Cyclic Implicit Complexity

LICS 2022, Haifa, Israel 2022

Gianluca Curzi

University of Birmingham

joint work with Anupam Das (University of Birmingham)

What is this presentation about?

- Goal:
  - circular proof systems characterising **FPTIME** and **FELEMENTARY**...
  - ...in the style of *implicit computational complexity* (ICC)

- Some motivations:
  - **new topic**, not much about complexity-theoretic aspects of circular reasoning;
  - circular proofs **subsume** several recursion schemes;
  - **hard to tame complexity**: introduce cyclic proof-theoretic conditions that identify computational and complexity-theoretic notions (uniformity, totality, stratification, ...)

.

ICC and safe recursion

- Function algebra B characterising **FPTIME** [Bellantoni and Cook 92].
- 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:

Circular proof systems based on safe recursion

Proof theoretic formulation of B

- **Formulas** $A, B, C \in \{N, \Box N\}$ and **contexts** $\Gamma, \Delta = A_1, \ldots, A_n$
- **Sequents** $\Box\vec{N}, \vec{N} \Rightarrow N$ represent functions $f(\vec{x}; \vec{y})$
- **Inference rules** for initial functions and closure operations of B:

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

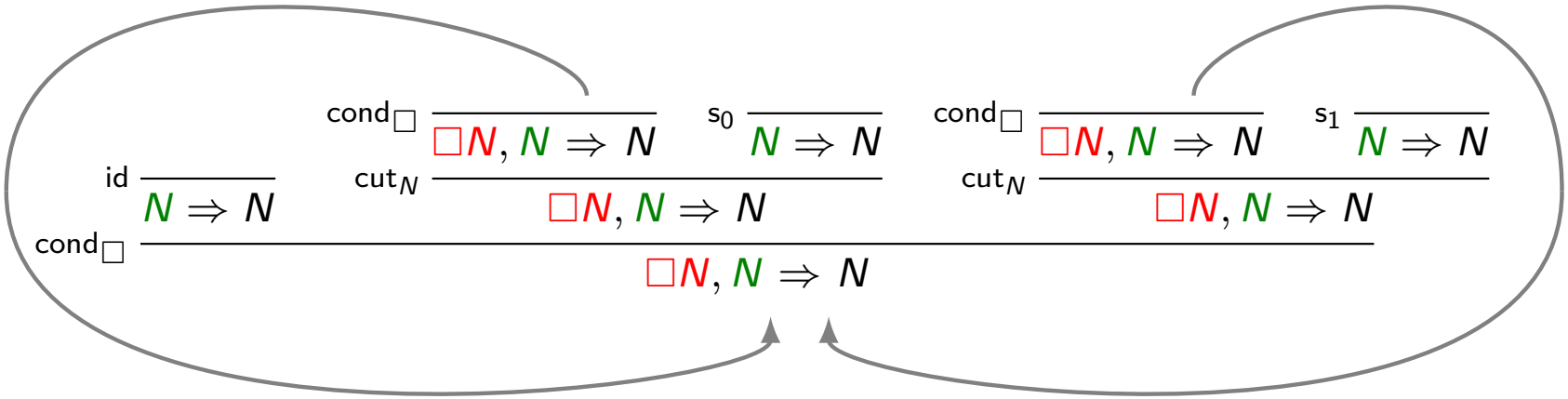
$$\Box_l\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \Box_r\ \frac{\Box N, \ldots, \Box N \Rightarrow N}{\Box N, \ldots, \Box N \Rightarrow \Box N} \qquad 0\ \frac{}{\Rightarrow N} \qquad \mathrm{s}_0\ \frac{}{N \Rightarrow N} \qquad \mathrm{s}_1\ \frac{}{N \Rightarrow N}$$

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

$$\boxed{\mathrm{srec}\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}}$$

Circular proofs

- Circular proofs = non-wellfounded proofs with **finitely** many distinct subproofs
- Circular proofs admit a finite, "cyclic" presentation

