A circular version of Gödel’s T and its abstraction complexity

Anupam Das

University of Birmingham

Journées 2021 du GT Scalp
Fontainebleau
3rd November 2021
Cyclic proofs: a Curry-Howard perspective

A circular version of Gödel’s T

From models to interpretations

Conclusions
Motivating example: circular typing for Ackermann-Péter

Consider the functions $I \colon (N \to N) \to N \to N$ and $A \colon N \to N \to N$ given by:

- $I f \_0 = f \_1$
- $I f \_s x = f (I f x)$
- $A \_0 = s$
- $A \_s x = I (A x)$

Can be written using only base types with 'circular' typing:

- $s \colon N \to N \times N, N \to N$
- $\cdot \colon N, N \to N$
- $\cdot \colon N \to N \times N, N \to N$

This appears to be non-wellfounded.

Why is the function well-defined?
Motivating example: circular typing for Ackermann-Péter

Consider the functions $I : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ and $A : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ given by:

\[
\begin{align*}
I f 0 &= f \, 1 \\
I f s x &= f (I f x) \\
A 0 &= s \\
A s x &= I (A x)
\end{align*}
\]
Consider the functions $I : (N \to N) \to N \to N$ and $A : N \to N \to N$ given by:

\[
\begin{align*}
I f^0 &= f^1 \\
I f^s x &= f(I f x) \\
A^0 &= s \\
A^s x &= I(A x)
\end{align*}
\]

Can be written using only base types with ‘circular’ typing:
Motivating example: circular typing for Ackermann–Péter

Consider the functions $I : (N \to N) \to N \to N$ and $A : N \to N \to N$ given by:

\[
\begin{align*}
I f 0 &= f 1 \\
I f s x &= f (I f x) \\
A 0 &= s \\
A s x &= I (A x)
\end{align*}
\]

Can be written using only base types with ‘circular’ typing:

- Apparently non-wellfounded.
- Why is the function well-defined?
There are now several distinct communities studying non-wellfounded reasoning. Some of these include:

| Algebra / Type systems | Modal logic    | Predicate logic |
|------------------------|----------------|-----------------|
| Linear logic + \( \mu, \nu \) | \( \mu \)-calculus | FOL + ind. dfns. |
| Kleene Algebra + \( \cap, \setminus, / \) | PDL & Game logic | Arithmetic |

**NB:** formula expressivity increases left-to-right.

**Some references:**

- **Algebra and type systems**: [Santocanale '02], [Fortier & Santocanale '13], [Baelde, Doumane & Saurin '16], [D. & Pous '17, '18], [Kuperberg, Pinault & Pous '21].
- **Modal logics**: [Niwinski & Walukiewicz '96], [Afshari & Leigh '17], [Enqvist, Hansen, Kupke, Marti & Venema '19].
- **Predicate logic**: [Brotherston & Simpson '07], [Simpson '17], [Berardi & Tatsuta '17], [D. '20].
Are cyclic proofs and inductive proofs equally powerful?
Are cyclic proofs and inductive proofs equally powerful?

The situation in arithmetic is now well-understood:

**Theorem (Simpson ’11)**

Cyclic Arithmetic (CA) is equivalent to Peano Arithmetic (PA).

**Theorem (D. ’20)**

$I\Sigma_{n+1}$ and $C\Sigma_n$ prove the same $\Pi_{n+1}$ theorems.
The Brotherston-Simpson conjecture

Are cyclic proofs and inductive proofs equally powerful?

The situation in arithmetic is now well-understood:

**Theorem (Simpson ’11)**

*Cyclic Arithmetic (CA) is equivalent to Peano Arithmetic (PA).*

**Theorem (D. ’20)**

$I\Sigma_{n+1}$ and $C\Sigma_n$ prove the same $\Pi_{n+1}$ theorems.

What about type theories?
1 Cyclic proofs: a Curry-Howard perspective

2 A circular version of Gödel’s $T$

3 From models to interpretations

4 Conclusions
Church's simple type theory

Finite types:

\[ \sigma, \tau ::= N \mid (\sigma \to \tau) \]

- **Language**: set of typed constants (always including equality \(=_{\sigma}\) at all \(\sigma\)).
- **Terms**: formed by typed application.
- **Theory**: set of axioms and rules (always including intensional equality).

