Pointer game semantics for polymorphism (work in progress)

Soren Lassen\textsuperscript{1} Paul Blain Levy\textsuperscript{2}

\textsuperscript{1}Google Sydney
\textsuperscript{2}University of Birmingham

March 21, 2010
1 No polymorphism
- CPS transform from call-by-push-value to calculus of no return
- Ultimate patterns
- The transition system
- Game semantics

2 Polymorphism
Call-by-push-value (with recursive types)

Value type \( A ::= UB \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A \)

Computation type \( B ::= FA \mid \prod_{i \in I} B_i \mid A \to B \mid X \mid \text{rec } X. B \)
Call-by-push-value (with recursive types)

value type \( A ::= UB \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A \)

computation type \( B ::= FA \mid \prod_{i \in I} B_i \mid A \rightarrow B \mid X \mid \text{rec } X. B \)

\( UB \) is the type of thunks of computations of type \( B \).
\( FA \) is the type of computations aiming to return a value of type \( A \).
Call-by-push-value (with recursive types)

value type \( A \) ::= \( UB \) | \( \sum_{i \in I} A_i \) | \( 1 \) | \( A \times A \) | \( X \) | \( \text{rec } X. A \)

computation type \( B \) ::= \( FA \) | \( \prod_{i \in I} B_i \) | \( A \rightarrow B \) | \( X \) | \( \text{rec } X. B \)

\( UB \) is is the type of thunks of computations of type \( B \).
\( FA \) is the type of computations aiming to return a value of type \( A \).

Value types denote dcpos, and computation types denote pointed dcpos. \( \llbracket FA \rrbracket \) is the lift of \( \llbracket A \rrbracket \), while \( \llbracket UB \rrbracket \) is just \( \llbracket B \rrbracket \).
Call-by-push-value (with recursive types)

value type \( A ::= UB | \sum_{i \in I} A_i | 1 | A \times A | X | \text{rec } X. A \)
computation type \( B ::= FA | \prod_{i \in I} B_i | A \to B | X | \text{rec } X. B \)

\( UB \) is the type of thunks of computations of type \( B \).
\( FA \) is the type of computations aiming to return a value of type \( A \).

Value types denote dcpo\(s\), and computation types denote pointed dcpo\(s\).
\( \llbracket FA \rrbracket \) is the lift of \( \llbracket A \rrbracket \), while \( \llbracket UB \rrbracket \) is just \( \llbracket B \rrbracket \).

\[
\begin{align*}
A \to_{\text{CBN}} B &= UA \to B \\
A +_{\text{CBN}} B &= F(UA + UB) \\
A \to_{\text{CBV}} B &= U(A \to FB)
\end{align*}
\]
CPS is a well-known transform that generates $\lambda$-terms in which functions never return. Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent’s LLP].
CPS is a well-known transform that generates \( \lambda \)-terms in which functions never return. Such terms can be arranged into a calculus \[\text{Lafont, Streicher, Reus (1993), cf. Laurent’s LLP}].

\[
A ::= \neg A \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A
\]

\( \neg A \) is the type of non-returning functions that take an argument of type \( A \).
CPS is a well-known transform that generates \( \lambda \)-terms in which functions never return. Such terms can be arranged into a calculus \([\text{Lafont, Streicher, Reus (1993), cf. Laurent's LLP}]\).

\[
A ::= \neg A \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A
\]

\( \neg A \) is the type of non-returning functions that take an argument of type \( A \).

\[
\text{value } V ::= x \mid \lambda x. M \mid \langle i, V \rangle \\
| \langle \rangle \mid \langle V, V \rangle \mid \text{fold } V
\]

\[
\text{non-returning command } M ::= V V \mid \text{match } V \text{ as } \{ \langle i, x \rangle. M \}_{i \in I} \\
| \text{match } V \text{ as } \langle \rangle. M \\
| \text{match } V \text{ as } \langle x, y \rangle. M \\
| \text{match } V \text{ as } \text{fold } x. M
\]
CPS is a well-known transform that generates $\lambda$-terms in which functions never return. Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent’s LLP].

$$A ::= \neg A \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A$$

$\neg A$ is the type of non-returning functions that take an argument of type $A$.

