Carnegie Mellon University

Session-Typed Recursive Processes and Circular Proofs

Farzaneh Derakhshan

PhD Prospectus Presentation

Programs as Proofs [1]

Functional programming ⇠ - - - - - ⇢ Intuitionistic Natural deduction[2]

$r.\mathtt{M}\colon A \vee B$

$$\dfrac{\begin{array}{c}\mathcal{M}\\ \Gamma \vdash B\end{array}}{\Gamma \vdash A \vee B}\ \vee I_r$$

$\mathsf{case}\ \mathtt{N}(l.x \Rightarrow \mathtt{N_a} \mid r.x \Rightarrow \mathtt{N_b})\colon C$

$$\dfrac{\begin{array}{c}\mathcal{N}\\ \Gamma \vdash A \vee B\end{array} \quad \begin{array}{c}\mathcal{N}_a\\ \Gamma, A \vdash C\end{array} \quad \begin{array}{c}\mathcal{N}_b\\ \Gamma, B \vdash C\end{array}}{\Gamma \vdash C}\ \vee E$$

[1] W. A. Howard, 1969 [2] Hilbert and Bernays, 1922 - Gentzen, 1932 - Prawitz, 1965

Programs as Proofs [1]

Functional programming ⟷ Intuitionistic Natural deduction[2]

input ⟶ output

$$r.\mathtt{M}\colon A \vee B$$

$$\frac{\begin{array}{c}\mathcal{M}\\ \Gamma \vdash B\end{array}}{\Gamma \vdash A \vee B}\ \vee I_r$$

$$\text{case}\ \mathtt{N}(l.x \Rightarrow \mathtt{N_a} \mid r.x \Rightarrow \mathtt{N_b})\colon C$$

$$\frac{\begin{array}{c}\mathcal{N}\\ \Gamma \vdash A \vee B\end{array} \quad \begin{array}{c}\mathcal{N}_a\\ \Gamma, A \vdash C\end{array} \quad \begin{array}{c}\mathcal{N}_b\\ \Gamma, B \vdash C\end{array}}{\Gamma \vdash C}\ \vee E$$

[1] W. A. Howard, 1969 [2] Hilbert and Bernays, 1922 - Gentzen, 1932 - Prawitz, 1965

Programs as Proofs[1]

Session-typed Processes ⟷ Sequent Calculus[2] (intuitionistic linear logic)

$$x_1{:}A_1, x_2{:}A_2, \cdots x_n{:}A_n \vdash \mathtt{P} :: (y{:}B) \qquad \overset{\mathcal{P}}{A_1 \cdots A_n \vdash B}$$

$$\Gamma \vdash y.\mathtt{r}; \mathtt{P} :: (y{:}A \oplus B) \qquad \dfrac{\overset{\mathcal{P}}{\Gamma \vdash B}}{\Gamma \vdash A \oplus B} \oplus R_r$$

$$\Gamma, y{:}A \oplus B \vdash \mathsf{case}\, y(l \Rightarrow \mathtt{P}_1 \mid r \Rightarrow \mathtt{P}_2) :: (z{:}C) \qquad \dfrac{\overset{\mathcal{P}_1}{\Gamma, A \vdash C} \quad \overset{\mathcal{P}_2}{\Gamma, B \vdash C}}{\Gamma, A \oplus B \vdash C} \oplus L$$

[1] Caires and Pfenning 2010 [2] Gentzen, 1932

Programs as Proofs

Session-typed Processes ⇠ - - - ⇢ Sequent Calculus

Sends message r along y (⟶) and continues as P (↓).

$$\Gamma \vdash y.\mathtt{r}; \mathtt{P} :: (y{:}A \oplus B) \longrightarrow \Gamma, y{:}A \oplus B \vdash \mathsf{case}\, y(l \Rightarrow \mathtt{P}_1 \mid r \Rightarrow \mathtt{P}_2) :: (z{:}C)$$

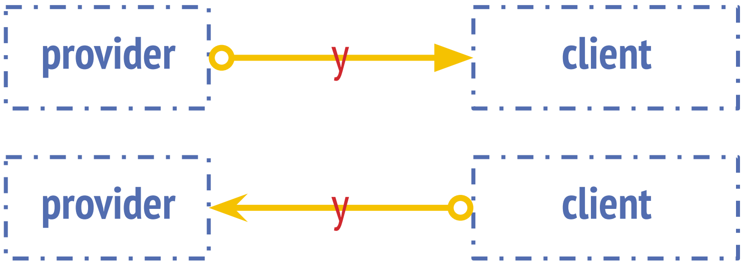
$$\Gamma \vdash \mathtt{P} :: (y{:}B) \; \text{- - y - -} \; \Gamma, y : B \vdash \mathtt{P}_2 :: (z{:}C)$$

Programs as Proofs

Session-typed Processes ⇠ ⇢ Sequent Calculus

Computation

Cut Reduction



Communications are bi-directional

Programs as Proofs

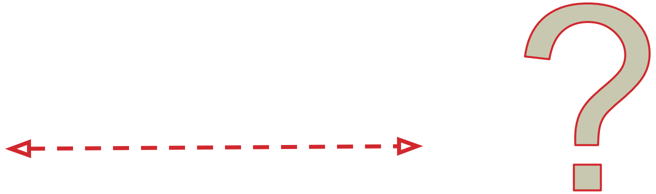
Session-typed Processes ↔ Sequent Calculus

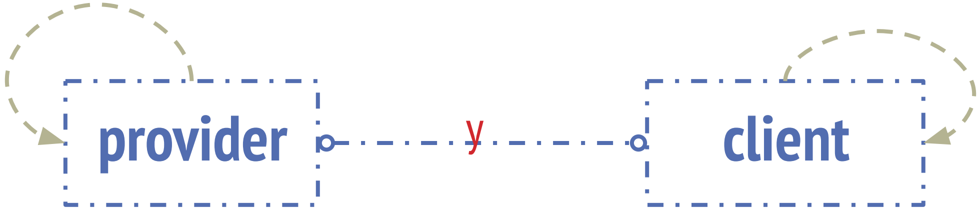
Termination

Cut Elimination

Programs as Proofs

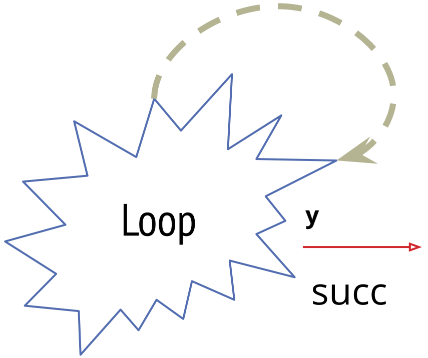
Recursive
Session-typed
Processes

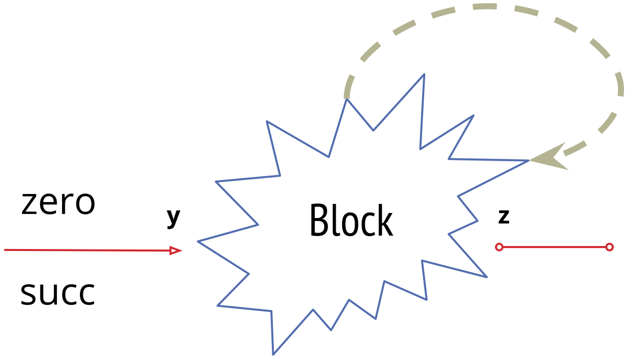




Recursion - An example of a process with only internal communications

$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\qquad \cdot \vdash \text{Loop} :: (y{:}\mathtt{nat})$ $\qquad y{:}\mathtt{nat} \vdash \text{Block} :: (z{:}1)$





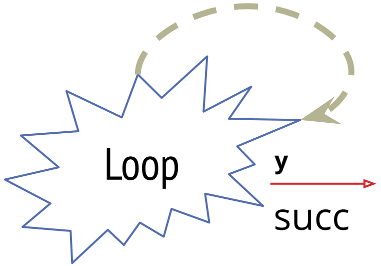
Loop sends a "succ" message along *y* and then calls itself recursively.

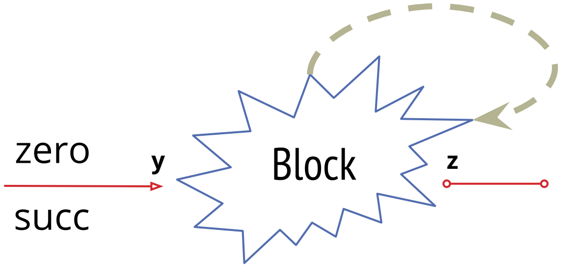
Block waits to receive a message along *y*, (a) if it is a "succ" it calls itself recursively, (b) if it is a "zero" it "closes" channel *z*.