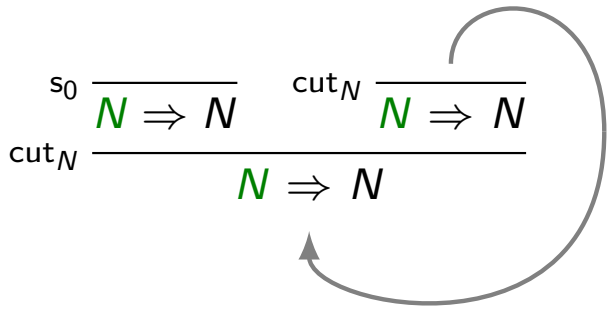


$$f_{\mathcal{D}}(0; y) = y$$
$$f_{\mathcal{D}}(s_0 x; y) = s_0(f_{\mathcal{D}}(x; y))$$
$$f_{\mathcal{D}}(s_1 x; y) = s_1(f_{\mathcal{D}}(x; y))$$

circularity $\rightsquigarrow$ computability

Progressiveness and safety

- Progressiveness condition = global criterion for **termination**



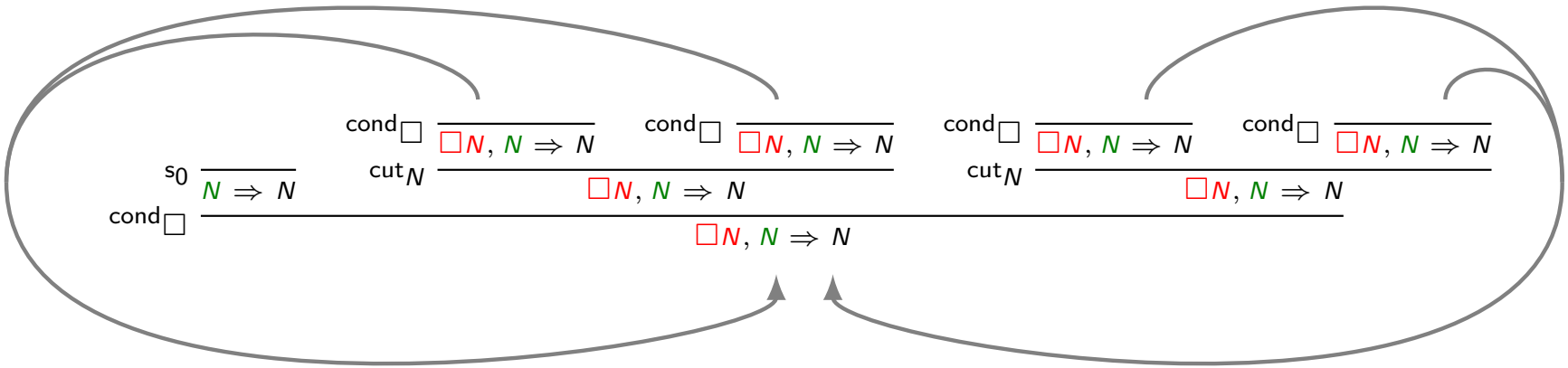
$$f_{\mathcal{D}}(;y) := f_{\mathcal{D}}(;s_0 y)$$

- Safety condition = global criterion enforcing stratification of data ($\Box N$ vs N)

progressiveness	$\rightsquigarrow$	extract recursion from cycles
progressiveness + safety	$\rightsquigarrow$	extract safe recursion from cycles

Nesting

- Circular proof system NCB = circular proofs that are both progressing and safe
- NCB defines safe nested recursion (hence exponential functions):



