Pointer game semantics for polymorphism (work in progress)

Soren Lassen[1] Paul Blain Levy[2]

[1]Google Sydney

[2]University of Birmingham

March 21, 2010

Outline

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)

$$\begin{array}{rrl} \text{value type} & A ::= & U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \\ \text{computation type} & \underline{B} ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{X}} \mid \mathtt{rec}\ \underline{\mathtt{X}}.\ \underline{B} \end{array}$$

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

$$\begin{array}{rrl} \text{value type} & A ::= & U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \\ \text{computation type} & \underline{B} ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{X}} \mid \mathtt{rec}\ \underline{\mathtt{X}}.\ \underline{B} \end{array}$$

$U\underline{B}$ is is the type of thunks of computations of type $\underline{B}$.
FA is the type of computations aiming to return a value of type A.

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

$$\begin{array}{rrl} \text{value type} & A ::= & U\underline{B} \mid \sum_{i \in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \\ \text{computation type} & \underline{B} ::= & FA \mid \prod_{i \in I} \underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{X}} \mid \mathtt{rec}\ \underline{\mathtt{X}}.\ \underline{B} \end{array}$$

$U\underline{B}$ is is the type of thunks of computations of type $\underline{B}$.
FA is the type of computations aiming to return a value of type A.

Value types denote dcpos, and computation types denote pointed dcpos.
$[\![FA]\!]$ is the lift of $[\![A]\!]$, while $[\![U\underline{B}]\!]$ is just $[\![\underline{B}]\!]$.

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

$$\begin{array}{rrl} \text{value type} & A ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \\ \text{computation type} & \underline{B} ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{X}} \mid \mathtt{rec}\ \underline{\mathtt{X}}.\ \underline{B} \end{array}$$

$U\underline{B}$ is is the type of thunks of computations of type $\underline{B}$.
FA is the type of computations aiming to return a value of type A.

Value types denote dcpos, and computation types denote pointed dcpos.
$[\![FA]\!]$ is the lift of $[\![A]\!]$, while $[\![U\underline{B}]\!]$ is just $[\![\underline{B}]\!]$.

$$\begin{array}{rcl} \underline{A} \to_{\mathbf{CBN}} \underline{B} & = & U\underline{A} \to \underline{B} \\ \underline{A} +_{\mathbf{CBN}} \underline{B} & = & F(U\underline{A} + U\underline{B}) \\ A \to_{\mathbf{CBV}} B & = & U(A \to FB) \end{array}$$

Calculus of no return (with recursive types)

CPS is a well-known transform that generates λ-terms in which functions never return.

Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent's LLP].

Calculus of no return (with recursive types)

CPS is a well-known transform that generates λ-terms in which functions never return.

Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent's LLP].

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

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

Calculus of no return (with recursive types)

CPS is a well-known transform that generates λ-terms in which functions never return.
Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent's LLP].

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

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

$$\begin{array}{lrl}
\text{value} & V ::= & \mathtt{x} \mid \lambda \mathtt{x}.M \mid \langle i, V \rangle \\
 & & \mid \langle\rangle \mid \langle V, V \rangle \mid \mathtt{fold}\ V \\
\text{non-returning command} & M ::= & V\ V \mid \mathtt{match}\ V\ \mathtt{as}\ \{\langle i, x \rangle.\ M\}_{i \in I} \\
 & & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle\rangle.\ M \\
 & & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\ M \\
 & & \mid \mathtt{match}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M
\end{array}$$

Calculus of no return (with recursive types)

CPS is a well-known transform that generates λ-terms in which functions never return.
Such terms can be arranged into a calculus [Lafont, Streicher, Reus (1993), cf. Laurent's LLP].

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

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

$$\begin{array}{rrl}
\text{value} & V ::= & \mathtt{x} \mid \lambda \mathtt{x}.M \mid \langle i, V \rangle \\
& & \mid \langle\rangle \mid \langle V, V \rangle \mid \mathtt{fold}\ V \\
\text{non-returning command} & M ::= & V\ V \mid \mathtt{match}\ V\ \mathtt{as}\ \{\langle i, x \rangle.\ M\}_{i \in I} \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle\rangle.\ M \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\ M \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M
\end{array}$$

Typing judgements are $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{n}} M$.

Typing the Calculus of No Return

The judgement for types is $\overrightarrow{\mathtt{X}} \vdash A$.

