Logic and Interaction
A Semantic Study of Totality

Pierre Clairambault

PPS — Université Paris 7

Outline of the talk

1. Introduction
   - Logical motivations
   - Semantic motivations
2. Pointer structures and normalization
   - Pointer structures
   - Pointer structures and λ-calculus
3. Inductive and Coinductive Types
   - Games model
   - Winning conditions
4. Conclusions & Perspectives

I. Introduction

1. Logical motivations

Tarski's notion of truth.

$$
\begin{array}{rcl}
\top & & \text{is true} \\
\bot & & \text{is false} \\
A \wedge B \text{ is true} & \Leftrightarrow & A \text{ is true and } B \text{ is true} \\
A \vee B \text{ is true} & \Leftrightarrow & A \text{ is true or } B \text{ is true} \\
\neg A \text{ is true} & \Leftrightarrow & A \text{ is false}
\end{array}
$$

Seems rather circular...

Games

Two players are arguing over the validity of a formula F.

Defender	Attacker
Verifier	Falsifier
$\exists$loïse	$\forall$bélard
Eve	Adam
Player	Opponent

Player	:	"F is true!"
Opponent	:	"F is false!"

Rules

$A \quad \wedge \quad B$

O attacks B

$A \quad \vee \quad B$

O : "Do you defend A or B ?"

P : "I defend A"

O attacks A

$\neg A$

O passes hand to P

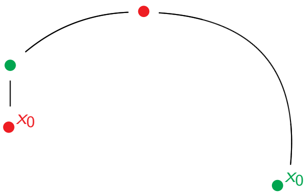
P attacks A

A formula is "true" if Player has a total strategy.

Backtrack

Players can backtrack to an earlier position

$$\forall x A(x) \quad \vee \quad \exists x \neg A(x)$$



O attacks

P defends $\forall x A(x)$

O attacks $A(x_0)$

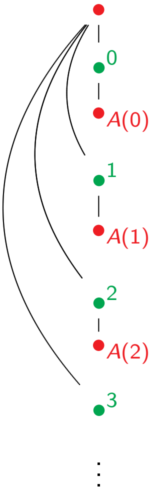
P defends $\neg A(x_0)$

This introduces **repetitions**.

Repetitions may lead to non-termination

Consider the following play:

$$\exists x \in \mathbb{N} \;\; A(x)$$



O attacks

P instanciates x by 0

O refutes $A(0)$

P backtracks with $x = 1$

$\vdots$

This strategy should be invalid.

Statement of the issue

Issue

What are the natural constraints on strategies to ensure the finiteness of their debates ?

The right answer depends on the considered formulas:

1. Finite or well-founded formulas
2. Infinite or non-well-founded formulas ((co)inductive types)

2. Semantic motivations

Game semantics

Game semantics is the study of the interactive behaviour of a **program** against its **environment** :

- A type A is interpreted by a **game**
- A program $M : A \Rightarrow B$ is interpreted as a **strategy**

Togother with a notion of composition of strategies.

Composition

Composition is defined by parallel interaction plus hiding.

$$\mathbb{B} \xRightarrow{\textbf{not}} \mathbb{B} \qquad \mathbb{B} \xRightarrow{\textbf{not}} \mathbb{B}$$

$$\color{red}{q}$$

Composition

Composition is defined by parallel interaction plus hiding.

$$\begin{array}{ccccccc}
\mathbb{B} & \xRightarrow{\textbf{not}} & \mathbb{B} & \qquad & \mathbb{B} & \xRightarrow{\textbf{not}} & \mathbb{B} \\
 & & & & & & q \\
 & & q & & q & &
\end{array}$$

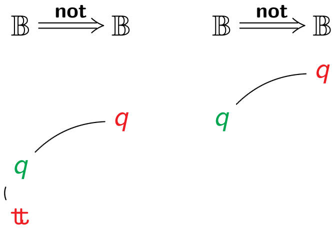
Composition

Composition is defined by parallel interaction plus hiding.

$$\begin{array}{ccccccc} \mathbb{B} & \xRightarrow{\textbf{not}} & \mathbb{B} & \qquad & \mathbb{B} & \xRightarrow{\textbf{not}} & \mathbb{B} \\ &&&&&& q \\ && q && q && \\ q &&&&&& \end{array}$$

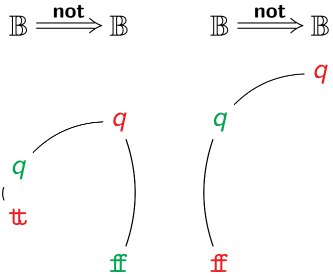
Composition

Composition is defined by parallel interaction plus hiding.



