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

Outline

1. Cyclic proofs: a Curry-Howard perspective
2. A circular version of Gödel's T
3. From models to interpretations
4. Conclusions

Motivating example: circular typing for Ackermann-Péter

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:

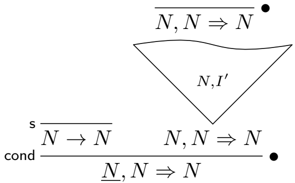
$$\begin{array}{rcl} I\,f\,0 & = & f\,1 \\ I\,f\,\mathsf{s}x & = & f\,(I\,f\,x) \end{array} \qquad \begin{array}{rcl} A\,0 & = & \mathsf{s} \\ A\,\mathsf{s}x & = & I\,(A\,x) \end{array}$$

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:

$$
\begin{array}{rclcrcl}
I\,f\,0 & = & f\,1 & \qquad & A\,0 & = & \mathsf{s} \\
I\,f\,\mathsf{s}x & = & f\,(I\,f\,x) & \qquad & A\,\mathsf{s}x & = & I\,(A\,x)
\end{array}
$$

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

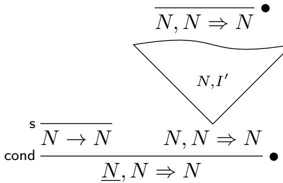


Motivating example: circular typing for Ackermann-Péter

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

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
I\,f\,0 & = & f\,1 & A\,0 & = & \mathsf{s} \\
I\,f\,\mathsf{s}x & = & f\,(I\,f\,x) & A\,\mathsf{s}x & = & I\,(A\,x)
\end{array}
$$

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



Apparently non-wellfounded.

Why is the function well-defined?

Landscape of cyclic proof theory

There are now several distinct communities studying non-wellfounded reasoning. *Some* of these include:

Algebra / Type systems	**Modal logic**	**Predicate logic**
Linear logic + ;	-calculus	FOL + ind. dfns.
Kleene Algebra + \ ; *n*; =	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].

The Brotherston-Simpson conjecture

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)

$I \ _{n+1}$ and $C \ _{n}$ prove the same $\ _{n+1}$ theorems.

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)

$I\ _{n+1}$ and $C\ _{n}$ prove the same $\ _{n+1}$ theorems.

What about type theories?

Outline

Finite types:

$$; \qquad ::= \quad N \quad j \quad (\quad ! \quad)$$

Language: set of typed constants (always including equality $=$ at all).

Terms: formed by typed application.

Theory: set of axioms and rules (always including intensional equality).

Example (Combinatory Algebra)

Language:

$$\begin{array}{lll} K & : & \quad ! \quad ! \\ S & : & (\quad ! \quad ! \quad) \; ! \; (\quad ! \quad) \; ! \quad ! \end{array}$$

Theory:

$$\begin{array}{lll} K\,x\,y & = & x \\ S\,x\,y\,z & = & x\,z\,(y\,z) \end{array}$$

Standard model N**:**

$$\begin{array}{rcl} N^{\mathfrak{N}} & := & \mathsf{N} \\ (\quad ! \quad)^{\mathfrak{N}} & := & ff : \; ^{\mathfrak{N}} \; ! \; ^{\mathfrak{N}}g \end{array}$$

Interpretations: take equational axioms as definitions left-to-right.

T extends combinatory algebra by **recursion cominators**:

$$\mathsf{rec} \; : \quad ! \; (N \; ! \quad ! \quad) \; ! \; N \; !$$

and (quantifier-free) axioms and rules:

$$\begin{array}{rcl} \mathsf{rec}\, f\, g\, 0 & = & g \\ \mathsf{rec}\, f\, g\, \mathsf{S}x & = & g\, x\, (\mathsf{rec}\, f\, g\, x) \end{array} \qquad \begin{array}{c} : \mathsf{S}x = 0 \\ \mathsf{S}x = \mathsf{S}y \quad x = y \end{array} \qquad ind \; \frac{'(0) \qquad '(x) \qquad '(\mathsf{S}x)}{'(t)}$$

T extends combinatory algebra by **recursion cominators**:

$$\mathsf{rec} \ : \quad ! \ (N \ ! \qquad ! \quad) \ ! \ N \ !$$

and (quantifier-free) axioms and rules:

$$\begin{array}{rcl} \mathsf{rec}\, f\, g\, 0 & = & g \\ \mathsf{rec}\, f\, g\, \mathsf{S}x & = & g\, x\, (\mathsf{rec}\, f\, g\, x) \end{array} \qquad \begin{array}{c} : \mathsf{S}x = 0 \\ \mathsf{S}x = \mathsf{S}y \quad x = y \end{array} \qquad ind\ \frac{'(0) \qquad '(x) \qquad '(\mathsf{S}x)}{'(t)}$$

Theorem (Gödel '41)

T is equiconsistent with Peano Arithmetic.

we can *trade off* quantifier complexity for abstraction complexity.