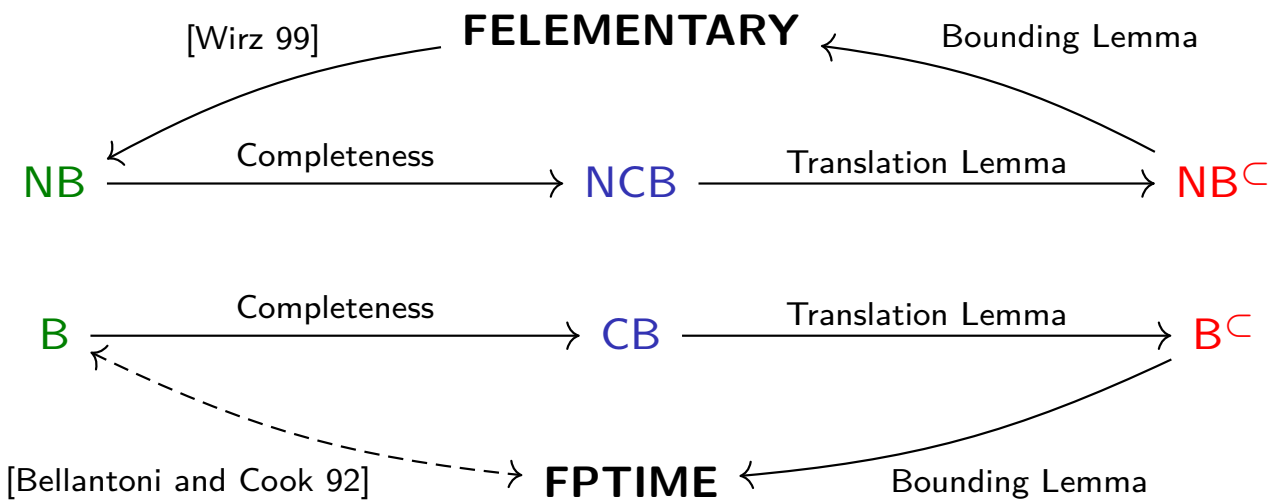
$$\exp(0; y) = s_0 y$$
$$\exp(s_i x; y) = \exp(x; \exp(x; y))$$

- Circular proof system CB = global "linearity" criterion on NCB

CB $\rightsquigarrow$ no duplication of recursive calls (hence, no nesting!)

Outline of the fundamental results

$$\begin{array}{rcl} \mathrm{NCB} & = & \textbf{FELEMENTARY} \\ \mathrm{CB} & = & \textbf{FPTIME} \end{array}$$



safe recursion	*on notation*	*on* $\subset$
unnested	B	$\mathrm{B}^{\subset}$
nested	NB	$\mathrm{NB}^{\subset}$

Thank you!
Questions?

Appendix

Semantics of non-wellfounded proofs for B

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

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

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

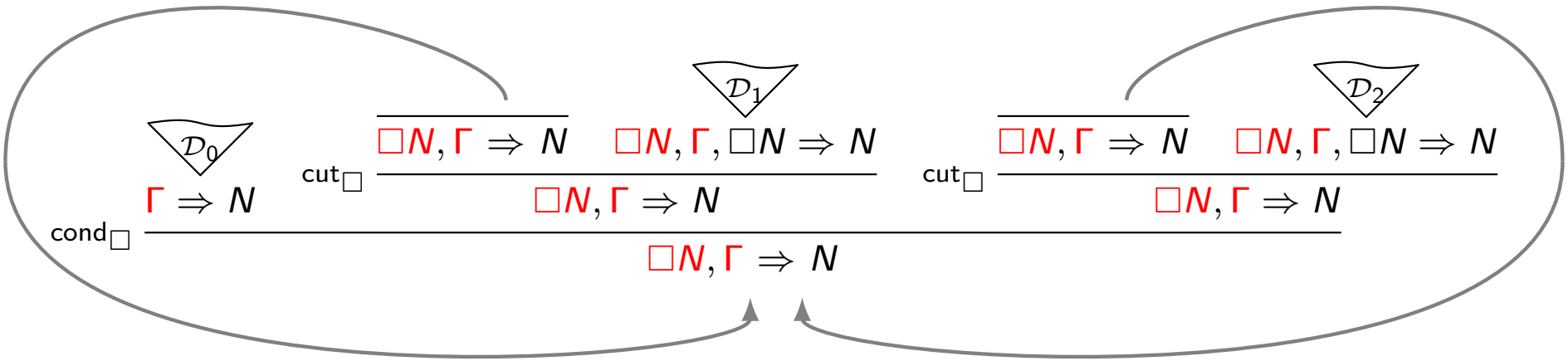
$$\text{cut}_{\Box}\ \frac{\mathcal{D}_0 \quad \mathcal{D}_1}{\Gamma \Rightarrow \Box N \quad \Box N, \Gamma \Rightarrow A \;/\; \Gamma \Rightarrow A} \qquad f_{\mathcal{D}}(\vec{x};\vec{y}) := f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x};\vec{y}), \vec{x};\vec{y})$$