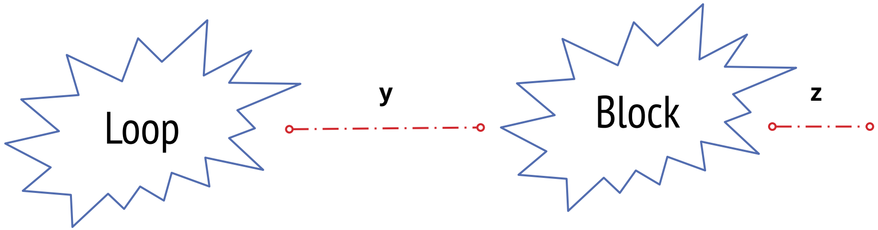
Deyoung and Pfenning 2016

Recursion - An example of a process with only internal communications

$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\quad \cdot \vdash \text{Loop} :: (y{:}\mathtt{nat})$ $\quad y{:}\mathtt{nat} \vdash \text{Block} :: (z{:}1)$

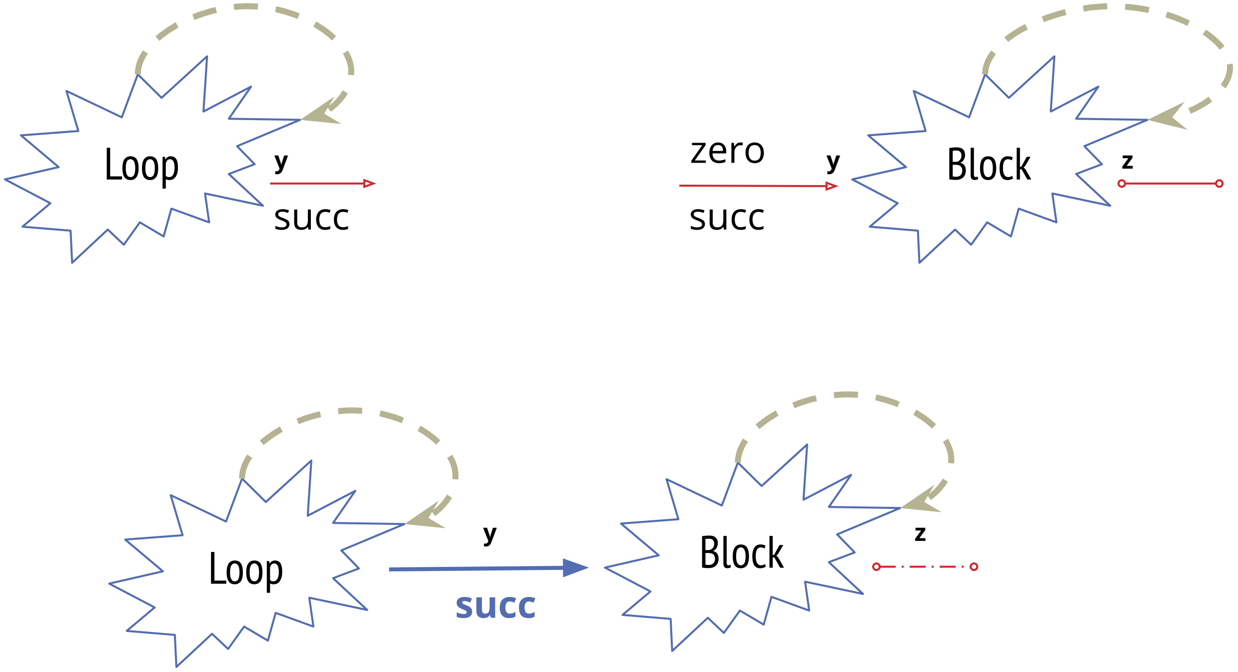






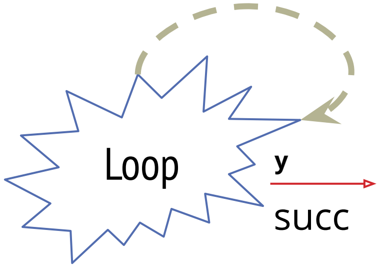
Recursion - An example of a process with only internal communications

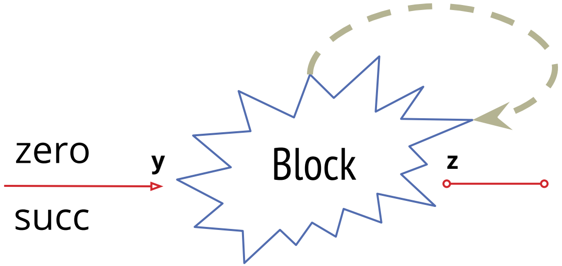
$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\quad \cdot \vdash \text{Loop} :: (y{:}\mathtt{nat})$ $\quad y{:}\mathtt{nat} \vdash \text{Block} :: (z{:}1)$

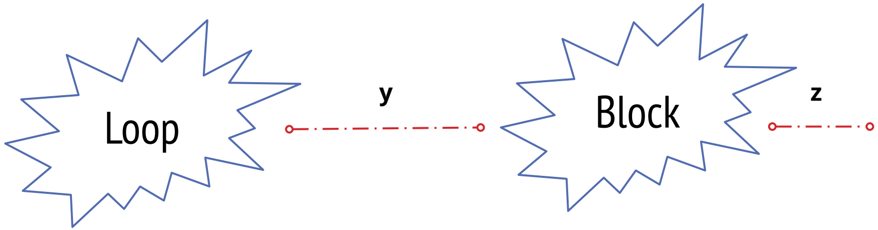


Recursion - An example of a process with only internal communications

$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\quad \cdot \vdash \text{Loop} :: (y{:}\mathtt{nat})$ $\quad y{:}\mathtt{nat} \vdash \text{Block} :: (z{:}1)$

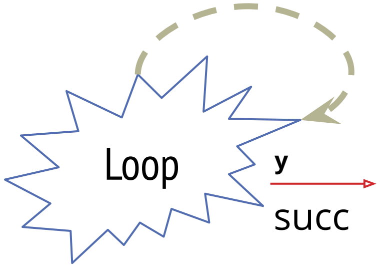


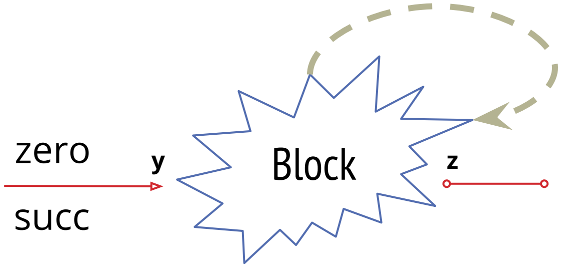


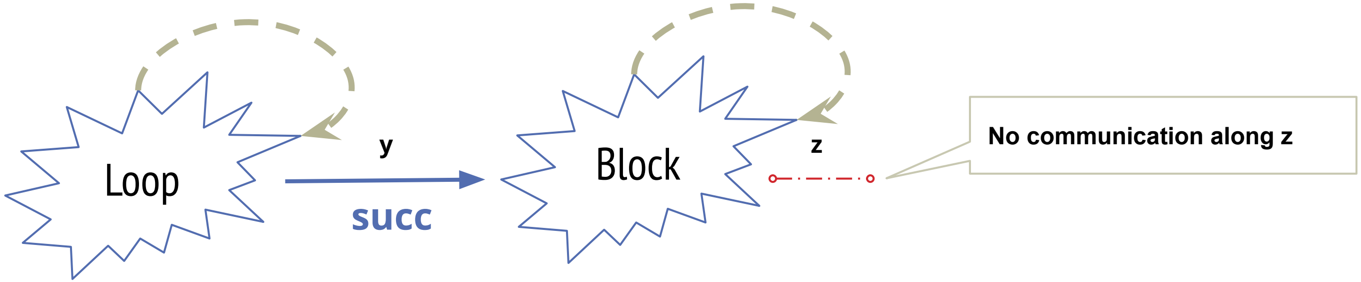


Recursion - An example of a process with only internal communications

$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\quad \cdot \vdash \text{Loop} :: (y{:}\mathtt{nat})$ $\quad y{:}\mathtt{nat} \vdash \text{Block} :: (z{:}1)$







Recursion - An example of a process with only internal communications

$\mathtt{nat} = \oplus\{zero : 1, succ : \mathtt{nat}\}$ $\quad \cdot \vdash \mathsf{Loop} :: (y{:}\mathtt{nat})$ $\quad y{:}\mathtt{nat} \vdash \mathsf{Block} :: (z{:}1)$

$$
\begin{aligned}
y \leftarrow \mathsf{Loop} = \\
&\quad y.succ; y \leftarrow \mathsf{Loop}
\end{aligned}
$$

$$
\begin{aligned}
z \leftarrow \mathsf{Block} \leftarrow y = \\
&\quad \mathbf{case}\, y(zero \Rightarrow \mathbf{wait} y; \mathbf{close} z \\
&\qquad\quad succ \Rightarrow z \leftarrow \mathsf{Block} \leftarrow y)
\end{aligned}
$$

Thesis statement

Even in the presence of recursion, we can retain the Curry-Howard isomorphism between **linear logic** and **session-typed concurrent programs** if we:

1. refine general **recursive session types** into **least and greatest fixed points**, and
2. impose *conditions* under which **recursively defined processes** correspond to **valid circular proofs**.

