Non-uniform complexity via non-wellfounded proofs

CSL 2023

Warsaw, 15 February 2023

Gianluca Curzi [1] Anupam Das [1]

[1]University of Birmingham, UK

Introduction

Follow up of a previous work [Curzi&Das 2022]:

- Cyclic proof system CB characterising **FP** (functions computable in polytime)
- CB is a circular version of B, an algebra of functions based on the principles of **implicit complexity** [Bellantoni&Cook 92]
- Alternative approach to implicit complexity: **Cyclic Implicit Complexity**

This talk in a nutshell:

- Cyclic proofs are special non-wellfounded proofs admitting finite presentation
- Finite presentability $\approx$ computational **uniformity**
- Relaxing finite presentabilty $\rightsquigarrow$ relaxing uniformity
- Non-wellfounded proof system nuB for **FP**/**poly** (functions computable in non-uniform polytime)

Introduction

Follow up of a previous work [Curzi&Das 2022]:

- Cyclic proof system CB characterising **FP** (functions computable in polytime)
- CB is a circular version of B, an algebra of functions based on the principles of **implicit complexity** [Bellantoni&Cook 92]
- Alternative approach to implicit complexity: **Cyclic Implicit Complexity**

This talk in a nutshell:

- Cyclic proofs are special non-wellfounded proofs admitting finite presentation
- Finite presentability $\approx$ computational **uniformity**
- Relaxing finite presentabilty $\rightsquigarrow$ relaxing uniformity
- Non-wellfounded proof system nuB for **FP/poly** (functions computable in non-uniform polytime)

Introduction

Follow up of a previous work [Curzi&Das 2022]:

- Cyclic proof system CB characterising **FP** (functions computable in polytime)
- CB is a circular version of B, an algebra of functions based on the principles of **implicit complexity** [Bellantoni&Cook 92]
- Alternative approach to implicit complexity: **Cyclic Implicit Complexity**

This talk in a nutshell:

- Cyclic proofs are special non-wellfounded proofs admitting finite presentation
- Finite presentability $\approx$ computational **uniformity**
- Relaxing finite presentabilty $\rightsquigarrow$ relaxing uniformity
- Non-wellfounded proof system nuB for **FP**/**poly** (functions computable in non-uniform polytime)

Introduction

Follow up of a previous work [Curzi&Das 2022]:

- Cyclic proof system CB characterising **FP** (functions computable in polytime)
- CB is a circular version of B, an algebra of functions based on the principles of **implicit complexity** [Bellantoni&Cook 92]
- Alternative approach to implicit complexity: **Cyclic Implicit Complexity**

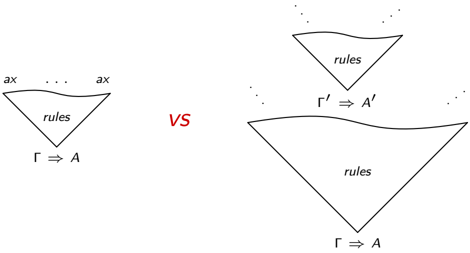
This talk in a nutshell:

- Cyclic proofs are special non-wellfounded proofs admitting finite presentation
- Finite presentability $\approx$ computational **uniformity**
- Relaxing finite presentabilty $\rightsquigarrow$ relaxing uniformity
- Non-wellfounded proof system nuB for **FP/poly** (functions computable in non-uniform polytime)

1. Cyclic Proofs and Computation
2. Cyclic Implicit Complexity Part I: uniform computation
3. Cyclic Implicit Complexity Part II: non-uniform computation

Non-wellfounded proofs

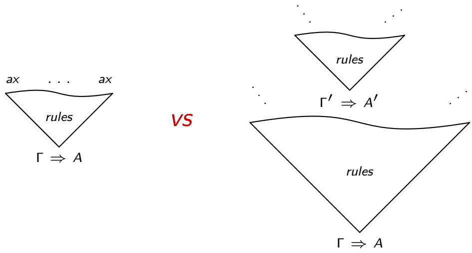
Non-wellfounded proofs = **infinitary** generalisations of the notion of proof



