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 \( \lambda \)-calculus

3. Inductive and Coinductive Types
   - Games model
   - Winning conditions

4. Conclusions & Perspectives
I. Introduction

1. Logical motivations
Tarski’s notion of truth.

⊤ is true

⊥ is false

\( A \land B \) is true \iff A is true and \( B \) is true

\( A \lor B \) is true \iff A is true or \( B \) is true

\( \neg A \) is true \iff A is false

Seems rather circular...
Two players are arguing over the validity of a formula $F$.

<table>
<thead>
<tr>
<th>Defender</th>
<th>Attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verifier</td>
<td>Falsifier</td>
</tr>
<tr>
<td>Eve</td>
<td>Artùlard</td>
</tr>
<tr>
<td>Player</td>
<td>Opponent</td>
</tr>
</tbody>
</table>

Player: “$F$ is true!”
Opponent: “$F$ is false!”
Rules

A ∧ B

• B

O attacks B

A ∨ B

•

O: “Do you defend A or B?”

P: “I defend A”

O attacks A

¬ A

•

A

O passes hand to P

P attacks A

A formula is “true” if Player has a total strategy.
Players can backtrack to an earlier position

$$\forall x A(x) \lor \exists x \neg A(x)$$

This introduces repetitions.
Repetitions may lead to non-termination

Consider the following play:

\[ \exists x \in \mathbb{N} \ A(x) \]

\[ \text{O attacks} \]
\[ P \text{ instanciates } x \text{ by 0} \]
\[ O \text{ refutes } A(0) \]
\[ P \text{ backtracks with } x = 1 \]
\[ \vdots \]

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

Together with a notion of composition of strategies.
Composition is defined by parallel interaction plus hiding.

\[ B \xrightarrow{\text{not}} B \quad \text{and} \quad B \xrightarrow{\text{not}} B \]

\[ q \]
Composition

Composition is defined by parallel interaction plus hiding.

\[
\begin{array}{c}
B \not \rightarrow B \\
q \quad q
\end{array}
\]
Composition is defined by parallel interaction plus hiding.
Composition

Composition is defined by parallel interaction plus hiding.

\[ B \xrightarrow{\text{not}} B \quad \text{and} \quad B \xrightarrow{\text{not}} B \]
Composition is defined by parallel interaction plus hiding.
Composition is defined by parallel interaction plus hiding.

\[ \begin{array}{c}
\text{B} \xrightarrow{\text{not}} \text{B} \\
q \xrightarrow{\text{q}} \text{f} \\
\end{array} \quad \begin{array}{c}
\text{B} \xrightarrow{\text{not}} \text{B} \\
q \xrightarrow{\text{q}} \text{t} \\
\end{array} \]
Composition is defined by parallel interaction plus hiding.

\[ \text{not,not} \]
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

Together with a notion of **composition** of strategies.
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad \text{and} \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

$A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C$
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

\[ A \xrightarrow{\sigma} B \quad B \xrightarrow{\tau} C \]
Composition may not preserve totality

$A \xrightarrow{\sigma} B$ \hspace{1cm} $B \xrightarrow{\tau} C$

\[ \ldots \quad \ldots \]
Composition may not preserve totality

\[ A \xrightarrow{\sigma \cdot \tau} C \]
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
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.

\[(\mathbb{B} \rightarrow \mathbb{B}) \rightarrow o\]

We lose notions of innocence and determinism...
The collapse

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

\[((B \times B) \Rightarrow o) \Rightarrow o\]

We lose notions of innocence and determinism...
The infinite interaction of $\delta\delta$
2. Pointer structures and λ-calculus
The unary $\lambda$-calculus

Collapsing amounts to restricting to the unary $\lambda$-calculus.

$$
\begin{align*}
0 & = \text{o} \\
\overline{k+1} & = \overline{k \to \text{o}}
\end{align*}
$$

---

**Unary $\lambda$-calculus**

\[
\begin{array}{c}
\Gamma \vdash M : k + 1 \\
\Gamma \vdash N : k
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash M \cdot N : 0 \\
\text{app}
\end{array}
\quad
\begin{array}{c}
\Gamma, x : k \vdash x : k \\
\text{ax}
\end{array}
\]

\[
\begin{array}{c}
\Gamma, x : k \vdash M : 0 \\
\text{lam}
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash M : k \\
\Gamma \vdash N : k
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash M + N : k \\
\text{plus}
\end{array}
\]
Syntactic collapse

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

\[(\lambda x_1 \ldots x_n . M)^* = \lambda x. (M[x/x_i])^*\]
\[(M \ U_1 \ \ldots \ U_p)^* = M^* \ (U_1^* + \cdots + U_p^*)\]
\[x^* = x\]

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. \( \odot \) \((\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))\)

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 \(\lambda\)-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 \( \lambda \)-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 \( \lambda \)-calculus.