Example (Combinatory Algebra)

**Language:**

\[
\begin{align*}
K_{\sigma \tau} & : \sigma \to \tau \to \sigma \\
S_{\rho \sigma \tau} & : (\rho \to \sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau
\end{align*}
\]

**Theory:**

\[
\begin{align*}
K \ x \ y & = x \\
S \ x \ y \ z & = x \ z \ (y \ z)
\end{align*}
\]

Standard model \(\mathcal{N}\):

\[
\begin{align*}
N^{\mathcal{N}} & ::= \mathbb{N} \\
(\sigma \to \tau)^{\mathcal{N}} & ::= \{ f : \sigma^{\mathcal{N}} \to \tau^{\mathcal{N}} \}
\end{align*}
\]

**Interpretations**: take equational axioms as definitions left-to-right.
$T$ extends combinatory algebra by **recursion combinators**:

$$\text{rec}_\sigma : \sigma \rightarrow (N \rightarrow \sigma \rightarrow \sigma) \rightarrow N \rightarrow \sigma$$

and (quantifier-free) axioms and rules:

$$\begin{align*}
\text{rec} f g 0 &= g \\
\text{rec} f g s x &= g x (\text{rec} f g x) \\
\neg s x &= 0 \\
\phi(0) &\Rightarrow \phi(s x) \\
\text{ind} &\Rightarrow \phi(t)
\end{align*}$$
System $T$

$T$ extends combinatory algebra by **recursion combinators**:

$$\text{rec}_\sigma : \sigma \rightarrow (N \rightarrow \sigma \rightarrow \sigma) \rightarrow N \rightarrow \sigma$$

and (quantifier-free) axioms and rules:

$$\begin{align*}
\text{rec } f \ g \ O &= g \\
\text{rec } f \ g \ s \ x &= g \ x \ (\text{rec } f \ g \ x) \\
\neg s \ x &= 0 \\
s \ x &= s \ y \supset x = y
\end{align*}$$

**Theorem (Gödel ’41)**

$T$ is **equiconsistent** with Peano Arithmetic.

$\leadsto$ we can *trade off* quantifier complexity for abstraction complexity.
System $T$

$T$ extends combinatory algebra by **recursion comonitors**:

$$\text{rec}_{\sigma} : \sigma \to (N \to \sigma \to \sigma) \to N \to \sigma$$

and (quantifier-free) axioms and rules:

\[
\begin{align*}
\text{rec } f \ g \ 0 &= g \\
\text{rec } f \ g \ sx &= g \ x \ (\text{rec } f \ g \ x) \quadدخل \ sx = 0 \quad دخل \ s \ x = 0 \quad دخل \ s \ y \supset x = y \\
\text{ind } \varphi(0) &\quad \varphi(x) \supset \varphi(sx) \\
\end{align*}
\]

**Theorem (Gödel ’41)**

$T$ is equiconsistent with Peano Arithmetic.

⇝ we can trade off quantifier complexity for abstraction complexity.

**Question**

*Can we interpret cyclic arithmetic (directly) in a circular version of $T$?*
Each instance of a rule is construed as a constant.

....the map (derivations → terms) is continuous.
Equational axiomatisation

\[
\begin{align*}
\text{id} \ x & = x \\
\text{ex} \ t \bar{x} x y \bar{y} & = t \bar{x} y x \bar{y} \\
\text{wk} \ t \bar{x} x & = t \bar{x} \\
\text{cntr} \ t \bar{x} x & = t \bar{x} x x \\
\text{rec} \ s t \bar{x} 0 & = s \bar{x} \\
\text{rec} \ s t \bar{x} sz & = t \bar{x} z (\text{rec} \ s t \bar{x} z)
\end{align*}
\]

\[
\begin{align*}
\text{cut} \ s t \bar{x} & = t \bar{x} (s \bar{x}) \\
\text{L} \ s t \bar{x} y & = t \bar{x} (y (s \bar{x})) \\
\text{R} \ t \bar{x} x & = t \bar{x} x \\
\text{cond} \ s t \bar{x} 0 & = s \bar{x} \\
\text{cond} \ s t \bar{x} sz & = t \bar{x} z
\end{align*}
\]

**NB:** gives interpretations of constants in \( \mathbb{N} \), using **meta-level induction**.
Coterms and coderivations

We can generalise term trees and derivation trees to non-wellfounded counterparts:

Definition

- **coterms** are generated **coinductively** from constants and variables.
- **coderivations** are generated **coinductively** from the rules.

**NB:** The ‘coterm of a coderivation’ is **well-defined**, thanks to continuity.
We can generalise term trees and derivation trees to non-wellfounded counterparts:

**Definition**

- **coterms** are generated coinductively from constants and variables.
- **coderivations** are generated coinductively from the rules.