T extends combinatory algebra by **recursion cominators**:

$$\mathsf{rec} \quad : \quad ! \ (N \ ! \quad ! \quad) \ ! \ N \ !$$

and (quantifier-free) axioms and rules:

$$\begin{array}{rcl} \mathsf{rec}\, f\, g\, 0 & = & g \\ \mathsf{rec}\, f\, g\, \mathsf{s}x & = & g\, x\, (\mathsf{rec}\, f\, g\, x) \end{array} \qquad \begin{array}{c} : \mathsf{s}x = 0 \\ \mathsf{s}x = \mathsf{s}y \quad x = y \end{array} \qquad ind\ \frac{'(0) \qquad '(x) \qquad '(\mathsf{s}x)}{'(t)}$$

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?

T-terms typed in sequent style

$$
\text{ex}\ \frac{\vec\rho, \sigma, \rho, \vec\sigma \Rightarrow \tau}{\vec\rho, \rho, \sigma, \vec\sigma \Rightarrow \tau}
\qquad
\text{wk}\ \frac{\vec\sigma \Rightarrow \tau}{\vec\sigma, \sigma \Rightarrow \tau}
\qquad
\text{cntr}\ \frac{\vec\sigma, \sigma, \sigma \Rightarrow \tau}{\vec\sigma, \sigma \Rightarrow \tau}
\qquad
\text{cut}\ \frac{\vec\sigma \Rightarrow \sigma \quad \vec\sigma, \sigma \Rightarrow \tau}{\vec\sigma \Rightarrow \tau}
$$

$$
\text{id}\ \frac{}{\sigma \Rightarrow \sigma}
\qquad
\text{L}\ \frac{\vec\sigma \Rightarrow \rho \quad \vec\sigma, \sigma \Rightarrow \tau}{\vec\sigma, \rho \to \sigma \Rightarrow \tau}
\qquad
\text{R}\ \frac{\vec\sigma, \sigma \Rightarrow \tau}{\vec\sigma \Rightarrow \sigma \to \tau}
$$

$$
0\ \frac{}{\Rightarrow N}
\qquad
\text{s}\ \frac{}{N \Rightarrow N}
\qquad
\text{cond}\ \frac{\vec\sigma \Rightarrow \tau \quad \vec\sigma, N \Rightarrow \tau}{\vec\sigma, N \Rightarrow \tau}
\qquad
\text{rec}_\tau\ \frac{\vec\sigma \Rightarrow \tau \quad \vec\sigma, N, \sigma \Rightarrow \tau}{\vec\sigma, N \Rightarrow \tau}
$$

Each instance of a rule is construed as a constant.

....the map (derivations ! terms) is continuous.

Equational axiomatisation

$$
\begin{aligned}
\mathsf{id}\ x &= x \\
\mathsf{ex}\ t\ \vec{x}\ x\ y\ \vec{y} &= t\ \vec{x}\ y\ x\ \vec{y} \\
\mathsf{wk}\ t\ \vec{x}\ x &= t\ \vec{x} \\
\mathsf{cntr}\ t\ \vec{x}\ x &= t\ \vec{x}\ x\ x
\end{aligned}
\qquad
\begin{aligned}
\mathsf{cut}\ s\ t\ \vec{x} &= t\ \vec{x}\ (s\ \vec{x}) \\
\mathsf{L}\ s\ t\ \vec{x}\ y &= t\ \vec{x}\ (y\ (s\ \vec{x})) \\
\mathsf{R}\ t\ \vec{x}\ x &= t\ \vec{x}\ x
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{rec}\ s\ t\ \vec{x}\ 0 &= s\ \vec{x} \\
\mathsf{rec}\ s\ t\ \vec{x}\ \mathsf{s}z &= t\ \vec{x}\ z\ (\mathsf{rec}\ s\ t\ \vec{x}\ z)
\end{aligned}
\qquad
\begin{aligned}
\mathsf{cond}\ s\ t\ \vec{x}\ 0 &= s\ \vec{x} \\
\mathsf{cond}\ s\ t\ \vec{x}\ \mathsf{s}z &= t\ \vec{x}\ z
\end{aligned}
$$

NB: gives interpretations of constants in $\mathtt{N}$, 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.

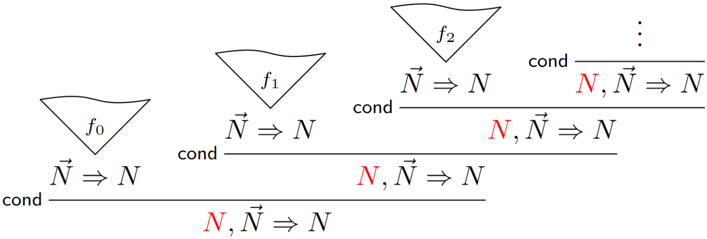
NB: The 'coterm of a coderivation' is well-defined, thanks to continuity.

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

Semantics: Kleene-Herbrand-Gödel style partial functionals.