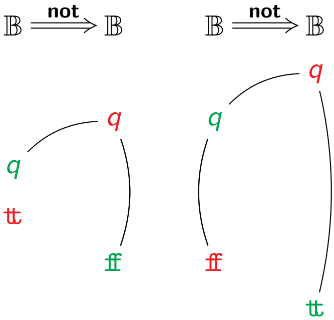
Composition

Composition is defined by parallel interaction plus hiding.



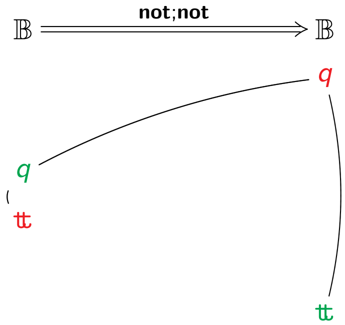
Composition

Composition is defined by parallel interaction plus hiding.



Composition

Composition is defined by parallel interaction plus hiding.



Semantics of proofs

Game semantics is the study of the interactive behaviour of a ~~program~~ proof against its ~~environment~~ counter-proofs :

- A ~~type~~ formula A is interpreted by a **game**
- A ~~program~~ proof $M : A \Rightarrow B$ is interpreted as a total strategy

Togother with a notion of composition of strategies.

Composition may not preserve totality

$$A \xRightarrow{\sigma} B \qquad\qquad B \xRightarrow{\tau} C$$

Composition may not preserve totality

$$A \xRightarrow{\sigma} B \qquad\qquad B \xRightarrow{\tau} C$$

Composition may not preserve totality

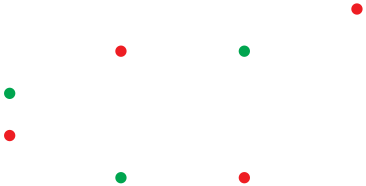
$$A \xRightarrow{\sigma} B \qquad\qquad B \xRightarrow{\tau} C$$

Composition may not preserve totality

$$A \xRightarrow{\sigma} B \qquad\qquad B \xRightarrow{\tau} C$$

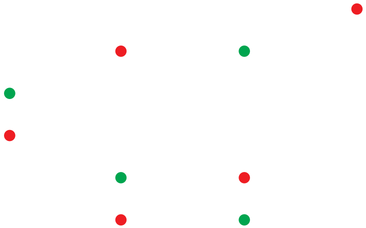
Composition may not preserve totality

$A \overset{\sigma}{\Longrightarrow} B \qquad B \overset{\tau}{\Longrightarrow} C$



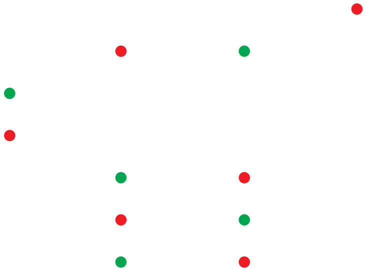
Composition may not preserve totality

$A \xRightarrow{\sigma} B \qquad B \xRightarrow{\tau} C$



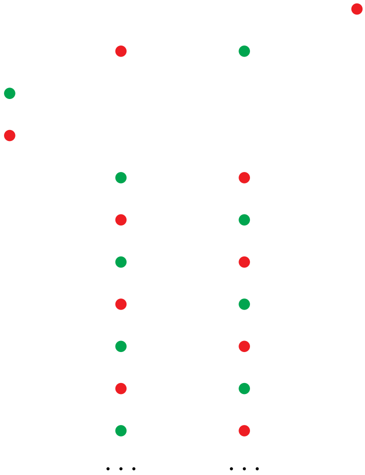
Composition may not preserve totality

$$A \xRightarrow{\sigma} B \qquad\qquad B \xRightarrow{\tau} C$$



Composition may not preserve totality

$A \xRightarrow{\sigma} B$ $B \xRightarrow{\tau} C$



Composition may not preserve totality

$$A \xRightarrow{\sigma;\tau} C$$

Semantics of proofs

To get a model of a proof system, we need to:

Issue

Isolate classes of total strategies which are stable under composition.

