Towards Cyclic Implicit Complexity

Continuity, Computability, Constructivity 2021

Special session in honour of Ulrich Berger

Gianluca Curzi

University of Birmingham

joint work with Anupam Das (University of Birmingham)

What is this presentation about?

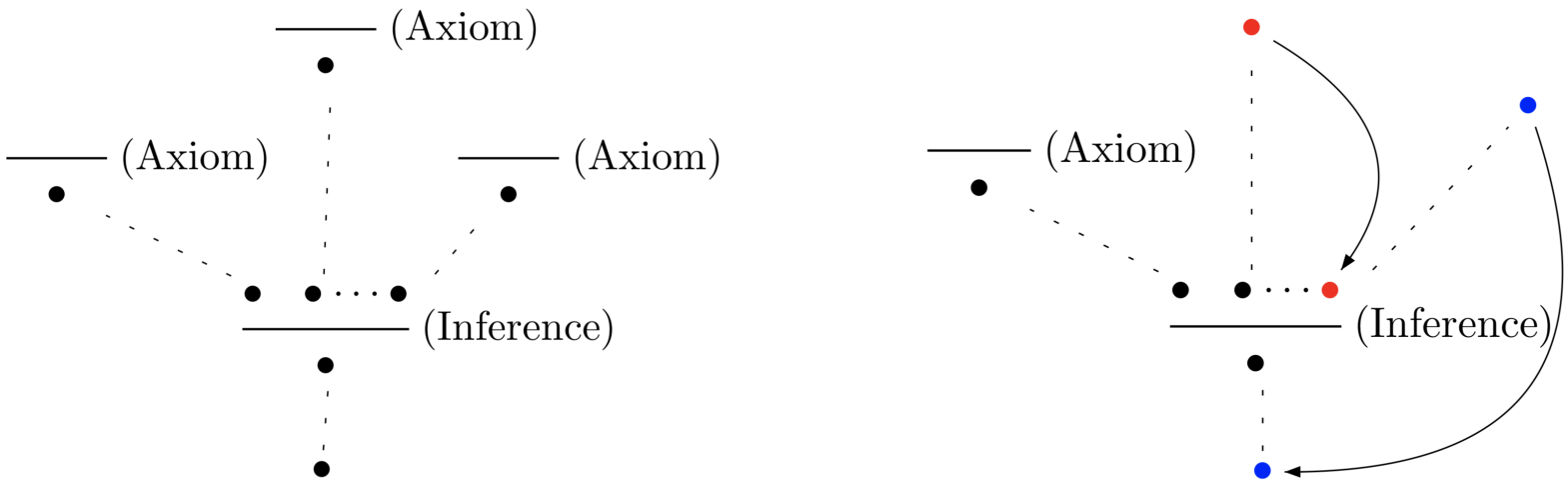


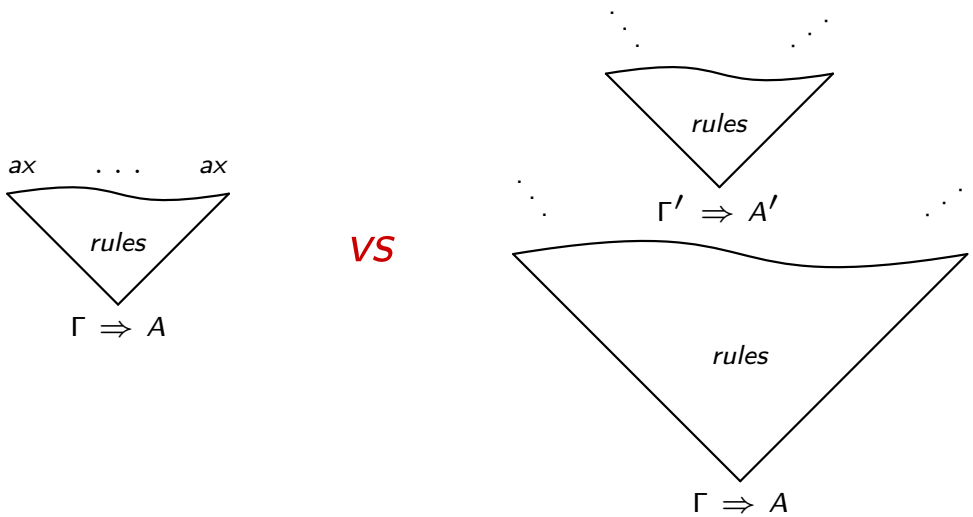
Figure: From "Introduction to cyclic proofs" (Brotherston 2008).

