Efficient coinductives through state-machine corecursors

Supervisor: Alex Keizer

Marking supervisor: Jamie Vicary

Director of studies: Russell Moore

Word count: 993

1 Work to be undertaken

Previous work has shown how coinductive types can be encoded in Lean as quotients of polynomial functors (QPFs)[1]. This is done through progressive approximation (Section 4.1.2). For example accessing the n index of a stream takes $\mathcal{O}(n^2)$ time. This becomes an issue when using Lean as a general purpose programming language. An alternative approach is using a state-machine based encoding (Section 4.1.1) of cofixed points. With these we get nice computational behaviour of the compiled code. In Figure 1 we can see in blue the performance from the current implementation, versus in orange the performance of a prototype implementation the state-machine based version.

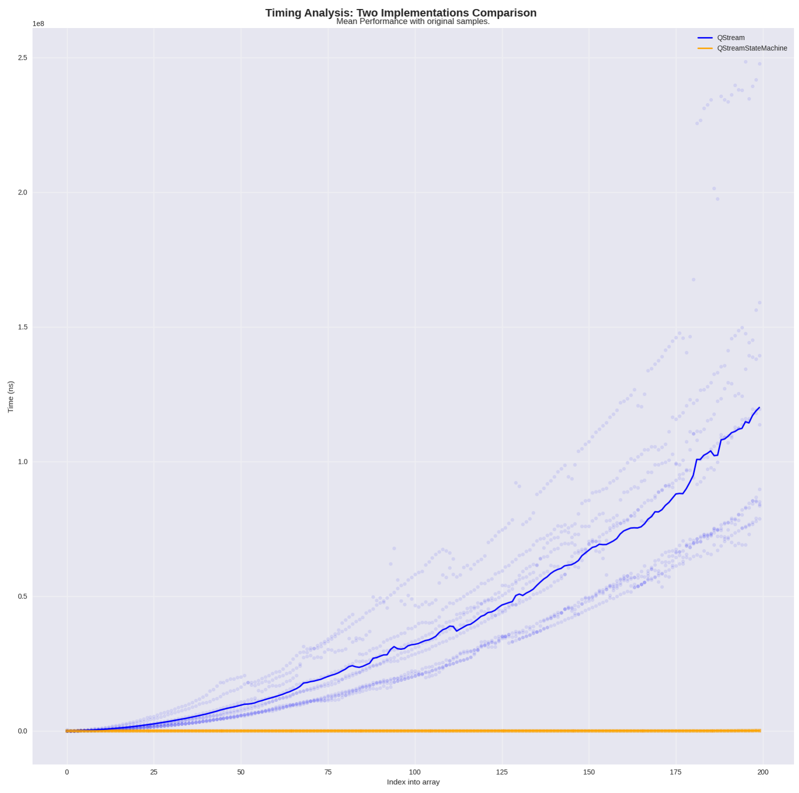


Figure 1: Graph plotting current performance v possible gains

2 Core project

This project will be implementing the state-machine encoding (Section 4.1.1) of coinductive types, and formalising the equivalence between these two engodings (Section 4.3). I will start with the special example of equivalence between `Streams` in the two representations, then building up to the general cofix structure.

After this I will be implementing a coinductive monad allowing us to encode non-termination as an effect using the state-machine representation. This will be an instance of the two representations.

3 Starting point

I have worked with QPFs on the meta-programming side for an internship between my Part Ia and Part Ib. During this I learnt of the basics of polynomial functors, and `TypeVecs`. This means I am aware of what the underlying structures are when it comes to the raw implementation. I have also done a feasibility assessment of the project by seeing how the current polynomials respond to universe levels. This lead to me making 2 pull requests (#28095, #28279) to mathlib on `TypeVec` in preparation for my project. I also tried to merge (#28112) an implementation of commutable Shrink to mathlib that later got reverted when it was found to be inconsistent.

There are more minor refactoring pull requests towards the mathlib repository which don't change any behaviour but in general all of these can be *found on this link*. I Include these for completeness and transparency.

4 Substance

Section 4.1.1, Section 4.3 and Section 4.4 are all discuss structures that will be implemented throughout the project. Section 4.1.2 and Section 4.2 both are implemented in Avigard et al [1].