With this approach we can retain the correspondence between **cut elimination**, and **meaningful communication** with type preservation and strong progress.

Contributions

1. Extend the Curry-Howard interpretation of circular derivations in linear logic as communicating processes to include least and greatest fixed points.

   A circular derivation is thus represented as a collection of mutually recursive process definitions.

2. A compositional criterion for validity of such programs, which is local in the sense that *each process definition can be checked independently*.
3. Local validity implies a strong progress property on programs and cut elimination on the circular proofs they correspond to.
4. Implement the local validity algorithm.

We have completed the first four steps for the **subsingleton fragment**.

5. An *infinitary sequent calculus for first order intuitionistic multiplicative additive linear logic* with least and greatest fixed points; A tool to reason about a rich signature of mutually defined inductive and coinductive predicates.

   It also allows using nonlinear first order theories.

Computational power and potential applications

Linear processes	Operations on Lists, tries, streams, etc.
Subsingleton fragment	Turing machines, Linear communicating automata
Only positive types	Finite state transducers (cut-free!), Data processing with limited state and time

Deyoung and Pfenning 2016

Previous works

1. James Brotherston. 2005. Cyclic proofs for first-order logic with inductive definitions. In International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. Springer, 78–92.

2. Luigi Santocanale. 2002. A Calculus of Circular Proofs and Its Categorical Semantics. In 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2002), M. Nielsen and U. Engberg (Eds.). Springer LNCS 2303, Grenoble, France, 357–371

3. Jérôme Fortier and Luigi Santocanale. 2013. Cuts for Circular Proofs: Semantics and Cut-Elimination. In 22nd Annual Conference on Computer Science Logic (CSL 2013), Simona Ronchi Della Rocca (Ed.). LIPIcs 23, Torino, Italy, 248–262.

4. David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In 25th Annual Conference on Computer Science Logic (CSL 2016), J.-M. Talbot and L. Regnier (Eds.). LIPIcs 62, Marseille, France, 42:1–42:17.

5. Amina Doumane. On the Infinitary Proof Theory of Logics with Fixed Points. PhD thesis, Paris Diderot University, France, June 2017.

Outline

Circular derivations in (the subsingleton fragment of) linear logic

Local validity for recursive session-typed processes

Negative results

An infinitary calculus for first-order IMALL with fixed points

Proposed work - next steps

Conclusion

Outline

Circular derivations in (the subsingleton fragment of) linear logic

- The subsingleton logic with fixed points : two examples
- A guard condition
- Cut elimination

Local validity for recursive session-typed processes

Negative results

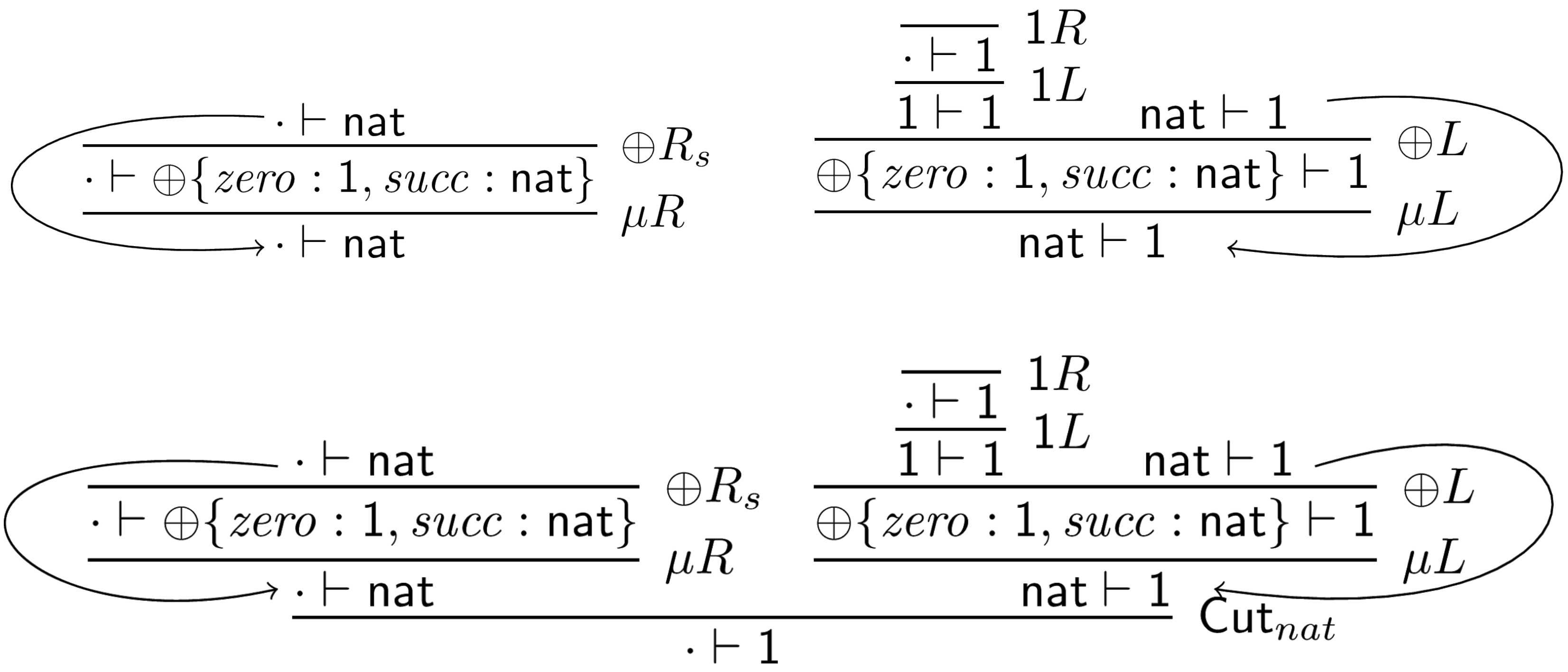
An infinitary calculus for first-order IMALL with fixed points

Proposed work - next steps

Conclusion

A Circular derivation in the subsingleton fragment

$$\texttt{nat} =^{1}_{\mu} \oplus\{zero : 1, succ : \texttt{nat}\}$$



Fortier and Santocanale, 2013; Santocanale 2002

A Circular derivation in the subsingleton fragment

$$\texttt{stream} =^{1}_{\nu} \&\{\mathit{next} : \texttt{cnat}\}$$

$$\texttt{cnat} =^{2}_{\mu} \oplus\{\mathit{cont} : \texttt{stream}, \mathit{succ} : \texttt{cnat}\}$$

$$\overline{1, \cdots} = ?\mathit{next}\,!\mathit{succ}\,!\mathit{cont} \cdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdot \vdash \mathsf{stream}}{\cdot \vdash \oplus\{\mathit{cont} : \mathsf{stream}, \mathit{succ} : \mathsf{cnat}\}}\ \mu R}{\cdot \vdash \mathsf{cnat}}\ \oplus R}{\cdot \vdash \oplus\{\mathit{cont} : \mathsf{stream}, \mathit{succ} : \mathsf{cnat}\}}\ \mu R}{\dfrac{\cdot \vdash \mathsf{cnat}}{\cdot \vdash \&\{\mathit{next} : \mathsf{cnat}\}}\ \& R}\ \nu R \quad \text{(top } \cdot \vdash \mathsf{stream} \text{ loops back to bottom } \cdot \vdash \mathsf{stream}\text{)}$$

$$\cdot \vdash \mathsf{stream}$$

Fortier and Santocanale, 2013; Santocanale 2002

Fortier and Santocanale's guard condition

Every cycle should be supported by the unfolding of

1. a **positive (least) fixed point** on the **antecedent**, or
2. a **negative (greatest) fixed point** on the **succedent**;

such that **the supporting fixed point for each cycle** is the **highest priority** among *all fixed points getting unfolded* in the cycle.