- Goal: cyclic proof systems to capture complexity classes in the style of ICC.
- Some motivations:
  - cyclic proofs **subsume** recursion;
  - relatively **new topic**;
  - hard to **tame complexity**.

.

Non-wellfounded proofs

- Inductive vs non-wellfounded proofs:



- **Non-wellfounded proofs** to reason about μ-calculus (e.g. [Dax, Hofmann and Lange 06], [Niwinski and Walukiewicz 96]), (co)induction (e.g. [Brotherston and Simpson 11]), Kleene algebra (e.g. [Das and Pous 17, 18]), linear logic (e.g. [Baelde, Doumane and Saurin 16]), continuous cut-elimination (e.g. [Mints 75] and [Fortier and Santocanale 13]).

- **Problem**. Any formula is derivable!

$$\dfrac{\dfrac{\vdots}{\Rightarrow A} \quad \dfrac{}{A \Rightarrow A}\,\mathsf{id}}{\Rightarrow A}\,\mathsf{cut} \qquad \dfrac{}{A \Rightarrow A}\,\mathsf{id}}{\Rightarrow A}\,\mathsf{cut}$$

- Progressiveness criterion = global criterion to guarantee consistency.

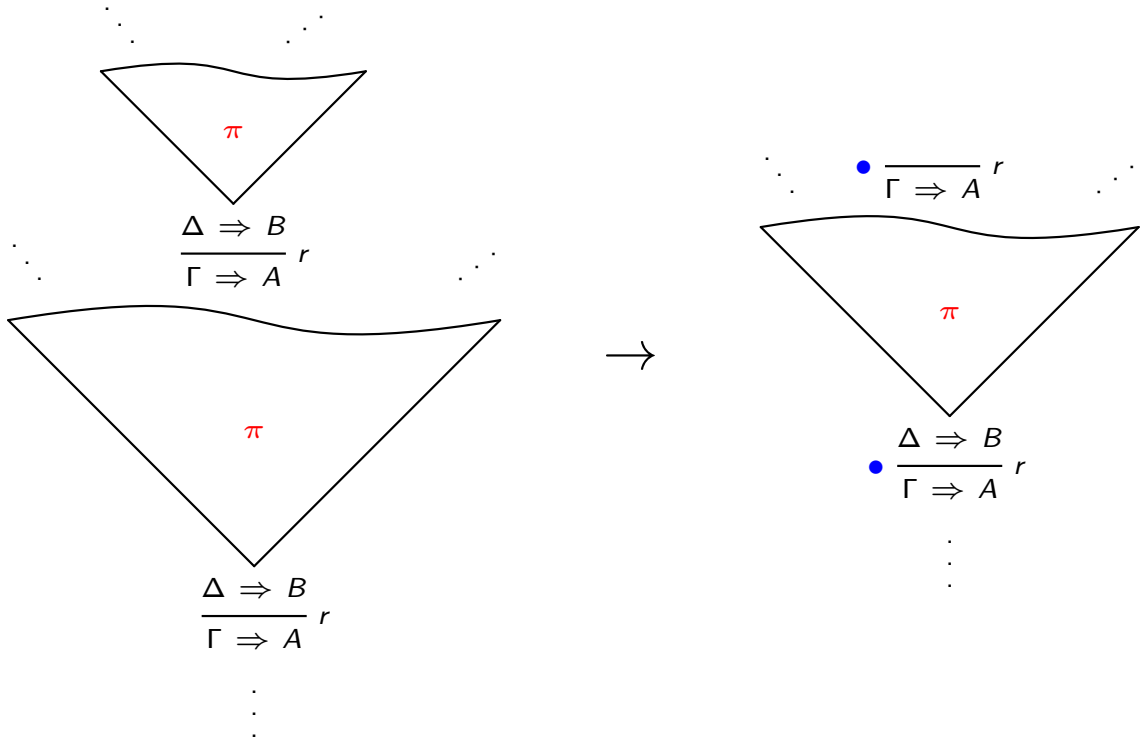
- **Problem**. Any formula is derivable!

$$\dfrac{\dfrac{\vdots}{\Rightarrow A} \quad \dfrac{}{A \Rightarrow A}\,\mathsf{id}}{\Rightarrow A}\,\mathsf{cut} \qquad \dfrac{}{A \Rightarrow A}\,\mathsf{id}}{\Rightarrow A}\,\mathsf{cut}$$

- Progressiveness criterion = global criterion to guarantee consistency.

