules are tree-like structures: they contain definitions of constants, inductives, modules, and module types.

[1] Slightly different from named function application for lambda calculus with constants: plus 2 3$\rightarrow_\delta$ 5

Representation of Global Environment and Modules

- Before: global environment is a list of name-definition pairs for constants and inductive types.
- After: Modules are tree-like structures: they contain definitions of constants, inductives, modules, and module types.
- Global environment is just a special case of a Module.
- Items in modules are referred to using pathnames (e.g. $M.a$) instead of defined names (e.g. a). The resolution of path name to its definition extends the δ-reduction[1] in Coq.
- **Goal: Formalize the above in TemplateCoq.**

[1] Slightly different from named function application for lambda calculus with constants: plus 2 $3 \rightarrow_\delta 5$

Some Properties to Verify

1. Definition: well-definedness of this structure: follow the Typing (Formation) Rules.
2. Definition-Alias: well-definedness of aliasing: no cycles.
3. Path: path points to the correct definition.
4. Path-Alias: aliased path points to the correct definition.
5. Related theorems about environment, typing, translation etc.

Example: TemplateToPCUICCorrectness.v

```
Lemma trans_lookup_module
  {cf} {Σ} {cst: kername} m {wfΣ : Typing.wf Σ}:
  (* If cst refers to a module m in the environment Σ, *)
  Ast.Env.lookup_env Σ cst = Some (Ast.Env.ModuleDecl m)
  (* Then one cannot find cst in the translated
  environment. *)
  -> lookup_env (trans_global_env Σ) cst = None.
Proof.
  destruct Σ as [univs decls retro].
  intros H.
  cbn -[fold_right].
```

Current Work

Current Work

1. Formalize modules (data structures, typing rules...)
2. From Coq to TemplateCoq (OCaml plugin)
3. From TemplateCoq To PCUIC (elaborating modules away, correctness...)
4. Verify Properties

Thank You!