Towards Formalising the Guard Checker of Rocq

Yee-Jian Tan[1,2] and Yannick Forster[1]

[1] Inria Paris, France
[2] Institut Polytechnique de Paris, France

Abstract

Rocq's consistency — and the consistency of constructive type theories in general — crucially depends on all recursive functions being terminating. Technically, in Rocq, this is ensured by a so-called guard checker, which is part of Rocq's kernel and ensures that fixpoints defined on inductive types are structurally recursive. Rocq's guard checker was 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 a first step preceeding the formalisation of the guard checker, namely, a faithful implementation of the guard checker of Rocq in Rocq, using the MetaRocq framework. We hope that our project will benefit the users of Rocq by providing an accurate and transparent implementation, as well as lay the foundations for future formalisation efforts or even mechanised consistency proofs of Rocq.

In Rocq, one can define inductive types which have parameters, are 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 0) (ps : ∀ (m: nat), P m → P (S m))
  (n : nat) {struct n}: P n :=
   match n with
     | 0 ⇒ p0
     | S m ⇒ ps m (nat_rec P p0 ps m)
   end.
```

All the mentioned features like mutuality and nesting introduce mostly orthogonal complexity in the definition of Rocq's guard checker, which is defined in the `kernel/inductive.ml` file of Rocq'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
   | 0 ⇒ 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 [15] along with a guard condition for fixpoints [14]. The first version of Rocq's Guard Checker was implemented in Rocq v5.10.2 by Paulin-Mohring in 1994 [5]. In 2012, Pierre Boutillier relaxed the guard checker via β-ι commutative cuts [3]. In 2014, Maxime Dénès restricted the guard checker to forbid an unwanted proof that propositional extensionality does not hold in Rocq [7]. 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 [9]. In 2024, Herbelin introduced a relaxation, allowing simpler implementation of nested recursive functions [10]. Furthermore, changes to the guard condition keep being proposed, see e.g. [11] and [12].

MetaRocq: a formalisation of Rocq in Rocq Two central parts of the MetaRocq project [16] are a formalisation of Rocq's type theory in Rocq [17] and, on top of that, verified implementations of a type checker [18] and a verified extraction function to OCaml [8]. The formalisation of type-theory faithfully captures typing rules of Rocq in an inductive predicate, and is parameterised in a guard checker function which is required 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 [18].

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 Rocq's Guard Checker in Rocq In the talk, we will report on the current state of the project: a full and faithful implementation of Rocq's guard checker in Rocq using the MetaRocq project [19], whose code can also be found online. 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 MetaRocq, i.e. talk about the syntax of Rocq as specified in MetaRocq. This specification can be an inductive predicate, meaning it does not have to be obviously decidable.

As a second step, a guard checker function deciding the guard condition predicate should be defined. This function will have to use MetaRocq'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 Rocq and working on syntax as specified by MetaRocq, 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 Rocq's type theory in Rocq. Namely, it then is feasible to define a simpler, more natural, and more modular variant of the guard condition and show that Rocq'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 Rocq's type theory is normalising (and thus consistent) if the simpler system is normalising. In particular, this means that the trust in Rocq'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 Rocq'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 Rocq [2].

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 [1, 13]. 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 [4, 6].

References

[1] Andreas Abel and Thorsten Altenkirch. A predicative analysis of structural recursion. *J. Funct. Program.*, 12(1):1–41, 2002. doi:10.1017/S0956796801004191.

[2] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-löf à la coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 230–245. ACM, 2024. doi:10.1145/3636501.3636951.

[3] Pierre Boutillier. A relaxation of Coq's guard condition. In *JFLA - Journées Francophones des langages applicatifs - 2012*, pages 1 – 14, Carnac, France, February 2012. URL: https://hal.science/hal-00651780.

[4] Joachim Breitner. Reference for equations compiler using brecon. https://leanprover.zulipchat.com/#narrow/stream/270676-lean4/topic/Reference.20for.20equations.20compiler.20using.20brecOn/near/465564128, 2024. URL: https://leanprover.zulipchat.com/#narrow/stream/270676-lean4/topic/Reference.20for.20equations.20compiler.20using.20brecOn/near/465564128.

[5] Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Gérard Huet, Pascal Manoury, César Munoz, Chetan Murthy, Catherine Parent, Christine Paulin-Mohring, Amokrane Saibi, and Benjamin Werner. The coq proof assistant-reference manual. *INRIA Rocquencourt and ENS Lyon, version*, 5, 1996.

[6] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer, 2021. doi:10.1007/978-3-030-79876-5_37.

[7] Maxime Dénès. Tentative fix for the commutative cut subterm rule. https://github.com/coq/coq/commit/9b272a861bc3263c69b699cd2ac40ab2606543fa, 2014. URL: https://github.com/coq/coq/commit/9b272a861bc3263c69b699cd2ac40ab2606543fa.

[8] Yannick Forster, Matthieu Sozeau, and Nicolas Tabareau. Verified extraction from coq to ocaml. *Proc. ACM Program. Lang.*, 8(PLDI), jun 2024. doi:10.1145/3656379.

[9] Hugo Herbelin. Check guardedness of fixpoints also in erasable subterms. https://github.com/coq/coq/pull/15434, 2022. URL: https://github.com/coq/coq/pull/15434.

[10] Hugo Herbelin. Extrude uniform parameters of inner fixpoints in guard condition check. https://github.com/coq/coq/pull/17986, 2024. URL: https://github.com/coq/coq/pull/17986.

[11] Hugo Herbelin. How much do system t recursors lift to dependent types? 2024. URL: https://types2024.itu.dk/abstracts.pdf.

[12] Hugo Herbelin. Size-preserving dependent elimination. 2024. URL: https://types2024.itu.dk/abstracts.pdf.

[13] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001. doi:10.1145/360204.360210.

[14] Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types. (Inductive Definitions in Type Theory)*. 1996. URL: https://tel.archives-ouvertes.fr/tel-00431817.

[15] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the calculus of constructions. In Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings*, volume 442 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 1989. URL: https://doi.org/10.1007/BFb0040259, doi:10.1007/BFB0040259.

[16] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The metacoq project. *J. Autom. Reason.*, 64(5):947–999, 2020. URL: https://doi.org/10.1007/s10817-019-09540-0, doi:10.1007/S10817-019-09540-0.

[17] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq coq correct! verification of type checking and erasure for coq, in coq. *Proc. ACM Program. Lang.*, 4(POPL):8:1–8:28, 2020. doi:10.1145/3371076.

[18] Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Botsch Nielsen, Nicolas Tabareau, and Théo

Winterhalter. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq. working paper or preprint, April 2023. URL: https://inria.hal.science/hal-04077552.

[19] Yee-Jian Tan and Yannick Forster. Towards Formalising the Guard Checker of Coq. Technical report, Ecole polytechnique ; Inria - Paris, March 2025. URL: https://inria.hal.science/hal-04983786.