Equivalently:

Issue

Find constraints on strategies to ensure the finiteness of their debates.

II. Pointer structures and normalization

1. Pointer structures

Pointer structures

To study finiteness, we forget the identity of moves and focus on **pointers**

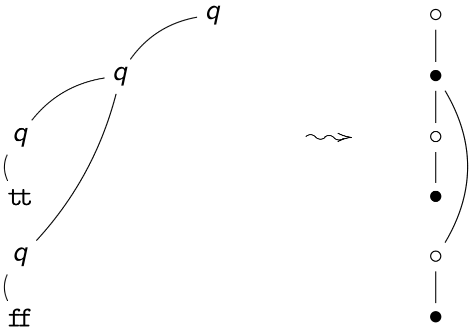
- Technical simplifications,
- Sufficient to study termination,
- Similar to **parity pointer functions** [HHM06] and **interaction sequences** [Coq95].

This simplification amounts to a **collapse** operation on plays.

The collapse

We consider only the **depth** of moves.

$$(\mathbb{B} \Longrightarrow \mathbb{B}) \Longrightarrow o$$

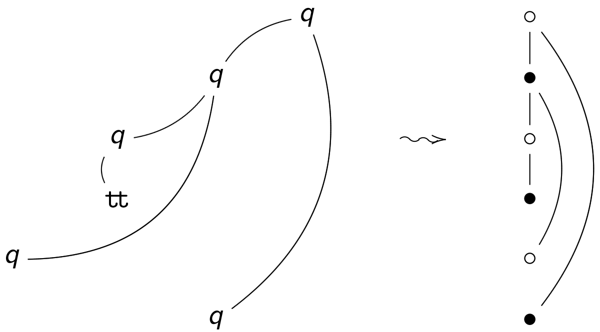


We lose notions of innocence and determinism...

The collapse

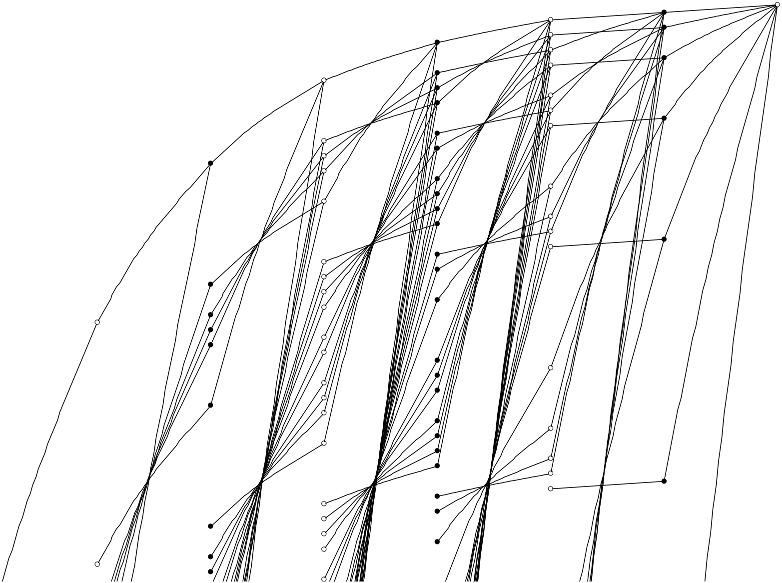
We consider only the **depth** of moves.

$$((\mathbb{B} \quad \times \quad \mathbb{B}) \Longrightarrow o) \Longrightarrow o$$



We lose notions of innocence and determinism...