Fortier and Santocanale, 2013

The guard condition assures cut elimination

Fortier and Santocanale's cut elimination algorithm uses a *reduction* function ***Treat*** that may never halt.

Treat halts on guarded proofs; it produces a cut-free inference.

For guarded proofs cut can be eliminated *productively*.

Fortier and Santocanale, 2013

Outline

Circular derivations in linear logic

Local validity for recursive session-typed processes

- Example: Copy
- Example: PingPong
- Strong progress

Negative results

An infinitary calculus for first-order IMALL with fixed points

Proposed work - next steps

Conclusion

Our local validity condition

Recursive Processes ⟷ Circular derivations

A **locally checkable, compositional** validity condition on processes.

We check validity of each process separately!

Copy: a valid program

$$\texttt{nat} =^{1}_{\mu} \oplus\{zero : 1, succ : \texttt{nat}\}$$

Copy receives a natural number along channel x and sends it along channel y.

	x	y
$y \leftarrow \texttt{Copy} \leftarrow x =$	0	0
$\textbf{case}\, x\, (\mu_{nat} \Rightarrow$	-1	0
$\textbf{case}\, x\, (zero \Rightarrow y.\mu_{nat};$	-1	1
$y.zero; \textbf{wait}\, x; \textbf{close}\, y$	-1	1
$succ \Rightarrow y.\mu_{nat};$	-1	1
$y.succ; y \leftarrow \texttt{Copy} \leftarrow x))$	-1	1

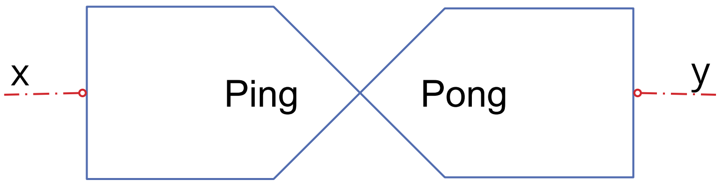
Ping-Pong: an invalid program

$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack},\quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1,\quad s : nat\}$$

$$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$$

$$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$$

$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$



Ping-Pong: an invalid program

$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack},\quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1,\quad s : nat\}$$

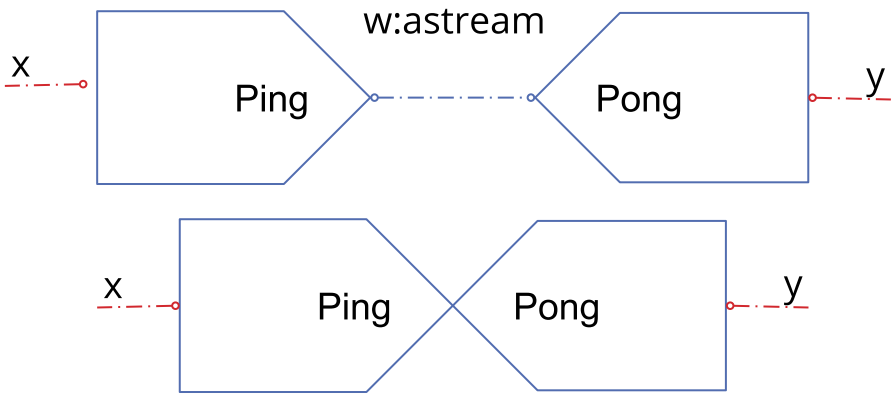
$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$

$$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$$

[0 , 0 , 0 , 0 , 0 , 0]

$$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$$

[0 , 0 , 0 , 0 , 0 , 0]



Ping-Pong: an invalid program

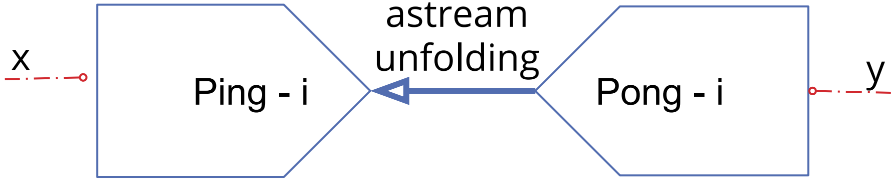
$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack}, \quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1, \quad s : nat\}$$

$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$

$$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$$

$$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$$

[0 , 0 , **-1** , 0 , 0 , 0]   w:&{head:ack, tail:astream}   [0 , 0 , 0 , **1** , 0 , 0]



Ping-Pong: an invalid program

$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack}, \quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1, \quad s : nat\}$$

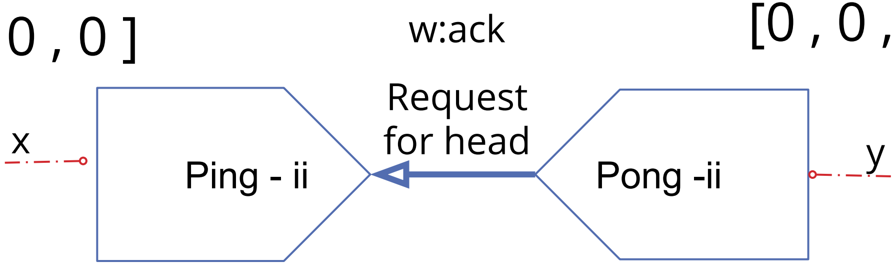
$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$

$$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$$

[0 , 0 , **-1** , 0 , 0 , 0]

$$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$$

[0 , 0 , 0 , **1** , 0 , 0]



Ping-Pong: an invalid program

$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack}, \quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1, \quad s : nat\}$$

$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$

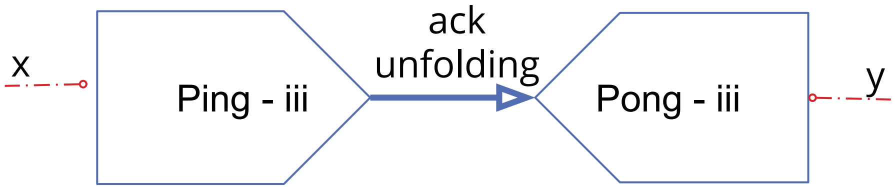
$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$

$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$

[0 , **1** , **-1** , 0 , 0 , 0]

w:+{ack:astream}

[**-1** , 0 , 0 , **1** , 0 , 0]



Ping-Pong: an invalid program

$$\Sigma := \mathsf{ack} =^1_\mu \oplus\{ack : \mathsf{astream}\},$$
$$\mathsf{astream} =^2_\nu \&\{head : \mathsf{ack}, \quad tail : \mathsf{astream}\},$$
$$\mathsf{nat} =^3_\mu \oplus\{z : 1, \quad s : nat\}$$

$$x : \mathsf{nat} \vdash \mathtt{Ping} :: (w : \mathsf{astream})$$
$$w : \mathsf{astream} \vdash \mathtt{Pong} :: (y : \mathsf{nat})$$
$$x : \mathsf{nat} \vdash \mathtt{PingPong} :: (y : \mathsf{nat})$$

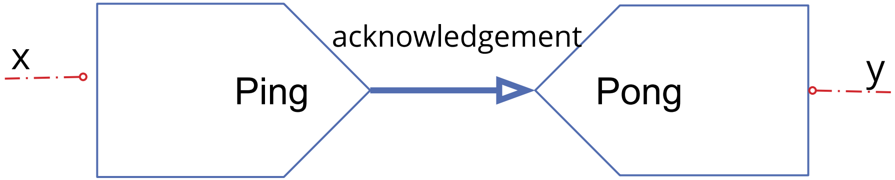
$[x_{\mathsf{ack}}, w_{\mathsf{ack}}, w_{\mathsf{astream}}, x_{\mathsf{astream}}, x_{\mathsf{nat}}, w_{\mathsf{nat}}]$

[0 , **1** , **-1** , 0 , 0 , 0]

w:astream

$[w_{\mathsf{ack}}, y_{\mathsf{ack}}, y_{\mathsf{astream}}, w_{\mathsf{astream}}, w_{\mathsf{nat}}, y_{\mathsf{nat}}]$

[**-1** , 0 , 0 , **1** , 0 , 0]



Back to the original configuration.