Progressiveness condition = global condition to guarantee **consistency**

Non-wellfounded proofs

Non-wellfounded proofs = **infinitary** generalisations of the notion of proof



Progressiveness condition = global condition to guarantee **consistency**

Cyclic proofs

Cyclic proofs = **regular** non-wellfounded proofs

Regular tree = only **finitely** many distinct subtrees

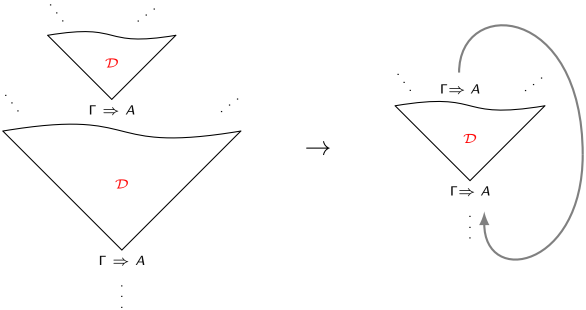
Cyclic proofs admit a finite, "circular" presentation:

Cyclic proofs

Cyclic proofs = **regular** non-wellfounded proofs

Regular tree = only **finitely** many distinct subtrees

Cyclic proofs admit a finite, "circular" presentation:



Cyclic proofs as programs

- Only one formula N corresponding to $\mathbb{N}$
- Inference rules correspond to algorithmic instructions
- The cyclic proof

$$\begin{array}{c}\mathcal{D}\\ \underbrace{N,\dots,N}_{n} \Rightarrow N\end{array}$$

corresponds to a program computing a function

$$f_{\mathcal{D}} : \underbrace{\mathbb{N}\times\ldots\times\mathbb{N}}_{n} \to \mathbb{N}$$

Cyclic proofs as programs

- Only one formula N corresponding to $\mathbb{N}$

- Inference rules correspond to algorithmic instructions

- The cyclic proof

$$\begin{array}{c} \mathcal{D} \\ \underbrace{N, \ldots, N}_{n} \Rightarrow N \end{array}$$

corresponds to a program computing a function

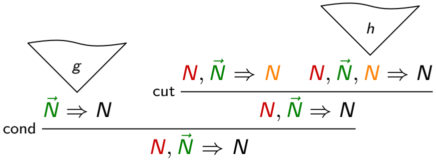
$$f_{\mathcal{D}} : \underbrace{\mathbb{N} \times \ldots \times \mathbb{N}}_{n} \to \mathbb{N}$$

An example

Example: primitive recursion

$$f(0, \vec{y}) = g(\vec{y})$$

$$f(x+1, \vec{y}) = h(x, \vec{y}, f(x, \vec{y}))$$



Computational meaning:

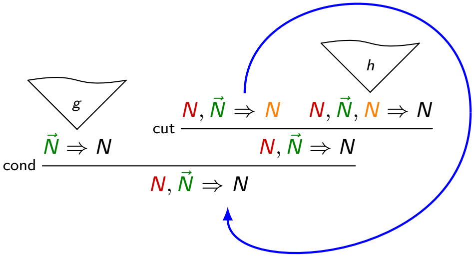
- regularity = **uniformity, computability** criterion
- progressiveness = **totality, termination** criterion

An example

Example: primitive recursion

$$f(0, \vec{y}) = g(\vec{y})$$

$$f(x+1, \vec{y}) = h(x, \vec{y}, f(x, \vec{y}))$$



Computational meaning:

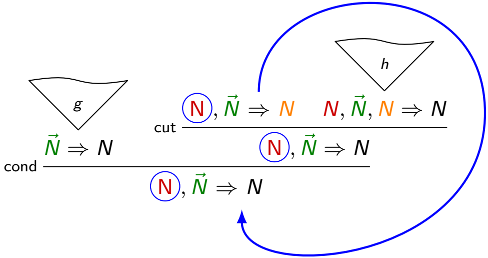
- regularity = **uniformity, computability** criterion
- progressiveness = **totality, termination** criterion

An example

Example: primitive recursion

$$f(0, \vec{y}) \;=\; g(\vec{y})$$

$$f(x+1, \vec{y}) \;=\; h(x, \vec{y}, f(x, \vec{y}))$$



Computational meaning:

- regularity = **uniformity, computability** criterion
- progressiveness = **totality, termination** criterion

ICC and safe recursion

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

- Function algebra B characterising **FP** in the style of ICC [Bellantoni&Cook 92].