Value

$$V ::= x \mid \lambda x. M \mid \langle i, V \rangle \mid \langle \rangle \mid \langle V, V \rangle \mid \text{fold } V$$

Non-returning command

$$M ::= V V \mid \text{match } V \text{ as } \{ \langle i, x \rangle. M \}_{i \in I}$$

$$\mid \text{match } V \text{ as } \langle \rangle. M$$

$$\mid \text{match } V \text{ as } \langle x, y \rangle. M$$

$$\mid \text{match } V \text{ as } \text{fold } x. M$$

**Typing judgements** are $\Gamma \vdash^v V : A$ and $\Gamma \vdash^n M$. 
The judgement for types is $\overrightarrow{X} \vdash A$.

The judgement for values is $\Gamma \vdash^v V : A$.

The judgement for non-returning commands is $\Gamma \vdash^n M$.

\[
\begin{align*}
\Gamma, x : A & \vdash^n M \\
\Gamma & \vdash^v \lambda x.M : \neg A \\
\Gamma & \vdash^v V : \neg A \quad \Gamma & \vdash^v W : A \\
\Gamma & \vdash^n V \ W
\end{align*}
\]
The CPS transform on types is given by

- \( U \mapsto \neg \)
- \( F \mapsto \neg \)
- \( \sum_{i \in I} \mapsto \sum_{i \in I} \)
- \( \Pi_{i \in I} \mapsto \sum_{i \in I} \)
- \( 1 \mapsto 1 \)
- \( \times \mapsto \times \)
- \( \rightarrow \mapsto \rightarrow \times \)
- \( X \mapsto X \)
- \( \overline{X} \rightarrow X \)
- \( \text{rec } X. \mapsto \text{rec } X. \)
- \( \overline{\text{rec } X.} \mapsto \text{rec } X. \)

In game semantics this

- erases the distinction between questions and answers
- alternatively, makes all moves into questions

No bracketing condition is required for calculus of no return.
The C-machine (on commands $\Gamma |-^n M$)

$$(\lambda x. M) \ V \quad \leadsto \quad M[\ V / x]$$

match $\langle \hat{i}, V \rangle$ as $\{ \langle i, x \rangle. \ M_i \}_{i \in I}$ $\leadsto \quad M_{\hat{i}}[\ V / x]$

match $\langle \rangle$ as $\langle \rangle. \ M$ $\leadsto \quad M$

match $\langle V, V' \rangle$ as $\langle x, y \rangle. \ M$ $\leadsto \quad M[\ V / x, \ V' / y]$

match fold $V$ as fold $x. \ M$ $\leadsto \quad M[\ V / x]$
The C-machine (on commands $\Gamma \vdash^n M$)

$(\lambda x. M) \ V \quad \leadsto \quad M[V/x]$

match $\langle \hat{i}, V \rangle$ as $\{\langle i, x \rangle. M_i\}_{i \in I} \quad \leadsto \quad M_{\hat{i}}[V/x]$

match $\langle \rangle$ as $\langle \rangle. M \quad \leadsto \quad M$

match $\langle V, V' \rangle$ as $\langle x, y \rangle. M \quad \leadsto \quad M[V/x, V'/y]$

match fold $V$ as fold $x. M \quad \leadsto \quad M[V/x]$

Assume all identifiers are functions—i.e. have $\neg$ type.

Then the C-machine runs until it hits $zV$, where $(z : \neg A) \in \Gamma$. 
The C-machine (on commands $\Gamma \vdash^n M$)

\[
\begin{align*}
(\lambda x. M) V & \Rightarrow M[V/x] \\
\text{match } \langle \hat{i}, V \rangle \text{ as } \{\langle i, x \rangle. M_i \}_{i \in I} & \Rightarrow M_{\hat{i}}[V/x] \\
\text{match } \langle \rangle \text{ as } \langle \rangle. M & \Rightarrow M \\
\text{match } \langle V, V' \rangle \text{ as } \langle x, y \rangle. M & \Rightarrow M[V/x, V'/y] \\
\text{match fold } V \text{ as fold } x. M & \Rightarrow M[V/x]
\end{align*}
\]

Assume all identifiers are functions—i.e. have $\neg$ type.