The infinite interaction of $\delta\delta$



2. Pointer structures and λ-calculus

The unary λ-calculus

Collapsing amounts to restricting to the **unary** λ-calculus.

$$\begin{aligned} \underline{0} &= o \\ \underline{k+1} &= \underline{k} \to o \end{aligned}$$

Unary λ-calculus

$$\frac{\Gamma \vdash M : \underline{k+1} \qquad \Gamma \vdash N : \underline{k}}{\Gamma \vdash MN : \underline{0}}\ app \qquad \frac{}{\Gamma, x : \underline{k} \vdash x : \underline{k}}\ ax$$

$$\frac{\Gamma, x : \underline{k} \vdash M : \underline{0}}{\Gamma \vdash \lambda x.M : \underline{k+1}}\ lam \qquad \frac{\Gamma \vdash M : \underline{k} \qquad \Gamma \vdash N : \underline{k}}{\Gamma \vdash M + N : \underline{k}}\ plus$$

Syntactic collapse

Each simply typed term can be collapsed, along with its possible reductions.

$$
\begin{array}{rcl}
(\lambda x_1 \ldots x_n.M)^\star & = & \lambda x.(M[x/x_i])^\star \\
(M\ U_1\ \ldots\ U_p)^\star & = & M^\star\ (U_1^\star + \cdots + U_p^\star) \\
x^\star & = & x
\end{array}
$$

non-deterministic sum

$\simeq$

breaking innocence

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $\circ$ $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1.		$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $\circ$ $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1.	$\bullet$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1.	$\bullet$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
	$\vert$	
1.	$\bullet$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
	$\vert$	
2.	$\circ$	$(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

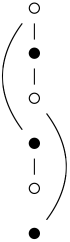
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary λ-calculus.

0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

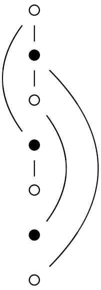
The PAM [DHR96] connects pointer structures with unary λ-calculus.



0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

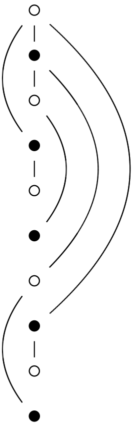
The PAM [DHR96] connects pointer structures with unary λ-calculus.



0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Pointer Abstract Machine

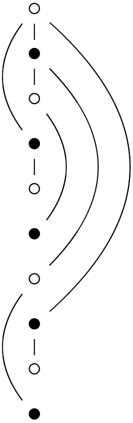
The PAM [DHR96] connects pointer structures with unary λ-calculus.



0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

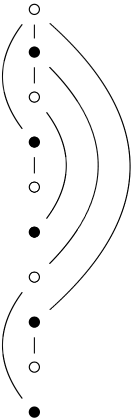
The PAM also relates to some states of the KAM.



0. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

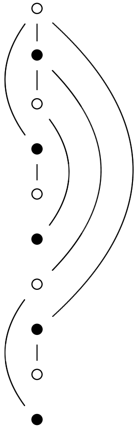
The PAM also relates to some states of the KAM.



0. $M_3 \star N_3$
1. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

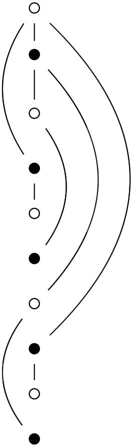
The PAM also relates to some states of the KAM.



0. $M_3 \star N_3$
1. $N_3 \star M_2^{f \mapsto N_3}$
2. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

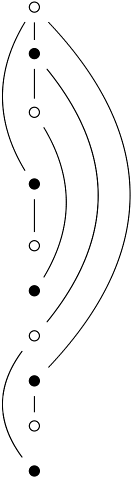
The PAM also relates to some states of the KAM.



0. $M_3 \star N_3$
1. $N_3 \star M_2^{f \mapsto N_3}$
2. $M_2^{f \mapsto N_3} \star N_2^{g \mapsto M_2^{f \mapsto N_3}}$
3. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

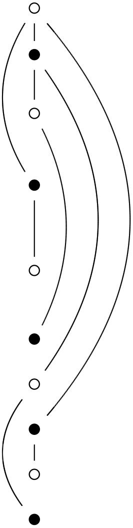
The PAM also relates to some states of the KAM.