- Function arguments partitioned into normal and safe:

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

- Safe recursion:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(x+1, \vec{x}; \vec{y}) = h(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

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

ICC and safe recursion

- Implicit computational complexity (ICC) = characterise complexity classes by means of languages/calculi **without** explicit reference to machine models or external resource bounds.
- Function algebra B characterising **FP** in the style of ICC [Bellantoni&Cook 92].
- Function arguments partitioned into normal and safe:

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

- Safe recursion:

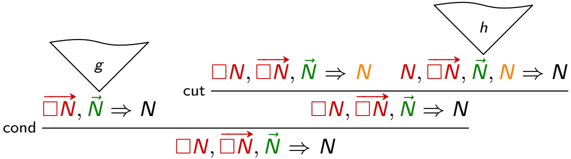
$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(x+1, \vec{x}; \vec{y}) = h(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

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

Cyclic proofs as polytime programs

Example: safe recursion [Bellantoni&Cook, 1992]

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(x+1, \vec{x}; \vec{y}) = h(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$

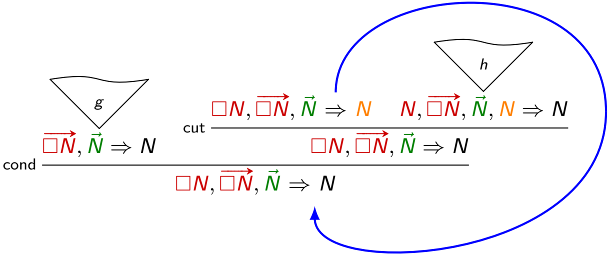


Idea: $\Box N$ vs N reflects normal vs safe distinction of function arguments

Cyclic proofs as polytime programs

Example: safe recursion [Bellantoni&Cook, 1992]

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(x+1, \vec{x}; \vec{y}) = h(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$



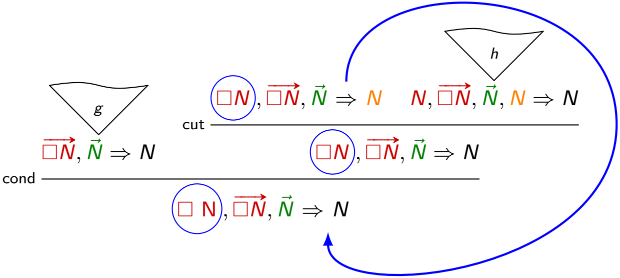
Idea: $\Box N$ vs N reflects normal vs safe distinction of function arguments

Cyclic proofs as polytime programs

Example: safe recursion [Bellantoni&Cook, 1992]

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$

$$f(x+1, \vec{x}; \vec{y}) = h(x, \vec{x}; \vec{y}, f(x, \vec{x}; \vec{y}))$$



Idea: $\Box N$ vs N reflects normal vs safe distinction of function arguments

Characterising the polynomial time (**FP**)

Cyclic proof system CB = regular and progressing non-wellfounded proofs satisfy the following global proof-theoretic conditions:

- Safety:
  - maintain globally the $\square N$ vs N distinction
  - **only** safe recursion schemes are representable
- Left-leaning: prevents **nested** safe recursion:

$$\exp(0; y) = y + 1$$

$$\exp(x + 1; y) = \exp(x; \exp(x; y))$$

  source of **exponential blow up**!

Theorem [Curzi&Das, 2022]:

- the functions representable in CB are exactly those in **FP**.
- the functions representable in CB **without the left-leaning condition** are exactly those in **FELEMENTARY**.