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

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

Safety condition

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

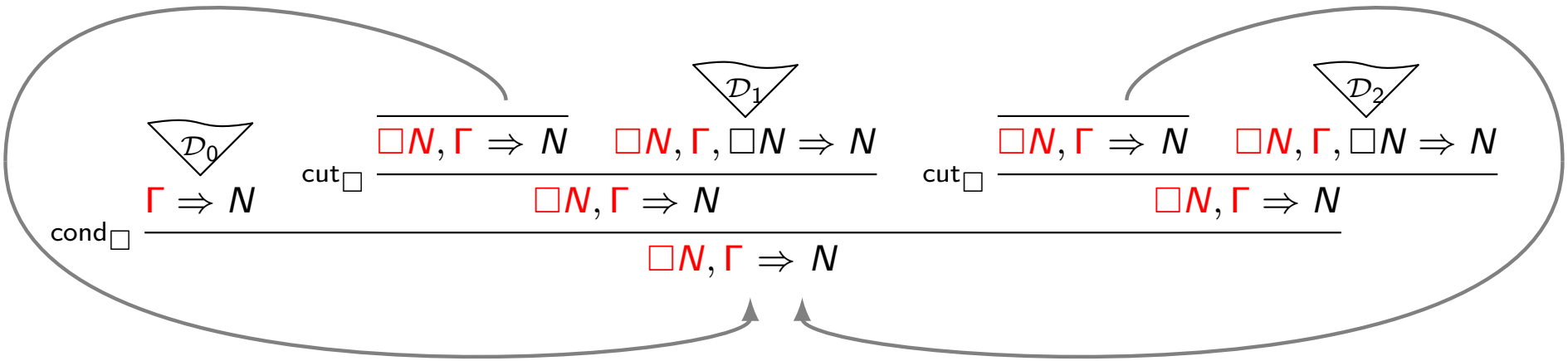


$$f_{\mathcal{D}}(0, \vec{x};) = f_{\mathcal{D}_0}(\vec{x};)$$
$$f_{\mathcal{D}}(s_i x, \vec{x};) = f_{\mathcal{D}_1}(x, \vec{x}, f(x, \vec{x});)$$

- Safe proof = any infinite branch crosses finitely many cut$_\Box$ rules.
- Circular proof system NCB = circular progressing safe proofs.
- Safety condition rules out non-safe recursion schemes.

