Matching (Co)patterns with Cyclic Proofs

Lide Grotenhuis and Daniël Otten
ILLC, Amsterdam

A cyclic proof

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

Example. Consider the language of arithmetic with axioms:

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

We have a cyclic proof:

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

with a cycle between the blue nodes.

Upshot: instead of guessing an induction hypothesis, we generate a proof until we find a repeat with progress.

A function defined by pattern matching

Proof assistants based on dependent type theory (like `Agda` and `Rocq`) allow the user to define functions using pattern matching:

Example. The Fibonacci function:

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

Upshot: much easier to use and read than recursive functions defined via the primitive elimination (i.e. induction) rules of dependent type theory.

Overview

We investigate connections between:

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

The type theory implemented by proof assistants can be seen as a cyclic proof system for dependent type theory:

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

We have two main goals:

- Explain how the Curry-Howard correspondence can be extended to cyclic proofs and definitions by (co)pattern matching.
- Use this correspondence to extend conservativity results: pattern matching can be reduced to primitive induction rules.

Soundness of cyclic proofs

For a cyclic proof system, we need to specify which 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 trace condition is: for every infinite path we can eventually trace an object that makes progress infinitely often.

Example. For arithmetic:

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

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

Two styles

Cyclic proof systems generally fall into two categories:

- systems where the sort is (co)inductive:

  natural numbers, ordinals, streams, ...

- systems where we allow fixpoint formulas:

$$\mu X.\,\phi \text{ is the smallest fixpoint of } X \mapsto \phi,$$
$$\nu X.\,\phi \text{ is the largest fixpoint of } X \mapsto \phi.$$

Example. In the modal μ-calculus:

$$\mu X.\,p \vee \Diamond X, \qquad \text{(there is a path to a node where } p \text{ holds)}$$
$$\nu X.\,p \vee \Diamond X. \qquad \text{(... or an infinite path)}$$

In the first-order μ-calculus:

$$R^+ := \mu Xxy.\,Rxy \vee \exists u(Xxu \wedge Ruy). \qquad \text{(transitive closure of } R\text{)}$$

Fixpoints in Dependent Type Theory

The setting of dependent type theory allows for both styles of cyclic proof systems:

- types can be seen as both sorts and formulas;
- inductive/coinductive types generalise smallest/largest fixpoints.

Example. We can define: $\mathbb{N} := \mu X. \mathbb{1} + X$ and Stream $A := \nu X. A \times X$. Or, we say $\mathbb{N}$ is the inductive type with constructors:

$$0 : \mathbb{N},$$
$$\mathsf{suc} : \mathbb{N} \to \mathbb{N}.$$

And we say Stream A is the coinductive type with destructors:

$$\mathsf{head} : \mathsf{Stream}\ A \to A,$$
$$\mathsf{tail} : \mathsf{Stream}\ A \to \mathsf{Stream}\ A.$$

Cycles in Dependent Type Theory

What are cyclic proofs in type theory? General idea:

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

Proof assistants (`Agda`, `Rocq`, ...) implement a type theory where functions are defined using (co)pattern matching and recursive calls.

To make sure that the function terminates, we put some conditions:

- `Rocq`: structural recursion. This is conservative over induction (with[1] and without[2] K).
- `Agda`: size-change termination. Conservativity is not known.

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

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

Structural Recursion

There is one inductive input that is structurally smaller in every recursive call.

Example. The Fibonacci function:

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

Limits of Structural Recursion

The following functions are not structurally recursive:

$$\mathsf{swap\text{-}add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$

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

$$\mathsf{ack} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$

$$\mathsf{ack}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto \mathsf{suc}\, n, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{ack}\, m'\, 1, \\ \mathsf{suc}\, n' \mapsto \mathsf{ack}\, m'\,(\mathsf{ack}\,(\mathsf{suc}\, m')\, n'). \end{cases} \end{cases}$$

$$\mathsf{f} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$

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

However, they do satisfy the size-change termination principle.

Size-change termination

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

Example.

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

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

Size-change termination (SCT)

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

Example.

$$\mathsf{ack} : \mathbb{N} \to \mathbb{N} \to \mathbb{N},$$

$$\mathsf{ack}\, m\, n := \mathsf{case}\, m \begin{cases} 0 \mapsto \mathsf{suc}\, n, \\ \mathsf{suc}\, m' \mapsto \mathsf{case}\, n \begin{cases} 0 \mapsto \mathsf{ack}\, m'\, 1, \\ \mathsf{suc}\, n' \mapsto \mathsf{ack}\, m'\, (\mathsf{ack}\, (\mathsf{suc}\, m')\, n'). \end{cases} \end{cases}$$

$$\begin{array}{ccc} m \xrightarrow{\;>\;} m & \qquad m \xrightarrow{\;>\;} m & \qquad m \longrightarrow m \\[2em] n \qquad\quad n & \qquad n \qquad\quad n & \qquad n \xrightarrow{\;>\;} n \end{array}$$

Size-change Termination (SCT)

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

Example.

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

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

SCT as GTC

SCT reminds us of the global trace condition (GTC) of cyclic proofs!

Indeed, we can view pattern matching definitions satisfying SCT as a cyclic proof system with a GTC.

Theorem (Leigh & Wehr, 2003)

Any cyclic proof with a GTC can be unfolded to a reset proof, that is a cyclic proof with

- *a local soundness condition: every cycle c has a progressing object x_c*
- *an induction order $\leq_{\mathsf{ind}}$ on cycles:*
  - *every strongly connected component has a $\leq_{\mathsf{ind}}$-maximal cycle;*
  - *if $c \leq_{\mathsf{ind}} c'$ then c preserves $x_{c'}$.*

The unfolding algorithm is based on the Safra construction for determinizing stream automata.