Characterising the polynomial time (**FP**)

Cyclic proof system CB = regular and progressing non-wellfounded proofs satisfy the following global proof-theoretic conditions:

- Safety:
  - maintain globally the $\Box N$ vs N distinction
  - **only** safe recursion schemes are representable
- Left-leaning: prevents **nested** safe recursion:

$$\exp(0; y) = y + 1$$

$$\exp(x + 1; y) = \exp(x; \exp(x; y))$$

  source of **exponential blow up**!

Theorem [Curzi&Das, 2022]:

- the functions representable in CB are exactly those in **FP**.
- the functions representable in CB **without the left-leaning condition** are exactly those in **FELEMENTARY**.

Characterising the polynomial time (**FP**)

Cyclic proof system CB = regular and progressing non-wellfounded proofs satisfy the following global proof-theoretic conditions:

- Safety:
  - maintain globally the $\Box N$ vs N distinction
  - **only** safe recursion schemes are representable
- Left-leaning: prevents **nested** safe recursion:

$$\mathrm{exp}(0; y) = y + 1$$

$$\mathrm{exp}(x + 1; y) = \mathrm{exp}(x; \mathrm{exp}(x; y))$$

source of **exponential blow up**!

Theorem [Curzi&Das, 2022]:

- the functions representable in CB are exactly those in **FP**.
- the functions representable in CB **without the left-leaning condition** are exactly those in **FELEMENTARY**.

Non-uniform polynomial time (**FP**/**poly**)

FP/**poly** = class of functions computable in non-uniform polynomial time

Theorem: $f \in$ **FP**/**poly** iff there are polynomial size circuits computing f.

FP($\mathbb{R}$) = class of functions computable in polynomial time by a Turing machine "querying bits of real numbers"

Theorem [Folklore]: **FP**/**poly** = **FP**($\mathbb{R}$).

Non-uniform polynomial time ($\mathbf{FP}/\mathbf{poly}$)

$\mathbf{FP}/\mathbf{poly}$ = class of functions computable in non-uniform polynomial time

Theorem: $f \in \mathbf{FP}/\mathbf{poly}$ iff there are polynomial size circuits computing f.

$\mathbf{FP}(\mathbb{R})$ = class of functions computable in polynomial time by a Turing machine "querying bits of real numbers"

Theorem [Folklore]: $\mathbf{FP}/\mathbf{poly} = \mathbf{FP}(\mathbb{R})$.

Non-wellfounded proofs as non-uniform polytime programs

Cyclic proofs = **regular** non-wellfounded proofs

regularity $\approx$ *computability*, *uniformity*

Idea: relaxing regularity to represent real numbers and characterise $\mathbf{FP}(\mathbb{R})$

weak *regularity* $\approx$ *computability* + **query on bits of real numbers**

... but since $\mathbf{FP}(\mathbb{R}) = \mathbf{FP}/\mathbf{poly}$ then:

weak *regularity* $\approx$ *non-uniformity*

Non-wellfounded proofs as non-uniform polytime programs

Cyclic proofs $=$ **regular** non-wellfounded proofs

$$\boxed{\textit{regularity} \quad \approx \quad \textit{computability}, \ \textit{uniformity}}$$

Idea: relaxing regularity to represent real numbers and characterise $\mathbf{FP}(\mathbb{R})$

$$\boxed{\textbf{weak } \textit{regularity} \quad \approx \quad \textit{computability} + \textbf{query on bits of real numbers}}$$

... but since $\mathbf{FP}(\mathbb{R}) = \mathbf{FP}/\mathbf{poly}$ then:

$$\boxed{\textbf{weak } \textit{regularity} \quad \approx \quad \textit{non-uniformity}}$$

Non-wellfounded proofs as non-uniform polytime programs

Cyclic proofs = **regular** non-wellfounded proofs

regularity $\approx$ *computability*, *uniformity*

Idea: relaxing regularity to represent real numbers and characterise $\mathbf{FP}(\mathbb{R})$

weak *regularity* $\approx$ *computability* + **query on bits of real numbers**

... but since $\mathbf{FP}(\mathbb{R}) = \mathbf{FP}/\mathbf{poly}$ then:

weak *regularity* $\approx$ *non-uniformity*

Weak regularity

Regular proof = finitely many distinct subproofs.