4.1 The M type

The M type is the name given to the terminal coalgebra of polynomial functors; the possibly infinitely deep trees. This means the implementations must have both: A a corecursor $\text{corec} : \{\alpha : \text{Type}\} \to (f : \alpha \to F\alpha) \to \alpha \to MF$, and a destructurer $\text{dest} : \text{M}F \to F(MF)$. We have two encodings of the M type.

4.1.1 State-machine encoding

The state-machine encoding is the naïve way to implement the terminal coalgebra. Given some polynomial functor F, the state-machine encoding is given by: some type $\alpha : \text{Type}$, a function $f : \alpha \to F\alpha$, and some witness $a : \alpha$. Then you quotient over bisimilarity, thereby only allowing direct usage of $\text{dest} : \text{Sme } F \to F(\text{Sme } F)$. The baseline implementation of this should be straight-forward.

4.1.2 Progressive approximation encoding

The other way is to generate them by progressive approximation where earlier trees must "agree" with the later ones. Agreement is given by them being the same up to the previous depth as seen in Figure 2.

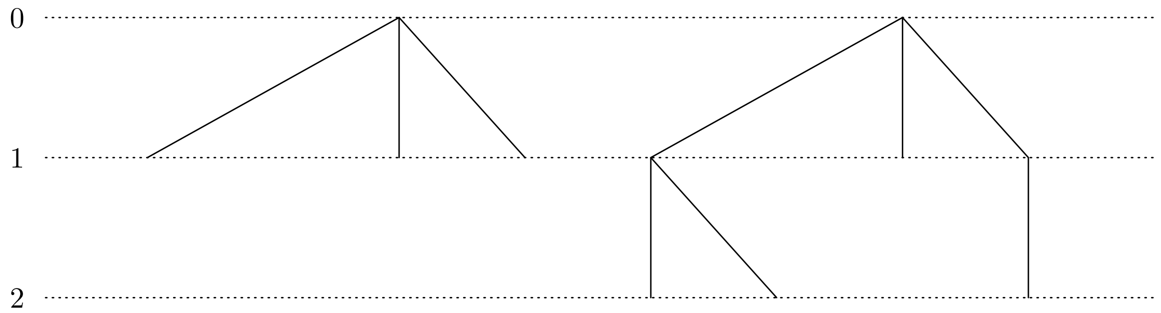


Figure 2: Agreement of two trees of height 1 and 2

4.2 The `Cofix` type

`Cofix` is the terminal coalgebra in QPFs, These are simply a quotient over M types lifting the quotient from the source QPF.

4.3 The equivalence

The equivalence between the current implementation of `Cofix` and M and the state-machine representation is the core of the project. The functions in both directions are given by f : $\text{corec}_{\text{Pae}}\text{dest}_{\text{Sme}}$ and f^{-1} : $\text{corec}_{\text{Sme}}\text{dest}_{\text{Pae}}$. The difficulty comes from proving $f \circ f^{-1} = \mathbb{1}$ and $f^{-1} \circ f = \mathbb{1}$. The proof of these equialities will be proven using bisimilarity, and other parts of the theory currently established for M types. Simpler versions of this equivalence also exist for the simpler representation.

4.4 The non-termination monad

The non-termination monad is a simple example of a coinductive. This will be useful for testing the state-machine representation's performance. The structure of the monad as a coinductive has two constructors as seen in this psudocode:

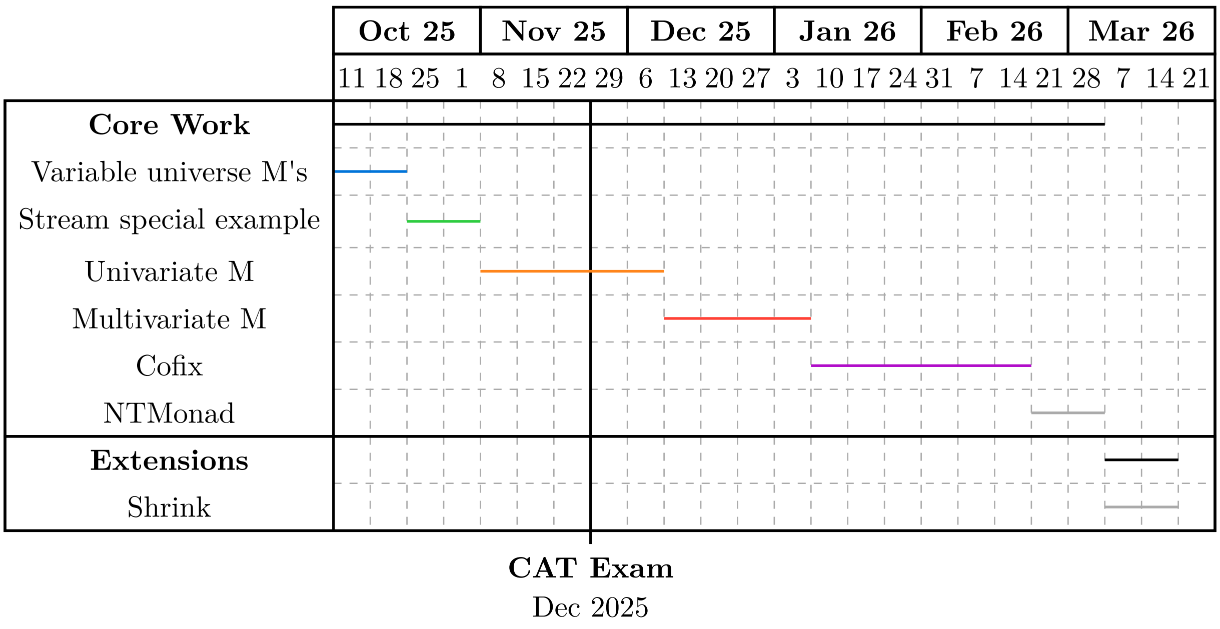
```
coinductive NTMonad (A : Type)
  | val : A -> NTMonad A
  | tau : NTMonad A -> NTMonad A
```

5 Evaluation

This projects success can be broken down into a few disjoint critrea.

1. If the equivalence (Section 4.3) has been proven,
2. how close to the theorised performance we get with the state-machine encoding,
3. and the ability to construct the example structures.

6 Core timeline



The plan for work would be divided into a few different stages.

6.1 Variable universe `Ms` (2025-10-11 2w 2025-10-24)

To begin with I have to make the corecursor universe polymorphic. As stated in Section 3 I have most of this code written. I will attempt to get this merged into mathlib, but if this review process takes too long I will be working on my own branch.

6.2 `Stream` special example (2025-10-25 2w 2025-11-7)

The first step would be implementing `Stream` under the two representations. Then I will familiarise myself with using bisimilarity to formalise the equivalence. This is a first step to just understand how these features work.

6.3 Univariate `M` (2025-11-8 2w 2025-11-21 + (CAT exam) + 2025-12-05 3w 2025-12-23)

Implementing the univariate `M` type is the natural next step from this. This will be the natural next step in difficulty. The work should lay the groundwork for the next proofs.

6.4 Multivariate `M` (2025-12-26 4w 2026-01-23)

The next step will be generalising the univariate implementation, the same structure of the proof should be the same for the multivariate case. The main difficulty here comes from working with `TypeVecs`. These are quite difficult to reason about.

6.5 `Cofix` (2026-01-24 6w 2026-03-06)

Finally, the `Cofix` type has to be implemented. This will be the most challanging part as I will have to work with quotients.

6.6 Implementing the NTMonad (2026-03-07 2w 2026-03-20)

This will be a demonstration of all of the previous work.

7 Extensions

7.1 Shrinking the representations (2026-03-21 2w 2026-04-04)

This involves finding a sound way allowing the current theory to use the efficient representation. The soundness of this will be difficult to reason about as it requires working with the Lean code generator. Therefore I leave this as an extension.

8 Resources

Access to the restricted side of the lab is needed for working with the greater team.

Bibliography

[1] J. Avigad, M. Carneiro, and S. Hudon, “Data Types as Quotients of Polynomial Functors,” in *10th International Conference on Interactive Theorem Proving (ITP 2019)*, J. Harrison, J. O'Leary, and A. Tolmach, Eds., in Leibniz International Proceedings in Informatics (LIPIcs), vol. 141. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 6:1–6:19. doi: 10.4230/LIPIcs.ITP.2019.6.