Safety condition

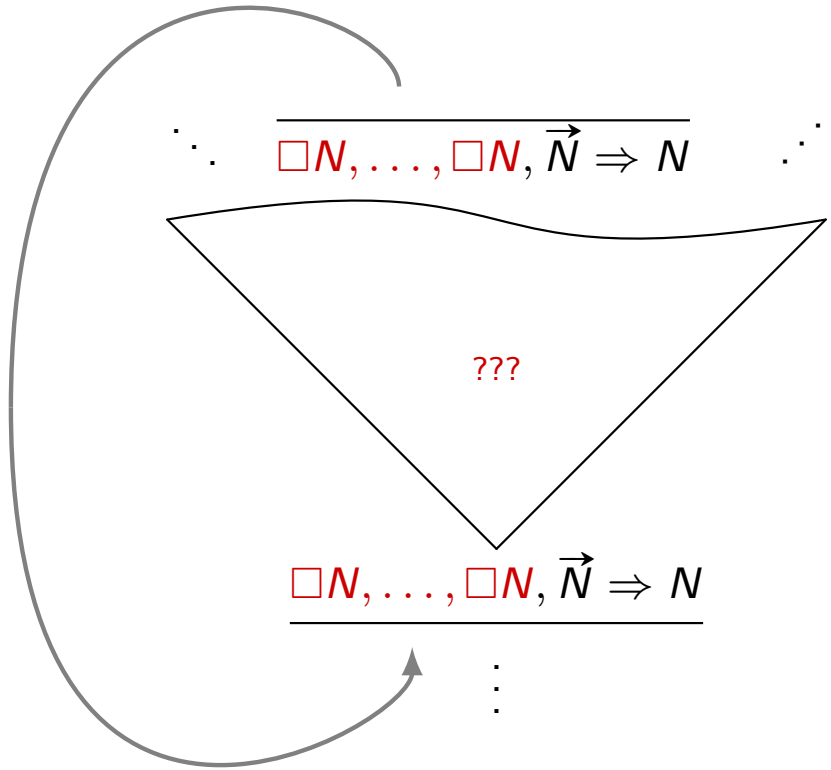
- **Example.** Modalities are not enough to enforce stratification in our setting. E.g. circular progressing proof $\mathcal{D}$ for primitive recursion (on notation):



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

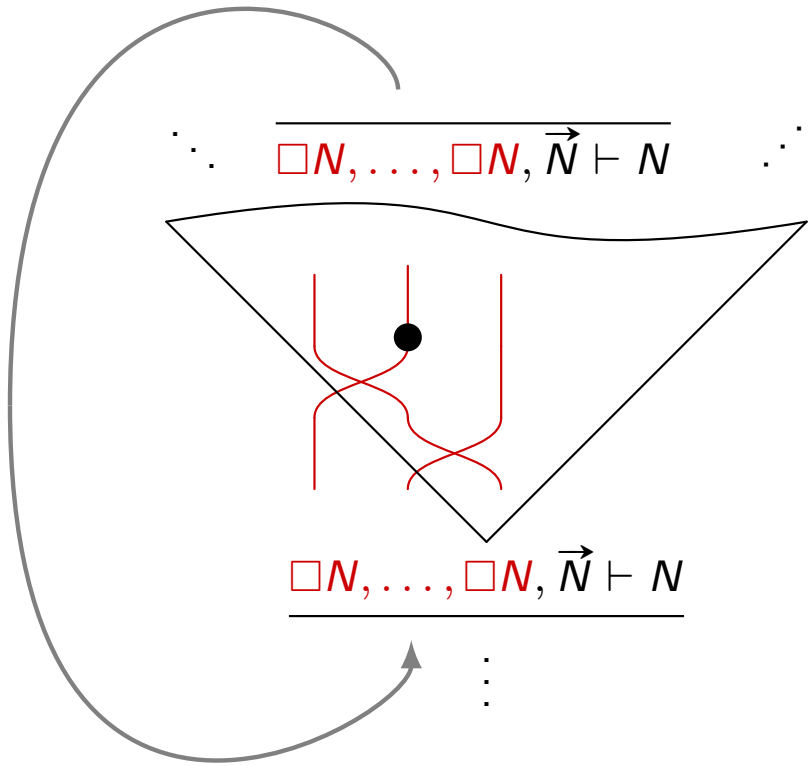
- Safe proof = any infinite branch crosses finitely many cut$_\Box$ rules.
- Circular proof system NCB = circular progressing safe proofs.
- Safety condition rules out non-safe recursion schemes.

