Supervisor:

Marking supervisor: Tobias Grosser

Team: Alex Keizer, Meven Lennon-Bertrand

1 Work to be undertaken

Previous work has shown how coinductives can be encoded in lean as quotients of polynomial functors (QPFs). These are highly expressive but accessing cofixed points to a depth n takes $\mathcal{O}(n^2)$ time, which compounds when you map data m times to $\mathcal{O}(n^{2^m})$, this is a problem when using lean as a general purpose programming language. Previous work has focused on trying to achieve this performance within the same universe. An alternative approach is trying rather to use a quite exotic type Shrink which has the behaviour of allowing types in higher universes to be shrunk to the lower universes under certain circumstances. By showing an equivalence between the state-machine implementation of a cofixed point and the current implementation, we can instantiate Shrink. Doing this gives us the computational behaviour of the state-machine implementation, meaning the entire expression collapses to $\mathcal{O}(n)$ resulting in usable coinductive types for general purpose programming in Lean. A demonstration of the possible gains can be seen in Figure 1, where the x-axis is the index into the array and the y-axis is a ms duration

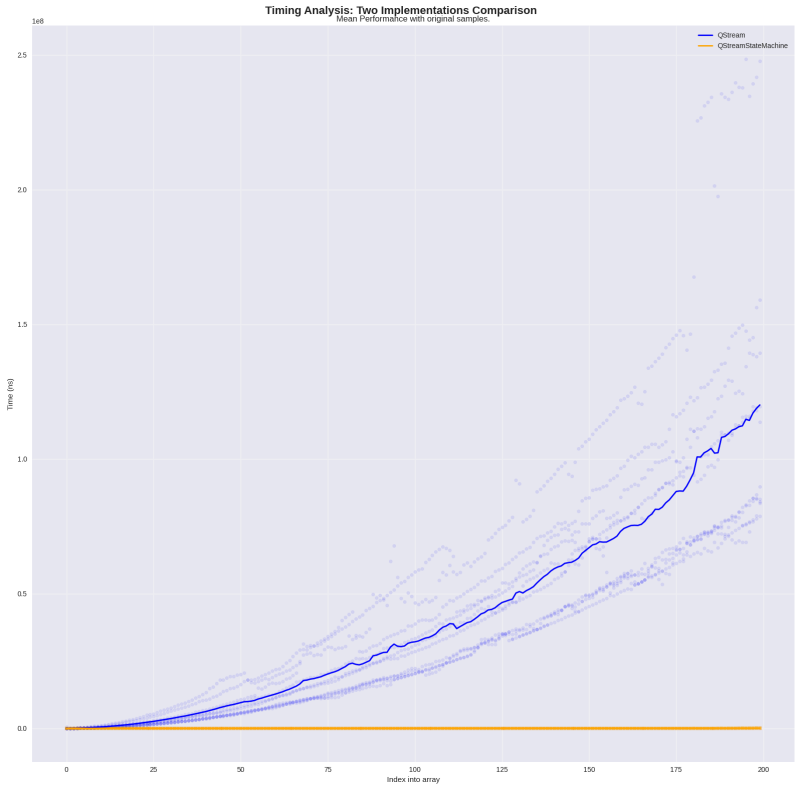

Figure 1: Graph plotting current performance v possible gains

2 Starting point

I have worked with QPFs on the meta-programming side for an internship between my Part Ia and Part Ib where 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 pheasability assesment of the project by seeing how the current polynomials respond to universe levels. This lead to me making 2 PRs (28095, 28279) to mathlib on TypeVec

in preperation for my project. I also tried to PR (28112) an implementation of computable Shrink to mathlib that later got reverted when it was found to be absurd.

Additionally my supervisors and I did a preliminary assessment to attempt to get a separate D2D supervisor on-board. This is what can be seen in Figure 1.

There are more minor refactoring PRs on `mathlib` 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.

3 Substance

There are a few structures that will be worked with during this project. Those are Section 3.1, Section 3.2 and Section 3.3.

3.1 `Shrink`

`Shrink` is temporarilly the choice used for doing the ABI translation between the two implementations. Given that the two types are equivalent then we can non-computably extract a model in the desired universe. Given two types $\alpha : \text{Type } u$ and $\beta : \text{Type } v$ for which an isomorphism exists, we can construct the type Shrink $\alpha\beta : \text{Type } v$ for which both diagrams in Figure 2 commute.

