Unravelling Abstract Cyclic Proofs into Proofs by Induction

Lide Grotenhuis and Daniël Otten
ILLC, University of Amsterdam

Cyclic Proofs

Idea: replace (co)induction rules with sound circular reasoning.

Example. We have a cyclic proof of $x + 0 = x$ in arithmetic:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{suc}\, x + 0 = \mathsf{suc}\,(x + 0)}\,\text{axiom} \qquad x + 0 = x}{\vdots}}{\dfrac{\dfrac{}{0 + 0 = 0}\,\text{axiom} \qquad \mathsf{suc}\, x + 0 = \mathsf{suc}\, x}{x + 0 = x}\,\mathsf{case}_x}\ \text{cycle}$$

Might look unfamiliar, but proof assistants allow this:

$$\begin{aligned} &\mathsf{unital} : (x : \mathbb{N}) \to (x + 0 = x), \\ &\mathsf{unital}\, 0 := \mathsf{refl}, \qquad \text{recursive call} \\ &\mathsf{unital}\,(\mathsf{suc}\, x) := \mathsf{ap}_{\mathsf{suc}}(\overbrace{\mathsf{unital}\, x}). \end{aligned}$$

Termination

Sound circular reasoning means terminating recursive functions.

Halting problem, so we pick a decidable termination condition:

- Structural Recursion (`Lean`, `Rocq`) (by default): every recursive call is to the same function and decreases the same inductive input.
- Size-Change Termination (`Agda`, `Lambdapi`, `Idris`, `Isabelle/HOL`): for every infinite sequence of recursive calls, we can eventually trace an inductive input that decreases infinitely often.

  This condition is a lot more permissive:

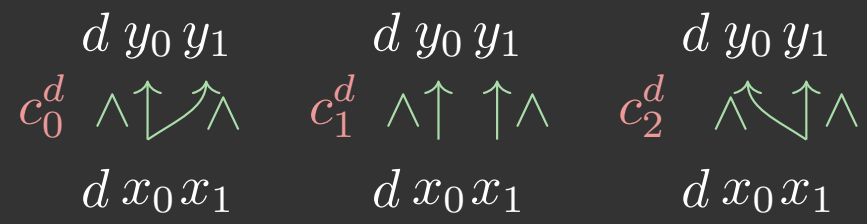
  - mutually recursive functions,
  - multiple inputs (lexicographic ordering, swapping, delayed tracking),
  - easier to extend to mixed (co)induction.

  Is it still sound?

Example: Distance

Consider the following inefficient distance function $d : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$:

$$
\begin{aligned}
d\,0\,x_1 &\coloneqq x_1, \\
d\,(\mathsf{suc}\,x_0')\,0 &\coloneqq \mathsf{suc}\,x_0', \\
d\,(\mathsf{suc}\,x_0')\,(\mathsf{suc}\,x_1') &\coloneqq \underbrace{d\,x_0'\,x_0'}_{c_0^d} + \underbrace{d\,x_0'\,x_1'}_{c_1^d} + \underbrace{d\,x_1'\,x_1'}_{c_2^d}.
\end{aligned}
$$



It is not structurally recursive, meaning: there exists no inductive input that stays in the same position and decreases in every recursive call.

But it is (size-change) terminating: every infinite sequence of recursive calls eventually has a path of an inductive input that decreases infinitely.

So, some proof assistants (`Agda`, `Lambdapi`, `Idris`, `Isabelle/HOL`) accept it, but can we translate it to a term in Martin-Löf type theory?

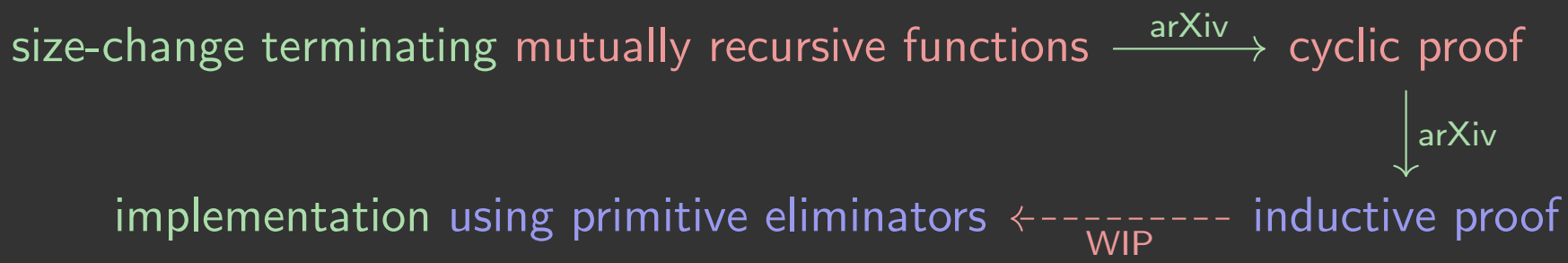
Compared with Induction

Both Structural Recursion and Size-Change Termination only accept terminating functions.