Example:

Weak regularity

Weakly regular proof = finitely many distinct subproofs **containing the inference rules . . . .**

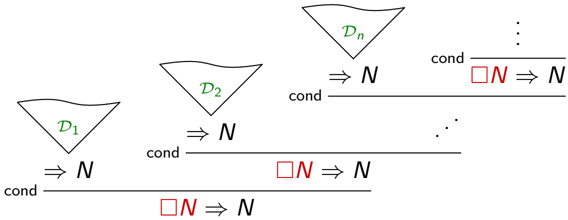
Example:

Weak regularity

Weakly regular proof = finitely many distinct subproofs **containing the inference rules . . . .**

Example:

$\mathcal{D} \quad :=$

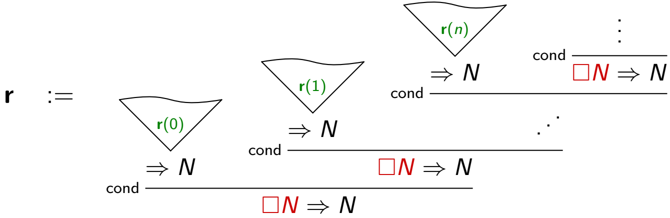


Idea: weak regularity implies $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n, \ldots$ are finitely many distinct.

Weak regularity

Weakly regular proof = finitely many distinct subproofs **containing the inference rules . . . .**

Example:



. . . which encodes a real number $\mathbf{r} = \langle \mathbf{r}(0), \mathbf{r}(1), \ldots, \mathbf{r}(n), \ldots \rangle$

Characterising **FP**/**poly**

Non-wellfounded proof system nuB = weakly regular version of CB.

Theorem [Curzi&Das 2023]: The functions representable in nuB are exactly those in **FP**/**poly**.

Idea of the proof: weak regularity allows a decomposition result

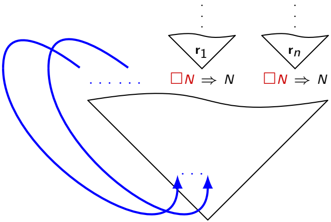
$\mathsf{nuB} = \mathsf{CB}(\mathbb{R}) = \mathbf{FP}(\mathbb{R}) = \mathbf{FP}/\mathbf{poly}$

Characterising $\mathbf{FP}/\mathbf{poly}$

Non-wellfounded proof system nuB = weakly regular version of CB.

Theorem [Curzi&Das 2023]: The functions representable in nuB are exactly those in $\mathbf{FP}/\mathbf{poly}$.

Idea of the proof: weak regularity allows a decomposition result



$$\text{nuB} = \text{CB}(\mathbb{R}) = \mathbf{FP}(\mathbb{R}) = \mathbf{FP}/\mathbf{poly}$$

Conclusion and future directions

- **Ongoing work:** non-wellfounded approaches to $\mathbf{FP}/\mathbf{poly}$ in the setting of linear logic.

- **Future work:** find proof-theoretic restrictions on nuB to characterise BPP (bounded-error probabilistic polynomial time).

Conclusion and future directions

- **Ongoing work:** non-wellfounded approaches to $\mathbf{FP}/\mathbf{poly}$ in the setting of linear logic.

- **Future work:** find proof-theoretic restrictions on nuB to characterise BPP (bounded-error probabilistic polynomial time).

Thank you!
Questions?

Appendix

Non-uniform complexity classes

- **FP** $=$ class of functions computable in polynomial time on a Turing machine.

- **FP**/**poly** is an extension of **FP** that intuitively has access to a 'small' amount of *advice*, determined only by the length of the input.

- **FP**/**poly** $=$ class of functions $f(\vec{x})$ for which there exists some strings $\alpha_{\vec{n}} \in \{0,1\}^*$ and a function $f'(x, \vec{x}) \in$ **FP** with:
  - $|\alpha_{\vec{n}}|$ is polynomial in $\vec{n}$.
  - $f(\vec{x}) = f'(\alpha_{|\vec{x}|}, \vec{x})$.

