Towards Formalising the Guard Condition of Coq

Yee-Jian Tan
Inria, Institute Polytechnique
Paris

Yannick Forster
Inria
Paris

Abstract

Coq's consistency — and the consistency of constructive type theories in general — crucially depends on all recursive functions being terminating. Technically, in Coq, this is ensured by a so-called guard checker, which is part of Coq's kernel and ensures that fixpoints defined on inductive types are structurally recursive. Coq's guard checker has been first implemented in the 1990s by Christine Paulin-Mohring, but was subsequently extended, adapted, and fixed by several contributors. As a result, there is no exact, abstract specification of it, meaning for instance that formal proofs about it are out of reach. We propose a talk on our ongoing work-in-progress project to synthesise a specification of Coq's guard checker as a guard condition predicate defined in Coq itself on top of the MetaCoq project. We hope that our project will benefit the users of Coq by providing an accurate and transparent description, as well as lay the foundations for future improvements or even mechanised consistency proofs of Coq.

In Coq, one can define inductive types which have parameters, be indexed, mutual, or nested. Common examples are natural numbers, lists (which have a parameter), vectors (which have indices), even/odd predicates (which are mutual), the accessibility predicate (which has nesting via a function type), and rose trees (which have nesting via inductive types). One can then define so-called fixpoints by structural recursion on elements of these inductive types. As an example here is a definition of the recursor on natural numbers as a fixpoint:

```
Fixpoint nat_rec (P : nat -> Set) (p0 : P O) (ps : forall (m: nat), P m -> P (S m))
   (n : nat) {struct n}: P n :=
     match n with
       | O => p0
       | S m => ps m (nat_rec P p0 ps m)
     end.
```

All of the mentioned features like mutuality and nesting introduce mostly orthogonal complexity in the definition of Coq's guard checker, which is defined in the `kernel/inductive.ml` file of Coq's code in about 1000 lines of OCaml code. To illustrate how a guard checker is crucial for consistency, one can look at the following two non-terminating fixpoints

```
Unset Guard Checking.
Fixpoint boom (x : nat) {struct x} : False := boom x.

Fixpoint inf (n : nat) {struct n} :=
   match n with
    | O => 0
    | S _ => S (inf n)
   end.
```

They allow proofs of `False` as `boom 0` and exploiting the contradictory property of `inf 1 = S (inf 1)`.

History of the guard condition

In the 1990s, Frank Pfenning and Christine Paulin-Mohring introduced inductive types in the calculus of constructions [1] along with a guard condition for fixpoints [2]. The first version of Coq's Guard Checker was implemented in Coq v5.10.2 by Paulin-Mohring in 1994 [3]. In 2012, Pierre Boutillier relaxed the guard checker via β-ι commutative cuts [4]. In 2014, Maxime Dénès restricted the guard checker to forbid an unwanted proof that propositional extensionality does not hold in Coq [5]. In 2022, Hugo Herbelin restricted the guard checker to ensure strong normalisation rather than just weak normalisation, which is a behaviour that seems to be more in line with the intuition of users [6]. In 2024, Herbelin introduced a relaxation, allowing simpler implementation of nested recursive functions [7]. Furthermore, changes to the guard condition keep being proposed, see e.g. [8] and [9].

MetaCoq: a formalisation of Coq in Coq

Two central parts of the MetaCoq project [10] are a formalisation of Coq's type theory in Coq [11] and, on top of that, verified implementations of a type checker [12] and a verified extraction function to OCaml [13]. The formalisation of type-theory faithfully captures typing rules of Coq in an inductive predicate, and is parameterised in a guard checker function which is requried to fulfil some basic properties such as being stable under reduction and substitution. Based on this formalisation, crucial properties such as subject reduction (types are preserved by reduction) and canonicity (normal forms of inductive types start with a constructor) are proved [12].

The verified type checker axiomatically assumes that reduction in the system is strongly normalising. The strong normalisation assumption can also be used to prove consistency, because any proof of `False` would have a normal form using strong normalisation, which would have the same type using subject reduction, and would start with a constructor of the inductive type `False` using canonicity – which is a contradiction, because `False` has no constructors.

An implementation of Coq's Guard Checker in Coq

In the talk, we will report on a the current state of the project: a full and faithful implementation of Coq's guard checker in Coq using the MetaCoq project, available on https://github.com/inria-cambium/m1-tan/tree/v1.0.0. We will also explain the workings of the guard checker via the data structures it uses and present its different dimensions of complexity, due to its many contributions by different authors.

Towards a verified guard checker

As future work, the first step to verifying the guard checker would be synthesising a guard condition predicate from the current OCaml implementation of the guard checker. This predicate will work similarly to the typing predicate in MetaCoq, i.e. talk about the syntax of Coq as specified in MetaCoq. This specification can be an inductive predicate, meaning it does not have to be obviosly decidable.

As a second step, a guard checker function deciding the guard condition predicate should be defined. This function will have to use MetaCoq's verified implementation of reduction, meaning it will have to rely on the strong normalisation axiom. Technically, a substantial challenge will lie in proving that the definition of this guard checker function, defined in Coq and working on syntax as specified by MetaCoq, is terminating. That this function indeed correctly computes the guard condition predicate can either be immediately proved by using dependent types and a correct-by-construction guard checker function, or in a separate additional step. Note that technically, proving the soundness of the function suffices, i.e. that whenever the function accepts a term, the guard condition predicate holds.