Cyclic proofs

- Cyclic proofs = only **finitely** many distinct subproofs.
- Cycle normal form = finite, "circular" presentation of a cyclic proof.



Implicit computational complexity (ICC)

- Implicit computational complexity (ICC) = characterize complexity classes by means of languages/calculi **without** explicit reference to machine models or external resource bounds.

- Originates in the 90's with the Bellantoni and Cook's paper on *safe recursion*.

- Pervasive notion of **stratification**: data are organized into *strata* (Bellantoni's safe recursion, Leivant's predicative/ramified/tiered recursion).

Safe recursion on notation

- Function algebra B characterizing **FPTIME** [Bellantoni and Cook 92].
- Two successors: $s_0 x = 2x$ and $s_1 x = 2x + 1$.
- Function arguments partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_0 x, \vec{x}; \vec{y}) = h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$
$$f(s_1 x, \vec{x}; \vec{y}) = h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

Idea. Recursive calls **only** in the safe zone:

Safe recursion on notation

- Function algebra B characterizing **FPTIME** [Bellantoni and Cook 92].
- Two successors: $s_0 x = 2x$ and $s_1 x = 2x + 1$.
- Function arguments partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_0 x, \vec{x}; \vec{y}) = h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$
$$f(s_1 x, \vec{x}; \vec{y}) = h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

Idea. Recursive calls **only** in the safe zone:

Safe recursion on notation

- Function algebra B characterizing **FPTIME** [Bellantoni and Cook 92].
- Two successors: $s_0 x = 2x$ and $s_1 x = 2x + 1$.
- Function arguments partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_0 x, \vec{x}; \vec{y}) = h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$
$$f(s_1 x, \vec{x}; \vec{y}) = h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

Idea. Recursive calls **only** in the safe zone:

Safe recursion on notation

- Function algebra B characterizing **FPTIME** [Bellantoni and Cook 92].
- Two successors: $s_0 x = 2x$ and $s_1 x = 2x + 1$.
- Function arguments partitioned into normal and safe:

$$f(x_1, \ldots, x_n \,;\, y_1, \ldots, y_m)$$

- Safe recursion on notation:

$$\begin{aligned} f(0, \vec{x}; \vec{y}) &= g(\vec{x}; \vec{y}) \\ f(s_0 x, \vec{x}; \vec{y}) &= h_0(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \\ f(s_1 x, \vec{x}; \vec{y}) &= h_1(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y})) \end{aligned}$$

Idea. Recursive calls **only** in the safe zone:

Non-wellfounded version of B

- Formulas $A, B, C \in \{N, \Box N\}$ and contexts $\Gamma, \Delta = A_1, \ldots, A_n$.

- Non-wellfounded proofs generated by the following rules:

$$\mathrm{id}\ \frac{}{N \Rightarrow N} \qquad \mathrm{w}_N\ \frac{\Gamma \Rightarrow B}{\Gamma, N \Rightarrow B} \qquad \mathrm{w}_\Box\ \frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \mathrm{e}\ \frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C}$$

$$\Box_l\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \Box_r\ \frac{\Box N, \ldots, \Box N \Rightarrow N}{\Box N, \ldots, \Box N \Rightarrow \Box N} \qquad 0\ \frac{}{\Rightarrow N} \qquad \mathrm{s}_0\ \frac{}{N \Rightarrow N} \qquad \mathrm{s}_1\ \frac{}{N \Rightarrow N}$$

$$\mathrm{cut}_N\ \frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow B}{\Gamma \Rightarrow B} \qquad \mathrm{cut}_\Box\ \frac{\Gamma \Rightarrow \Box N \quad \Box N, \Gamma \Rightarrow B}{\Gamma \Rightarrow B}$$

$$\mathrm{cond}_N\ \frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow N \quad \Gamma, N \Rightarrow N}{\Gamma, N \Rightarrow N} \qquad \mathrm{cond}_\Box\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