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

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

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{v}} \lambda\mathtt{x}.M : \neg A}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \neg A \quad \Gamma \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{n}} V\ W}$$

The CPS transform

The CPS transform on types is given by

$$
\begin{array}{rcl}
U & \mapsto & \neg \\
\sum_{i\in I} & \mapsto & \sum_{i\in I} \\
1 & \mapsto & 1 \\
\times & \mapsto & \times \\
\mathtt{X} & \mapsto & \mathtt{X} \\
\mathtt{rec}\ \mathtt{X}. & \mapsto & \mathtt{rec}\ \mathtt{X}.
\end{array}
\qquad
\begin{array}{rcl}
F & \mapsto & \neg \\
\prod_{i\in I} & \mapsto & \sum_{i\in I} \\
 & & \\
\to & \mapsto & \times \\
\underline{\mathtt{X}} & \to & \mathtt{X} \\
\mathtt{rec}\ \underline{\mathtt{X}}. & \mapsto & \mathtt{rec}\ \mathtt{X}.
\end{array}
$$

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 $\Gamma \vdash^{\mathsf{n}} M$)

$$
\begin{array}{lcl}
(\lambda \mathrm{x}.M)\ V & \rightsquigarrow & M[V/\mathrm{x}] \\
\mathtt{match}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathrm{x}\rangle.\ M_i\}_{i \in I} & \rightsquigarrow & M_{\hat{\imath}}[V/\mathrm{x}] \\
\mathtt{match}\ \langle\rangle\ \mathtt{as}\ \langle\rangle.\ M & \rightsquigarrow & M \\
\mathtt{match}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathrm{x}, \mathrm{y}\rangle.\ M & \rightsquigarrow & M[V/\mathrm{x}, V'/\mathrm{y}] \\
\mathtt{match}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M & \rightsquigarrow & M[V/\mathrm{x}]
\end{array}
$$

The C-machine (on commands $\Gamma \vdash^{\mathsf{n}} M$)

$$
\begin{array}{lcl}
(\lambda \mathrm{x}.M)\ V & \rightsquigarrow & M[V/\mathrm{x}] \\
\mathtt{match}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i, \mathrm{x}\rangle.\ M_i\}_{i \in I} & \rightsquigarrow & M_{\hat{\imath}}[V/\mathrm{x}] \\
\mathtt{match}\ \langle\rangle\ \mathtt{as}\ \langle\rangle.\ M & \rightsquigarrow & M \\
\mathtt{match}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathrm{x}, \mathrm{y}\rangle.\ M & \rightsquigarrow & M[V/\mathrm{x}, V'/\mathrm{y}] \\
\mathtt{match}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathrm{x}.\ M & \rightsquigarrow & M[V/\mathrm{x}]
\end{array}
$$

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

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

The C-machine (on commands $\Gamma \vdash^{\mathsf{n}} M$)

$$
\begin{array}{lcl}
(\lambda \mathrm{x}.M)\ V & \rightsquigarrow & M[V/\mathrm{x}] \\
\texttt{match}\ \langle \hat{\imath}, V\rangle\ \texttt{as}\ \{\langle i, \mathrm{x}\rangle.\ M_i\}_{i \in I} & \rightsquigarrow & M_{\hat{\imath}}[V/\mathrm{x}] \\
\texttt{match}\ \langle\rangle\ \texttt{as}\ \langle\rangle.\ M & \rightsquigarrow & M \\
\texttt{match}\ \langle V, V'\rangle\ \texttt{as}\ \langle \mathrm{x}, \mathrm{y}\rangle.\ M & \rightsquigarrow & M[V/\mathrm{x}, V'/\mathrm{y}] \\
\texttt{match fold}\ V\ \texttt{as fold x}.\ M & \rightsquigarrow & M[V/\mathrm{x}]
\end{array}
$$

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

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

What then?

Ultimate pattern matching theorem

A value $\Gamma \vdash^{\mathsf{v}} V : A$, 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

$\langle i, \langle j, \langle \lambda \mathrm{x}.M, \mathrm{y} \rangle\rangle\rangle$
Ultimate pattern is $\langle i, \langle j, \langle -, - \rangle\rangle\rangle$
Filling is $\lambda \mathrm{x}.M, \mathrm{y}$

Ultimate pattern matching theorem

A value $\Gamma \vdash^{\mathsf{v}} V : A$, 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

$\langle i, \langle j, \langle \lambda \mathrm{x}.M, \mathrm{y} \rangle \rangle \rangle$
Ultimate pattern is $\langle i, \langle j, \langle -, - \rangle \rangle \rangle$
Filling is $\lambda \mathrm{x}.M, \mathrm{y}$

Proof by induction on V.