However, can we implement these functions using primitive eliminators?

- Structural Recursion: Yes (known):
  - implement pattern matching (with[0] and without[1] axiom K),
  - one induction on the structural well-order for the inductive input.
- Size-Change Termination: Work in progress:
  - implement pattern matching using known techniques[0,1],
  - do various nested inductions on different functions and different inputs in the correct order.

We approach this by proving an abstract cyclic proof theory result:



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

Overview

Unravelling cycles into induction:

- So far, this has been done for various specific proof systems[0].
- We show how to do this in a general abstract way.
- In addition, we want to preserve the structure of the proof.

In particular:

- We start with an abstract cyclic proof system $\mathcal{C}$.
- We define an induced (non-cyclic) proof system $\mathcal{C}_{\text{ind}}$ that extends the rules of $\mathcal{C}$ with intuitionistic well-founded induction.
- We show how any cyclic proof π in $\mathcal{C}$ can be transformed into a finite proof π_{ind} in $\mathcal{C}_{\text{ind}}$ with the same conclusion and structure.

Our approach is greatly inspired by the PhD thesis of Dominik Wehr:

- Our main improvement is covering non-linear inductive sorts.

[0] arithmetic, modal/linear/first-order μ-calculus, inductive definitions, ...

Cyclic Proof System

Definition (cyclic proof system)

A *proof system* consists of:

- a set Judg *(the judgments)*, a set Rule *(the rule instances)*,
- for each rule r : Rule an associated finite list of *premises* $\mathsf{prem}\, r : \mathsf{Judg}$ and a *conclusion* $\mathsf{concl}\, r : \mathsf{Judg}$.

A *cyclic proof system* is a proof system with additional structure:

- for each judgment A : Judg a finite set of *(trace) objects* $\mathsf{Ob}\, A$,
- for each rule r : Rule and each premise index i a *size-change graph*

$$(\mathsf{graph}\, r)_i : \mathsf{Ob}\,(\mathsf{concl}\, r) \to \mathsf{Ob}\,(\mathsf{prem}\, r)_i.$$

This is a bipartite graph where edges are labelled as either:

$$\geq \text{ (preserving)} \qquad \text{or} \qquad > \text{ (progressing)}.$$

Proof

Definition (Proof)

A *derivation* is a (possibly infinite) tree where every node is labelled by a rule in a compatible way.

A proof is a *regular* derivation (it only has finitely many subderivations) such that for every infinite branch, we can eventually find a path of trace objects that has infinitely many progressing edges.

Every set of mutually recursive functions induces a cyclic proof system:

- for every function f we have a judgment $\downarrow f$ (read: f *terminates*) where the trace objects are the inputs, and a rule

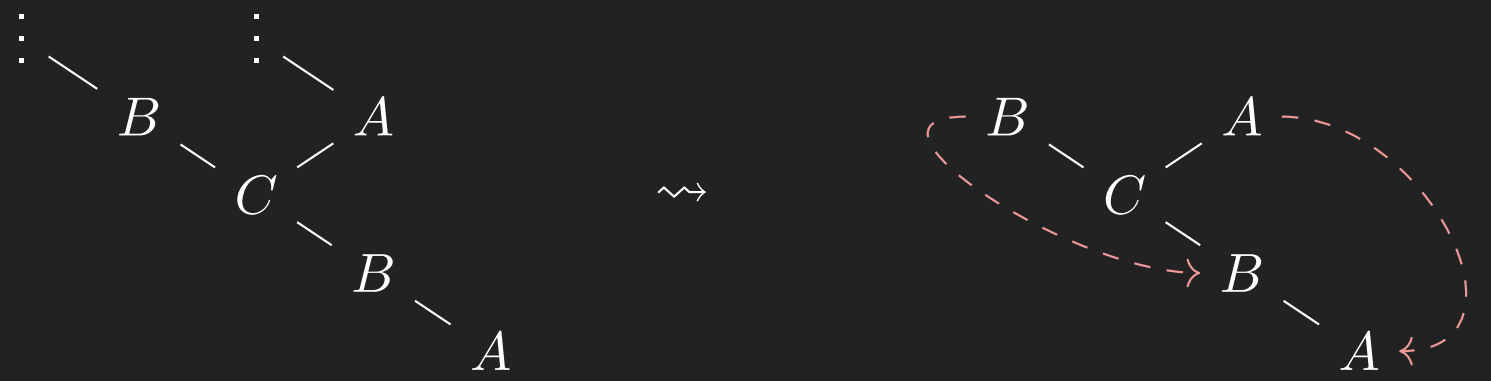
$$\frac{\downarrow g \qquad \text{for every call within } f \text{ to } g}{\downarrow f}\, f,$$

  where $(\text{graph}\, r)_i : \text{Input}\, f \to \text{Input}\, g$ is given by the structural order.
- f is Size-Change Terminating iff unique derivation of f is a proof.