Then the C-machine runs until it hits $z V$, where $(z : \neg A) \in \Gamma$.

What then?
A value $\Gamma \vdash^v V : A$, where all identifiers are functions, is uniquely of the form $p[W]$

$p$ is an ultimate pattern—it consists of tags

$p$ is the filling—it consists of functions.

**Example of ultimate pattern-matching**

\[
\langle i, \langle j, \langle \lambda x. M, y \rangle \rangle \rangle
\]

Ultimate pattern is $\langle i, \langle j, \langle -, - \rangle \rangle \rangle$

Filling is $\lambda x. M, y$
A value $\Gamma \vdash^v V : A$, where all identifiers are functions, is uniquely of the form $p[W]$

$p$ is an **ultimate pattern**—it consists of tags

$p$ is the **filling**—it consists of functions.

**Example of ultimate pattern-matching**

$$\langle i, \langle j, \langle \lambda x. M, y \rangle \rangle \rangle$$

Ultimate pattern is $\langle i, \langle j, \langle -, -, \rangle \rangle \rangle$

Filling is $\lambda x. M, y$

Proof by induction on $V$. 
Inductive definition:

\[ p ::= - \text{ (of type } \neg A) \mid \langle i, p \rangle \mid \langle \rangle \mid \langle p, p \rangle \mid \text{fold } p \]

ulpatt(A) is the set of ultimate patterns of type A.
Inductive definition:

\[ p ::= - \text{ (of type } \neg A) \mid \langle i, p \rangle \mid \langle \rangle \mid \langle p, p \rangle \mid \text{fold } p \]

ulpatt(A) is the set of ultimate patterns of type A.

An ultimate pattern \( p \) has a sequence of holes, each with \( \neg \) type.
Inductive definition:

\[ p ::= \neg (\text{of type } \neg A) \mid \langle i, p \rangle \mid \langle \rangle \mid \langle p, p \rangle \mid \text{fold } p \]

\text{ulpatt}(A) \text{ is the set of ultimate patterns of type } A.

An ultimate pattern \( p \) has a sequence of holes, each with \( \neg \) type.

We write \( H(p) \) for the sequence of these types.
How play proceeds [Jagadeesan, Pitcher, Riely 2007; Laird 2007; Lassen, Levy 2007]

Players pass functions to each other.

After some time, each player has some functions acquired from the other.
Players pass functions to each other.

After some time, each player has some functions acquired from the other. \( f : \neg A | | g \leftrightarrow V : \neg B \) indicates that

- Proponent has functions \( \overrightarrow{f} \) — they could be anything
- Opponent has functions \( \overrightarrow{g} \) — and \( g \) is actually bound to \( V \).
Nodes of the transition system

**Passive node (Opponent to play)**

A passive node takes the form

\[
\begin{align*}
    f : \neg A & \parallel g \rightarrow V : \neg B \\
\end{align*}
\]

**Active node (Proponent to play)**

An active node takes the form

\[
\begin{align*}
    f : \neg A & \parallel g \rightarrow V : \neg B \models^n M \\
\end{align*}
\]

where \( f : \neg A \models^n M \)
Nodes of the transition system

Passive node (Opponent to play)

A passive node takes the form

\[
\overrightarrow{f} : \neg A \parallel \overrightarrow{g} \mapsto V : \neg B
\]

Active node (Proponent to play)

An active node takes the form

\[
\overrightarrow{f} : \neg A \parallel \overrightarrow{g} \mapsto V : \neg B \vdash^{n} M
\]

where \( \overrightarrow{f} : \neg A \vdash^{n} M \)

We begin with an active node \( \overrightarrow{f} : \neg A \parallel \vdash^{n} M \).
Proponent move

Let $n$ be an active node $f : \neg A \parallel g \leftrightarrow V : \neg B \vdash^n M$.

- If $M \sim^* f p [\overrightarrow{W}]$, then $n$ outputs $f p$.
  
  $\begin{align*}
  n & \sim^* f p \\
  \overrightarrow{f : \neg A \parallel g \leftrightarrow V : \neg B, h \leftrightarrow \overrightarrow{W} : H(p)}
  \end{align*}$

- If $M \sim^\omega$ then $n \uparrow$
**Transitions**

### Proponent move

Let $n$ be an active node $\overrightarrow{f : \neg A} \parallel \overrightarrow{g \leftrightarrow V : \neg B} \vdash^n M$.