Ultimate patterns

Inductive definition:

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

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

Ultimate patterns

Inductive definition:

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

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

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

Ultimate patterns

Inductive definition:

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

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

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

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.

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.

$\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B}$ indicates that

- Proponent has functions $\overrightarrow{\mathtt{f}}$—they could be anything
- Opponent has functions $\overrightarrow{\mathrm{g}}$—and g is actually bound to V.

Nodes of the transition system

Passive node (Opponent to play)

A passive node takes the form

$$\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B}$$

Active node (Proponent to play)

An active node takes the form

$$\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B} \vdash^{\mathsf{n}} M$$

where $\overrightarrow{\mathtt{f} : \neg A} \vdash^{\mathsf{n}} M$

Nodes of the transition system

Passive node (Opponent to play)

A passive node takes the form

$$\overrightarrow{\mathtt{f} : \neg A} \| \overrightarrow{\mathrm{g} \mapsto V : \neg B}$$

Active node (Proponent to play)

An active node takes the form

$$\overrightarrow{\mathtt{f} : \neg A} \| \overrightarrow{\mathrm{g} \mapsto V : \neg B} \vdash^{\mathsf{n}} M$$

where $\overrightarrow{\mathtt{f} : \neg A} \vdash^{\mathsf{n}} M$

We begin with an active node $\overrightarrow{\mathtt{f} : \neg A} \| \vdash^{\mathsf{n}} M$.

Transitions

Proponent move

Let n be an active node $\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B} \vdash^{\mathsf{n}} M$.

- If $M \rightsquigarrow^* \mathtt{f}\,{\color{red}p}[{\color{blue}\overrightarrow{W}}]$, then n outputs $\mathtt{f}\,p$.

$$n \overset{\mathtt{f}p}{\rightsquigarrow} \overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B}{\color{green}, \overrightarrow{h \mapsto W} : H(p)}$$

- If $M \rightsquigarrow^{\omega}$ then $n \Uparrow$

Transitions

Proponent move

Let n be an active node $\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B} \vdash^{\mathrm{n}} M$.

- If $M \rightsquigarrow^* \mathtt{f}\,p[\overrightarrow{W}]$, then n outputs $\mathtt{f}\,p$.

$$n \overset{\mathtt{f}p}{\rightsquigarrow} \overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B}, \overrightarrow{h \mapsto W} : H(p)$$

- If $M \rightsquigarrow^\omega$ then $n \Uparrow$

Opponent move

Let n be a possive node $\overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathrm{g} \mapsto V : \neg B}$. Then n can input any $\mathrm{g}q$.

$$n : (\mathrm{g}q) = \overrightarrow{\mathtt{f} : \neg A}, \overrightarrow{\mathtt{h}} : H(q) || \overrightarrow{\mathrm{g} \mapsto V : \neg B} \vdash^{\mathrm{n}} V q[\overrightarrow{\mathtt{h}}]$$

Full Abstraction

Put general references and an error into the language.

Full Abstraction

Put general references and an error into the language.

Nodes must then include Proponent's private state.

Full Abstraction

Put general references and an error into the language.

Nodes must then include Proponent's private state.

Theorem

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

Full Abstraction

Put general references and an error into the language.

Nodes must then include Proponent's private state.

Theorem

Let $\Gamma \vdash^{\mathsf{n}} M, M'$ 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.

Full Abstraction

Put general references and an error into the language.

Nodes must then include Proponent's private state.

Theorem

Let $\Gamma \vdash^{\mathsf{n}} M, M'$ 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.

Arenas

An arena is a forest.

Arenas

An arena is a forest.

Semantics of function contexts

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

Arenas

An arena is a forest.

Semantics of function contexts

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

Semantics of types

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

Domains of strategies

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

Domains of strategies

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

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

The domain equations

$$\begin{array}{rcl} \mathsf{Pstrat}(R||S) & = & (\sum_{a \in \mathsf{rt}\, R} \mathsf{Ostrat}(R||S \uplus R_a))_\perp \\ \mathsf{Ostrat}(R||S) & = & \prod_{b \in \mathsf{rt}\, S} \mathsf{Pstrat}(R \uplus b||S) \end{array}$$

Solving these gives the domain of strategies with justification pointers.

Domains of strategies

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

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

The domain equations

$$\begin{array}{rcl} \mathsf{Pstrat}(R||S) & = & (\sum_{a \in \mathsf{rt}\, R} \mathsf{Ostrat}(R||S \uplus R_a))_\perp \\ \mathsf{Ostrat}(R||S) & = & \prod_{b \in \mathsf{rt}\, S} \mathsf{Pstrat}(R \uplus b||S) \end{array}$$

Solving these gives the domain of strategies with justification pointers.