Semantics of non-wellfounded proofs for B

$$0\ \frac{}{\Rightarrow N} \qquad f_{\mathcal{D}}(;) := 0$$

$$\mathsf{s}_i\ \frac{}{N \Rightarrow N} \qquad f_{\mathcal{D}}(;x) := \mathsf{s}_i x$$

$$\mathrm{cut}\ \frac{\mathcal{D}_0 \quad \mathcal{D}_1}{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow A}{\Gamma \Rightarrow A} \qquad f_{\mathcal{D}}(\vec{x};\vec{y}) := f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y}))$$

$$\mathrm{cut}_{\Box}\ \frac{\mathcal{D}_0 \quad \mathcal{D}_1}{\Gamma \Rightarrow \Box N \quad \Box N, \Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad f_{\mathcal{D}}(\vec{x};\vec{y}) := f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x};\vec{y}), \vec{x};\vec{y})$$

$$\mathrm{cond}_{\Box}\ \frac{\mathcal{D}_0 \quad \mathcal{D}_1 \quad \mathcal{D}_2}{\Gamma \Rightarrow N \quad \Box N, \Gamma, \Rightarrow N \quad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

$$\begin{aligned} f_{\mathcal{D}}(0, \vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\ f_{\mathcal{D}}(\mathsf{s}_0 x, \vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(x, \vec{x};\vec{y}) \\ f_{\mathcal{D}}(\mathsf{s}_1 x, \vec{x};\vec{y}) &:= f_{\mathcal{D}_2}(x, \vec{x};\vec{y}) \end{aligned}$$

Cyclicity

- Cyclic proof = finitely many distinct subproofs.
- **Idea**. Cyclicity = computability.
- **Example.** A cyclic proof $\mathcal{D}$:

$$\dfrac{\dfrac{}{N \Rightarrow N}\,\mathsf{s}_0 \qquad \dfrac{}{\Box N, N \Rightarrow N}\,\mathsf{cut}_N \bullet}{\Box N, N \Rightarrow N}\,\mathsf{cut}_N \bullet$$

$$f_{\mathcal{D}}(x; y) \quad := \quad f_{\mathcal{D}}(x; \mathsf{s}_0 y)$$

Progressiveness

- Progressive proof = every infinite branch contains a $\Box N$-*thread* with infinitely many principal formulas of the rule $\mathrm{cond}_\Box$.

- **Example.** A progressing proof:

$$\dfrac{\mathrm{id}\dfrac{}{N \Rightarrow N} \quad \mathrm{cond}_\Box\dfrac{}{\Box N, N \Rightarrow N}\bullet \quad \mathrm{cut}_N\dfrac{\mathrm{s}_0\dfrac{}{N \Rightarrow N} \quad \Box N, N \Rightarrow N}{\Box N, N \Rightarrow N} \quad \mathrm{cut}_N\dfrac{\mathrm{cond}_\Box\dfrac{}{\Box N, N \Rightarrow N}\bullet \quad \mathrm{s}_1\dfrac{}{N \Rightarrow N} \quad \Box N, N \Rightarrow N}{\Box N, N \Rightarrow N}}{\mathrm{cond}_\Box\dfrac{\Box N, N \Rightarrow N \quad \Box N, N \Rightarrow N}{\Box N, N \Rightarrow N}\bullet}$$

$$\begin{aligned} f_{\mathcal{D}}(0; y) &= y \\ f_{\mathcal{D}}(s_0 x; y) &= s_0(f_{\mathcal{D}}(x; y)) \\ f_{\mathcal{D}}(s_1 x; y) &= s_1(f_{\mathcal{D}}(x; y)) \end{aligned}$$

- **Idea**. Progressiveness = totality.

Safety

- **Problem**. Modalities are not enough to enforce stratification in an non-wellfounded setting.