- If $M \leadsto^* \overrightarrow{fp[\overrightarrow{W}]}$, then $n$ outputs $\overrightarrow{fp}$.

\[
\begin{align*}
\overrightarrow{fp} & \\
n & \leadsto^* \overrightarrow{fp} & \\
\overrightarrow{f : \neg A} \parallel \overrightarrow{g \leftrightarrow V : \neg B}, & \overrightarrow{h \leftrightarrow W} & \vdash H(p)
\end{align*}
\]

- If $M \leadsto^\omega$ then $n \uparrow$

### Opponent move

Let $n$ be a passive node $\overrightarrow{f : \neg A} \parallel \overrightarrow{g \leftrightarrow V : \neg B}$. Then $n$ can input any $gq$.

\[
\begin{align*}
n : (gq) = & \\
n : (gq) & = \overrightarrow{f : \neg A, h : H(q)} \parallel \overrightarrow{g \leftrightarrow V : \neg B} \vdash^n Vq[\overrightarrow{h}]
\end{align*}
\]
Put general references and an error into the language.
Put general references and an error into the language. Nodes must then include Proponent’s private state.
Put general references and an error into the language.
Nodes must then include Proponent’s private state.

**Theorem**

Let $\Gamma \vdash^n M, M'$ be two commands.
Then $M$ and $M'$ have the same set of traces iff they are observationally equivalent.
Put general references and an error into the language.
Nodes must then include Proponent's private state.

**Theorem**

Let $\Gamma \vdash^n M, M'$ be two commands.
Then $M$ and $M'$ have the same set of traces iff they are observationally equivalent.

These trace sets can be made into a denotational game semantics.
Put general references and an error into the language.
Nodes must then include Proponent’s private state.

**Theorem**

Let $\Gamma \vdash^n M, M'$ be two commands. Then $M$ and $M'$ have the same set of traces iff they are observationally equivalent.

These trace sets can be made into a denotational game semantics.
It is the arena model of Abramsky, Honda and McCusker.
An arena is a forest.
An arena is a forest.

**Semantics of function contexts**

Any function context $\Gamma$ gives an arena $[[\Gamma]]$. Each $xp$ is a root, where $(x : \neg A) \in \Gamma$ and $p \in \text{ulpatt}(A)$. Under the root $xp$, put the arena $[[H(p)]]$. 
An arena is a forest.

**Semantics of function contexts**

Any function context $\Gamma$ gives an arena $[\Gamma]$. Each $x p$ is a root, where $(x : \neg A) \in \Gamma$ and $p \in \text{ulpatt}(A)$. Under the root $x p$, put the arena $[H(p)]$.

**Semantics of types**

Any (closed) type $A$ gives a family of arenas $\{[H(p)]\}_{p \in \text{ulpatt}(A)}$. 
Domains of strategies

A pair $f : \neg A \parallel g : \neg B$ represents a pair of arenas $R \parallel S$. 
A pair $\overrightarrow{f}: \neg A \parallel \neg B$ represents a pair of arenas $R \parallel S$.

We give domains $\text{Ostrat}(R \parallel S)$ and $\text{Pstrat}(R \parallel S)$ by equations.

### The domain equations

\[
\begin{align*}
\text{Pstrat}(R \parallel S) &= \left( \sum_{a \in \text{rt } R} \text{Ostrat}(R \parallel S \cup R_a) \right) \perp \\
\text{Ostrat}(R \parallel S) &= \prod_{b \in \text{rt } S} \text{Pstrat}(R \cup b \parallel S)
\end{align*}
\]

Solving these gives the domain of strategies with justification pointers.
Domains of strategies

A pair $f: \neg A \parallel g: \neg B$ represents a pair of arenas $R \parallel S$.

We give domains $\text{Ostrat}(R \parallel S)$ and $\text{Pstrat}(R \parallel S)$ by equations.