Cyclic Representation

Proofs can be infinite but regular, so we can represent them with cycles:

Example. The following infinite proof has a cyclic representation:



Inductive Proof System

Given a cyclic proof system $\mathcal{C}$ we define a (non-cyclic) proof system $\mathcal{C}_{\mathsf{ind}}$.
The judgements of $\mathcal{C}_{\mathsf{ind}}$ are sequents $\gamma_0, \dots, \gamma_{n-1} \vdash \delta$ of formulas:

$$\phi, \psi, \dots \coloneqq A\,\overline{x} \mid x > y \mid x \geq y \mid \phi \to \psi \mid \forall x\, \psi.$$

Here A is a judgment of $\mathcal{C}$, treated as a relation symbol of arity $\mathsf{Ob}\, A$.

$\mathcal{C}_{\mathsf{ind}}$ has intuitionistic $\to$ and $\forall$ rules, quasi-order rules for $>$ and $\geq$, and:

$$x' \text{ fresh } \frac{\Gamma, \forall x' \,(x > x' \to \phi[x'/x]) \vdash \phi}{\Gamma \vdash \forall x\, \phi} >_{\mathsf{ind}_x}.$$

Lastly, for every $\mathcal{C}$-rule

$$\frac{A_i \qquad \text{for every } i < n}{A}\, r,$$

with size-change graphs $(g_i : A \to A_i)_{i<n}$, we have a $\mathcal{C}_{\mathsf{ind}}$-rule

$$\overline{y} \text{ fresh } \frac{\Gamma, \overline{x} \gtrsim_{g_i} \overline{y} \vdash A_i\, \overline{y} \qquad \text{for every } i < n}{\Gamma \vdash A\, \overline{x}}\, r.$$

Here $\overline{x} \gtrsim_{g_i} \overline{y}$ contains (strict/non-strict) inequalities according to g_i.

General Idea

We will transform a (possibly infinite) $\mathcal{C}$-proof π of A_0 into a finite $\mathcal{C}_{\mathsf{ind}}$-proof π_{ind} of $\vdash A_0\,\overline{x}_0$:

- We first translate π into a (possibly infinite) derivation $\pi_{\mathsf{ind}}^{\mathsf{inf}}$ in $\mathcal{C}_{\mathsf{ind}}$:

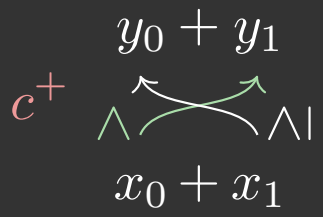
$$\dfrac{\dfrac{\dfrac{\vdots}{A_3}}{A_2}\,r_2}{A_1}\,r_1}{A_0}\,r_0 \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\vdots}{\overline{x_0} \gtrsim_{g_0} \overline{x_1}, \overline{x_1} \gtrsim_{g_1} \overline{x_2}, \overline{x_2} \gtrsim_{g_2} \overline{x_3} \vdash A_3\,\overline{x_3}}\,r_2}{\overline{x_0} \gtrsim_{g_0} \overline{x_1}, \overline{x_1} \gtrsim_{g_1} \overline{x_2} \vdash A_2\,\overline{x_2}}\,r_1}{\overline{x_0} \gtrsim_{g_0} \overline{x_1} \vdash A_1\,\overline{x_1}}\,r_0}{\vdash A_0\,\overline{x_0}}$$

- We define π_{ind} by 'cutting' the infinite branches of $\pi_{\mathsf{ind}}^{\mathsf{inf}}$ short: we follow a nice cyclic representation π_{cyc} of π; for each cycle we introduce an induction hypothesis below and apply it above.

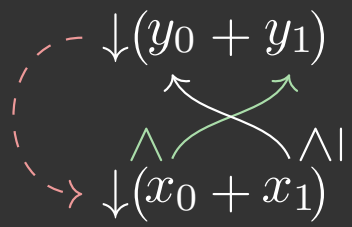
A Simple Example

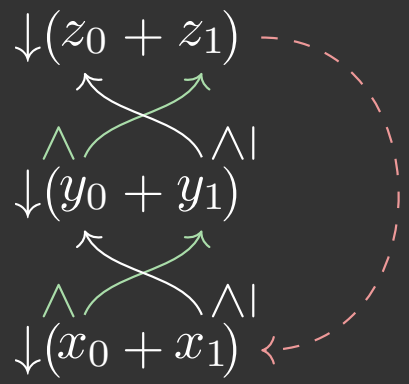
Consider the swapped addition function:

$$0 + x_1 \coloneqq x_1,$$
$$(\mathsf{suc}\, x_0') + x_1 \coloneqq \mathsf{suc}\,(\underbrace{x_1 + x_0'}_{c^+}),$$



The unique proof has (among others) the following two cyclic representations, where we have visualized the size-change graphs:





The second representation can be used to prune the infinite derivation $\pi_{\mathsf{ind}}^{\mathsf{inf}}$ into a finite proof π_{ind} (since there is an input with progress).