- Note, in particular, that **FP**/**poly** admits undecidable problems. E.g. the function $f(x) = 1$ just if $|x|$ is the code of a halting Turing machine (and 0 otherwise) is in **FP**/**poly**. Indeed, the point of the class **FP**/**poly** is to rather characterise a more non-uniform notion of computation.

- **Theorem**: $f(\vec{x}) \in$ **FP**/**poly** iff there are poly-size circuits computing $f(\vec{x})$.

- The class $\mathbf{FP}(\mathbb{R})$ consists of just the functions computable in polynomial time by a Turing machine with access to oracles from:

$$\mathbb{R} := \{f(x) : \mathbb{N} \to \{0, 1\} \mid |x| = |y| \implies f(x) = f(y)\}$$

- Note that the notation $\mathbb{R}$ is suggestive here, since its elements are essentially maps from lengths/positions to Booleans, and so may be identified with Boolean streams.

- **Theorem** [Folklore]: $\mathbf{FP}/\mathbf{poly} = \mathbf{FP}(\mathbb{R})$.

Rules for the non-wellfounded proof system nuB

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

$$\text{w}_N\ \frac{\Gamma \Rightarrow B}{\Gamma, N \Rightarrow B} \qquad \text{w}_\Box\ \frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \text{e}\ \frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C} \qquad \Box_l\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \Box_r\ \frac{\Box\Gamma \Rightarrow N}{\Box\Gamma \Rightarrow \Box N}$$

$$0\ \frac{}{\Rightarrow N} \qquad 1\ \frac{}{\Rightarrow N} \qquad \text{s}_0\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \text{s}_1\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \text{srec}\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

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

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

Semantics of non-wellfounded proofs for nuB

$$i \; \frac{i \in \{0,1\}}{\Rightarrow N} \qquad\qquad f_{\mathcal{D}}(;) := i$$

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

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

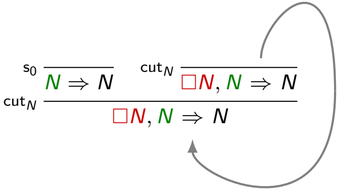
$$\mathsf{cut}_{\Box} \; \frac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow \Box N} \quad \overset{\mathcal{D}_1}{\Box N, \Gamma \Rightarrow A}}{\Gamma \Rightarrow A} \qquad\qquad f_{\mathcal{D}}(\vec{x}; \vec{y}) := f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x}; \vec{y}), \vec{x}; \vec{y})$$

$$\mathsf{cond}_{\Box} \; \frac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_1}{\Box N, \Gamma, \Rightarrow N} \quad \overset{\mathcal{D}_2}{\Box N, \Gamma \Rightarrow N}}{\Box N, \Gamma \Rightarrow N} \qquad\qquad \begin{array}{rcl} f_{\mathcal{D}}(0, \vec{x}; \vec{y}) & := & f_{\mathcal{D}_0}(\vec{x}; \vec{y}) \\ f_{\mathcal{D}}(\mathsf{s}_0 x, \vec{x}; \vec{y}) & := & f_{\mathcal{D}_1}(x, \vec{x}; \vec{y}) \\ f_{\mathcal{D}}(\mathsf{s}_1 x, \vec{x}; \vec{y}) & := & f_{\mathcal{D}_2}(x, \vec{x}; \vec{y}) \end{array}$$

$$|\mathsf{cond}|_{\Box} \; \frac{\overset{\mathcal{D}_0}{\Gamma \Rightarrow N} \quad \overset{\mathcal{D}_2}{\Box N, \Gamma \Rightarrow N}}{\Box N, \Gamma \Rightarrow N} \qquad\qquad \begin{array}{rcl} f_{\mathcal{D}}(0, \vec{x}; \vec{y}) & := & f_{\mathcal{D}_0}(\vec{x}; \vec{y}) \\ f_{\mathcal{D}}(\mathsf{s}_i x, \vec{x}; \vec{y}) & := & f_{\mathcal{D}_2}(x, \vec{x}; \vec{y}) \end{array}$$