The domain equations

\[
\begin{align*}
\text{Pstrat}(R \parallel S) &= \left( \sum_{a \in \text{rt } R} \text{Ostrat}(R \parallel S \cup R_a) \right) \perp \\
\text{Ostrat}(R \parallel S) &= \prod_{b \in \text{rt } S} \text{Pstrat}(R \cup b \parallel S)
\end{align*}
\]

Solving these gives the domain of strategies with justification pointers.

For a command $\Gamma \vdash^n M$, the trace set is $[M] \in \text{Pstrat}([\Gamma] \parallel \emptyset)$.
Compositionality is a theorem, not a definition.
Compositionality is a theorem, not a definition.

**Compositionality for terms**

Example: $[VW] = \psi([V], [W])$
Compositionality

Compositionality is a theorem, not a definition.

**Compositionality for terms**

Example: $[VW] = \psi([V], [W])$

**Compositionality for types**

Example: $[\neg A] \cong \theta([A])$
Two categories of arenas:

- in $C$, morphisms are strategies that are OP-visible
- in $D$, morphisms are forest isomorphisms.
Two categories of arenas:

- in $\mathcal{C}$, morphisms are strategies that are OP-visible
- in $\mathcal{D}$, morphisms are forest isomorphisms.

We have a functor $J : \mathcal{D} \rightarrow \mathcal{C}$. 

Theorem [Laurent] $J$ is fully faithful. Conjectured to also hold without the visibility constraint.
Full abstraction for types

Two categories of arenas:
- in $\mathcal{C}$, morphisms are strategies that are OP-visible
- in $\mathcal{D}$, morphisms are forest isomorphisms.

We have a functor $J : \mathcal{D} \to \mathcal{C}$.

**Theorem [Laurent]**

$J$ is fully faithful.

Conjectured to also hold without the visibility constraint.
Adding Polymorphism to Call-By-Push-Value

\[
A ::= \text{UB} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A \mid \sum X. A \mid \sum X. A
\]

\[
B ::= FA \mid \prod_{i \in I} B_i \mid A \rightarrow B \mid X \mid \text{rec } X. B \mid \prod X. B \mid \prod X. B
\]
Adding Polymorphism to Calculus of No Return

\[ A ::= \neg A \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A \mid \sum X. A \]
Adding Polymorphism to Calculus of No Return

\[ A ::= \neg A \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid X \mid \text{rec } X. A \mid \sum X. A \]

value \( V ::= \) x \mid \lambda x. M \mid \langle i, V \rangle

\mid \langle \rangle \mid \langle V, V \rangle \mid \text{fold } V \mid \langle A, V \rangle

non-returning command \( M ::= \) V V \mid \text{match } V \text{ as } \{ \langle i, x \rangle. M \}_{i \in I}

\mid \text{match } V \text{ as } \langle \rangle. M

\mid \text{match } V \text{ as } \langle x, y \rangle. M

\mid \text{match } V \text{ as fold } x. M

\mid \text{match } V \text{ as } \langle X, y \rangle. M
A value that I pass to you contains

tags
functions

ultimate pattern
filling
Ultimate patterns and fillings

A value that I pass to you contains

- tags
- functions
- types
- opaque values

ultimate pattern filling
ingning
A value that I pass to you contains

tags
functions
types
opaque values
of type I’ve received from you
of type I’ve sent to you

---

We define ultimate patterns $\text{ulpatt}(\cdots)$ by the grammar

$$p ::= \neg A | \langle i, p \rangle | \langle \rangle | \langle p, p \rangle | \text{fold } p | \langle \neg, p \rangle | \neg (\text{of type } Y) | \neg : x$$
Ultimate patterns and fillings

A value that I pass to you contains

- tags
- ultimate pattern
- functions
- filling
- types
- filling
- opaque values
- of type I’ve received from you
- ultimate pattern
- of type I’ve sent to you
- filling

We define ultimate patterns

\[
\text{ulpatt}(\overrightarrow{X}, \overrightarrow{x} : \Xi || \overrightarrow{Y} \vdash D)
\]

by the grammar

\[
p ::= \neg \ (\text{of type } \neg A) \mid \langle i, p \rangle \mid \langle \rangle \mid \langle p, p \rangle \mid \text{fold } p \\
\mid \langle \neg, p \rangle \mid \neg \ (\text{of type } \neg Y) \mid \neg : x
\]
Ultimate pattern matching theorem