**NB:** The ‘coterm of a coderivation’ is well-defined, thanks to continuity.

A coderivation is **regular** or **circular** if it has only finitely many distinct sub-coderivations.
We can generalise term trees and derivation trees to non-wellfounded counterparts:

**Definition**

- **coterms** are generated coinductively from constants and variables.
- **coderivations** are generated coinductively from the rules.

**NB:** The ‘coterm of a coderivation’ is well-defined, thanks to continuity.

A coderivation is **regular** or **circular** if it has only finitely many distinct sub-coderivations.

**Semantics:** Kleene-Herbrand-Gödel style partial functionals.
Let $f : \mathbb{N} \times \mathbb{N}^k \to \mathbb{N}$ and write $f_i(\vec{x}) := f(i, \vec{x})$. 
Unbounded search $\mu x (f \ x = 0)$ is given by $H \ 0$ with:

$$H \ x \ := \ \text{cond} \ (f \ x) \ x \ (H \ s x)$$

$H$ is computed by the following regular coderivation:
A totality criterion

$\sigma^1$ is an **immediate ancestor** of $\sigma^2$ if they are in the premiss and conclusion, resp., and have the ‘same colour’.

**Definition (Threads and progress)**

- A **thread** is a maximal path in the graph of immediate ancestry.
- An $N$-thread is **progressing** if it is infinitely often **principal** for cond.
- A coderivation is **progressing** if each infinite branch has a progressing $N$-thread.
A totality criterion

σ₁ is an immediate ancestor of σ² if they are in the premiss and conclusion, resp., and have the ‘same colour’.

Definition (Threads and progress)

- A thread is a maximal path in the graph of immediate ancestry.
- An N-thread is progressing if it is infinitely often principal for cond.
- A coderivation is progressing if each infinite branch has a progressing N-thread.

Definition (Circular systems)

CT is the simple type theory that has a symbol for every progressing regular coderivation, and is axiomatised by all previous equations (over coterms).

- Tₙ is the restriction of T allowing only types of level n in typing derivations.
- CTₙ is the restriction of CT allowing only types of level n in typing derivations.
Example: Ackermann-Péter

\[
A(0, y) := y + 1 \\
A(x + 1, 0) := A(x, 1) \\
A(x + 1, y + 1) := A(x, A(x + 1, y))
\]

**NB.** Not representable in $T_0$!
Example: Ackermann-Péter

\[
A(0, y) := y + 1 \\
A(x + 1, 0) := A(x, 1) \\
A(x + 1, y + 1) := A(x, A(x + 1, y))
\]

**NB.** Not representable in \(T_0\)! However:

\[
\begin{array}{c}
\frac{1}{\text{cut}} \quad \frac{\Rightarrow N}{N, N \Rightarrow N} \\
\frac{s}{N \Rightarrow N} \\
\frac{\text{wk}}{N, N \Rightarrow N} \\
\frac{\text{cond}}{N, N, N \Rightarrow N} \\
\frac{\text{cntr}}{N, N \Rightarrow N}
\end{array}
\]

\[
\begin{array}{c}
\frac{\Rightarrow N}{N, N \Rightarrow N} \\
\frac{N \Rightarrow N}{N, N \Rightarrow N} \\
\frac{\text{cut}}{N, N, N \Rightarrow N} \\
\frac{\Rightarrow N}{N, N \Rightarrow N}
\end{array}
\]

\[
\begin{array}{c}
\frac{1}{\text{cut}} \quad \frac{\Rightarrow N}{N, N \Rightarrow N} \\
\frac{N \Rightarrow N}{N, N \Rightarrow N} \\
\frac{\text{cond}}{N, N, N \Rightarrow N} \\
\frac{\text{cntr}}{N, N \Rightarrow N}
\end{array}
\]
Example: Ackermann-Péter

\[
\begin{align*}
A(0, y) & := y + 1 \\
A(x + 1, 0) & := A(x, 1) \\
A(x + 1, y + 1) & := A(x, A(x + 1, y))
\end{align*}
\]

**NB.** Not representable in $T_0$! However:

\[
\begin{align*}
\text{\(1\)} & \quad \frac{1 \Rightarrow N}{N, N \Rightarrow N} \\
\text{\(2\)} & \quad \frac{N \Rightarrow N}{N, N \Rightarrow N} \\
\text{\(3\)} & \quad \frac{N, N \Rightarrow N}{N, N, N \Rightarrow N}
\end{align*}
\]

**Question**

*What is the relative abstraction complexity of functionals in $T$ and $CT$?*
Proposition (Well-definedness)

A progressing coderivation computes a well-defined total functional.
Proposition (Well-definedness)

A progressing coderivation computes a well-defined total functional.