Towards normalisation proofs

Having a formal description of the guard condition now allows relative normalisation proofs of Coq's type theory in Coq. Namely, it then is feasible to define a simpler, more natural, and more modular variant of guard condition and show that Coq's type theory with the current implementation of the guard condition can be interpreted in this simpler theory. This would result in a relative normalisation proof, meaning that Coq's type theory is normalising (and thus consistent) if the simpler system is normalising. In particular, this means that the trust in Coq's consistency could be moved to trusting that such a simpler theory is terminating. In the long term future, it then even could be possible to *prove* termination of Coq's type theory (for a restricted number of universes to get around Gödel incompleteness issues) by reducing it in many steps to an extension of the currently also ongoing formalisation of MLTT in Coq [14].

Related Work

Termination checking in Agda is done semantically via sized types: a special, implicit type used by the type checker to determine if the recursion done on a strictly smaller argument [15, 16]. Lean, on the other hand, only has recursors (or eliminators) in its type theory, thus user-written recursive functions are represented in the kernel by recursors [17, 18].

Bibliography

[1] F. Pfenning and C. Paulin-Mohring, "Inductively Defined Types in the Calculus of Constructions," in *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings*, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, Eds., in Lecture Notes in Computer Science, vol. 442. Springer, 1989, pp. 209–228. doi: 10.1007/BFB0040259.

[2] C. Paulin-Mohring, *Définitions Inductives en Théorie des Types. (Inductive Definitions in Type Theory)*. 1996. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00431817

[3] C. Cornes *et al.*, "The Coq Proof Assistant-Reference Manual," *INRIA Rocquencourt and ENS Lyon, version*, vol. 5, 1996.

[4] P. Boutillier, "A relaxation of Coq's guard condition," in *JFLA - Journées Francophones des langages applicatifs - 2012*, Feb. 2012, pp. 1–14. [Online]. Available: https://hal.science/hal-00651780

[5] M. Dénès, "Tentative fix for the commutative cut subterm rule." [Online]. Available: https://github.com/coq/coq/commit/9b272a861bc3263c69b699cd2ac40ab2606543fa

[6] H. Herbelin, "Check guardedness of fixpoints also in erasable subterms." [Online]. Available: https://github.com/coq/coq/pull/15434

[7] H. Herbelin, "Extrude uniform parameters of inner fixpoints in guard condition check." [Online]. Available: https://github.com/coq/coq/pull/17986

[8] H. Herbelin, "Size-preserving dependent elimination," 30th International Conference on Types for Proofs, Programs, TYPES 2024, pp. 35–37, 2024. [Online]. Available: https://types2024.itu.dk/abstracts.pdf

[9] H. Herbelin, "How much do System T recursors lift to dependent types?," 30th International Conference on Types for Proofs, Programs, TYPES 2024, pp. 38–39, 2024. [Online]. Available: https://types2024.itu.dk/abstracts.pdf

[10] M. Sozeau *et al.*, "The MetaCoq Project," *J. Autom. Reason.*, vol. 64, no. 5, pp. 947–999, 2020, doi: 10.1007/S10817-019-09540-0.

[11] M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter, "Coq Coq correct! verification of type checking and erasure for Coq, in Coq," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, pp. 1–28, 2020, doi: 10.1145/3371076.

[12] M. Sozeau, Y. Forster, M. Lennon-Bertrand, J. B. Nielsen, N. Tabareau, and T. Winterhalter, "Correct and Complete Type Checking and Certified Erasure for Coq, in Coq," Apr. 2023. [Online]. Available: https://inria.hal.science/hal-04077552

[13] Y. Forster, M. Sozeau, and N. Tabareau, "Verified Extraction from Coq to OCaml," *Proc. ACM Program. Lang.*, vol. 8, no. PLDI, Jun. 2024, doi: 10.1145/3656379.

[14] A. Adjedj, M. Lennon-Bertrand, K. Maillard, P.-M. Pédrot, and L. Pujet, "Martin-Löf à la Coq," in *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, A. Timany, D. Traytel, B. Pientka, and S. Blazy, Eds., ACM, 2024, pp. 230–245. doi: 10.1145/3636501.3636951.

[15] C. S. Lee, N. D. Jones, and A. M. Ben-Amram, "The size-change principle for program termination," in *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, C. Hankin and D. Schmidt, Eds., ACM, 2001, pp. 81–92. doi: 10.1145/360204.360210.

[16] A. Abel and T. Altenkirch, "A predicative analysis of structural recursion," *J. Funct. Program.*, vol. 12, no. 1, pp. 1–41, 2002, doi: 10.1017/S0956796801004191.

[17] L. de Moura and S. Ullrich, "The Lean 4 Theorem Prover and Programming Language," in *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, A. Platzer and G. Sutcliffe, Eds., in Lecture Notes in Computer Science, vol. 12699. Springer, 2021, pp. 625–635. doi: 10.1007/978-3-030-79876-5_37.

[18] J. Breitner, "Reference for equations compiler using brecOn." [Online]. Available: https://leanprover.zulipchat.com/#narrow/stream/270676-lean4/topic/Reference.20for.20equations.20compiler.20using.20brecOn/near/465564128