Reset proofs

Our definition of the Ackermann function is already a 'reset proof'.

Example.

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

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

Reset proofs

For swap-add, we need to unfold once.

Example.

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

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

Conservativity

Want to show: every function defined by pattern matching with SCT can be defined using primitive induction rules.

On the proof-theoretic side, conservativity of cycles over an explicit induction rule is known for:

- First-order μ-calculus with ordinal approximations.[3]
- Peano and Heyting arithmetic.[4]

The proofs referenced here try to preserve the computational content of the cyclic proof.

[3] Sprenger & Dam 2003
[4] Wehr 2023

Conservativity

Proof sketch of proof-theoretic conservativity:

- Start with a cyclic proof that has an induction order (e.g. to a reset proof);
- Unfold the proof that the tree structure of the proof reflects the induction order: more important repeats occur lower in the tree.
- Starting at the root, replace each cycle by an appropriate inductive argument.

We are trying to use this idea to show conservativity of pattern matching with SCT.

Inductive Families

An added difficulty of type theory is that we can also define inductive families (or indexed inductive types) by specifying constructors.

Example. $(\mathsf{List}\, A\, n)_{n:\mathbb{N}}$ is the inductive family with constructors:

$$\mathsf{nil} : \mathsf{List}\, A\, 0,$$
$$\mathsf{cons} : \{n : \mathbb{N}\} \to A \to \mathsf{List}\, A\, n \to \mathsf{List}\, A\, (\mathsf{suc}\, n).$$

Example. $(a = a')_{a,a':A}$ is the inductive family with constructor:

$$\mathsf{refl} : \{a : A\} \to (a = a),$$

Unification

For pattern matching on inductive families, we need to use unification.

Example. $(m \leq n)_{m,n:\mathbb{N}}$ is the inductive family with constructors:

$$\mathsf{leq}_0 : \{n : \mathbb{N}\} \to (0 \leq n),$$

$$\mathsf{leq}_{\mathsf{suc}} : \{m, n : \mathbb{N}\} \to (m \leq n) \to (\mathsf{suc}\, m \leq \mathsf{suc}\, n),$$

$$
\begin{aligned}
&\mathsf{trans} : \{l, m, n : \mathbb{N}\} \to (a : l \leq m) \to (b : m \leq n) \to (l \leq n),\\
&\mathsf{trans}\, a\, b := \mathsf{case}\, a\, \{\\
&\qquad \mathsf{leq}_0 \mapsto \mathsf{leq}_0, && (l \equiv 0)\\
&\qquad \mathsf{leq}_{\mathsf{suc}}\, a' \mapsto \mathsf{case}\, b\, \{ && (l \equiv \mathsf{suc}\, l', m \equiv \mathsf{suc}\, m')\\
&\qquad\qquad \mathsf{leq}_0 \mapsto \lightning, && (m \equiv 0)\\
&\qquad\qquad \mathsf{leq}_{\mathsf{suc}}\, b' \mapsto \mathsf{leq}_{\mathsf{suc}}\, (\mathsf{trans}\, a'\, b'). && (m \equiv \mathsf{suc}\, m'', n \equiv \mathsf{suc}\, n')
\end{aligned}
$$

We don't need to cover the $\lightning$ case because $\mathsf{suc}\, m'$ can't be unified with 0. In the last case, b' has the correct type when we unify $\mathsf{suc}\, m'$ and $\mathsf{suc}\, m''$.

Unification: with or without K

When showing conservativity of pattern matching on inductive families, we need to internalise unification into the core type theory.

However, unrestricted use of unification leads to additional assumptions.

Example. With the 'standard' unification algorithm, axiom K is provable by pattern matching on the inductive family $(a = a')_{a,a':A}$:

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

Without axiom K, we need to restrict unification.

For structural recursion, we have conservativity with and without axiom K (Goguen, McBride & McKinna 2006; Cockx, Devriese & Piessens 2014).

What about coinduction?

In future work it would be interesting to look at coinductive types.

Example. For copattern matching we need to make progress before calling the function:

$$\text{zip} : \text{Stream}\, A \to \text{Stream}\, A \to \text{Stream}\, A,$$

$$\text{zip}\, s\, t := \text{record} \begin{cases} \text{head} \mapsto \text{head}\, s, \\ \quad \text{tail} \mapsto \text{zip}\, t\, (\text{tail}\, s). \end{cases}$$

And we can mix this with pattern matching

$$:\!\text{complog} : \mathbb{N} \to \mathbb{N} \to \text{Stream}\, \mathbb{N},$$

$$\text{complog}\, t\, r := \text{case}\, t \begin{cases} \quad 0 \mapsto \text{record} \begin{cases} \text{head} \mapsto r, \\ \quad \text{tail} \mapsto \text{complog}\, r\, r, \end{cases} \\ \text{suc}\, t' \mapsto \text{complog}\, t'\, (r!). \end{cases}$$

Conclusion

To summarize:

- The Curry-Howard correspondence extends to recursive functions and cyclic proofs.
- Via this correspondence, results from cyclic proof theory may be useful for type theory.
- `Agda` admits more functions than `Roqc`. Conservativity is only known for `Roqc`, we are trying to prove it for `Agda`.

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 - From GTC to Reset: Generating reset proof systems from cyclic proof systems
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