Ping-Pong: an invalid program - code

$$
\begin{array}{ll}
y \leftarrow \texttt{PingPong} \leftarrow x = & \\
\quad w \leftarrow \texttt{Ping} \leftarrow x; & \%\ \textit{spawn process } \texttt{Ping}_w \\
\quad\quad y \leftarrow \texttt{Pong} \leftarrow w & \%\ \textit{continue with a tail call}
\end{array}
$$

$$
\begin{array}{ll}
w \leftarrow \texttt{Ping} \leftarrow x = & [0, 0, 0, 0, 0, 0] \\
\quad \textbf{case } Rw\ (\nu_{astream} \Rightarrow & [0, 0, -1, 0, 0, 0] \\
\quad\quad \textbf{case } Rw\ (head \Rightarrow Rw.\mu_{ack}; & [0, 1, -1, 0, 0, 0] \\
\quad\quad\quad Rw.ack; w \leftarrow \texttt{Ping} \leftarrow x & [0, 1, -1, 0, 0, 0] \\
\quad\quad \mid tail \Rightarrow w \leftarrow \texttt{Ping} \leftarrow x)) & [0, 0, -1, 0, 0, 0]
\end{array}
$$

$$
\begin{array}{ll}
y \leftarrow \texttt{Pong} \leftarrow w = & [0, 0, 0, 0, 0, 0] \\
\quad Lw.\nu_{astream}; & [0, 0, 0, 1, 0, 0] \\
\quad Lw.head; & [0, 0, 0, 1, 0, 0] \\
\quad \textbf{case } Lw\ (\mu_{ack} \Rightarrow & [-1, 0, 0, 1, 0, 0] \\
\quad\quad \textbf{case } Lw\ (& [-1, 0, 0, 1, 0, 0] \\
\quad\quad\quad ack \Rightarrow Ry.\mu_{nat}; & [-1, 0, 0, 1, 0, 1] \\
\quad\quad\quad Ry.s; & [-1, 0, 0, 1, 0, 1] \\
\quad\quad\quad y \leftarrow \texttt{Pong} \leftarrow w)) & [-1, 0, 0, 1, 0, 1]
\end{array}
$$

Our validity condition implies the guard condition

Theorem 1. Our local condition implies guard condition of the underlying derivation; therefore it implies termination of reduction function ***Treat*** and cut elimination.

Strong progress and cut elimination

Theorem 2. A valid program *always terminates* either in an empty configuration or one attempting to communicate along external channels.

Strong Progress ⇠ - - - - - - ⇢ Cut elimination

Outline

Circular derivations in linear logic

Local validity for recursive session-typed processes

Negative results

Turing machines and undecidability of strong progress

Binary counter: a negative example

An infinitary calculus for first-order intuitionistic MALL with fixed points

Proposed work - next steps

Conclusion

Turing machines and undecidability of strong progress

Cut reduction on circular pre-proofs in *subsingleton logic* with recursive types has the computational power of Turing machines.[1]

Theorem. Recognizing all programs that satisfy a compositional *strong progress property* is ***undecidable***.

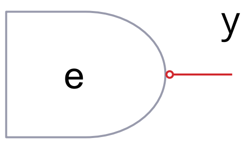
Proof. Termination of a Turing machine can be encoded as strong progress.

[1] Deyoung and Pfenning 2016

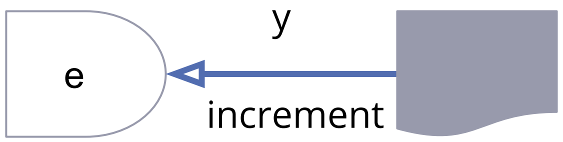
Binary counter: a negative example

$$\Sigma_8 := \mathsf{ctr} =^1_\nu \&\{inc : \mathsf{ctr}, \quad val : \mathsf{bin}\},$$
$$\mathsf{bin} =^2_\mu \oplus\{b0 : \mathsf{bin}, b1 : \mathsf{bin}, \$: 1\}$$

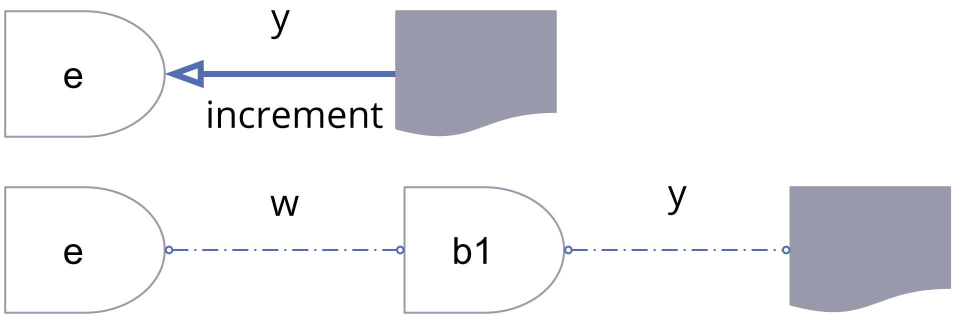
Start with an empty counter that offers
along channel y:ctr