0. $M_3 \star N_3$
1. $N_3 \star M_2^{f \mapsto N_3}$
2. $M_2^{f \mapsto N_3} \star N_2^{g \mapsto M_2^{f \mapsto N_3}}$
3. $N_3 \star M_1^{x \mapsto N_2^{g \mapsto M_2^{f \mapsto N_3}}}$
4. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

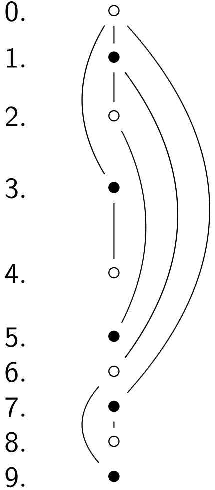
The PAM also relates to some states of the KAM.



0. $M_3 \star N_3$
1. $N_3 \star M_2^{f\mapsto N_3}$
2. $M_2^{f\mapsto N_3} \star N_2^{g\mapsto M_2^{f\mapsto N_3}}$
3. $N_3 \star M_1^{x\mapsto N_2^{g\mapsto M_2^{f\mapsto N_3}}}$
4. $M_1^{x\mapsto N_2^{g\mapsto M_2^{f\mapsto N_3}}} \star N_2^{g\mapsto M_1^{x\mapsto N_2^{g\mapsto M_2^{f\mapsto N_3}}}}$
5. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
6. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
7. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
8. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$
9. $(\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))$

The Krivine Abstract Machine

The PAM also relates to some states of the KAM.



$$M_3 \star N_3$$

$$N_3 \star M_2^{f \mapsto N_3}$$

$$M_2^{f \mapsto N_3} \star N_2^{g \mapsto M_2^{f \mapsto N_3}}$$

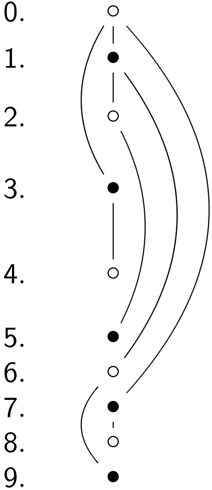
$$N_3 \star M_1^{x \mapsto N_2^{g \mapsto M_2^{f \mapsto N_3}}}$$

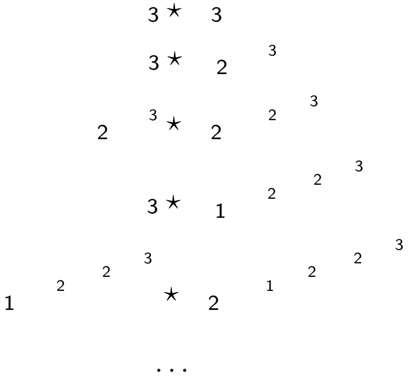
$$M_1^{x \mapsto N_2^{g \mapsto M_2^{f \mapsto N_3}}} \star N_2^{g \mapsto M_1^{x \mapsto N_2^{g \mapsto M_2^{f \mapsto N_3}}}}$$

$$\ldots$$

Agents

. . . which collapses to an operation on integers.

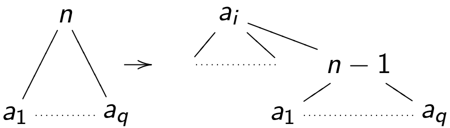




Agents

Definition

An **agent** *is a tree of integers.*



Theorem

1. *This reduction* **bisimulates** *visible pointer structures*
2. *This reduction always terminates (very short proof)*

Corollaries

P-views correspond to branches of cut-free terms

Definition

1. **Finite strategies** have a finite number of P-views
2. **Bounded strategies** have bounded P-views
3. **Nœtherian strategies** have well-founded P-views

Theorem (Compacity)

In case of innocent strategies in finite arenas, the three notions are equivalent.

Theorem

These three classes are stable by composition, and ensure preservation of totality.