Safety = simpler $\Box$-thread structure



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

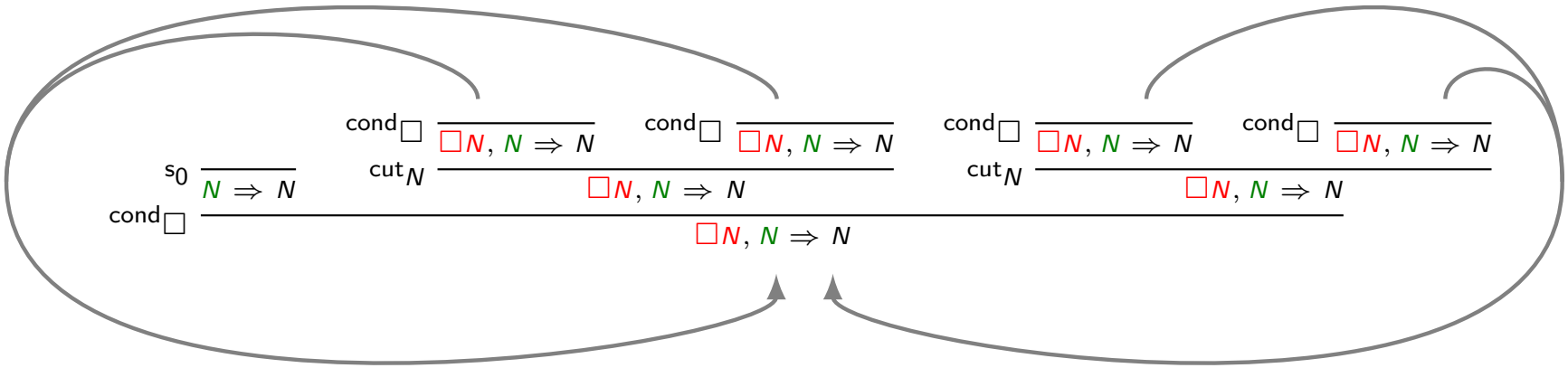
Safety = simpler $\Box$-thread structure



$$\mathrm{w}\Box\ \frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \Box_I\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \mathrm{cut}\Box\ \frac{\Gamma \Rightarrow \Box N \qquad \Box N, \Gamma \Rightarrow B}{\Gamma \Rightarrow B} \qquad \mathrm{cond}\Box\ \frac{\Gamma \Rightarrow N \qquad \Box N, \Gamma \Rightarrow N \qquad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