For a command $\Gamma \vdash^{\mathsf{n}} M$, the trace set is $[M] \in \mathsf{Pstrat}([\Gamma]||\emptyset)$.

Compositionality

Compositionality is a theorem, not a definition.

Compositionality

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: $[\neg A] \cong \theta([A])$

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.

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 $J : \mathcal{D} \longrightarrow \mathcal{C}$.

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 $J : \mathcal{D} \longrightarrow \mathcal{C}$.

Theorem [Laurent]

J is fully faithful.

Conjectured to also hold without the visibility constraint.

Adding Polymorphism to Call-By-Push-Value

$$
\begin{array}{lcl}
A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mathtt{rec}\ \mathtt{X}.\ A \mid \sum \mathtt{X}.\ A \mid \sum \underline{\mathtt{X}}.\ A \\
\underline{B} & ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{X}} \mid \mathtt{rec}\ \underline{\mathtt{X}}.\ \underline{B} \mid \prod \mathtt{X}.\underline{B} \mid \prod \underline{\mathtt{X}}.\ \underline{B}
\end{array}
$$

Adding Polymorphism to Calculus of No Return

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

Adding Polymorphism to Calculus of No Return

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

$$\begin{array}{rrl}
\text{value} & V ::= & \mathtt{x} \mid \lambda \mathtt{x}.M \mid \langle i, V \rangle \\
& & \mid \langle\rangle \mid \langle V, V \rangle \mid \mathtt{fold}\ V \mid \langle A, V \rangle \\
\text{non-returning command} & M ::= & V\ V \mid \mathtt{match}\ V\ \mathtt{as}\ \{\langle i, x \rangle.\ M\}_{i \in I} \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle\rangle.\ M \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle.\ M \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M \\
& & \mid \mathtt{match}\ V\ \mathtt{as}\ \langle \mathtt{X}, \mathtt{y} \rangle.\ M
\end{array}$$

Ultimate patterns and fillings

A value that I pass to you contains

tags	ultimate pattern
functions	filling

Ultimate patterns and fillings

A value that I pass to you contains

tags	ultimate pattern
functions	filling
types	filling
opaque values	

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

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

$$\mathrm{ulpatt}(\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x} : \Xi} || \overrightarrow{\mathtt{Y}} \vdash \mathrm{D})$$

by the grammar

$$\begin{array}{rl} p ::= & - \text{ (of type } \neg A) \mid \langle i, p \rangle \mid \langle \rangle \mid \langle p, p \rangle \mid \mathtt{fold}\ p \\ & \mid \langle -, p \rangle \mid - \text{ (of type } \mathtt{Y}) \mid - : \mathtt{x} \end{array}$$

Ultimate pattern matching theorem

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

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

where each Ξ is drawn from $\overrightarrow{\mathtt{X}}$

Ultimate pattern matching theorem

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

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

where each Ξ is drawn from $\overrightarrow{\mathtt{X}}$

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

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

and filling w.

Ultimate pattern matching theorem

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

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

where each Ξ is drawn from $\overrightarrow{\mathtt{X}}$

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

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

and filling w.

Proof by induction on V.

Transition system [Lassen, Levy 2008]

Passive node (Opponent to play)

A passive node takes the form

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x} : \Xi}, \overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y} : \Upsilon}, \overrightarrow{\mathtt{g} \mapsto V : \neg B}$$

with each Ξ drawn from $\overrightarrow{\mathtt{X}}$ and each Υ drawn from $\overrightarrow{\mathtt{Y}}$

Active node (Proponent to play)

An active node takes the form

$$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{x} : \Xi}, \overrightarrow{\mathtt{f} : \neg A} || \overrightarrow{\mathtt{Y}}, \overrightarrow{\mathtt{y} : \Upsilon}, \overrightarrow{\mathtt{g} \mapsto V : \neg B} \vdash^{\mathsf{n}} M$$

Full Abstraction

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

Conjecture

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

Full Abstraction

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

Conjecture

Let $\Gamma \vdash^{\mathsf{n}} M, M'$ 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?

Isomorphisms

De Lataillade gave a complete list of isomorphisms that hold up to $\beta\eta$-equality.

Isomorphisms

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

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

Isomorphisms

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

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

How can we generalize Laurent's result to the polymorphic setting?

Related work

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

Also recent work by Laird.