A Simple Example

The infinite derivation $\pi^{\mathsf{inf}}_{\mathsf{ind}}$ is:

$$\dfrac{\dfrac{\dfrac{\vdots}{x_0 > y_1, x_1 \geq y_0, y_0 > z_1, y_1 \geq z_0 \vdash \downarrow(z_0 + z_1)}+}{x_0 > y_1, x_1 \geq y_0 \vdash \downarrow(y_0 + y_1)}+}{\vdash \downarrow(x_0 + x_1)}+$$

The cyclic representation of the previous slide yields π_{ind}:

$$\dfrac{\dfrac{\dfrac{(\text{use } \mathsf{hyp}[x_0] \text{ on } x'_0 := z_0 \text{ and } x'_1 := z_1)}{\mathsf{hyp}[x_0], x_0 > y_1, x_1 \geq y_0, y_0 > z_1, y_1 \geq z_0 \vdash \downarrow(z_0 + z_1)}+}{\mathsf{hyp}[x_0], x_0 > y_1, x_1 \geq y_0 \vdash \downarrow(y_0 + y_1)}+}{\dfrac{\mathsf{hyp}[x_0] \vdash \downarrow(x_0 + x_1)}{\vdash \downarrow(x_0 + x_1)} >_{\mathsf{ind}'_{x_0}},}$$

where $\mathsf{hyp}[x_0] := \forall x'_0 x'_1\, (x_0 > x'_0 \to \downarrow(x'_0 + x'_1))$.

A Simple Example

The cyclic representation gives an inlined swapped addition:

$$\begin{aligned} 0 + x_1 &:= x_1, \\ (\mathsf{suc}\, x_0') + x_1 &:= \mathsf{suc}\, (x_1 + x_0'), \end{aligned} \quad \rightsquigarrow \quad \begin{aligned} 0 + x_1 &:= x_1, \\ (\mathsf{suc}\, x_0') + 0 &:= \mathsf{suc}\, x_0', \\ (\mathsf{suc}\, x_0') + (\mathsf{suc}\, x_1') &:= \mathsf{suc}\, (\mathsf{suc}\, (x_0' + x_1')). \end{aligned}$$

We can implement the inlined definition using primitive eliminators:

$$x_0 + x_1 := <\text{-ind}\, x_0\, \{x_0 : \mathbb{N}, x_1 : \mathbb{N}, h : \Pi_{x_0', x_1'}\, ((x_0 > x_0') \to \mathbb{N}) \mapsto \mathsf{inner},$$

$$\mathsf{inner} := \mathsf{case}\, x_0 \begin{cases} 0 \mapsto x_1, \\ \mathsf{suc}\, x_0' \mapsto \mathsf{case}\, x_1 \begin{cases} 0 \mapsto \mathsf{suc}\, x_0', \\ \mathsf{suc}\, x_1' \mapsto \mathsf{suc}\, (\mathsf{suc}\, (h\, x_0'\, x_1'\, \mathsf{ineq}_{x_0}^{x_0'})). \end{cases} \end{cases}$$

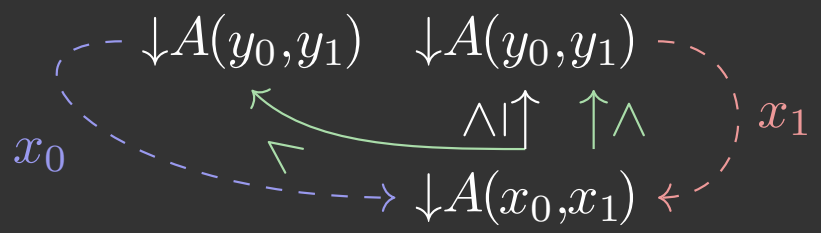
where $<$-ind and case are defined using ind.

Warning: The inlined definition and primitive implementation do not reduce $(\mathsf{suc}\, x_0') + x_1$ to $\mathsf{suc}\, (x_1 + x_0')$ but they compute on closed terms.

Homework: Implement swapped addition while preserving full computational behaviour or prove that this is impossible.

Importance of Induction Order

The termination proof of the Ackermann function has cyclic repr:



We introduce an induction hypothesis for x_0 and x_1 in order:

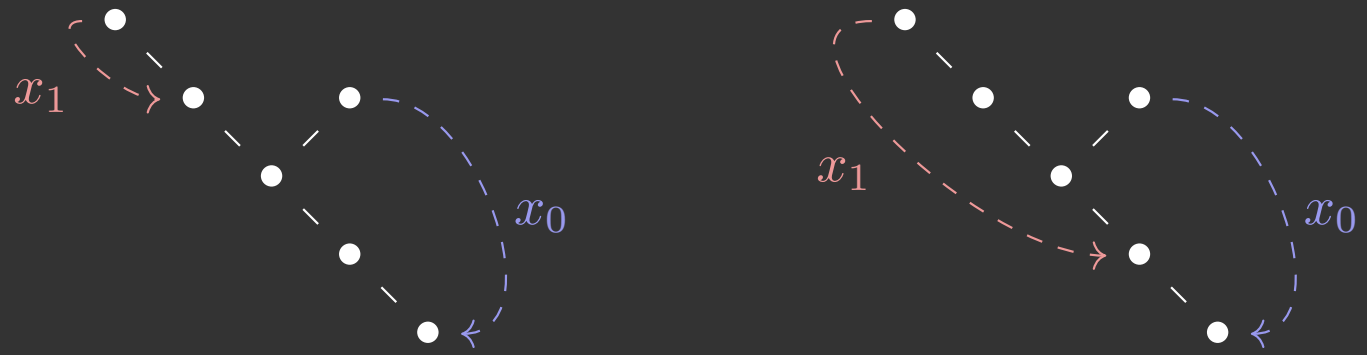
$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{use } \mathsf{hyp}_0(x_0) \text{ on } x_i' := y_i) \quad (\text{use } \mathsf{hyp}_1(x_1) \text{ on } x_i' := y_i)}{\ldots, x_0 > y_0 \vdash \downarrow A(y_0,y_1) \quad \ldots, x_0 \geq y_0, x_1 > y_1 \vdash \downarrow A(y_0,y_1)}\, A}{\mathsf{hyp}_0(x_0), \mathsf{hyp}_1(x_1) \vdash \downarrow A(x_0,x_1)}\, >_{\mathsf{ind}'_{x_1}}}{\mathsf{hyp}_0(x_0) \vdash \downarrow A(x_0,x_1)}\, >_{\mathsf{ind}'_{x_0}}}{\vdash \downarrow A(x_0,x_1)},$$

where the induction hypotheses are:

$$\mathsf{hyp}_0(x_0) := \forall x_0' \, \forall x_1' \, (x_0 > x_0' \to \downarrow A(x_0', x_1')),$$
$$\mathsf{hyp}_1(x_1) := \forall x_0' \, \forall x_1' \, (x_1 > x_1' \to \mathsf{hyp}_0(x_0') \to \downarrow A(x_0', x_1')).$$

Importance of Forgetting

Consider the following two forms of cyclic representations:



For both, if we want to transform them into finite $\mathcal{C}_{\text{ind}}$-proofs, we have to introduce an induction hypothesis at the root for x_0:

- in the first example, we can forget the x_0 induction hypothesis (using weakening) before introducing the x_1 induction hypothesis;
- in the second example this is not possible, so in this example the back-edge for x_1 has to 'preserve' x_0 in some way.

General Technique

The main technical difficulty is finding a suitable cyclic representation:

- We use annotations based on 'reset proofs' and the 'Safra construction' to find a cyclic representation with an induction order.
- We modify reset proofs to deal with non-linear inductive sorts.
- We use a technique of Sprenger and Dam to unfold the tree until the order of cycles in the tree respects the induction order.

After this, we use the annotations to guide what we forget, and the cycles to determine when we should introduce an induction hypothesis.

Theorem

For every $\mathcal{C}$-proof π of A, there exists a $\mathcal{C}_{\text{ind}}$-proof π_{ind} of $\vdash A\,\overline{x}_0$. Moreover, π_{ind} preserves the structure of π.

(Cyclic) Heyting Arithmetic

Example. Robinson Arithmetic (Q) is intuitionistic first-order logic with function symbols $0, \mathsf{suc}, +, \times$, the following axioms

$$\begin{aligned} 0 + y &= y, & \mathsf{suc}\,(x') + y &= \mathsf{suc}\,(x' + y), \\ 0 \times y &= 0, & \mathsf{suc}\,(x') \times y &= (x' \times y) + y, \\ 0 &\neq \mathsf{suc}\,(y') & \mathsf{suc}\,(x') &= \mathsf{suc}\,(y') \to x' = y', \end{aligned}$$

and an additional rule:

$$\frac{(\Gamma \vdash \delta)[0/x] \qquad (\Gamma \vdash \delta)[\mathsf{suc}\,(x)/x]}{\Gamma \vdash \delta}\,\mathsf{case}_x$$

Heyting Arithmetic (HA) is Q extended with induction.

Cyclic Heyting Arithmetic (CHA) is Q as a cyclic proof system: the trace objects are the free variables, variable x progresses iff it passes case_x.

Application: (Cyclic) Heyting Arithmetic

Our results can be used to transform a CHA (cyclic Heyting arithmetic) proof into a normal HA (Heyting arithmetic) proof:

- First we translate the CHA-proof into a $\mathrm{CHA}_{\mathrm{ind}}$-proof.
- Because HA and CHA are already first-order theories containing intuitionistic rules for $\to$ and $\forall$, and we can define

$$(n \leq m) \coloneqq \exists x\,(n + x = m), \qquad (n < m) \coloneqq (\mathsf{suc}\,(n) \leq m),$$

  and translate the $\mathrm{CHA}_{\mathrm{ind}}$-proof into a HA-proof.