III. Inductive and Coinductive Types

1. A logic with fixpoints : μLJ

$\mu LJ = LJ + \mu$-calculus

- Formulas are built by the following grammar:

$$S, T \ ::= \ S \Rightarrow T \mid S + T \mid S \times T \mid \mu X.T \mid \nu X.T \mid X \mid 1 \mid 0$$

- Bound type variables have to occur **positively**

Examples

$$\begin{array}{rcl}
\texttt{bool} & = & 1 + 1 \\
\texttt{nat} & = & \mu X.1 + X \\
\texttt{list}(A) & = & \mu X.1 + A \times X \\
\texttt{stream}(A) & = & \nu X.1 + A \times X \\
\texttt{tree} & = & \mu X.(\mu Y.1 + X \times Y) \\
\texttt{tree}(\texttt{stream}(\texttt{bool})) & = & \mu X.(\nu Z.1 + \texttt{bool} \times Z) \times (\mu Y.1 + X \times Y) \\
 & & \\
? & = & \mu X.((X \Rightarrow \texttt{bool}) \Rightarrow \texttt{bool})
\end{array}$$

Deduction rules

Deduction rules are LJ's rules, plus :

Fixpoints

$$\frac{\Gamma \vdash T[\mu X.T/X]}{\Gamma \vdash \mu X.T}\,\mu_r \qquad \frac{\Gamma, T[A/X] \vdash A}{\Gamma, \mu X.T \vdash A}\,\mu_l$$

$$\frac{\Gamma, T[\nu X.T/X] \vdash B}{\Gamma, \nu X.T \vdash B}\,\nu_l \qquad \frac{\Gamma, A \vdash T[A/X]}{\Gamma, A \vdash \nu X.T}\,\nu_r$$

Functors

$$\frac{\Gamma, A \vdash B}{\Gamma, T(A) \vdash T(B)}\,[T] \qquad \frac{\Gamma, B \vdash A}{\Gamma, N(A) \vdash N(B)}\,[N]$$

Regarded as a very explicit total programming language.

Cut reduction

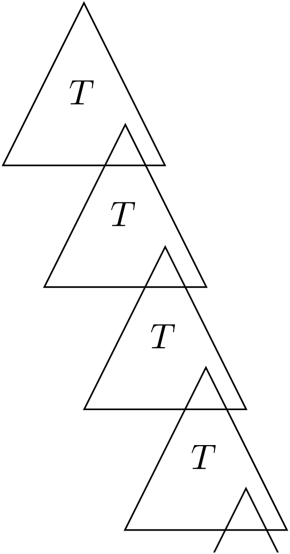
$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma \vdash T[\mu X.T/X]}}{\Gamma \vdash \mu X.T}\,\mu_r \qquad \dfrac{\dfrac{\pi_2}{T[A/X] \vdash A}}{\mu X.T \vdash A}\,\mu_l}{\Gamma \vdash A}\,\mathit{Cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma \vdash T[\mu X.T/X]} \qquad \dfrac{\dfrac{\dfrac{\pi_2}{T[A/X] \vdash A}}{\mu X.T \vdash A}\,\mu_l}{T[\mu X.T/X] \vdash T[A/X]}\,[T]}{\Gamma \vdash T[A/X]}\,\mathit{Cut} \qquad \dfrac{\pi_2}{T[A/X] \vdash A}}{\Gamma \vdash A}\,\mathit{Cut}$$

- We add unfolding reductions for functors
- Rules for ν are dual
- This is a 2-cell in the diagram of initial algebra !

2. Games and recursive types

McCusker's model of recursive types

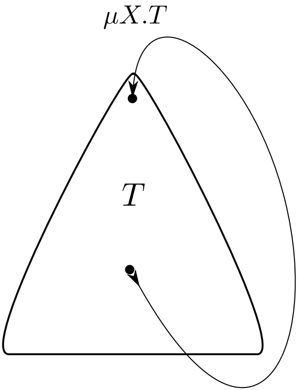
$\mu X.T$



The basic ingredients:

- Type with free variables are interpreted by **strong functors**
- Recursive types are obtained by infinite expansion of these functors

Our model of recursive types



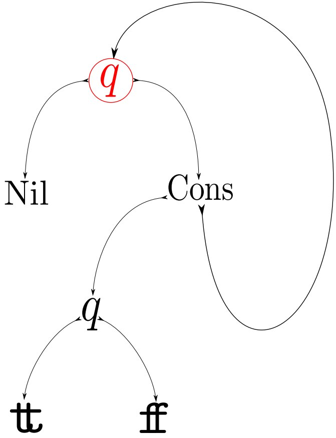
The basic ingredients:

- Type with free variables are interpreted by **open arenas**
- Open arenas automatically give rise to **strong functors**
- Recursive types are then obtained by a **loop construction**

The two resulting arenas are isomorphic by Laurent's theorem.

