Rational Codata as Syntax-with-Binding:
Correct-by-Construction Foundations of the Modal μ-Calculus*

Sean Watters

University of Strathclyde, UK
sean.watters@strath.ac.uk

The modal μ-calculus is an extension of the basic modal logic K with least and greatest fixpoint operators. It is of foundational importance in the field of model checking [1], and also of considerable theoretical interest to logicians for its rich properties. The model checking and satisfiability problems for the μ-calculus are decidable, yet the logic is highly expressive — it subsumes the temporal logics LTL, CTL, CTL* and propositional dynamic logic, and is equivalent to the bisimulation-invariant fragment of monadic second-order logic [5]. Of interest to the practising type theorist, the μ-calculus admits a constructive semantics in containers which is currently in the early stages of being explored [10].

This (still work-in-progress) project is concerned with formalising the correctness and finiteness proofs for a particular key syntactic construction on μ-calculus formulas, namely their (Fischer-Ladner) *closure*. This can be thought of as a succinct graph representation of the formula. The definition of the closure is not structurally recursive, and indeed, the proof of its finiteness (originally due to Kozen [6]) is fairly technical, particularly in a formal setting where issues like variable binding and substitution must be treated seriously.

A key feature of our work is the use of syntax-with-binding as an inductive presentation of rational codata — data structures that loop back on themselves — in a manner similar to the technique first demonstrated by Ghani et al [3], and Hamana [4].

We work in Agda, using guarded corecursion. At the time of submission, the bulk of the formalisation is complete (up to and including Theorem 1). Theorem 2 is still a work-in-progress.

The μ-Calculus Formulas in the modal μ-calculus are generated by the following grammar, parameterised by some set A of atomic propositions. We represent formulas in Agda as an inductive data type using well-scoped de Bruijn indices for the variables, but for convenience we write variable names here. Note the lack of implication or arbitrary negations; this is required to maintain strict positivity, which is important when giving the semantics of the fixpoint operators.

$$\mu\mathsf{ML} := \top \mid \bot \mid A \mid \neg A \mid \mu\mathsf{ML} \wedge \mu\mathsf{ML} \mid \mu\mathsf{ML} \vee \mu\mathsf{ML} \mid \Box\, \mu\mathsf{ML} \mid \Diamond\, \mu\mathsf{ML} \mid \mu x.\mu\mathsf{ML} \mid \nu x.\mu\mathsf{ML}$$

We can define substitution for these formulas in standard fashion, which allows us to define the *unfolding* of a fixpoint formula as its direct subformula with the outermost variable substituted for the whole formula: $\mathsf{unfold}\ (\mu x.\varphi) = \varphi\ [x := \mu x.\varphi]$.

The Closure The closure of a μ-calculus formula φ is the least set of formulas that contains φ and is closed under the "single-step closure relation", which relates modal and propositional formulas to their direct subformulas, and fixpoint formulas to their unfoldings. We define the membership relation for the closure $- \in_C -$ as the transitive reflexive closure of this single-step relation, where "$\psi \in_C \varphi$" means "ψ is in the closure of φ".

*Work-in-progress Agda formalisation available at https://github.com/Sean-Watters/mu-calc-de-bruijn

The membership relation $\in_C$ allows us to characterise the closure, but does not provide an obviously terminating algorithm to compute it. Note that it is not immediate from the definition of $\in_C$ that the closure graph only contains finitely many nodes, as the unfolding of φ is not structurally smaller than φ, and it is not immediately clear that a chain of unfoldings would eventually loop back on itself.

Kozen's original proof of finiteness involves an alternative definition of the closure which is structurally recursive. The key to it is the so-called *expansion map*, which can be thought of as the "maximal unfolding" — the expansion map sequentially instantiates every free variable for its binder, in order from most recently bound to least. This definition of the closure is finite by construction, but it is difficult to directly prove it equivalent to the standard definition in a formal setting.

Our approach is to first define two closure algorithms; one that produces an infinite cotree via guarded corecursion directly from the definition of $\in_C$ (which is trivially correct), and one via Kozen's inductive algorithm (which is intrisically finite). Crucially, the inductive data structure we use for the latter includes a notion of *back-edges*, which allows us to unfold it to an infinite codata structure. We then prove that the coinductive algorithm is bisimilar to the unfolding of the inductive one, and by doing so, obtain correctness and finiteness for both constructions.