Nesting condition

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

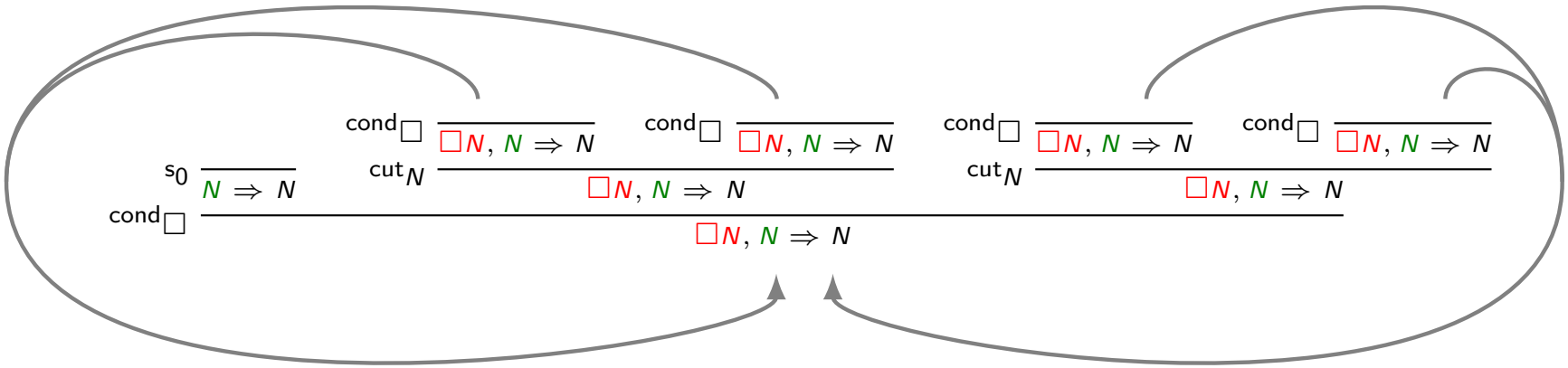


$$\exp(0; y) = s_0 y$$
$$\exp(s_i x; y) = \exp(x; \exp(x; y))$$

- Left-leaning proof = any branch goes right at a cut_N rule only finitely often.
- Circular proof system CB = circular progressing safe left-leaning proofs.

Nesting condition

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



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

- Left-leaning proof = any branch goes right at a cut_N rule only finitely often.
- Circular proof system CB = circular progressing safe left-leaning proofs.

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*):

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

x g

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

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

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

Nesting and higher-order recursion

- 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 circular 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.

Nesting and higher-order recursion

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

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

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

- **Takeaway**. Type n circular 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.

Nesting and higher-order recursion

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

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

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

- **Takeaway**. Type n circular 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.

Nested safe recursion

- Function algebra NB, which generalises B to **nested safe recursion**. E.g.

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

- Defining nested safe recursion requires introduction of oracles:

$$f(\vec{a})(\vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$$

with $\vec{a} = a_1, \ldots, a_n$ oracles.

- Nested safe recursion: from $g(\vec{x}; \vec{y}) \in \mathrm{NB}(\varnothing)$ and $h(a)(x, \vec{x}; \vec{y}) \in \mathrm{NB}(a, \vec{a})$, define $f(x, \vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$ by:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_i x, \vec{x}; \vec{y}) = h_i(\underbrace{\lambda \vec{v}. f(x, \vec{x}; \vec{v})}_{a})(x, \vec{x}; \vec{y})$$

Nested safe recursion

- Function algebra NB, which generalises B to **nested safe recursion**. E.g.

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

- Defining nested safe recursion requires introduction of oracles:

$$f(\vec{a})(\vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$$

with $\vec{a} = a_1, \ldots, a_n$ oracles.

- Nested safe recursion: from $g(\vec{x}; \vec{y}) \in \mathrm{NB}(\varnothing)$ and $h(a)(x, \vec{x}; \vec{y}) \in \mathrm{NB}(a, \vec{a})$, define $f(x, \vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$ by:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_i x, \vec{x}; \vec{y}) = h_i(\underbrace{\lambda \vec{v}. f(x, \vec{x}; \vec{v})}_{a})(x, \vec{x}; \vec{y})$$

Nested safe recursion

- Function algebra NB, which generalises B to **nested safe recursion**. E.g.

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

- Defining nested safe recursion requires introduction of oracles:

$$f(\vec{a})(\vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$$

  with $\vec{a} = a_1, \ldots, a_n$ oracles.

- Nested safe recursion: from $g(\vec{x}; \vec{y}) \in \mathrm{NB}(\varnothing)$ and $h(a)(x, \vec{x}; \vec{y}) \in \mathrm{NB}(a, \vec{a})$, define $f(x, \vec{x}; \vec{y}) \in \mathrm{NB}(\vec{a})$ by:

$$f(0, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$$
$$f(s_i x, \vec{x}; \vec{y}) = h_i(\underbrace{\lambda \vec{v}. f(x, \vec{x}; \vec{v})}_{a})(x, \vec{x}; \vec{y})$$

Permutation of prefixes

- Prefix order: $x \subseteq y$ (resp. $x \subset y$) iff the binary notation of x is a prefix (resp. strict prefix) of y
- **Example**. $s_0(s_1(x)) \subset s_0(s_1(s_0(s_1 x)))$
- Permutation of prefixes: if $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (x_1, \ldots, y_n)$:

  $\vec{x} \subseteq \vec{y}$ if, for some permutation π, each x_i is prefix of $y_{\pi(i)}$