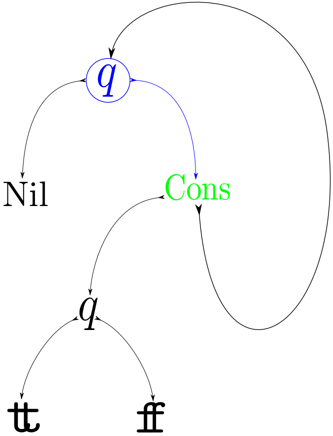
Example

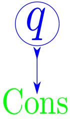
The cyclic arena of boolean lists



Example

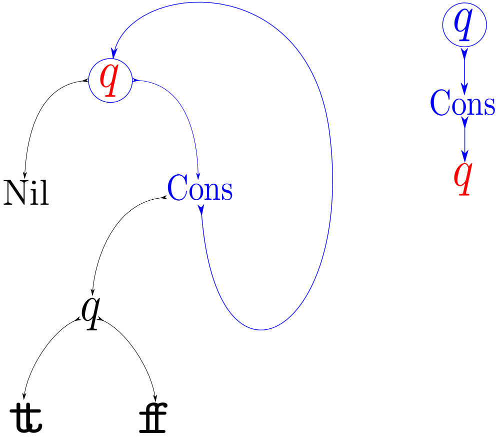
The cyclic arena of boolean lists





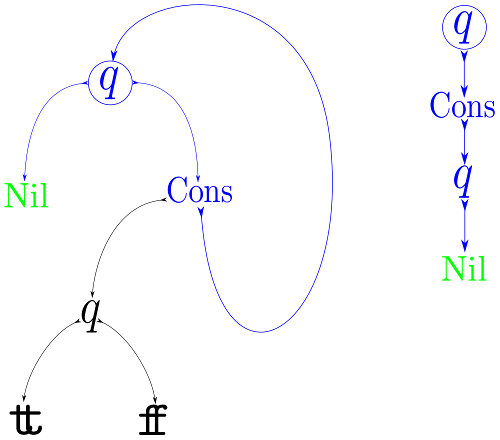
Example

The cyclic arena of boolean lists



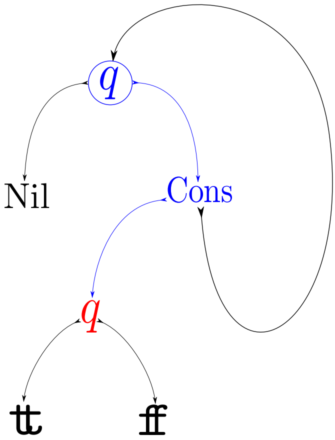
Example

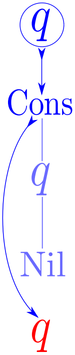
The cyclic arena of boolean lists



Example

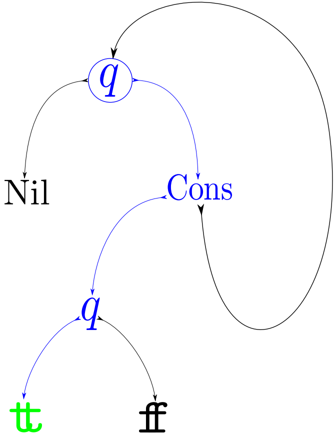
The cyclic arena of boolean lists

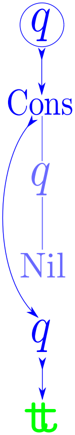




Example

The cyclic arena of boolean lists





(Co)inductive types = recursive types + totality

A complete model of induction and coinduction should consist in the following components:

- A model of recursive types
- A way to ensure totality
- Then, recursive types should split into inductive and coinductive types

We will use **winning conditions**, taking inspiration from **parity games**.