Proof sketch.

• Each rule preserves totality top-down, so preserves non-totality bottom-up.
• \( \Rightarrow \) we may build a leftmost ‘non-total’ infinite branch.
• Assign to a progressing \( N \)-thread the least natural numbers witnessing non-totality of the corresponding coderivations.
• This sequence will be monotone decreasing but cannot converge.
Outline

1. Cyclic proofs: a Curry-Howard perspective
2. A circular version of Gödel’s T
3. From models to interpretations
4. Conclusions
We may construe the equations of $T$ and $CT$ as a rewrite system:

\[
\begin{align*}
\text{id } x & \rightsquigarrow x \\
\text{ex } t \overrightarrow{x} x y \overrightarrow{y} & \rightsquigarrow t \overrightarrow{x} y x \overrightarrow{y} \\
\text{wk } t \overrightarrow{x} x & \rightsquigarrow t \overrightarrow{x} \\
\text{cntr } t \overrightarrow{x} x & \rightsquigarrow t \overrightarrow{x} x x \\
\text{cut } s t \overrightarrow{x} & \rightsquigarrow t \overrightarrow{x} (s \overrightarrow{x}) \\
\text{L } s t \overrightarrow{x} y & \rightsquigarrow t \overrightarrow{x} (y (r \overrightarrow{x})) \\
\text{R } t \overrightarrow{x} x & \rightsquigarrow t \overrightarrow{x} x \\
\text{rec } s t \overrightarrow{x} 0 & \rightsquigarrow s \overrightarrow{x} \\
\text{rec } s t \overrightarrow{x} sy & \rightsquigarrow t \overrightarrow{x} (\text{rec } s t \overrightarrow{x} y) \\
\text{cond } s t \overrightarrow{x} 0 & \rightsquigarrow s \overrightarrow{x} \\
\text{cond } s t \overrightarrow{x} sy & \rightsquigarrow t \overrightarrow{x} y
\end{align*}
\]

Write $\approx$ for reflexive symmetric transitive closure of $\rightsquigarrow$. 

Theorem (Confluence for $CT$, RCA$^0$)

If $s \rightsquigarrow^* t_0$ and $s \rightsquigarrow^* t_1$ then there is some $t$ with $t_0 \rightsquigarrow^* t$ and $t_1 \rightsquigarrow^* t$. 

Confluence of $T$
Confluence of $T$

We may construe the equations of $T$ and $CT$ as a rewrite system:

\[
\begin{align*}
\text{id} & \ x \ \xrightarrow{\approx} \ x \\
\text{ex} & \ t \ \vec{x} \ x \ y \ \vec{y} \ \xrightarrow{\approx} \ t \ \vec{x} \ y \ x \ \vec{y} \\
\text{wk} & \ t \ \vec{x} \ x \ \xrightarrow{\approx} \ t \ \vec{x} \\
\text{cntr} & \ t \ \vec{x} \ x \ \xrightarrow{\approx} \ t \ \vec{x} \ x \ x \\
\text{cut} & \ s \ t \ \vec{x} \ \xrightarrow{\approx} \ t \ \vec{x} \ (s \ \vec{x}) \\
\text{L} & \ s \ t \ \vec{x} \ y \ \xrightarrow{\approx} \ t \ \vec{x} \ (y \ (r \ \vec{x})) \\
\text{R} & \ t \ \vec{x} \ x \ \xrightarrow{\approx} \ t \ \vec{x} \ x \\
\text{rec} & \ s \ t \ \vec{x} \ 0 \ \xrightarrow{\approx} \ s \ \vec{x} \\
\text{rec} & \ s \ t \ \vec{x} \ sy \ \xrightarrow{\approx} \ t \ \vec{x} \ (\text{rec} \ s \ t \ \vec{x} \ y) \\
\text{cond} & \ s \ t \ \vec{x} \ 0 \ \xrightarrow{\approx} \ s \ \vec{x} \\
\text{cond} & \ s \ t \ \vec{x} \ sy \ \xrightarrow{\approx} \ t \ \vec{x} \ y
\end{align*}
\]

Write $\approx$ for reflexive symmetric transitive closure of $\xrightarrow{\approx}$.

