# Matching (Co)patterns with Cyclic Proofs

Lide Grotenhuis and Daniël Otten
University of Amsterdam

# Teaser

Agda accepts the following functions that Rocq rejects:

$$\mathsf{swap\text{-}add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$
$$\mathsf{swap\text{-}add}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto n, \\ \mathsf{suc}\, m' \mapsto \mathsf{suc}\,(\mathsf{swap\text{-}add}\, n\, m'); \end{cases}$$

$$\mathsf{g} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$
$$\mathsf{g}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto 0, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{suc}\, 0, \\ \mathsf{suc}\, n' \mapsto \mathsf{g}\, m'\, m' + \mathsf{g}\, n'\, n'. \end{cases} \end{cases}$$

Why do they terminate? Can we define them with induction?

# Overview

We connect:

cyclic proof theory and recursive functions with (co)pattern matching.

Cyclic proof systems replace (co)induction rules with circular reasoning.

**Example.** Consider arithmetic with axioms:

$$\frac{}{x + 0 = x}\, +_0, \qquad \frac{}{x + \mathsf{suc}\, y = \mathsf{suc}\,(x + y)}\, +_{\mathsf{suc}}.$$

We have a cyclic proof:

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

with a cycle between the blue nodes.

# Overview

We connect:

cyclic proof theory and recursive functions with (co)pattern matching.

Cyclic proof systems replace (co)induction rules with circular reasoning.

Good for proof search:

- (co)induction: guess a (co)induction hypothesis.
- cycles: generate until our current goal matches a previous goal; check for progress.

The type theory implemented by proof assistants can be seen as cyclic:

| Cyclic Proof | Recursive Function |
|---|---|
| Fixpoint Formula | (Co)inductive Type |
| Cycle | Recursive Function Call |
| Soundness Conditions | Termination Checking |

# Goals

Two main goals:

- Explain how the Curry-Howard correspondence can be extended to cyclic proofs and recursive functions.
- Extend conservativity results that show that pattern matching can be reduced to induction rules (with[1] and without[2] K).

[1] Goguen, McBride, McKinna 2006
[2] Cockx, Devriese, Piessens 2014

# Soundness Condition

For a cyclic proof system we specify when cycles are allowed:

- we want to be restrictive enough to be sound;
- we want to be admissive enough to be complete, and easy to use.

This is called the soundness condition.

The global soundness condition is: for every infinite path we can eventually trace an object that makes progress infinitely often.

**Example.** For arithmetic:

- objects: variables,
- progress: passing through a case distinction.

In general, checking the global soundness condition is PSPACE-complete.

# Two Styles

Cyclic proof systems generally fall into two styles:

- systems where the sort is (co)inductive:

  natural numbers, ordinals, streams, ...

- systems where the formulas contain fixpoints:

  $R^*$ is the smallest relation such that

$$xR^*y \quad \leftrightarrow \quad x = y \vee \exists x'(xRx' \wedge x'R^*y),$$
$$\vdots$$

We want a system that generalises both styles.

Dependent type theory is a natural candidate:

- types can be seen as both sorts and formulas.

# Termination Checking

What are cyclic proofs in type theory? General idea:

- A sequent $\Gamma \vdash a : A$ gives a function sending $\Gamma$ to $a : A$.
- A cycle uses the function inside the function (recursive call).

Proof assistants (`Agda`, `Rocq`, ...) implement recursive calls.

To ensure termination, we check:

- `Roqr`: structural recursion. This is conservative over induction (with[3] and without[4] K).
- `Agda`: size-change termination. Conservativity is not known.

These conditions are sufficient but not necessary (halting problem).

[3] Goguen, McBride, McKinna 2006
[4] Cockx, Devriese, Piessens 2014

# Structural Recursion

One input is structurally smaller in every recursive call:

Example. The Fibonacci function:

$$\begin{aligned} &\mathsf{fib} : \mathbb{N} \to \mathbb{N}, \\ &\mathsf{fib}\, n := \mathsf{case}\, n \begin{cases} 0 \mapsto 0, \\ \mathsf{suc}\, n' \mapsto \mathsf{case}\, n' \begin{cases} 0 \mapsto 1, \\ \mathsf{suc}\, n'' \mapsto \mathsf{fib}\, n'' + \mathsf{fib}\, n'. \end{cases} \end{cases} \end{aligned}$$

# Size-change termination

Every infinite sequence of calls eventually has a path that decreases infinitely often:

**Example.**

$$
\begin{aligned}
&\mathsf{swap\text{-}add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},\\
&\mathsf{swap\text{-}add}\, m\, n := \mathsf{case}\, m \begin{cases} \quad 0 \mapsto n, \\ \mathsf{suc}\, m' \mapsto \mathsf{suc}\,(\mathsf{swap\text{-}add}\, n\, m'). \end{cases}
\end{aligned}
$$

$$
\begin{tikzcd}
m \arrow[dr, ">"] & m \\
n \arrow[ur, "\geq"'] & n
\end{tikzcd}
$$

# Size-change Termination

Every infinite sequence of calls eventually has a path that decreases infinitely often:

**Example.**

$$
\begin{aligned}
&\mathsf{g} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}, \\
&\mathsf{g}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto 0, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{suc}\, 0, \\ \mathsf{suc}\, n' \mapsto \mathsf{g}\, m'\, m' + \mathsf{g}\, n'\, n'. \end{cases} \end{cases}
\end{aligned}
$$

$$
\begin{tikzcd}
m \arrow[r, ">"] \arrow[dr, ">"'] & m \\
n & n
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
m & m \\
n \arrow[ur, ">"] \arrow[r, ">"'] & n
\end{tikzcd}
$$

# Size-change Termination

Every infinite sequence of calls eventually has a path that decreases infinitely often.