- **Example.** A cyclic progressing proof $\mathcal{D}$ for primitive recursion (on notation):

$$\mathsf{cond}_{\Box}\ \dfrac{\mathcal{D}_0\ \ \Gamma \Rightarrow N \qquad \mathsf{cut}_{\Box}\ \dfrac{\dfrac{}{\Box N, \Gamma \Rightarrow N}\bullet \qquad \mathcal{D}_1\ \ \Box N, \Gamma, \Box N \Rightarrow N}{\Box N, \Gamma \Rightarrow N} \qquad \mathsf{cut}_{\Box}\ \dfrac{\dfrac{}{\Box N, \Gamma \Rightarrow N}\bullet \qquad \mathcal{D}_2\ \ \Box N, \Gamma, \Box N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}}{\Box N, \Gamma \Rightarrow N}\bullet$$

$$\begin{aligned} f_{\mathcal{D}}(0, \vec{x};) &= f_{\mathcal{D}_0}(\vec{x};) \\ f_{\mathcal{D}}(s_0 x, \vec{x};) &= f_{\mathcal{D}_1}(x, \vec{x}, f(x, \vec{x});) \\ f_{\mathcal{D}}(s_1 x, \vec{x};) &= f_{\mathcal{D}_2}(x, \vec{x}, f(x, \vec{x};);) \end{aligned}$$

- Safe proof = any branch crosses finitely many $\mathsf{cut}_{\Box}$-steps.
- Cyclic proof system NCB = cyclic progressing safe proofs.

Safety

- **Problem**. Modalities are not enough to enforce stratification in an non-wellfounded setting.

- **Example.** A cyclic progressing proof $\mathcal{D}$ for primitive recursion (on notation):

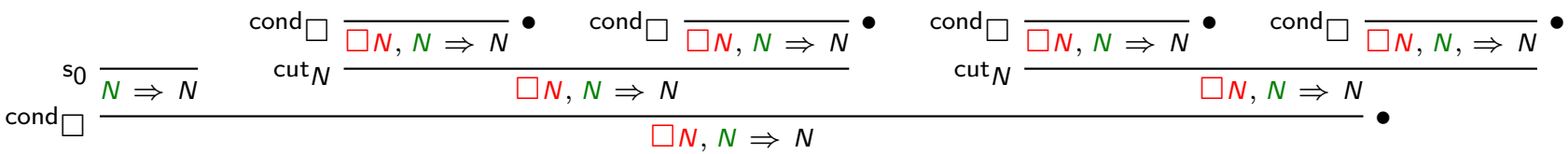
$$\dfrac{\mathcal{D}_0 \quad \Gamma \Rightarrow N \qquad \dfrac{\dfrac{}{\Box N, \Gamma \Rightarrow N}\bullet \qquad \mathcal{D}_1 \quad \Box N, \Gamma, \Box N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}\,\mathrm{cut}_\Box \qquad \dfrac{\dfrac{}{\Box N, \Gamma \Rightarrow N}\bullet \qquad \mathcal{D}_2 \quad \Box N, \Gamma, \Box N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}\,\mathrm{cut}_\Box}{\Box N, \Gamma \Rightarrow N}\,\mathrm{cond}_\Box\ \bullet$$

$$\begin{aligned} f_{\mathcal{D}}(0, \vec{x};) &= f_{\mathcal{D}_0}(\vec{x};) \\ f_{\mathcal{D}}(s_0 x, \vec{x};) &= f_{\mathcal{D}_1}(x, \vec{x}, f(x, \vec{x});) \\ f_{\mathcal{D}}(s_1 x, \vec{x};) &= f_{\mathcal{D}_2}(x, \vec{x}, f(x, \vec{x};);) \end{aligned}$$

- Safe proof = any branch crosses finitely many $\mathrm{cut}_\Box$-steps.
- Cyclic proof system NCB = cyclic progressing safe proofs.

