Matching (Co)patterns with Cyclic Proofs

Lide Grotenhuis, University of Amsterdam
Daniël Otten, University of Amsterdam

Overview. We investigate connections between cyclic proof theory and recursive functions defined by (co)pattern matching. Cyclic proof systems replace (co)induction rules with sound forms of circular reasoning. For example, by adding a cycle between the green nodes we get a cyclic proof:

$$\dfrac{\dfrac{}{0+0=0}\,{+_0} \qquad \dfrac{\dfrac{}{0+\mathsf{suc}\,x'=\mathsf{suc}(0+x')}\,{+_{\mathsf{suc}}} \qquad \dfrac{0+x'=x'}{\mathsf{suc}(0+x')=\mathsf{suc}\,x'}\,\mathrm{cong}_{\mathsf{suc}}}{0+\mathsf{suc}\,x'=\mathsf{suc}\,x'}\,\mathrm{trans}}{0+x=x}\,\mathrm{case}_x,$$

This proof is sound because the variable x is decreased to its predecessor x' before we cycle back, and so it represents a proof by infinite descent. The advantage of these systems lies in proof search: to apply (co)induction we need to guess the right (co)induction hypothesis, whereas with cycles we can start generating the proof until our current goal matches one that we have seen before. Under the Curry-Howard correspondence, such cycles correspond to recursive function calls, while the soundness condition ensure that the function always terminates:

cyclic proof	recursive function
fixpoint formula	(co)inductive type
cycle	recursive function call
soundness condition	termination checking

In this way, recursive functions defined with dependent (co)pattern matching can be seen as a proof-relevant and dependent generalisation of cyclic proofs. In addition, there is a correspondence between type-based termination checking, in the form of sized-types, and ordinal approximations of fixpoint formulas. Our goal is to explain these correspondences and to use these connections to extend type theoretic conservativity results.

Proof Assistants. Proof assistants like `Agda`, `Dedukti`, `Rocq`, and `Lean` allow the user to define functions using recursive calls. Which functions are accepted depends on a unification algorithm, and this determines in particular whether we can prove axiom K. For both cases — with and without K — we have conservativity results: we can already implement these functions using the primitive rules (turn these cyclic proofs into (co)inductive proofs) [GMM06, CDP16, Thi20]. Such a reduction is needed to show that any function accepted by the proof assistant has an interpretation in any model of the type theory. However, these conservativity results only cover 'simple cycles': there is one inductive input that is decreased in every recursive call, or one coinductive output that is productive before every recursive call. Some proof assistants, like `Agda` and `Dedukti`, allow more complex interleaving of recursive calls. To implement these functions using primitive (co)induction rules, we need to apply induction and coinduction multiple times, and often in a specific order. For example, for the Ackermann function we need

a lexicographical order on inputs:

$$\begin{array}{l}\mathsf{ack} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}, \\ \mathsf{ack}\, m\, n \coloneqq \mathsf{case}\, m \begin{cases} 0 \mapsto \mathsf{suc}\, n, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{ack}\, m'\, 1, \\ \mathsf{suc}\, n' \mapsto \mathsf{ack}\, m'\, (\mathsf{ack}\, (\mathsf{suc}\, m')\, n'). \end{cases} \end{cases}\end{array}$$

While the following example mixes induction and coinduction:

$$\begin{array}{l}\mathsf{complog} : \mathbb{N} \to \mathbb{N} \to \mathsf{Stream}\, \mathbb{N}, \\ \mathsf{complog}\, t\, r \coloneqq \mathsf{case}\, t \begin{cases} 0 \mapsto \mathsf{record} \begin{cases} \mathsf{head} \mapsto r, \\ \mathsf{tail} \mapsto \mathsf{complog}\, r\, r, \end{cases} \\ \mathsf{suc}\, t' \mapsto \mathsf{complog}\, t'\, (r!). \end{cases}\end{array}$$