The same strategy gives a transformation from CPA into PA.

We need a similar translation for type theory:

- We worked this out for non-indexed inductive types but we need to consider computational behaviour more carefully.
- For indexed inductive types, we need to combine this with work on eliminating pattern matching (with[0] and without[1] axiom K).

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

Conclusion

Theorem

For every $\mathcal{C}$-proof π of A, there exists a $\mathcal{C}_{\text{ind}}$-proof π_{ind} of $\vdash A(\overline{x}_0)$. Moreover, π_{ind} preserves the structure of π.

Future work:

- Work out the application to size-change termination:
  - check preservation computational behaviour,
  - extend to indexed inductive types.
- Extend to a mix between inductive and coinductive sorts:
  - either by allowing both inductive and coinductive traces,
  - or through ordinal approximations or sized types.

Preprint

https://arxiv.org/abs/2602.12054

Size-change Graph

Definition (size-change graph)

A *size-change graph* $G : X \to Y$ is a bipartite graph from a finite set X to a finite set Y where edges are labelled as either:

$$\geq \text{ (preserving)} \qquad \text{or} \qquad > \text{ (progressing)}.$$

A *trace* through a sequence of size-change graphs $(G_i : X_i \to X_{i+1})_i$ consists of a starting time $k \in \mathbb{N}$ and nodes $(x_i \in X_i)_{i \geq k}$ such that x_i and x_{i+1} are connected by an edge in G_i.

We call an infinite sequence *progressing* if there exists a trace where the connecting edge is progressing infinitely often.

Examples:

Structural Well-order

For $\mathrm{W}_{x:A}\, B$ we define the (definitional) structural well-order:

$$\frac{}{c\, b \lessdot_{\mathrm{W}_{x:A}\, B} \mathsf{tree}\, a\, c}\ \lessdot\text{-base}, \qquad \frac{t' \lessdot_{\mathrm{W}_{x:A}\, B} c\, b}{t' \lessdot_{\mathrm{W}_{x:A}\, B} \mathsf{tree}\, a\, c}\ \lessdot\text{-step},$$

We define the propositional structural well-order as an indexed inductive type $<_{\mathrm{W}_{x:A}\, B} : \mathrm{W}_{x:A}\, B \to \mathrm{W}_{x:A}\, B \to \mathrm{Type}$ with constructors:

$$\begin{aligned}
\mathsf{base} &: \{x : A, z : (y : B) \to \mathrm{W}_{x:A}\, B, y : B\} \to \\
&\quad (z\, y <_{\mathrm{W}_{x:A}\, B} \mathsf{tree}\, x\, z) \\
\mathsf{step} &: \{x : A, z : (y : B) \to \mathrm{W}_{x:A}\, B, y : B, w' : \mathrm{W}_{x:A}\, B\} \to \\
&\quad (w' <_{\mathrm{W}_{x:A}\, B} x\, y) \to (w' <_{\mathrm{W}_{x:A}\, B} \mathsf{tree}\, x\, z)
\end{aligned}$$

This generalises to any indexed inductive type: for example, for $\mathrm{Vec}\, A : \mathbb{N} \to \mathrm{Type}$ we have $< : \Sigma_{n:\mathbb{N}} \mathrm{Vec}\, A\, n \to \Sigma_{n:\mathbb{N}} \mathrm{Vec}\, A\, n \to \mathrm{Type}$:

$$\begin{aligned}
&\mathsf{base} : \{n : \mathbb{N}, l : \mathrm{Vec}\, A\, n, a : A\} \to (n, l) < (\mathsf{suc}\, n, l :: a), \\
&\mathsf{step} : \{n, n', l, l', a\} \to (n', l') < (n, l) \to (n', l') < (\mathsf{suc}\, n, l :: a).
\end{aligned}$$

Structural Well-order

The definitional order implies the propositional order: if $t' \ll t$, then we will write $\mathsf{ineq}_t^{t'} : t' < t$ for the canonical term build using base and step.

For this notion, we can derive well-founded induction:

$$\frac{\Gamma \vdash t : \mathrm{W}_{x:A}\, B \qquad \Gamma, w : \mathrm{W}_{x:A}\, B \vdash P : \mathsf{Type} \qquad \Gamma, w : \mathrm{W}_{x:A}\, B, h : \Pi_{w',s:w'<w}\, P[w'] \vdash p : P}{\begin{array}{l}\Gamma \vdash <\text{-}\mathsf{ind}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t\,\{w, h \mapsto p\} : P[t] \\ \Gamma \vdash \mathsf{below}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t\,\{w, h \mapsto p\} : \Pi_{w',s:w'<t}\, P[w']\end{array}} <\text{-ind},$$

$$\frac{}{\begin{array}{l}\Gamma \vdash <\text{-}\mathsf{ind}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t\,\{w, h \mapsto p\} \equiv p[t, \mathsf{below}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t\,\{w, h \mapsto p\}] \\ \Gamma \vdash \mathsf{below}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t\,\{w, h \mapsto p\}\, t'\, \mathsf{ineq}_t^{t'} \equiv <\text{-}\mathsf{ind}_{w.\,P}^{\mathrm{W}_{x:A}\, B}\, t'\,\{w, h \mapsto p\}\end{array}} <\text{-comp}.$$