Given a type $\overrightarrow{X}, \overrightarrow{Y} \vdash B$ and types $\overrightarrow{Y} \leftrightarrow B$,
and a value $\overrightarrow{X}, \overrightarrow{x} : \Xi, \overrightarrow{f} : \neg A[B/Y] \vdash^v V : D[B/Y]$ where each $\Xi$ is drawn from $\overrightarrow{X}$
Ultimate pattern matching theorem

Given a type $\overrightarrow{X}, \overrightarrow{Y} \vdash B$ and types $\overrightarrow{Y} \mapsto B$,

and a value $\overrightarrow{X}, \overrightarrow{x} : \overrightarrow{\Xi}, \overrightarrow{f} : \neg A[\overrightarrow{B}/\overrightarrow{Y}] \vdash^v V : D[\overrightarrow{B}/\overrightarrow{Y}]$

where each $\overrightarrow{\Xi}$ is drawn from $\overrightarrow{X}$

$V$ is uniquely of the form $p[\overrightarrow{B}/\overrightarrow{Y}, w]$

for ultimate pattern $p$ on $\overrightarrow{X}, \overrightarrow{x} : \overrightarrow{\Xi} \parallel \overrightarrow{Y} \vdash D$

and filling $w$. 
Given a type $\overrightarrow{X}, \overrightarrow{Y} \vdash B$ and types $Y \mapsto \overrightarrow{B}$,

and a value $\overrightarrow{X}, \overrightarrow{x : \Xi}, f : \neg A[B/Y] \vdash^v V : D[B/Y]$

where each $\Xi$ is drawn from $\overrightarrow{X}$

$V$ is uniquely of the form $p[B/Y, w]$ for ultimate pattern $p$ on $\overrightarrow{X}, \overrightarrow{x : \Xi||Y} \vdash D$

and filling $w$.

Proof by induction on $V$. 
Transition system [Lassen, Levy 2008]

Passive node (Opponent to play)

A passive node takes the form

\[ \overrightarrow{X}, x: \Xi, f: \neg A \parallel \overrightarrow{Y}, y: \Upsilon, g \mapsto V: \neg B \]

with each \( \Xi \) drawn from \( \overrightarrow{X} \) and each \( \Upsilon \) drawn from \( \overrightarrow{Y} \)

Active node (Proponent to play)

An active node takes the form

\[ \overrightarrow{X}, x: \Xi, f: \neg A \parallel \overrightarrow{Y}, y: \Upsilon, g \mapsto V: \neg B \vdash^n M \]
Full Abstraction

Put general references and an error into the language. Nodes must then include Proponent’s private state.

**Conjecture**

Let $\Gamma \triangleright^n M, M'$ be two commands.
Then $M$ and $M'$ have the same set of traces iff they are observationally equivalent.
Put general references and an error into the language. Nodes must then include Proponent’s private state.

**Conjecture**

Let $\Gamma \vdash^n M, M'$ be two commands. Then $M$ and $M'$ have the same set of traces iff they are observationally equivalent.

Can we turn these trace sets into a denotational game semantics?
De Lataillade gave a complete list of isomorphisms that hold up to $\beta\eta$-equality.

How can we generalize Laurent’s result to the polymorphic setting?
De Lataillade gave a complete list of isomorphisms that hold up to \( \beta\eta \)-equality.

But up to observational equivalence, there are many more.

**Example isomorphism**

For a type \( A[\_, \_] \) we have

\[
\sum X.(X^n \times A[X, X^m]) \cong \sum X.A[m \times X + n, X]
\]
De Lataillade gave a complete list of isomorphisms that hold up to $\beta\eta$-equality.

But up to observational equivalence, there are many more.

**Example isomorphism**

For a type $A[−, +]$ we have

$$\sum X. (X^n \times A[X, X^m]) \cong \sum X. A[m \times X + n, X]$$

How can we generalize Laurent’s result to the polymorphic setting?
Related work

- Hughes
- Murawski, Ong: affine polymorphism
- Abramsky, Jagadeesan
- de Lataillade
- polymorphic $\pi$-calculus [Pierce, Sangiorgi; Berger, Honda, Yoshida]

Also recent work by Laird.