3. Winning conditions

Winning for arena games

Definition

We define **winning plays** *by:*

- *s is winning if each of its threads is winning*
- *s is winning on $A \times B$ or $A + B$ if it is winning on A and B*
- *s is winning on $A \Rightarrow B$ if (if it is winning on A then it is winning on B)*

A strategy σ is winning if all its infinite plays are winning.

Theorem

Total winning strategies are stable under composition.

Winning for arena games

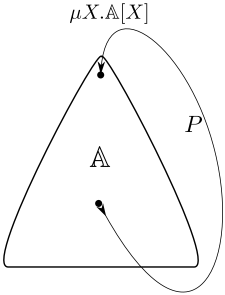
For the moment, we get the same category of games:

Theorem

On finite games (without loops), winning strategies are exactly **nœtherian** *strategies.*

How to extend winning to the loop construction ?

Least fixed point

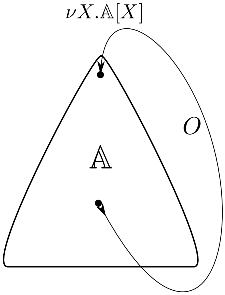


s is winning if and only if both these conditions are satisfied :

- There is $N \in \mathbb{N}$ such that no path of s crosses the external more than N times, **and**
- s is winning on A

This defines an **initial algebra** for $\mathbb{A}[X]$.

Greatest fixed point



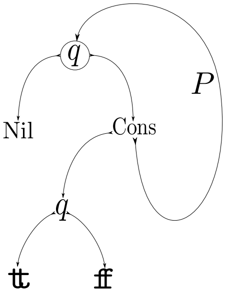
s is winning if and only if one of these conditions are satisfied :

- For any bound $N \in \mathbb{N}$, there is a path of s crossing the external loop more than N times, **or**
- s is winning on A

This defines an **terminal coalgebra** for $\mathbb{A}[X]$.

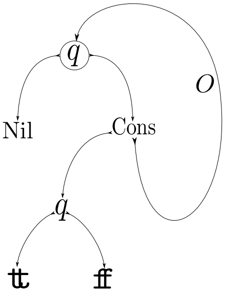
Example

$$List(\mathbb{B}) = \mu X.1 + (\mathbb{B} \times X)$$



$[1; 2; 3; 4; \ldots; n]$

$$Stream(\mathbb{B}) = \nu X.1 + (\mathbb{B} \times X)$$



$[1; 2; 3; 4; \ldots]$

Results

Theorem

Soundness and completeness:

1. *Winning games defines a sound model for μLJ*
2. *The model is complete with respect to an infinitary extension of μLJ*
3. *However, it is not faithful*

Theorem

Definability terminates on all formulas where

- *μ only appears in positive position*
- *ν only appears in negative position*

Thus on these formulas, cut is admissible.

IV. Conclusions & Perspectives

Achievements

- An account of totality in game semantics [CH09]
- A games model of inductive and coinductive types [Cla09a]
- Categories with strong types and μ-closed categories [Cla09b]
- Open functors

[CH09] Pierre Clairambault and Russ Harmer. Totality in arena games. Annals of Pure and Applied Logic, 2009.

[Cla09a] Pierre Clairambault. Least and greatest fixpoints in game semantics. In FOSSACS, pages 16–31, 2009.

[Cla09b] Pierre Clairambault. Least and greatest fixpoints in game semantics. In FICS, 2009.

Perspectives

Game semantics and pointer structures.

- Use agents to evaluate lengths of linear head reduction sequences
- Link agents with **revealed** game semantics

Fixed points.

- Prove that winning conditions on P-views are sufficient
- Improve and simplify the categorical model for fixed points (the "strengthening conjecture")
- Generalize open functors
- Try to achieve completeness
- Investigate isomorphisms

Dependent types...

Faithfullness

Consider the following programs:

```
let rec iter f n b =
   if n = 0 then b
   else iter f (n-1) (f b)

let rec iter' f n b =
   if n = 0 then b
   else not (iter' f (n-1) (f (not b)))
```

- $[\![\texttt{not (iter f n (not b))}]\!] = [\![\texttt{iter}' \texttt{ f n b}]\!]$
- But they cannot be convertible to each other.