**Theorem (Confluence for $CT$, RCA$_0$)**

If $s \xrightarrow{\approx}^* t_0$ and $s \xrightarrow{\approx}^* t_1$ then there is some $t$ with $t_0 \xrightarrow{\approx}^* t$ and $t_1 \xrightarrow{\approx}^* t$. 
Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure HR on coterminals.
Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure $\text{HR}$ on coterminals. In particular, for any $\text{CT}_n$-coterm $t : \tau$:

**Theorem ($\text{RCA}_0 + I\Sigma_{n+2}$)**

$t \in \text{HR}_{\tau}$. 
Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure $\text{HR}$ on coterminals. In particular, for any $CT_n$-coterm $t : \tau$:

**Theorem (RCA$_0 + I\Sigma_{n+2}$)**

$t \in \text{HR}_\tau$.

**Proof idea.**

- **Formalise** the totality argument wrt $\text{HR}$ structure.
- **Well-definedness** of infinite branch achieved by *minimisation principles*.
- **Logical complexity** controlled by *arithmetical approximation* of progress.
Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure \( HR \) on coterms. In particular, for any \( CT_n \)-coterm \( t : \tau \):

**Theorem (RCA\(_0\) + I\(\Sigma_{n+2}\))**

\[ t \in HR_\tau. \]

**Proof idea.**

- **Formalise** the totality argument wrt \( HR \) structure.
- Well-definedness of infinite branch achieved by **minimisation principles**.
- Logical complexity controlled by **arithmetical approximation** of progress.

This implies that \( HR \) is a **model** of \( CT \).
Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure $\mathbb{HR}$ on coterms. In particular, for any $CT_n$-coterm $t : \tau$:

**Theorem ($\mathsf{RCA}_0 + I\Sigma_{n+2}$)**

$t \in \mathbb{HR}_\tau$.

**Proof idea.**

- Formalise the totality argument wrt $\mathbb{HR}$ structure.
- Well-definedness of infinite branch achieved by minimisation principles.
- Logical complexity controlled by arithmetical approximation of progress.

This implies that $\mathbb{HR}$ is a model of $CT$. In particular for any $CT_n$-coderivation $t$:

**Corollary ($\mathsf{RCA}_0 + I\Sigma_{n+2}$)**

$t$ is weakly normalising wrt $\leadsto$. 


**NB:** all results are *arithmetised* within fragments of *second-order arithmetic*.

We can apply well-known *program extraction* techniques in order to recover an *interpretation* of $CT$ into $T$.
**NB**: all results are *arithmetised* within fragments of *second-order arithmetic*.

We can apply well-known *program extraction* techniques in order to recover an *interpretation* of $CT$ into $T$.

**Theorem (Interpretation)**

If $CT_n \vdash s = t$ then $T_{n+1} \vdash s \approx t$.

**Corollary (Computation at type 1)**

Any *type 1 function* representable in $CT_n$ is also representable in $T_{n+1}$. 
Outline

1. Cyclic proofs: a Curry-Howard perspective
2. A circular version of Gödel’s $T$
3. From models to interpretations
4. Conclusions
By formalising a model of 'convertibility' à la Tait, we obtain:

Theorem (Strong normalisation)
Let t be representable in CT. Then

\[ \text{ACA}_0 \]
proves that t is strongly normalising.

Via a form of cut-elimination and a realisation of the deduction theorem:

Theorem (Converse interpretation)

\[ T_{n+1} \]
is interpreted into CT

\[ T_n \]
(over the level \( n + 1 \) theory).

Corollary

\[ T_{n+1} \] and CT

\[ T_n \]
are equiconsistent.

By formalising termination of 'runs' along progressing coderivations in

\[ \text{ACA}_0 \]
, we recover recursion along progressing coderivations directly in

\[ T \]:

Theorem (Functional equivalence)

CT and T compute the same functionals, at all types.
By formalising a model of ‘convertibility’ à la Tait, we obtain:

**Theorem (Strong normalisation)**

*Let t be representable in CT. Then ACA₀ proves that t is strongly normalising.*
By formalising a model of ‘convertibility’ à la Tait, we obtain:

**Theorem (Strong normalisation)**

Let \( t \) be representable in \( CT \). Then \( \text{ACA}_0 \) proves that \( t \) is **strongly normalising**.

Via a form of cut-elimination and a realisation of the deduction theorem:

**Theorem (Converse interpretation)**

\( T_{n+1} \) is interpreted into \( CT_n \) (over the level \( n + 1 \) theory).