So, $\Pi_{w',s:w'<w}\, P[w']$ is an alternative for:

$$\mathsf{Below}_{w.\,P} \coloneqq \mathsf{ind}_{w.\mathcal{U}}^{\mathrm{W}_{x:A}\, B}\, w\,\{\mathsf{tree}\, x\, z, h \mapsto \Pi_{y:B}\, (h\, y \times P[z\, y])\}.$$

From Induction to Cycles

We can translate an HA proof to a CHA proof; we just replace the induction axiom with the cyclic proof:

$$
\dfrac{\dfrac{h \vdash \forall x\, \phi}{\dfrac{\dfrac{\dfrac{\dfrac{}{h \vdash h}\,\text{assum}}{h \vdash \phi[0/x]}\,\land\text{-elim}_0 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{h \vdash h}\,\text{assum}}{h \vdash \forall x\,(\phi \to \phi[\mathsf{suc}\,x/x])}\,\land\text{-elim}_1}{h \vdash \phi \to \phi[\mathsf{suc}\,x/x]}\,\forall\text{-elim} \qquad h \vdash \phi}{h \vdash \phi[\mathsf{suc}\,x/x]}\,\to\text{-elim}}{h \vdash \phi}\,\mathsf{case}_x}{}\,\forall\text{-intro}}{\vdash \phi[0/x] \land \forall x'\,(\phi[x'/x] \to \phi[\mathsf{suc}\,x'/x]) \to \forall x\,\phi}\,\to\text{-intro}
$$

x progresses

where $h := \phi[0/x] \land \forall x'\,(\phi[x'/x] \to \phi[\mathsf{suc}\,x'/x])$.

The other direction is more involved and we will cover it in general.

Example: Mutually Recursive Functions

Every set of mutually recursive functions induces a cyclic proof system:

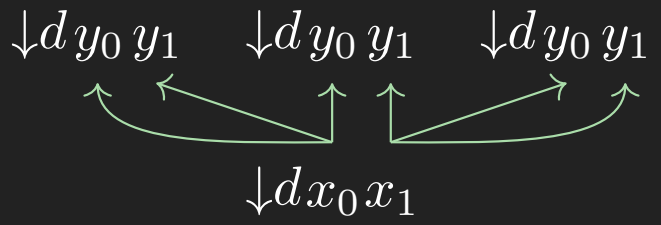
- for every function f we have a judgment $\downarrow f$ (read: f *terminates*) where the trace objects are the inputs, and a rule

$$\frac{\downarrow g \qquad \text{for every call within } f \text{ to } g}{\downarrow f}\, f,$$

  where $(\text{graph}\, r)_i : \text{Input}\, f \to \text{Input}\, g$ is given by the structural order.
- f is size-change terminating iff the unique derivation of f is a proof.

Example. For the distance function d we get:

$$\frac{\downarrow d \qquad \downarrow d \qquad \downarrow d}{\downarrow d}\, d$$



Inductive Proof System

Lastly, for every $\mathcal{C}$-rule

$$\frac{A_i \qquad \text{for every } i < n}{A}\, r,$$

with size-change graphs $(g_i : A \to A_i)_{i<n}$, we have a $\mathcal{C}_{\mathsf{ind}}$-rule

$$\overline{y} \text{ fresh } \frac{\Gamma, \overline{x} \gtrsim_{g_i} \overline{y} \vdash A_i\, \overline{y} \qquad \text{for every } i < n}{\Gamma \vdash A\, \overline{x}}\, r.$$

Here $\overline{x} \gtrsim_{g_i} \overline{y}$ contains (strict/non-strict) inequalities according to g_i.

Example. For the distance function d we get:

$$\frac{\downarrow d \qquad \downarrow d \qquad \downarrow d}{\downarrow d}\, d \quad \rightsquigarrow \quad \frac{\begin{array}{l}\Gamma, x_0 > y_0, x_0 > y_1 \vdash \downarrow d\, y_0\, y_1 \\ \Gamma, x_0 > y_0, x_1 > y_1 \vdash \downarrow d\, y_0\, y_1 \\ \Gamma, x_1 > y_0, x_1 > y_1 \vdash \downarrow d\, y_0\, y_1\end{array}}{\Gamma \vdash \downarrow d\, x_0\, x_1}\, d$$