This corresponds to the PSPACE-complete global soundness condition.

In cyclic proof theory, there are results showing that in some cases, this condition is conservative over induction:

- For first-order $\mu$-calculus with ordinal approximations.[5]
- For natural numbers.[6]

We hope to prove a similar result for type theory.

[5] Sprenger, Dam 2003
[6] Leigh, Wehr 2023

# Unification

For inductive families such as $=$-types, pattern matching uses unification.

Example. With normal unification, axiom K is provable:

$$K : (C : a = a \to \mathsf{Type}) \to C\,\mathsf{refl} \to (\alpha : a = a) \to C\,\alpha,$$
$$K\,C\,c\,\alpha := \mathsf{case}\,\alpha\,\{\mathsf{refl} \mapsto c.$$

Without K we have to restrict unification.

# Conservativity

We are trying to combine ideas:

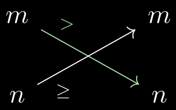
- Type theory: how to deal with unification and axiom K.
- Cyclic proof theory: how to deal with the global soundness condition.

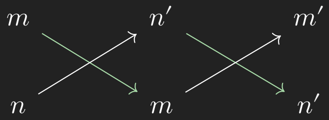
The main idea is that we unfold the definitions some more.

# Unfold the Tree

Example.

$$\mathsf{swap\text{-}add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$
$$\mathsf{swap\text{-}add}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto n, \\ \mathsf{suc}\, m' \mapsto \mathsf{suc}\,(\mathsf{swap\text{-}add}\, n\, m'). \end{cases}$$



# Unfold the Tree

**Example.** $g : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$,

$$
\begin{tikzcd}
m \arrow[r, ">"] \arrow[dr, ">"'] & m \\
n & n
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
m & m \\
n \arrow[ur, ">"] \arrow[r, ">"'] & n
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & & m'' \\
 & m' \arrow[ur] \arrow[r] & m'' \\
 & m' \arrow[r] \arrow[dr] & m'' \\
m \arrow[uur] \arrow[ur] & & m'' \\
n \arrow[dr] \arrow[ddr] & & n'' \\
 & n' \arrow[ur] \arrow[r] & n'' \\
 & n' \arrow[r] \arrow[dr] & n'' \\
 & & n''
\end{tikzcd}
$$

# Algorithm

We have an algorithm to determine how much to unfold:

- Start unfolding with annotations to track inputs.
  The annotations are based on the Safra construction, which makes nondeterministic $\omega$-automata deterministic.
- If annotations start repeating, then we can stop.
- Such an annotated function corresponds to a reset proof, where we have an equivalent local soundness condition.
- The annotations give us an idea of the order in which to apply induction, and the local condition ensures structurally smaller input.
- By following the annotations, we add induction hypotheses.
- We replace recursive calls with appeals to induction hypotheses.

# Conclusion

To summarize:

- The Curry-Howard correspondence extends to recursive functions and cyclic proofs.
- Cyclic proof theory can be useful for type theory.
- `Agda` admits more functions than `Rocr`. Conservativity is only known for `Rocr`, we are trying to prove it for `Agda`.

Our approach is a bit more general than we have seen here:
mix of arbitrary inductive families and mutually recursive functions.

In future work it would be interesting to look at copattern matching.

# Literature

Abel, Cocqx 2020 - Elaborating Dependent Copattern Matching
Abel, Pientka, Thibodeau, Setzer 2013 - Copatterns: Programming Infinite Structures by Observations Abel, Pientka 2016 - Well-founded Recursion with Copatterns and Sized Types
Afshari, Leigh 2027 - Cut-free Completeness for Modal Mu-Calculus Cockx 2017 - Dependent Pattern Matching and Proof-relevant Unification
Cockx, Devriese, Piessens 2014 - Pattern Matching Without K
Cockx, Devriese 2016 - Eliminating Dependent Pattern Matching Without K
Goguen, McBride, McKinna 2006 - Eliminating Dependent Pattern Matching
McBride, Goguen, McKinna 2004 - A Few Constructions on Constructors
Leigh, Wehr 2023 - Unravelling Cyclic First-Order Arithmetic
Sprenger, Dam 2003 - On the Structure of Inductive Reasoning, Circular and Tree-shaped Proofs in the mu-Calculus
Thibodeau 2020 - An Intensional Type Theory of Coinduction using Copatterns
Wehr 2023 - Representation Matters in Cyclic Proof Theory