Nesting

- **Problem**. NCB can express **nested recursion**.
- **Example.** A cyclic progressing safe proof for the **exponential** function $\exp(x)(y) = 2^{2^{|x|}} \cdot y$:

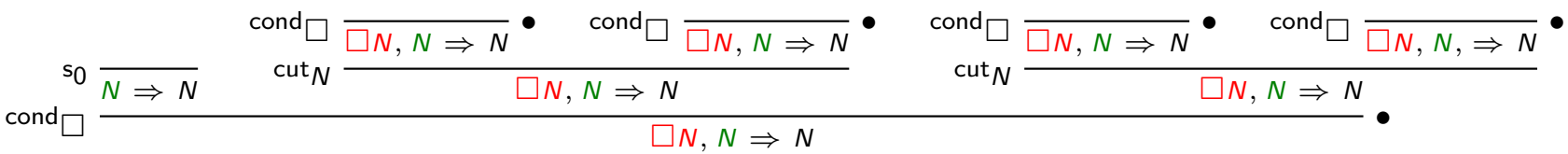


$$\exp(0; y) = \mathsf{s}_0 y$$
$$\exp(\mathsf{s}_0 x; y) = \exp(x; \exp(x; y))$$
$$\exp(\mathsf{s}_1 x; y) = \exp(x; \exp(x; y))$$

- Left-leaning proof = any branch goes right at a cut_N-step only finitely often.
- Cyclic proof system CB = cyclic progressing safe left-leaning proofs.

Nesting

- **Problem**. NCB can express **nested recursion**.
- **Example.** A cyclic progressing safe proof for the **exponential** function $\exp(x)(y) = 2^{2^{|x|}} \cdot y$:



$$\exp(0; y) = \mathsf{s}_0 y$$
$$\exp(\mathsf{s}_0 x; y) = \exp(x; \exp(x; y))$$
$$\exp(\mathsf{s}_1 x; y) = \exp(x; \exp(x; y))$$

- Left-leaning proof = any branch goes right at a cut_N-step only finitely often.
- Cyclic proof system CB = cyclic progressing safe left-leaning proofs.

Hofmann's type system SLR

- Two function spaces: $\Box A \to B$ (*modal*) and $A \multimap B$ (*linear*).

- Safe linear recursion operator (with A $\Box$-*free*):

$$\mathrm{rec}_A : \Box N \to \underbrace{(\Box N \to A \multimap A)}_{h} \to \underbrace{A}_{x} \to \underbrace{A}_{g}$$

  where $f(x) = \mathrm{rec}_A(x, h, g)$ means:

$$\begin{array}{rcl} f(0) & = & g \\ f(\mathsf{s}_0 x) & = & h(x, f(x)) \\ f(\mathsf{s}_1 x) & = & h(x, f(x)) \end{array}$$

- SLR captures exactly **FPTIME**.

Nesting and abstraction complexity

- Nested recursion in SLR if higher-order types are not handled **linearly**:

$$\begin{array}{lcll} A & = & N \to N & \\ g & = & \mathsf{s}_0 & : A \\ h & = & \lambda x : \Box N.\lambda u : N \to N.\lambda y : N.u(uy) & : \Box N \to A \to A \to A \end{array}$$

$$\exp(x; y) = \mathsf{rec}_A(x, h, g)(y)$$

- **Takeaway**. Type n cyclic proofs can represent type n+1 recursion [Das 21].

Nesting and abstraction complexity

- Nested recursion in SLR if higher-order types are not handled **linearly**:

$$\begin{array}{lcll} A & = & N \to N & \\ g & = & \mathsf{s}_0 & : A \\ h & = & \lambda x : \Box N.\lambda u : N \to N.\lambda y : N.u(uy) & : \Box N \to A \to A \to A \end{array}$$

$$\exp(x; y) = \mathsf{rec}_A(x, h, g)(y)$$

- **Takeaway**. Type n cyclic proofs can represent type n+1 recursion [Das 21].

1. Cyclic proofs
2. ICC and safe recursion
3. A cyclic proof system based on safe recursion
4. Safety and nesting
5. Characterizing **FPTIME** and **FELEMENTARY**

Results and perspectives

Characterization results:

- **Theorem**. NCB captures exactly **FELEMENTARY**.
- **Theorem**. CB captures exactly **FPTIME**.

Conclusions and future directions:

- CB = circular version of B
- NCB = generalization of B to nested safe recursion schemes
- Higher-order version of cyclic proof systems based on Hofmann's SLR?
- Cyclic proof systems to characterize other complexity classes, like **FPSPACE**, **ALOGTIME**, **NC**?

Thank you!
Questions?