  $\vec{x} \subset \vec{y}$ iff $\vec{x} \subseteq \vec{y}$ and some x_i is a proper prefix of $y_{\pi(i)}$.
- Guarded abstraction:

$$(\lambda \vec{v} \subset \vec{x}.f(\vec{v}; \vec{y}))(\vec{u}) := \begin{cases} f(\vec{u}; \vec{y}) & \vec{u} \subset \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Permutation of prefixes

- Prefix order: $x \subseteq y$ (resp. $x \subset y$) iff the binary notation of x is a prefix (resp. strict prefix) of y
- **Example**. $s_0(s_1(x)) \subset s_0(s_1(s_0(s_1 x)))$
- Permutation of prefixes: if $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (x_1, \ldots, y_n)$:

  $\vec{x} \subseteq \vec{y}$ if, for some permutation π, each x_i is prefix of $y_{\pi(i)}$

  $\vec{x} \subset \vec{y}$ iff $\vec{x} \subseteq \vec{y}$ and some x_i is a proper prefix of $y_{\pi(i)}$.
- Guarded abstraction:

$$(\lambda \vec{v} \subset \vec{x}.f(\vec{v}; \vec{y}))(\vec{u}) := \begin{cases} f(\vec{u}; \vec{y}) & \vec{u} \subset \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Permutation of prefixes

- Prefix order: $x \subseteq y$ (resp. $x \subset y$) iff the binary notation of x is a prefix (resp. strict prefix) of y
- **Example**. $s_0(s_1(x)) \subset s_0(s_1(s_0(s_1 x)))$
- Permutation of prefixes: if $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (x_1, \ldots, y_n)$:

  $\vec{x} \subseteq \vec{y}$ if, for some permutation π, each x_i is prefix of $y_{\pi(i)}$

  $\vec{x} \subset \vec{y}$ iff $\vec{x} \subseteq \vec{y}$ and some x_i is a proper prefix of $y_{\pi(i)}$.
- Guarded abstraction:

$$(\lambda \vec{v} \subset \vec{x}.f(\vec{v}; \vec{y}))(\vec{u}) := \begin{cases} f(\vec{u}; \vec{y}) & \vec{u} \subset \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Recursion on permutation of prefixes

- Generalise nested safe recursion to permutation of prefixes:

$$f(\vec{a})(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$$

with $\vec{a} = a_1, \ldots, a_n$ oracles

- Nested safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

- (Unnested) safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

Recursion on permutation of prefixes

- Generalise nested safe recursion to permutation of prefixes:

$$f(\vec{a})(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$$

with $\vec{a} = a_1, \ldots, a_n$ oracles

- Nested safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

- (Unnested) safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

Recursion on permutation of prefixes

- Generalise nested safe recursion to permutation of prefixes:

$$f(\vec{a})(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$$

with $\vec{a} = a_1, \ldots, a_n$ oracles

- Nested safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{NB}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

- (Unnested) safe recursion on permutation of prefixes: from $h(a)(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(a,\vec{a})$ define $f(\vec{x};\vec{y}) \in \mathrm{B}^{\subset}(\vec{a})$ by:

$$f(\vec{x};\vec{y}) = h(\underbrace{\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v})}_{a})(\vec{x};\vec{y})$$

Further results and future work

- **Further results**: relaxing circularity in CB (= **FPTIME**) $\rightsquigarrow$ **FP**/**poly**.
- **Future work**:
  - Higher-order version of circular proof systems based on Hofmann's SLR?
  - Circular proof systems to characterise other complexity classes, like **FPSPACE**, **ALOGTIME**, **NC**? Hint: "parallel cut".

$$\text{pcut}\ \frac{\Gamma \Rightarrow N \quad \cdots \quad \Gamma \Rightarrow N \quad \Gamma, N, \ldots, N \Rightarrow N}{\Gamma \Rightarrow N}$$