Rational Codata Just as a rational number is one whose decimal expansion is finitely presentable as a prefix plus a loop, an element of a rational codata type is an infinite data structure for which every infinite path eventually falls into a repeating cycle. They arise via the *rational* fixpoint of endofuctors, which is a particular sub-coalgebra of the final coalgebra [8][9].

We argue that the correct way to think about the closure of a μ-calculus formula is as a rational codata structure. Inductive structures such as lists and trees forget the backedges, which are crucial to the properties we wish to prove, meanwhile simply using cotrees forgets any evidence we may have had about the closure being finitely presentable.

In the style of Ghani et al. [3], we represent the closure as an inductive "μ-calculus formula-shaped tree". That is: such trees store data at both nodes and leaves, nodes are labelled by formula constructors $\{\wedge, \vee, \Box, \Diamond, \mu, \nu\}$ (and have the appropriate arities), and we also have variables which we consider as pointers to $\{\mu, \nu\}$-nodes. Thus, this presentation is a syntax-with-binding. We define their unfolding to non-wellfounded cotrees (with the same node arities and labelling) via guarded corecursion.

Bisimilarity for cotrees is defined as the pointwise lifting of propositional equality, for which we create a DSL for compositional proof, following the technique first demonstrated by Danielsson [2]. We now have all we need to prove that the two definitions of the closure are equivalent:

Theorem 1. *The direct coinductive definition of the closure is bisimilar to the unfolding of the inductive syntax-with-binding definition.*

Our notions of correctness for each algorithm relate the paths through the tree produced to $\in_C$, in the sense that a path exists from φ to ψ iff $\psi \in_C \varphi$. For both kinds of tree, this is an instance of the "Any" predicate transformer (or rather, in the inductive case, a variant of it that allows backedge traversal). If we can transport proofs of Any across bisimulations of the form found in Theorem 1, then we can prove the finite-by-construction inductive algorithm correct. The proof of this fact is still work-in-progress.

Theorem 2 (WIP)**.** *Let T be the tree with backedges produced by the inductive closure algorithm applied to φ. Then for all formulas ψ there is a path to ψ in T iff $\psi \in_C \varphi$. That is, T really is the closure of φ.*

Conclusions and Future Work We believe this to be the first serious attempt at formalising the Fischer-Ladner closure of the modal μ-calculus in a proof assistant, though others have previously formalised different aspects of the μ-calculus in Agda [10], Rocq [7], and LEGO [11]. A natural next step in this programme would be to formalise the game semantics of the μ-calculus (where the closure plays a key role), and in doing so obtain a correct-by-construction model checker.

In our opinion, the modal μ-calculus and type theory are ripe for further cross-fertilisation. For example, we believe that rational codata types are currently under-explored, and that this work provides a good example of their utility. In particular, we would like to further investigate the inductive syntax-with-binding presentation and its unfolding, and whether these generalise usefully to containers. Our hope is to develop a type-theoretic framework for ergonomically reasoning about finitely-presentable infinite data.

References

[1] Julian Bradfield and Igor Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer International Publishing, Cham, 2018.

[2] Nils Anders Danielsson. Beating the productivity checker using embedded languages. *arXiv preprint arXiv:1012.4898*, 2010.

[3] Neil Ghani, Makoto Hamana, Tarmo Uustalu, and Varmo Vene. Representing cyclic structures as nested datatypes. In *Proc. of 7th Symp. on Trends in Functional Programming, TFP*, volume 2006, 2006.

[4] Makoto Hamana. Initial algebra semantics for cyclic sharing tree structures. *Logical Methods in Computer Science*, 6, 2010.

[5] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *International Conference on Concurrency Theory*, pages 263–277. Springer, 1996.

[6] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical computer science*, 27(3):333–354, 1983.

[7] Marino Miculan. On the formalization of the modal µ-calculus in the calculus of inductive constructions. *Inf. Comput.*, 164:199–231, 2001.

[8] Stefan Milius. A sound and complete calculus for finite stream circuits. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 421–430. IEEE, 2010.

[9] Stefan Milius. Proper functors and their rational fixed point. In *7th Conference on Algebra and Coalgebra in Computer Science (CALCO 2017)*, pages 18–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.

[10] Ivan Todorov and Casper Bach Poulsen. Modal μ-calculus for free in agda. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Type-Driven Development*, pages 16–28, 2024.

[11] Shenwei Yu and Zhaohui Luo. Implementing a model checker for lego. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME '97: Industrial Applications and Strengthened Foundations of Formal Methods*, pages 442–458, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.