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

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 \( \lambda \)-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. \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \(\lambda\)-calculus.

\begin{align*}
0. & \quad \circ \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
1. & \quad \bullet \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
2. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\end{align*}
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \( \lambda \)-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 \(\lambda\)-calculus.

\[
\begin{align*}
0. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g.g(\lambda z. g(\lambda w. z))) \\
1. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g.g(\lambda z. g(\lambda w. z))) \\
2. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g.g(\lambda z. g(\lambda w. z)))
\end{align*}
\]
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \( \lambda \)-calculus.

0. \( \bigcirc \) \[ (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \]

1. \( \bullet \) \[ (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \]

2. \( \bigcirc \) \[ (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \]

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 $\lambda$-calculus.

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

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

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

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

3. \( \bullet \) \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \( \lambda \)-calculus.

\[
\begin{align*}
0. & \quad \circ \quad (\lambda f. f (\lambda x. f (\lambda y. x))) (\lambda g. g (\lambda z. g (\lambda w. z))) \\
1. & \quad \bullet \quad (\lambda f. f (\lambda x. f (\lambda y. x))) (\lambda g. g (\lambda z. g (\lambda w. z))) \\
2. & \quad \circ \quad (\lambda f. f (\lambda x. f (\lambda y. x))) (\lambda g. g (\lambda z. g (\lambda w. z))) \\
3. & \quad \bullet \quad (\lambda f. f (\lambda x. f (\lambda y. x))) (\lambda g. g (\lambda z. g (\lambda w. z)))
\end{align*}
\]
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary $\lambda$-calculus.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.</td>
<td>$\circ$</td>
<td>$(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))$</td>
</tr>
<tr>
<td>1.</td>
<td>$\bullet$</td>
<td>$(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))$</td>
</tr>
<tr>
<td>2.</td>
<td>$\circ$</td>
<td>$(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))$</td>
</tr>
<tr>
<td>3.</td>
<td>$\bullet$</td>
<td>$(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))$</td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td>$(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))$</td>
</tr>
</tbody>
</table>
The Pointer Abstract Machine

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

0. \( \bullet \)  
\[
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\]

1. \( \circ \)  
\[
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\]

2. \( \circ \)  
\[
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\]

3. \( \bullet \)  
\[
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\]

4. \( \bullet \)  
\[
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\]
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \( \lambda \)-calculus.

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

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

2. \( \bullet \)  \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)

3. \( \bullet \)  \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)

4. \( \bullet \)  \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \(\lambda\)-calculus.

\[
\begin{align*}
0. & \quad \circ \\
1. & \quad \bullet \\
2. & \quad \circ \\
3. & \quad \bullet \\
4. & \quad \circ \\
\end{align*}
\]

\[
\begin{align*}
& (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
& (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
& (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
& (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
& (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\end{align*}
\]
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. \( \circ \) \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)

3. \( \bullet \) \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)

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

5. \( \bullet \) \( (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \)
The Pointer Abstract Machine

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

0.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
1.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
2.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
3.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
4.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
5.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
6.  (λf.f(λx.f(λy.x)))(λg.g(λz.g(λw.z)))
The Pointer Abstract Machine

The PAM [DHR96] connects pointer structures with unary \(\lambda\)-calculus.

\[
\begin{align*}
0. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
1. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
2. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
3. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
4. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
5. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
6. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
7. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
8. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z))) \\
9. & & (\lambda f.f(\lambda x.f(\lambda y.x)))(\lambda g.g(\lambda z.g(\lambda w.z)))
\end{align*}
\]
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.

\[ M_3 \ast N_3 \]

\[
\begin{align*}
0. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
1. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
2. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
3. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
4. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
5. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
6. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
7. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
8. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
9. & \quad (\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\end{align*}
\]
The PAM also relates to some states of the KAM.