Example: type 1 completeness of coderivations

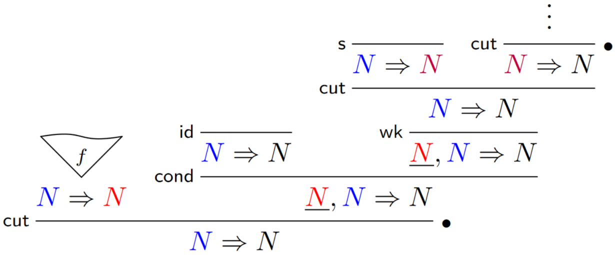
Let $f : \mathsf{N} \times \mathsf{N}^k \to \mathsf{N}$ and write $f_i(x) := f(i; x)$.



Unbounded search $x(f\,x = 0)$ is given by $H\,0$ with:

$$H\,x \;:=\; \mathsf{cond}\,(f\,x)\,x\,(H\,\mathsf{s}x)$$

H is computed by the following regular coderivation:



A totality criterion

^{1} is an **immediate ancestor** of ^{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

^{1} is an **immediate ancestor** of ^{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.

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).

T_n is the restriction of T allowing only types of level n in typing derivations.

CT_n is the restriction of CT allowing only types of level n in typing derivations.

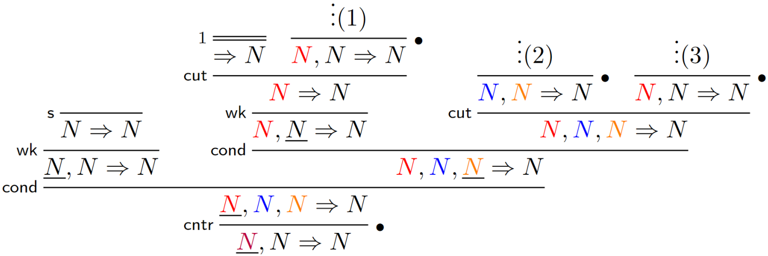
Example: Ackermann-Péter

$$\begin{aligned} A(0; y) &:= y + 1 \\ A(x+1; 0) &:= A(x; 1) \\ A(x+1; y+1) &:= A(x; A(x+1; y)) \end{aligned}$$

NB. Not representable in T_0!

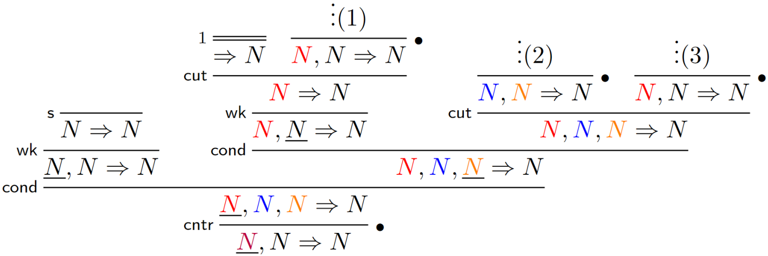
$$
\begin{aligned}
A(0; y) &:= y + 1 \\
A(x + 1; 0) &:= A(x; 1) \\
A(x + 1; y + 1) &:= A(x; A(x + 1; y))
\end{aligned}
$$

NB. Not representable in T_0! However:



$$
\begin{aligned}
A(0;y) &:= y+1\\
A(x+1;0) &:= A(x;1)\\
A(x+1;y+1) &:= A(x;A(x+1;y))
\end{aligned}
$$

NB. Not representable in T_0! However:



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.

Progress =) totality

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.

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

Confluence of T

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

Write for reflexive symmetric transitive closure of .

Confluence of T

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

Write for reflexive symmetric transitive closure of .

Theorem (Confluence for CT, RCA_0)

If s t_0 and s t_1 then there is some t with t_0 t and t_1 t.

Thanks to confluence, we can recast the model of hereditary recursive operations as a type structure HR on coterms.

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 : :

Theorem (RCA$_0$ + I $_{n+2}$)

t 2 HR .

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 : :

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

t 2 HR .

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. □

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 :   :

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

t 2 HR .

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 HR on coterms. In particular, for any CT_n-coterm t : :

Theorem (RCA$_0$ + I $_{n+2}$)

t 2 HR .

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. In particular for any CT_n-coderivation t:

Corollary (RCA$_0$ + I $_{n+2}$)

t is weakly normalising wrt .

Interpretation 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.

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* $T_{n+1} \vdash s \quad t$.

Corollary (Computation at type 1)

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

Outline

Other results

By formalising a model of 'convertibility' à la Tait, we obtain:

Theorem (Strong normalisation)

Let t be representable in CT. Then ACA_0 proves that t is strongly normalising.

Other results

By formalising a model of 'convertibility' à la Tait, we obtain:

Theorem (Strong normalisation)

Let t be representable in CT. Then 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.

Other results

By formalising a model of 'convertibility' à la Tait, we obtain:

Theorem (Strong normalisation)

Let t be representable in CT. Then 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 ACA_0, we recover recursion along progressing coderivations directly in T:

Theorem (Functional equivalence)

CT and T comptue the same functionals, at all types.

Summary and open questions

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.

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 primitive recursive functions (at type 1).

Progressing coterms primitive recursive function*als* (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 primitive recursive functions (at type 1).
- Progressing coterms primitive recursive function*als* (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 primitive recursive functions (at type 1).

Progressing coterms primitive recursive function*als* (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.