**Corollary**

\( T_{n+1} \) and \( CT_n \) are **equiconsistent**.
By formalising a model of ‘convertibility’ à la Tait, we obtain:

**Theorem (Strong normalisation)**

Let \( t \) be representable in \( CT \). Then \( \text{ACA}_0 \) proves that \( t \) is **strongly normalising**.

Via a form of **cut-elimination** and a realisation of the **deduction theorem**:

**Theorem (Converse interpretation)**

\( T_{n+1} \) is interpreted into \( CT_n \) (over the level \( n + 1 \) theory).

**Corollary**

\( T_{n+1} \) and \( CT_n \) are **equiconsistent**.

By **formalising termination** of ‘runs’ along progressing coderivations in \( \text{ACA}_0 \), we recover **recursion** along progressing coderivations directly in \( T \):

**Theorem (Functional equivalence)**

\( CT \) and \( T \) compute the **same functionals**, at all types.
Summary and open questions

We interpreted $\text{CT}_n$ into $T_{n+1}$ and vice-versa, and showed various equivalences. See https://arxiv.org/abs/2012.14421 for details.

Related work: Kuperberg, Pinault & Pous '21 have also considered a variation of $\text{CT}$-terms:

- Affine progressing coterms $\approx$ primitive recursive functions (at type 1).
- Progressing coterms $\approx$ primitive recursive functionals (at type 1).

Future work:

- Proof interpretations from arithmetic to type systems. [w.i.p. with Thomas Powell].
- Extensions by arbitrary inductive definitions. [w.i.p. with Lukas Holter Melgaard], cf. also [Berardi & Tatsuta '18].
- Cyclic implicit complexity based on ramified recursion. [w.i.p. with Gianluca Curzi]

Thank you.
Summary and open questions

We interpreted $CT_n$ into $T_{n+1}$ and vice-versa, and showed various equivalences. See https://arxiv.org/abs/2012.14421 for details.

Related work:
- Kuperberg, Pinault & Pous '21 have also considered a variation of $CT$-terms:
  - Affine progressing coterms $\approx$ primitive recursive functions (at type 1).
  - Progressing coterms $\approx$ primitive recursive functions also (at type 1).

Future work:
- Proof interpretations from arithmetic to type systems. [w.i.p. with Thomas Powell]
- Extensions by arbitrary inductive definitions. [w.i.p. with Lukas Holter Melgaard], cf. also [Berardi & Tatsuta '18].
- Cyclic implicit complexity based on ramified recursion. [w.i.p. with Gianluca Curzi]

Thank you.
We interpreted $CT_n$ into $T_{n+1}$ and vice-versa, and showed various equivalences. See https://arxiv.org/abs/2012.14421 for details.

**Related work:**
Kuperberg, Pinault & Pous ’21 have also considered a variation of $CT$-terms:
- **Affine** progressing coterms $\approx$ primitive recursive functions (at type 1).
- Progressing coterms $\approx$ primitive recursive functionals (at type 1).
Summary and open questions

We interpreted $CT_n$ into $T_{n+1}$ and vice-versa, and showed various equivalences. See https://arxiv.org/abs/2012.14421 for details.

**Related work:**
Kuperberg, Pinault & Pous ’21 have also considered a variation of $CT$-terms:
- **Affine** progressing coterms $\approx$ primitive recursive functions (at type 1).
- Progressing coterms $\approx$ primitive recursive functionals (at type 1).

**Future work:**
- **Proof interpretations** from arithmetic to type systems. [w.i.p. with Thomas Powell].
- Extensions by arbitrary inductive definitions. [w.i.p. with Lukas Holter Melgaard], cf. also [Berardi & Tatsuta ’18].
- Cyclic implicit complexity based on ramified recursion. [w.i.p. with Gianluca Curzi]
Summary and open questions

We interpreted $CT_n$ into $T_{n+1}$ and vice-versa, and showed various equivalences. See https://arxiv.org/abs/2012.14421 for details.

Related work:

Kuperberg, Pinault & Pous ’21 have also considered a variation of $CT$-terms:

• Affine progressing coterms $\approx$ primitive recursive functions (at type 1).
• Progressing coterms $\approx$ primitive recursive functionals (at type 1).

Future work:

• Proof interpretations from arithmetic to type systems.
  [w.i.p. with Thomas Powell].
• Extensions by arbitrary inductive definitions.
  [w.i.p. with Lukas Holter Melgaard], cf. also [Berardi & Tatsuta ’18].
• Cyclic implicit complexity based on ramified recursion.
  [w.i.p. with Gianluca Curzi]

Thank you.