0. $M_3 \star N_3$

1. $N_3 \star M_2^f \hookrightarrow N_3$

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 PAM also relates to some states of the KAM.

\[ M_3 \star N_3 \]
\[ N_3 \star M_2^{f \mapsto N_3} \]
\[ M_2^{f \mapsto N_3} \star N_2^g \mapsto M_2^{f \mapsto N_3} \]

\[
\begin{align*}
0. & \quad \circ \quad M_3 \star N_3 \\
1. & \quad \bullet \quad N_3 \star M_2^{f \mapsto N_3} \\
2. & \quad \circ \quad M_2^{f \mapsto N_3} \star N_2^g \mapsto M_2^{f \mapsto N_3} \\
3. & \quad \bullet \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
4. & \quad \circ \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
5. & \quad \bullet \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
6. & \quad \circ \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
7. & \quad \bullet \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
8. & \quad \circ \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z}))) \\
9. & \quad \bullet \quad (\lambda \mathbf{f}. \mathbf{f}(\lambda \mathbf{x}. \mathbf{f}(\lambda \mathbf{y}. \mathbf{x}))) (\lambda \mathbf{g}. \mathbf{g}(\lambda \mathbf{z}. \mathbf{g}(\lambda \mathbf{w}. \mathbf{z})))
\end{align*}
\]
The Krivine Abstract Machine

The PAM also relates to some states of the KAM.

\[
\begin{align*}
M_3 & \star N_3 \\
N_3 & \star M_2^f \mapsto N_3 \\
M_2^f \mapsto N_3 & \star N_2^g \mapsto M_2^f \mapsto N_3 \\
N_3 & \star M_1^x \mapsto N_2^g \mapsto M_2^f \mapsto N_3 \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z))) \\
(\lambda f. f(\lambda x. f(\lambda y. x)))(\lambda g. g(\lambda z. g(\lambda w. z)))
\end{align*}
\]
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. \[ M_2 f \mapsto N_3 \star N_2 \]

3. \[ N_3 \star M_1 x \mapsto N_2 g \mapsto M_1 f \mapsto N_3 \]

4. \[ M_1 x \mapsto N_2 g \mapsto M_2 f \mapsto N_3 \]

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 PAM also relates to some states of the KAM.

0. $M_3 \star N_3$

1. $N_3 \star M_2$

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

3. $N_3 \star M_1^{x \mapsto N_2}$

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

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

6. $N_2^{g \mapsto M_2^{f \mapsto N_3}}$

7. ...

8. ...

9. ...
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. **Noetherian 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: $\mu LJ$
\( \mu LJ = LJ + \mu \)-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**

\[
\begin{align*}
\text{bool} & = 1 + 1 \\
\text{nat} & = \mu X.1 + X \\
\text{list}(A) & = \mu X.1 + A \times X \\
\text{stream}(A) & = \nu X.1 + A \times X \\
\text{tree} & = \mu X.(\mu Y.1 + X \times Y) \\
\text{tree} (\text{stream}(\text{bool})) & = \mu X. (\nu Z.1 + \text{bool} \times Z) \times (\mu Y.1 + X \times Y) \\
? & = \mu X. ((X \Rightarrow \text{bool}) \Rightarrow \text{bool})
\end{align*}
\]
Deduction rules are $LJ$’s rules, plus:

**Fixpoints**

\[
\begin{align*}
\Gamma & \vdash T[\mu X.T/X] \\ 
\Gamma & \vdash \mu X.T \quad \mu_r \\
\Gamma, T[A/X] & \vdash A \\ 
\Gamma, \mu X.T & \vdash A \quad \mu_l \\
\Gamma, T[\nu X.T/X] & \vdash B \\ 
\Gamma, \nu X.T & \vdash B \quad \nu_l \\
\Gamma, A & \vdash T[A/X] \\ 
\Gamma, A & \vdash \nu X.T \quad \nu_r
\end{align*}
\]

**Functors**

\[
\begin{align*}
\Gamma, A & \vdash B \\ 
\Gamma, T(A) & \vdash T(B) \quad [T] \\
\Gamma, B & \vdash A \\ 
\Gamma, N(A) & \vdash N(B) \quad [N]
\end{align*}
\]

Regarded as a very explicit total programming language.
We add unfolding reductions for functors

Rules for \( \nu \) 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 two resulting arenas are isomorphic by Laurent’s theorem.
Example

The cyclic arena of boolean lists

\[
\text{Nil} \rightarrow q \rightarrow \text{Cons} \rightarrow q \rightarrow \text{true, false}
\]
Example

The cyclic arena of boolean lists
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
## 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 \( \sigma \) 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 noetherian strategies.*

How to extend winning to the loop construction?
Least fixed point

\[ \mu X. A[X] \]

\( s \) 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 \( A[X] \).
Greatest fixed point

\[ \nu X.A[X] \]

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 \( A[X] \).
Example

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

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

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

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

Theorem

*Soundness and completeness:*

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

Theorem

*Definability terminates on all formulas where*

- $\mu$ only appears in positive position
- $\nu$ 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 $\mu$-closed categories [Cla09b]
- Open functors


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...
Consider the following programs:

```ocaml
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)))
```

\[ \text{not (iter } f \ n \ (\text{not } b) \text{)} = \text{iter' } f \ n \ b \]

But they cannot be convertible to each other.