# Monadic Equational Reasoning for `while` loop in Rocq

Ryuji Kawakami[1], Jacques Garrigue[1], Takahumi Saikawa[1], and Reynald Affeldt[2]

[1] Nagoya University, Japan
[2] National Institute of Advanced Industrial Science and Technology (AIST), Japan

**Monadic equational reasoning.** Pure functional programs can be reasoned about using equational reasoning thanks to their referential transparency. For programs containing computational effects, Gibbons and Hinze [GH11] proposed monadic equational reasoning, which extends equational reasoning to the verification of programs designed around monads. The interface of each monad is defined as a collection of operators and equations, that allow to manipulate effects.

**Monae.** Monae [ANS19] is a library that enables verification using monadic equational reasonig in Rocq. Monae consists of interfaces and models. The models guarantee the consistency of the interfaces. Rocq guarantees the correctness of the verification, and the math library MathComp/SSReflect [GM10] makes it possible to write concise proofs. Monae implements a hierarchy of interfaces using Hierarchy Builder [CST20], and allows for the combination of multiple monads and reusable lemmas. Dijkstra monads [SHK+16] also provide an alternative formal framework to verify monadic programs, albeit using Hoare logic rather than equational reasoning.

**Non-structurally recursive functions.** Proof assistants such as Rocq, which allow for the reduction of programs, do not permit the definition of non-terminating functions to guarantee consistency. Rocq's `Fixpoint` command can only define structurally recursive functions. For non-structural recursion, it is necessary to use an additional accessibility predicate, corresponding to a well-founded order, either directly or through the `Function` or `Equations` commands. For example, McCarthy's 91 function `mc91`, which performs complex recursion, cannot be defined through direct structural recursion in Rocq.

```
let rec mc91 m = if 100 < m then m - 10 else mc91 (mc91 (m + 11))
```

Moreover, functions whose termination is unknown, such as the Collatz predicate, cannot be defined as recursive functions in Rocq.

**Defining functions containing `while` statements by a coinductive type.** Another way of dealing with non-structurally recursive functions in such proof assistants is to use corecursive definitions, which allow for defining infinite sequences of data. Since Rocq allows an infinite number of constructor applications as long as the guard constraint is satisfied, it is possible to use corecursive definitions to make recursive calls without limit.

The Delay monad proposed by Capretta [Cap05] can be used to represent such corecursive functions as monadic programs. Interaction Trees [SZ21] are a natural generalization of the delay monad to represent non-structurally recursive functions with events. In our work, by defining the interface to the Delay monad as a complete Elgot monad [AMV10], we are able to reason about functions with `while` statements that need not be guaranteed to terminate, such as McCarthy's 91 function and the Collatz predicate.

**Complete Elgot monad.** The theory for complete Elgot monads corresponds to Iteration theory [BÉ93], which deals with recursive structures algebraically.

In our study, we use the function `while` to define a complete Elgot monad. It is defined using the `CoFixpoint` command, where the right embedded value `inr x` is the continuation of iterations with value `x` and the left embedded value `inl a` is the end of the iteration with the return value `a`.

```
CoFixpoint while {X A} (body: X -> M (A + X)) : X -> M A :=
  fun x => (body x) >>= (fun xa => match xa with
                                   | inr x => DLater (while body x)
                                   | inl a => DNow a end).
```

Uustalu and Veltri [UV17] have shown, by quotienting by an equivalence relation that ignores a finite number of computational steps, that the delay monad is a complete Elgot monad.

**Combining computational effects using monad transformers.** By using a complete Elgot monad to represent a `while` statement, we can handle functions that contain `while` statements, but only pure functions. For example, a factorial computed using references and `while` statements cannot be expressed using only a complete Elgot monad.

```
let fact n =
  let r = ref 1 in
  let l = ref 1 in



  while !l <= n do

    r := !r * !l;
    l := !l + 1;

  done;
  !r
```

```
Definition factdts n :=
  do r <- cnew ml_int 1;
  do l <- cnew ml_int 1;
  do _ <-
  while (fun (_:unit) =>
         do i <- cget l;
         if i <= n
         then do v <- cget r;
              do _ <- cput r (i * v);
              do _ <- cput l (i.+1);
              Ret (inr tt)
         else Ret (inl tt)) tt;
  do v <- cget r; Ret v.
```

We use monad transformers [AN20] to combine any complete Elgot monad with the exception monad and the typed store monad introduced to represent OCaml references [AGS25, Section 5]. In our work, we show the exception monad transformer, the store monad transformer and the typed store monad transformer preserve the complete Elgot monad structure. This allows verifying programs with multiple effects in Monae.

**Contribution.** Our contributions are as follows.

1. We define the interface of the delay monad as a complete Elgot monad and show its consistency. This allows Monae to verify functions containing `while` statements.

2. By using monad transformers for combination and the `setoid_rewrite` tactic for generalised rewriting, we have confirmed that verification with Monae is practical for functions involving `while` statements together with other effects.

The code for this work can be found at:

https://github.com/affeldt-aist/monae/pull/147

**Acknowledgments.** The authors acknowledge support of the JSPS KAKENHI Grants Number 22H00520 and Number 22K11902.

## References


[AGS25] Reynald Affeldt, Jacques Garrigue, and Takafumi Saikawa. A practical formalization of monadic equational reasoning in dependent-type theory. *Journal of Functional Programming*, 35:e1, 2025.

[AMV10] Jirí Adámek, Stefan Milius, and Jirí Velebil. Equational properties of iterative monads. *Information and Computation*, 208(12):1306–1348, 2010.

[AN20] Reynald Affeldt and David Nowak. Extending equational monadic reasoning with monad transformers. In *26th International Conference on Types for Proofs and Programs (TYPES 2020), March 2–5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 2:1–2:21, 2020.

[ANS19] Reynald Affeldt, David Nowak, and Takafumi Saikawa. A hierarchy of monadic effects for program verification using equational reasoning. In *13th International Conference on Mathematics of Program Construction (MPC 2019), Porto, Portugal, October 7–9, 2019*, volume 11825 of *LNCS*, pages 226–254. Springer, 2019.

[BÉ93] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories—The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993.

[Cap05] Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), 2005.

[CST20] Cyril Cohen, Kazuhiko Sakaguchi, and Enrico Tassi. Hierarchy Builder: Algebraic hierarchies made easy in Coq with Elpi (system description). In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), June 29–July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 34:1–34:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[GH11] Jeremy Gibbons and Ralf Hinze. Just do it: simple monadic equational reasoning. In *16th ACM SIGPLAN international conference on Functional Programming (ICFP 2011), Tokyo, Japan, September 19–21, 2011*, pages 2–14. ACM, 2011.

[GM10] Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formalized Reasoning*, 3(2):95–152, 2010.

[SHK+16] Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in F*. *Proc. ACM Program. Lang.*, 51(POPL):256–270, 2016.

[SZ21] Lucas Silver and Steve Zdancewic. Dijkstra monads forever: termination-sensitive specifications for interaction trees. 5(POPL):1–28, 2021.

[UV17] Tarmo Uustalu and Niccolò Veltri. The delay monad and restriction categories. In *14th International Colloquium on Theoretical Aspects of Computing (ICTAC 2017), Hanoi, Vietnam, October 23–27, 2017*, volume 10580 of *Lecture Notes in Computer Science*, pages 32–50. Springer, 2017.