And this one uses induction on both inputs simultaneously:

$$\begin{array}{l}\mathsf{aggr} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}, \\ \mathsf{aggr}\, m\, n \coloneqq \mathsf{case}\, m \begin{cases} 0 \mapsto 0, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{suc}\, 0, \\ \mathsf{suc}\, n' \mapsto \mathsf{aggr}\, m'\, m' + \mathsf{aggr}\, n'\, n'. \end{cases} \end{cases}\end{array}$$

In addition, there are more complicated examples using dependent types and mutually recursive functions. The general soundness condition is the following: for any infinite sequence of function calls that might occur, there should eventually be an input/output that we can track, where progress is made infinitely often. This can either be an inductive input that is decreased infinitely often, or a coinductive output that is productive infinitely often. This is known as the size-change termination principle [LJBA01, Wah07], which can be implemented both with and without sized-types. Although this problem refers to infinite sequences, it can be shown to be PSPACE-complete, and can be solved using ω-automata or call graphs. This principle ensures termination, but it is no longer clear how the accepted functions can be implemented using primitive (co)induction rules.

Cyclic proofs. A similar problem has been studied in the cyclic proof theory literature [SD03b, CD23]. Here the setting is non-dependent and proof-irrelevant, but we see the same complex interleaving of cycles. Instead of tracing inputs and outputs for functions, we are tracing formulas through an infinite sequent calculus derivation, which is obtained by unfolding the cyclic derivation. We then have a similar PSPACE-complete soundness condition: for every infinite branch of the derivation, we can eventually trace a fixpoint formula that makes progress infinitely often [SD03a]. By a well-known strategy from cyclic proof theory, one can turn this global soundness condition into a local one by adding so-called *annotations* to formulas [Sti14, Jun10, DKMV23, LW24]. These annotations are based on Safra's determinisation construction [Saf88] for ω-automata.

Outlook. Inspired by the proof-theoretic picture, we hope to extend the current type theoretic conservativity results by allowing more complex interleaving function calls. Our hope is that by annotating the call graph, we can move from a global to a local soundness condition. These annotations then inform us on the order of (co)induction, which in turn should allow us to turn cyclic proofs into inductive proofs, following a similar approach to Sprenger and Dam [SD03b].

References

[CD23] Gianluca Curzi and Anupam Das. Computational expressivity of (circular) proofs with fixed points. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2023.

[CDP16] Jesper Cockx, Dominique Devriese, and Frank Piessens. Eliminating dependent pattern matching without k. *Journal of functional programming*, 26:e16, 2016.

[DKMV23] Maurice Dekker, Johannes Kloibhofer, Johannes Marti, and Yde Venema. Proof systems for the modal-calculus obtained by determinizing automata. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 242–259. Springer, 2023.

[GMM06] Healfdene Goguen, Conor McBride, and James McKinna. *Eliminating Dependent Pattern Matching*, pages 521–540. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Jun10] Natthapong Jungteerapanich. *Tableau systems for the modal μ-calculus.* PhD thesis, University of Edinburgh, 2010.

[LJBA01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. *SIGPLAN Not.*, 36(3):81–92, January 2001.

[LW24] Graham E. Leigh and Dominik Wehr. From gtc to image 1: Generating reset proof systems from cyclic proof systems. *Annals of Pure and Applied Logic*, 175(10):103485, 2024.

[Saf88] S. Safra. On the complexity of omega -automata. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, 1988.

[SD03a] Christoph Sprenger and Mads Dam. On global induction mechanisms in a μ-calculus with explicit approximations. *RAIRO-Theoretical Informatics and Applications*, 37(4):365–391, 2003.

[SD03b] Christoph Sprenger and Mads Dam. On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In Andrew D. Gordon, editor, *Foundations of Software Science and Computation Structures*, pages 425–440, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Sti14] Colin Stirling. A tableau proof system with names for modal mu-calculus. In *HOWARD-60*, 2014.

[Thi20] David Thibodeau. *An Intensional Type Theory of Coinduction Using Copatterns.* PhD thesis, McGill University Montréal, QC, Canada, 2020.

[Wah07] David Wahlstedt. *Dependent Type Theory with Parameterized First-Order Data Types and Well-Founded Recursion.* Chalmers Tekniska Hogskola (Sweden), 2007.