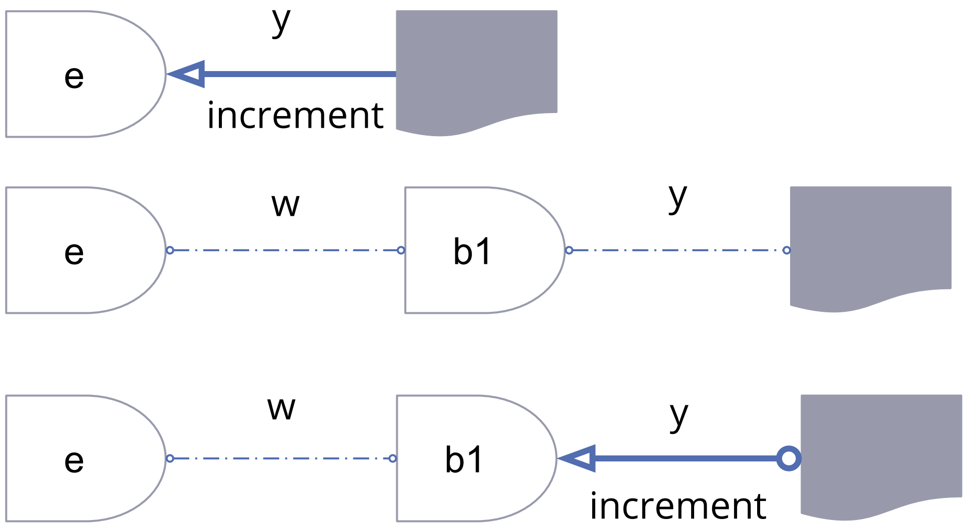
Binary counter: a negative example



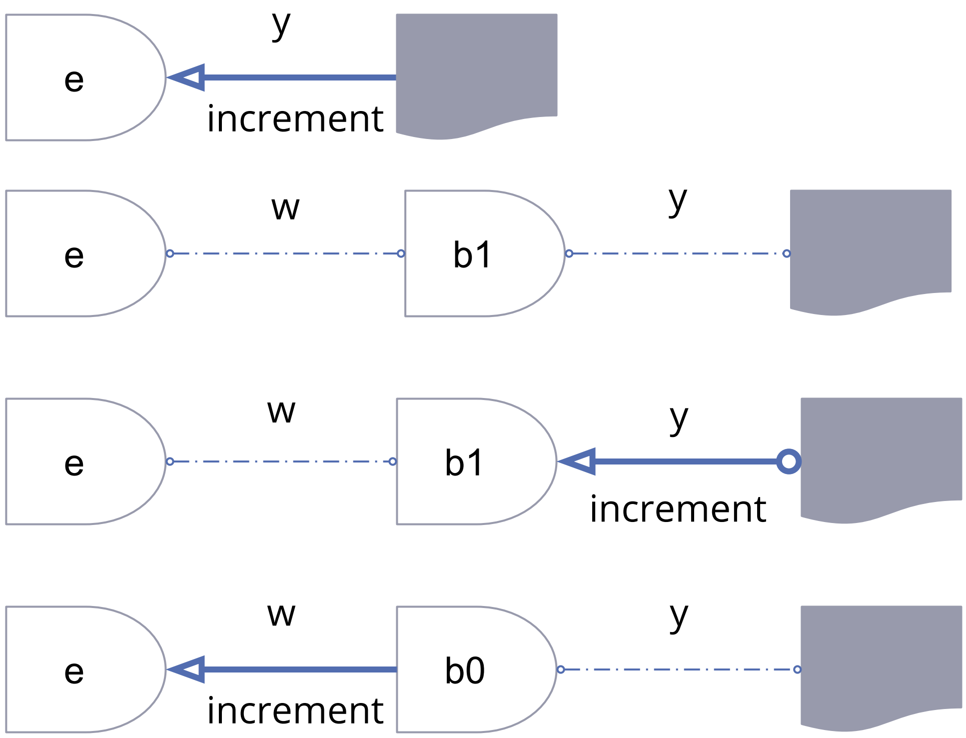
Binary counter: a negative example



Binary counter: a negative example



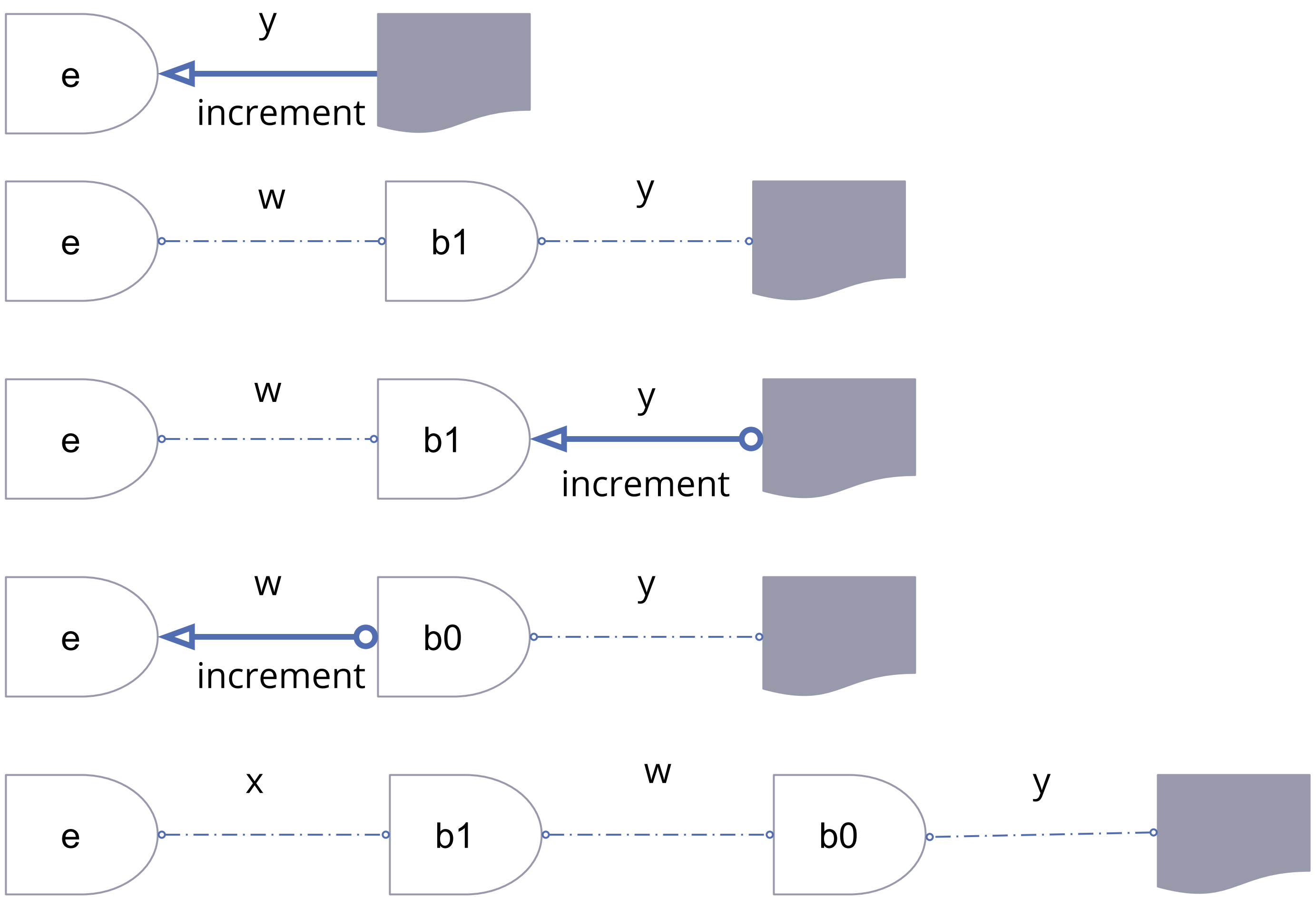
Binary counter: a negative example



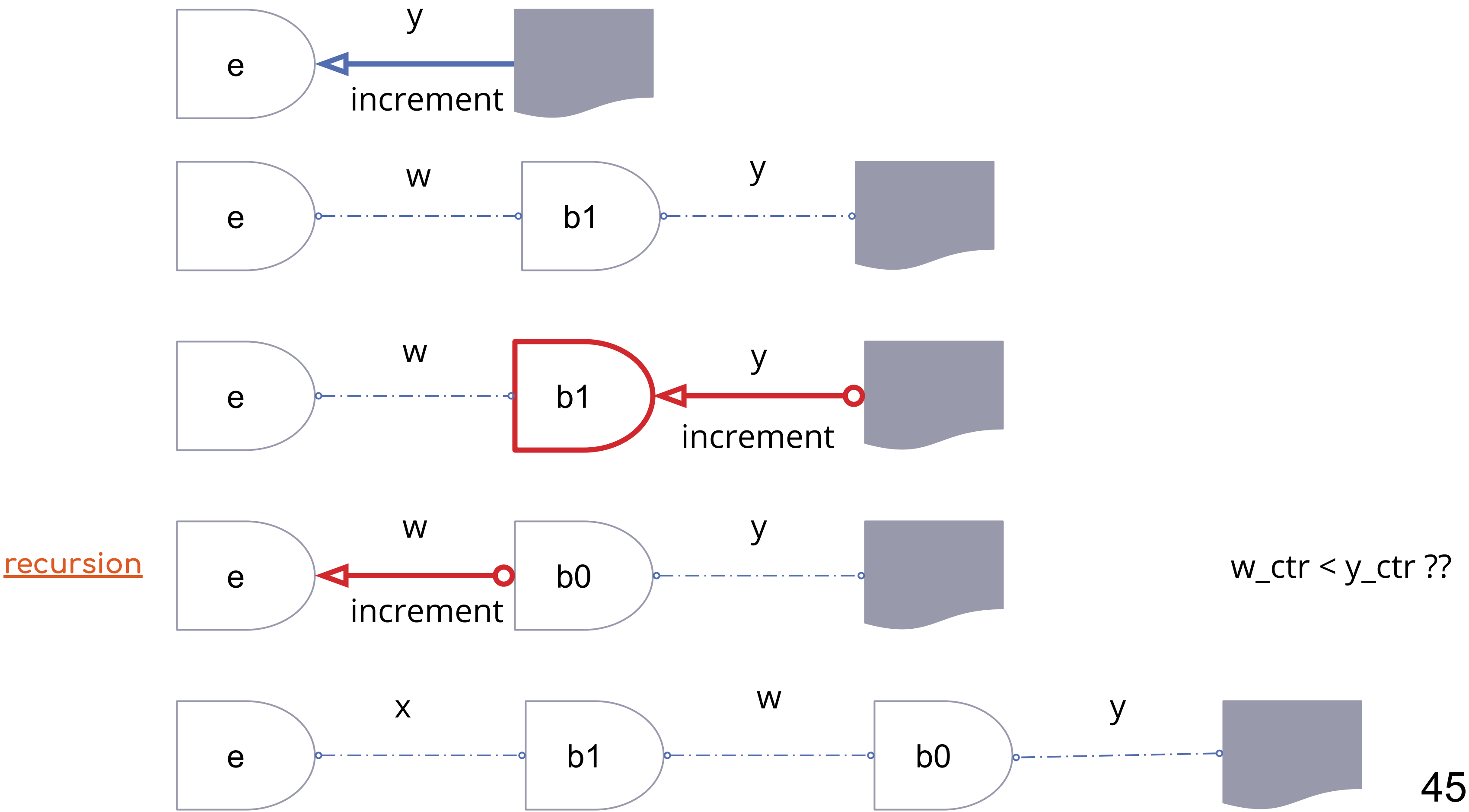
recursion

Binary counter: a negative example

recursion



Binary counter: a negative example



Binary counter: a negative example - code

$$\Sigma_8 := \mathsf{ctr} =^1_\nu \&\{inc : \mathsf{ctr},\ \ val : \mathsf{bin}\},$$
$$\mathsf{bin} =^2_\mu \oplus\{b0 : \mathsf{bin}, b1 : \mathsf{bin}, \$: 1\}$$