Progressiveness

- **Example.** A cyclic proof $\mathcal{D}$ representing a partial function:

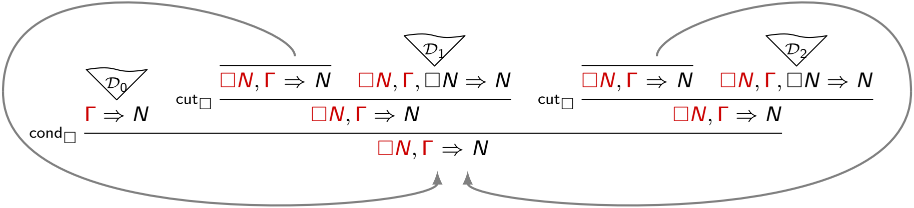


$$f_{\mathcal{D}}(x; y) \quad := \quad f_{\mathcal{D}}(x; \mathsf{s}_0 y)$$

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

Safety condition

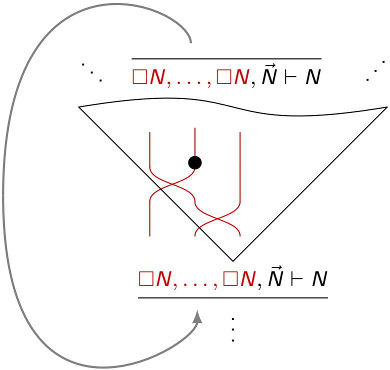
- **Example.** Modalities are not enough to enforce stratification in our setting. E.g. cyclic progressing proof $\mathcal{D}$ for primitive recursion (on notation):



$$f_{\mathcal{D}}(0, \vec{x};\,) = f_{\mathcal{D}_0}(\vec{x};\,)$$
$$f_{\mathcal{D}}(s_i x, \vec{x};\,) = f_{\mathcal{D}_1}(x, \vec{x}, f(x, \vec{x});\,)$$

- Safe proof = any infinite branch crosses finitely many $\mathrm{cut}_\Box$ rules.
- Safety condition rules out non-safe recursion schemes.

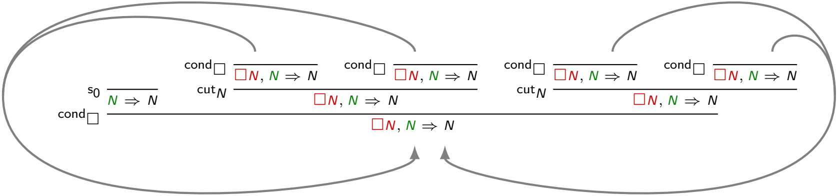
Safety condition induces a simpler □-thread structure



$$\mathsf{w}_{\Box}\ \frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \Box_l\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \mathsf{cut}_{\Box}\ \frac{\Gamma \Rightarrow \Box N \qquad \Box N, \Gamma \Rightarrow B}{\Gamma \Rightarrow B} \qquad \mathsf{cond}_{\Box}\ \frac{\Gamma \Rightarrow N \qquad \Box N, \Gamma \Rightarrow N \qquad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

Left-leaning condition

- Safety condition is not enough! We can express **nested safe recursion**.
- **Example.** A cyclic progressing safe proof for the **exponential** function $\exp(x)(y) = 2^{2^{|x|}} \cdot y$:



$$\exp(0; y) = s_0 y$$
$$\exp(s_i x; y) = \exp(x; \exp(x; y))$$

- Left-leaning proof = any branch goes right at a cut_N rule only finitely often.

Hofmann's type system SLR [Hofmann 97]

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

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

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

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

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

- terms $t : (\Box N)^n \to N^m \multimap N$ represent *exactly* the functions in **FP**.

Nesting and higher-order recursion

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

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

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

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

- **Left-leaning is a linearity condition**: it prevents duplication of recursive calls, and hence their nesting.