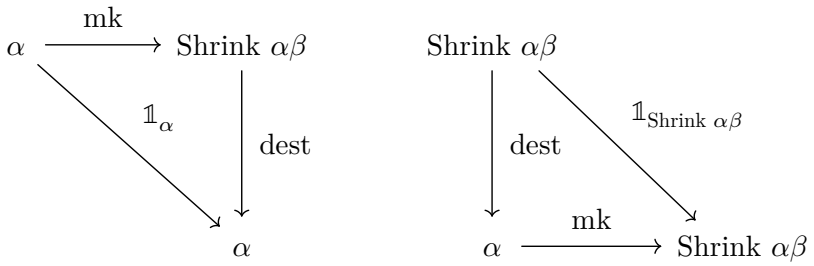

Figure 2: Operations on Shrink

An attempt was made to make this computable by using `unsafeCasts` but these had to be reverted. For now the exact type is undecided as there is a PR by the Lean FRO adding specialised types for this purpose.

3.2 `M`

The `M` type is the name given to the terminal coalgebra of polynomial functors; the possibly infinitely deep trees. These are generated 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. A visual example is given in Figure 3. We can have approximations for any n, thereby letting the trees take any depth including infinite depth. Trees can be terminiated by having no children as one might expect.

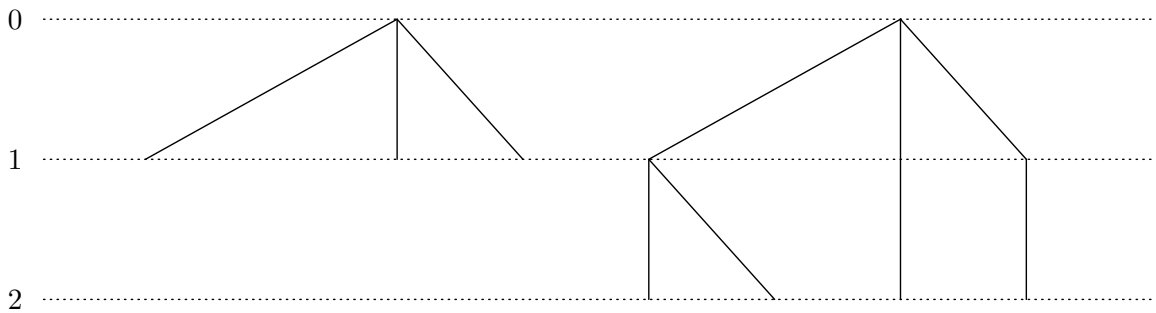

Figure 3: Agreement of two trees of height 1 and 2

The current QPF implementation has 2 `M` types; univariate and multivariate implementations. These both have the undesired computational behaviour. `M` types are polynomial

3.3 `Cofix`

`Cofix` is the terminal coalgebra in QPFs, (possibly) infinitely big quotiented trees. This is the slighly concerning part of the project and by far the highest risk section as working with `Quot` in Lean is a painful experiance.

4 Evaluation

The success of this project can be given by how close to the theorised performance we can get to. The goal would be getting to the same order or magnitude.

5 Core

The plan for work would be devided into a few different stages

5.1 Variable universe `M`s

To begin, the pull requests created from before the start of the project will have to be completed and merged into `mathlib`.

5.2 `List` special example

An early step would be to familizarise myself with using the bisimilarity features given by the `M` type to see how pheasable it is to prove equivilences of two `M` types with a simple functor.

5.3 Univariate `M`

After a special example I would move over to the univariate example. This will be much easier than the multivariate case as I dont have to suffer with `Typevec`s.

5.4 Multivariate `M`

This will be the next natural step. Will me much harder than the univariate.

5.5 `Cofix`

Finally, it has to be proven for `Cofix`. This will be hard as I will have to suffer with Quot which is really concerning to work with.

6 Extentions

6.1 A fast implementation of `Precoroutines`

Precoroutines are a type Alex Keizer is interested in. They generalize interaction trees and other similar datatypes useful for denotational purposes. These leverage some of the powers of QPFs. This should come for free from the prior the core.

7 Resources

Access to the restricted side of the lab would be nice to be able to work with the group. This is not strictly necessary but would be highly convinient.