$$x : \mathsf{ctr} \vdash y \leftarrow \texttt{Bit0Ctr} \leftarrow x :: (y : \mathsf{ctr})$$
$$x : \mathsf{ctr} \vdash y \leftarrow \texttt{Bit1Ctr} \leftarrow x :: (y : \mathsf{ctr})$$
$$\cdot \vdash y \leftarrow \texttt{Empty} :: (y : \mathsf{ctr})$$

$$
\begin{array}{ll}
y^\beta \leftarrow \texttt{Bit0Ctr} \leftarrow x^\alpha = & [0, 0, 0, 0] \\
\quad \textbf{case}\, Ry^\beta\ (\nu_{ctr} \Rightarrow & [-1, 0, 0, 0] \\
\quad\quad \textbf{case}\, Ry^{\beta+1}\ (inc \Rightarrow\ y^{\beta+1} \leftarrow \texttt{Bit1Ctr} \leftarrow x^\alpha & [-1, 0, 0, 0] \\
\quad\quad\quad \mid val \Rightarrow Ry^{\beta+1}.\mu_{bin}; Ry^{\beta+2}.b0; Lx^\alpha.\nu_{ctr}; Lx^{\alpha+1}.val; y^{\beta+2} \leftarrow x^{\alpha+1})) & [-1, 1, 0, 1] \\
y^\beta \leftarrow \texttt{Bit1Ctr} \leftarrow x^\alpha = & [0, 0, 0, 0] \\
\quad \textbf{case}\, Ry^\beta\ (\nu_{ctr} \Rightarrow & [-1, 0, 0, 0] \\
\quad\quad \textbf{case}\, Ry^{\beta+1}\ (inc \Rightarrow\ Lx^\alpha.\nu_{ctr}; Lx^{\alpha+1}.inc; y^{\beta+1} \leftarrow \texttt{Bit0Ctr} \leftarrow x^{\alpha+1} & [-1, 1, 0, 0] \\
\quad\quad\quad \mid val \Rightarrow Ry^{\beta+1}.\mu_{bin}; Ry^{\beta+2}.b1; Lx^\alpha.\nu_{ctr}; Lx^{\alpha+1}.val; y^{\beta+2} \leftarrow x^{\alpha+1})) & [-1, 1, 0, 1] \\
y^\beta \leftarrow \texttt{Empty} \leftarrow \cdot = & [0, \_, \_, 0] \\
\quad \textbf{case}\, Ry^\beta\ (\nu_{ctr} \Rightarrow & [-1, \_, \_, 0] \\
\quad\quad \textbf{case}\, Ry^{\beta+1}\ (inc \Rightarrow\ w^0 \leftarrow \texttt{Empty} \leftarrow \cdot; & [\infty, \_, \_, \infty] \\
\quad\quad\quad\quad y^{\beta+1} \leftarrow \texttt{Bit1Ctr} \leftarrow w^0 & [-1, \infty, \infty, 0] \\
\quad\quad\quad \mid val \Rightarrow Ry^{\beta+1}.\mu_{bin}; Ry^{\beta+2}.\$; \textbf{close}\, Ry^{\beta+2})) & [-1, \_, \_, 1]
\end{array}
$$

Generalize the local validity condition?

We cannot rely on the guard condition anymore.

We need an alternative technique to prove strong progress:

- *Proof using logical relations*

Simultaneous induction/coinduction

Outline

Circular derivations in linear logic

Local validity for recursive session-typed processes

Negative results

An infinitary calculus for first-order intuitionistic MALL with fixed points

- Previous work
- Example: productivity of run(x,t)
- Strong progress of locally valid processes

Proposed work - next steps

Conclusion

Our goal

A calculus to reason about data-types defined as mutual least and greatest fixed points.

Reason about session-typed programs.

Use circular derivations to prove theorems by simultaneous induction and coinduction.

Previous work: Calculi for inductive and coinductive proofs

- Coinduction principle [Kozen and Silva, 2017]
- An infinitary calculus for first-order logic with inductive definitions [Brotherston, 2005]
- A finitary calculus for least and greatest fixed points in linear logic [Baelde, 2007]
- Well founded recursion with copatterns and sized types [Abel and Pientka, 2016]

An infinitary sequent calculus for first order intuitionistic MALL with fixed points

To ***reason about programs*** in a meta-circular way.

Our calculus is mainly designed for linear reasoning but we also allow appealing to first order theories such as arithmetic, by adding an adjoint downgrade modality.

A *condition* to identify (a subset of) *valid proofs* among all infinite derivations.

We proved cut elimination for the valid proofs.

Programming with mutual least and greatest fixed points

run(x,t): A stream producer where x is the list of operations, and t is the output stream.

Skip one step and do nothing

Put z as the head of output stream and inserts the new list of operations x to the original one.

$$\begin{aligned}
\mathtt{run}(\cdot\,, t) &=^1_\mu \quad 1 \\
\mathtt{run}(skip; x, t) &=^1_\mu \quad \mathtt{run}(x, t) \\
\mathtt{run}(put\langle x\rangle; y, t) &=^1_\mu \quad \mathtt{nrun}\,(x, y, t) \\
\mathtt{nrun}(x, y, t) &=^2_\nu \quad \mathtt{hd}\, t = \mathtt{o} \,\&\, \mathtt{run}(x; y, \mathtt{tl}\ t)
\end{aligned}$$

Run on any list of operations produces a (possibly infinite) list of elements "o"

$$
\begin{aligned}
\texttt{run}(\cdot\,, t) &=^1_\mu 1 \\
\texttt{run}(skip; x, t) &=^1_\mu \texttt{run}(x, t) \\
\texttt{run}(put\langle x\rangle; y, t) &=^1_\mu \texttt{nrun}\,(x, y, t) \\
\texttt{nrun}(x, y, t) &=^2_\nu \texttt{hd}\,t = \texttt{o}\,\&\,\texttt{run}(x; y, \texttt{tl}\,t)
\end{aligned}
$$

$$
\begin{aligned}
\texttt{list}_\texttt{o}(t) &=^1_\mu \oplus\{\mathsf{nil} : 1, \mathsf{next} : \texttt{stream}_\texttt{o}(t)\} \\
\texttt{stream}_\texttt{o}(t) &=^2_\nu \&\{\mathsf{hd} : \texttt{hd}\,t = \texttt{o}, \mathsf{tl} : \texttt{list}_\texttt{o}\,(\texttt{tl}\,t)\}
\end{aligned}
$$

$$(\dagger)\ \texttt{run}(x, t) \vdash \texttt{list}_\texttt{o}(t)$$

$$(\star)\ \texttt{nrun}(x, y, t) \vdash \texttt{stream}_\texttt{o}(t)$$

Run produces a list$_o$ - proof

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash 1}\;1R}{\cdot \vdash \oplus\{\mathsf{nil} : 1, \mathsf{next} : \mathtt{stream}_o(t)\}}\;\oplus R}{\cdot \vdash \mathtt{list}_o(t)}\;\mu_{\mathtt{list}_o}R}{1 \vdash \mathtt{list}_o(t)}\;1L}{\dagger\, \mathtt{run}(\cdot\,, t) \vdash \mathtt{list}_o(t)}\;\mu_{\mathtt{run}}L
$$

$$
\dfrac{\overset{\dagger}{\mathtt{run}(x, t) \vdash \mathtt{list}_o(t)}}{\dagger\, \mathtt{run}((skip; x), t) \vdash \mathtt{list}_o(t)}\;\mu_{\mathtt{run}}L
$$