Induction on the entire sequent

Instead of

$$x' \text{ fresh} \frac{\Gamma, \forall x' \left((x > x') \to \phi[x'/x]\right) \vdash \phi}{\Gamma \vdash \forall x\, \phi} >\text{-ind}_x$$

we will use the following variation

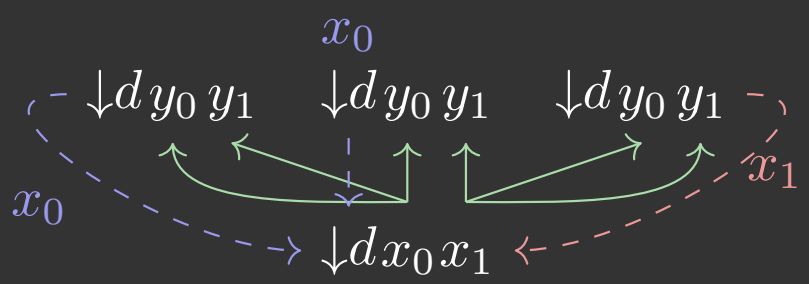
$$x', \overline{z}' \text{ fresh} \frac{\Gamma, \forall x', \overline{z}' \left((x > x') \to (\Gamma \to \delta)[x', \overline{z}'/x, \overline{z}]\right) \vdash \delta}{\Gamma \vdash \delta} >\text{-inds}_x$$

These two induction rules are interderivable; one can derive $>$-inds$_x$ by applying $>$-ind$_x$ to the formula representing the entire sequent:

$$\phi := \forall \overline{z}\, (\Gamma \to \delta).$$

Example: Non-linear

Consider the inefficient distance function d:



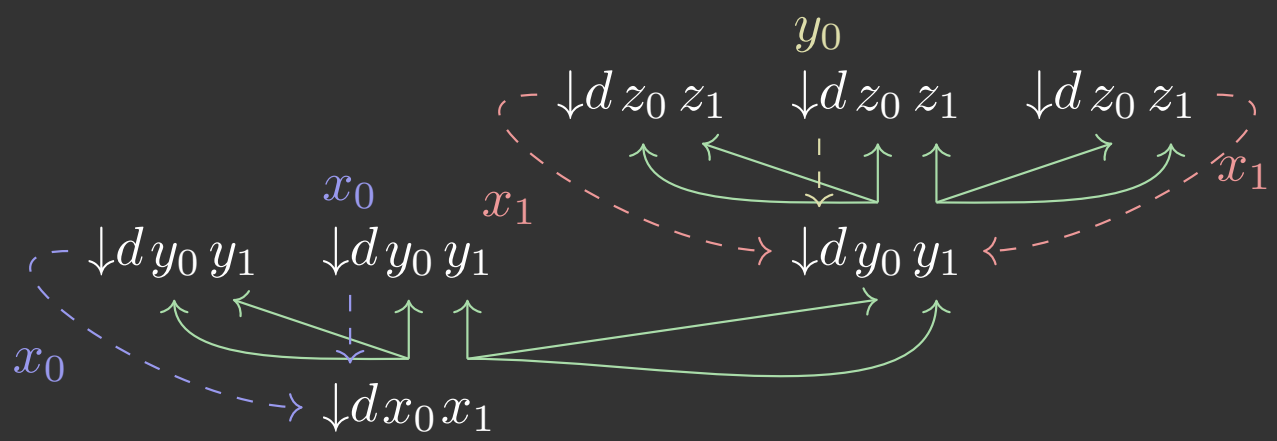
Both induction orders for x_0 and x_1 will not work directly.

Previous work solves this issue by doing induction on $\max\, x_0\, x_1$.

However, this does not work for non-linear inductive sorts, and it is not good for computational behaviour.

Example: Non-linear

Our approach: first unfold one of the cycles as follows



Now we can include history in induction hypotheses:

$$\mathsf{hyp}_0[x_0] \coloneqq \forall x_0'\, \forall x_1'\, ((x_0 > x_0') \to \downarrow d\, x_0'\, x_1'),$$
$$\mathsf{hyp}_1[x_1] \coloneqq \forall x_0'\, \forall x_1'\, \forall y_0'\, \forall y_1'\, ((x_1 > x_1') \to (x_1' > y_0', y_1') \to \downarrow d\, y_0'\, y_1'),$$
$$\mathsf{hyp}_2[y_0] \coloneqq \forall x_0'\, \forall x_1'\, \forall y_0'\, \forall y_1'\, ((y_0 > y_0') \to (x_1' > y_0', y_1') \to \mathsf{hyp}_1[x_1'] \to \downarrow d\, y_0'\, y_1')$$

This uses two tricks:

- we can forget $\mathsf{hyp}_0[x_0]$ before introducing $\mathsf{hyp}_1[x_1]$;
- $x_1' > y_0, y_1$, so we can do induction on x_1 instead of $\max\, y_0\, y_1$.

Pattern Matching

For indexed inductive types, proof assistants unify after every split.

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

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

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

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 $\mathsf{suc}\, m''$.