$$
\dfrac{\dfrac{\dfrac{\overset{\star}{\mathtt{nrun}\,(x, y, t) \vdash \mathtt{stream}_o(t)}}{\mathtt{nrun}\,(x, y, t) \vdash \oplus\{\mathsf{nil} : 1, \mathsf{next} : \mathtt{stream}_o(t)\}}\;\oplus R}{\mathtt{nrun}\,(x, y, t) \vdash \mathtt{list}_o(t)}\;\mu_{\mathtt{list}_o}R}{\dagger\, \mathtt{run}(put\langle x\rangle; y, t) \vdash \mathtt{list}_o(t)}\;\mu_{\mathtt{run}}L
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{hd}\,t = \mathsf{o} \vdash \mathsf{hd}\,t = \mathsf{o}}\;\mathtt{ID}}{\&\{\mathsf{hd} : \mathsf{hd}\,t = \mathsf{o}, \mathsf{tl} : \mathtt{run}(x; y, \mathtt{tl}\,t)\} \vdash \mathsf{hd}\,t = \mathsf{o}}\;\&L}{\mathtt{nrun}\,(x, y, t) \vdash \mathsf{hd}\,t = \mathsf{o}}\;\nu_{\mathtt{nrun}}L \qquad \dfrac{\dfrac{\overset{\dagger}{\mathtt{run}(x; y, \mathtt{tl}\,t) \vdash \mathtt{list}_o(\mathsf{tl}\,t)}}{\&\{\mathsf{hd} : \mathsf{hd}\,t = \mathsf{o}, \mathsf{tl} : \mathtt{run}(x; y, \mathtt{tl}\,t)\} \vdash \mathtt{list}_o(\mathsf{tl}\,t)}\;\&L}{\mathtt{nrun}\,(x, y, t) \vdash \mathtt{list}_o(\mathsf{tl}\,t)}\;\nu_{\mathtt{nrun}}L}{\dfrac{\mathtt{nrun}\,(x, y, t) \vdash \&\{\mathsf{hd} : \mathsf{hd}\,t = \mathsf{o},\ \mathsf{tl} : \mathtt{list}_o(\mathsf{tl}\,t)\}}{\star\, \mathtt{nrun}\,(x, y, t) \vdash \mathtt{stream}_o(t)}\;\nu_{\mathtt{stream}_o}R}\;\&R
$$

A valid configuration of processes satisfies strong progress

We define strong progress as a predicate

$$\mathcal{C} \in [\![x : A]\!]$$

$$\cdot \vdash \mathcal{C} :: (x{:}A) \quad \overset{\text{Bisimulation}}{\Longleftrightarrow} \quad \cdot \vdash \mathcal{C} \in [\![x : A]\!]$$

Theorem. *If configuration C is well-typed then there is an infinite derivation for its strong progress property. Moreover, if it C is valid, the infinite derivation is a proof.*

Outline

Circular derivations in linear logic

Local validity for recursive session-typed processes

Negative results

An infinitary calculus for first-order intuitionistic MALL with fixed points

Proposed work - next steps

- Subsingleton fragment - revisited
- Linear logic
- Mode shifts

Conclusion

1. A more general local validity condition for the subsingleton fragment

$$
\begin{array}{ll}
y^{\beta} \leftarrow \texttt{Empty} \leftarrow \cdot = & [0, \_, \ \_, \ 0] \\
\quad \textbf{case}\ Ry^{\beta}\ (\nu_{ctr} \Rightarrow & [-1, \_, \_, 0] \\
\quad\quad \textbf{case}\ Ry^{\beta+1}\ (inc \Rightarrow\ w^{0} \leftarrow \texttt{Empty} \leftarrow \cdot; & [\infty, \ \_, \ \_, \infty] \\
\quad\quad\quad\quad y^{\beta+1} \leftarrow \texttt{Bit1Ctr} \leftarrow w^{0} & [-1,\ \infty, \infty, 0] \\
\quad\quad\quad \mid val \Rightarrow Ry^{\beta+1}.\mu_{bin}; Ry^{\beta+2}.\$; \textbf{close}\ Ry^{\beta+2})) & [-1, \_, \_, 1]
\end{array}
$$

We need to know that Bit1Ctr output is "smaller" than its input.

Use our calculus to prove strong progress property for the generalized version using *logical relations*.

2. Linear logic

Processes defined based on linear logic may use more than one resource.

$$x_1{:}A_1, x_2{:}A_2, \cdots x_n{:}A_n \vdash \mathtt{P} :: (y{:}B)$$

Track the values of all channels on the left and the one on the right for each fixed point in the signature.

A lexicographic order on the list of all channels.

An example: Append two finite lists

$$\Sigma_6 := \mathsf{list}_A =^1_\mu \oplus\{nil : 1, cons : A \otimes \mathsf{list}_A\}$$

$$l_1 : \mathsf{list}, l_2 : \mathsf{list} \vdash \texttt{Append} :: (l : \mathsf{list})$$

If l1 is an empty list (nil): forward l2 to l.

If l1=cons(x, --): send x to l and call Append on l2 and the remaining of l1.

	l_1	l_2	l
$l \leftarrow \texttt{Append} \leftarrow l_1, l_2 =$	0	0	0
$\quad \textbf{case}\, l_1\, (\mu_{list} \Rightarrow$	-1	0	0
$\quad\quad \textbf{case}\, l_1\, (nil \Rightarrow \textbf{wait}\, l_1; l \leftarrow l_2$	-1	0	0
$\quad\quad cons \Rightarrow l.\mu_{list};$	-1	0	1
$\quad\quad\quad x \leftarrow \textbf{recv}\, l;$	-1	0	1
$\quad\quad\quad l.cons; \textbf{send}\, l\, x; l \leftarrow \texttt{Append} \leftarrow l_1\, l_2))$	-1	0	1

Polarity shifts

Type t appears in both positive and negative positions

$$t =^1_\mu t \multimap t$$

$$\Downarrow$$

$$t^- =^1_\mu \downarrow^-_+ t^- \multimap t^-$$

3. Shift for modes

$$A_l ::= \cdots \mid \uparrow_s^l A_s$$

$$A_s ::= \cdots \mid \downarrow_s^l A_l$$

Outline

Circular derivations in linear logic

Local validity for recursive session-typed processes

Negative results

An infinitary calculus for first-order intuitionistic MALL with fixed points

Proposed work - next steps

Conclusion

Conclusion

- *Results we accomplished so far:*
    a. A local validity condition for recursive session-typed processes in the subsingleton fragment
    b. Our local validity ensures the guard condition; thus it implies strong progress
    c. Implementation of the condition as a static check in SML
    d. A first order infinitary calculus to reason about programs
    e. A validity condition that ensures cut elimination
    f. Prove strong progress of locally valid processes directly

- *Next steps:*
    a. A more generalized version of local validity condition for the subsingleton fragment
    b. A local validity condition for linear logic; a special treatment of function types
    c. Prove strong progress for locally valid processes
        i. To use our first order calculus
- *Time permitting:*
    a. Generalize the results for the calculus with adjoint modalites for mode shifts

Thank you!

References

1. Frank Pfenning. Substructural logics. Lecture notes for course given at Carnegie Mellon University, Fall 2016, December 2016.
2. Farzaneh Derakhshan and Frank Pfenning. 2019. Circular Proofs as Session-Typed Processes: A Local Validity Condition. arXiv preprint arXiv:1908.01909 (2019).
3. Farzaneh Derakhshan and Frank Pfenning. 2020. Circular Proofs in First-Order Linear Logic with Least and Greatest Fixed Points. arXiv preprint arXiv:2001.05132 (2020).
4. Andreas Abel and Brigitte Pientka. 2016. Well-founded recursion with copatterns and sized types. Journal of Functional Programming 26(2016).
5. David Baelde and Dale Miller. 2007. Least and greatest fixed points in linear logic. In International Conference on Logic for Programming Artificial Intelligence and Reasoning. Springer, 92–106
6. Amina Doumane. 2017.On the infinitary proof theory of logics with fixed points. Ph.D. Dissertation.
7. David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary proof theory: the multiplicative additive case. (2016).
8. James Brotherston. 2005. Cyclic proofs for first-order logic with inductive definitions. In International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. Springer, 78–92.
9. Dexter Kozen and Alexandra Silva. 2017. Practical coinduction.Mathematical Structures in Computer Science 27, 7 (2017), 1132–1152.
10. Gentzen, 1932 - Prawitz, 1965
11. W. A. Howard. The formulae-as-types notion of construction. Unpublished note. An annotated version appeared in: To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, 479–490, Academic Press (1980), 1969.
12. Hilbert D, Bernays P (1921-22) Grundlagen der Mathematik. In David Hilbert's Lectures on the Foundations of Mathematics and Physics 1917-1933. W. B. Ewald and W. Sieg, editors, Springer, 2013:431-518
13. Gentzen G (1932-3) Urdissertation, Wissenschaftshistorische Sammlung, Eidgenossische Technische Hochschule. Zurich, Bernays Nachlass, Ms ULS (A detailed description of the manuscript is found in (von Plato 2009))