Towards a denotational semantics for the $\rho$-calculus

Germain Faure  
LORIA-Nancy

Alexandre Miquel  
PPS-Paris 7
Menu of the next 30 minutes

- Presentation of the $\rho$-calculus (a fragment).
- Scott semantics.
- Discussions (weakness & new insights).
- On the horizon.
The $\rho$-calculus as a generalization of $\lambda$-calculus

“Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- Uniform integration of rewriting and $\lambda$-calculus.
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
  - **Pattern abstractions:**
    
    $\lambda x . M \leadsto \lambda P . M$
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- Uniform integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
  - Pattern abstractions:

    $$\lambda xyz . zyx \leadsto \lambda \text{cons}(x, l) . x$$
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
  - **Pattern abstractions:**
    \[
    \lambda xyz . zxy \leadsto \lambda cons(x, l) . x
    \]
  - **Matching constraints:**
    \[
    [cons(x, l) \ll \text{nil}]x
    \]
The \( \rho \)-calculus as a generalization of \( \lambda \)-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and \( \lambda \)-calculus.

- Main ideas of the \( \rho \)-calculus:
  - **Pattern abstractions:**
    \[
    \lambda xyz . zxy \leadsto \lambda \text{cons}(x, l) . x
    \]
  - **Matching constraints:**
    \[
    \text{[cons}(x, l) \ll \text{nil}]x \quad \lambda x . [f(y) \ll x]y
    \]
The \(\rho\)-calculus as a generalization of \(\lambda\)-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- Uniform integration of rewriting and \(\lambda\)-calculus.

- Main ideas of the \(\rho\)-calculus:
  - Pattern abstractions:
    \[
    \lambda xyz . zxy \leadsto \lambda \text{cons}(x, l) . x
    \]
  - Matching constraints:
    \[
    [\text{cons}(x, l) \ll \text{nil}]x \quad \lambda x . [f(y) \ll x]y
    \]
  - Structures:
    \[
    1 \quad 2
    \]
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
  - **Pattern abstractions:**
    \[
    \lambda xyz . zxy \leadsto \lambda \text{cons}(x, l) . x
    \]
  - **Matching constraints:**
    \[
    [\text{cons}(x, l) \ll \text{nil}]x \quad \lambda x . [f(y) \ll x]y
    \]
  - **Structures:**
    \[ Y \]
The $\rho$-calculus as a generalization of $\lambda$-calculus

- “Main design concept: to make all the basic ingredients of rewriting explicit objects” (from IGPAL-01)

- **Uniform** integration of rewriting and $\lambda$-calculus.

- Main ideas of the $\rho$-calculus:
  - **Pattern abstractions:**
    \[
    \lambda xyz \cdot zxy \leadsto \lambda \text{cons}(x, l) \cdot x
    \]
  - **Matching constraints:**
    \[
    [\text{cons}(x, l) \ll \text{nil}]x \quad \lambda x \cdot [f(y) \ll x]y
    \]
  - **Structures:**
    \[
    1; 2
    \]
Presentation of the $\rho$-calculus
Syntax of the $\rho$-calculus

| Terms          | $M, N ::= x$ (Variables) |
|               | $c$ (Constructors)       |
|               | $\lambda P . M$ (Abstraction) |
|               | $M \cdot N$ (Functional application) |
|               | $[P \ll N]M$ (Delayed matching constraints) |
|               | $M; N$ (Structure)       |

| Patterns       | $P ::= x$ (Variables) |
|               | $cP_1 \ldots P_n$ (Constructors) |
Semantics of the $\rho$-calculus

\( (\rho) \ (\lambda P . M) \cdot N \rightarrow [P \ll N] \cdot M \)

\( (\sigma) \ [P \ll N] \cdot M \rightarrow \sigma M \) 
if \( \sigma = P \ll N \)

\( (\delta) \ (M_1; M_2) \cdot N \rightarrow M_1 \cdot N; M_2 \cdot N \)
Examples

\((\lambda \text{cons}(x, l) \cdot x) \cdot \text{cons}(1, \text{nil})\)  
\[\quad \mapsto_\rho [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})] x \]
\[\quad \mapsto_\sigma 1\]
Examples

$$(\lambda \text{cons}(x, l) . x) \cdot \text{cons}(1, \text{nil})$$

$\mapsto_\rho [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})] x$

$\mapsto_\sigma 1$

$$(\lambda \text{cons}(x, l) . x) \cdot \text{nil}$$

$\mapsto_\rho [\text{cons}(x, l) \ll \text{nil}] x$$
Examples

\[(\lambda \text{cons}(x, l) \cdot x) \bullet \text{cons}(1, \text{nil})\]
\[\rightarrow_{\rho} [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})] x\]
\[\rightarrow_{\sigma} 1\]

\[(\lambda \text{cons}(x, l) \cdot x) \bullet \text{nil}\]
\[\rightarrow_{\rho} [\text{cons}(x, l) \ll \text{nil}] x\]

\[(\lambda \circ \cdot \text{stop}; \lambda \circ \cdot \text{go}; \lambda \circ \cdot \text{go}; \lambda \circ \cdot \text{stop}) \bullet \circ\]
Examples

\[(\lambda \text{cons}(x, l) . x) \bullet \text{cons}(1, \text{nil})\]
\[\mapsto_\rho \ [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})]x\]
\[\mapsto_\sigma 1\]

\[(\lambda \text{cons}(x, l) . x) \bullet \text{nil}\]
\[\mapsto_\rho \ [\text{cons}(x, l) \ll \text{nil}]x\]

\[(\lambda \circ . \text{stop}; \lambda \circ . \text{go}; \lambda \circ . \text{go}; \lambda \circ . \text{stop}) \bullet \circ\]
\[\mapsto_\delta \ (\lambda \circ . \text{stop}) \bullet \circ; (\lambda \circ . \text{go}) \bullet \circ; (\lambda \circ . \text{go}) \bullet \circ; (\lambda \circ . \text{stop}) \bullet \circ\]
\[\mapsto_\rho \ [\circ \ll \circ \text{stop}; [\circ \ll \circ \text{go}; [\circ \ll \circ \text{go}; [\circ \ll \circ \text{stop}\]
\[\mapsto_\sigma  [\circ \ll \circ \text{stop}; [\circ \ll \circ \text{go}; \text{go}; \text{stop}\]
Examples

\((\lambda\ \text{cons}(x, l)\ .\ x)\cdot\text{cons}(1, \text{nil})\)
\[\mapsto_{\rho} [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})] x\]
\[\mapsto_{\sigma} 1\]

\((\lambda\ \text{cons}(x, l)\ .\ x)\cdot\text{nil}\)
\[\mapsto_{\rho} [\text{cons}(x, l) \ll \text{nil}] x\]

\((\lambda\quad\text{stop}; \lambda\quad\text{go}; \lambda\quad\text{go}; \lambda\quad\text{stop})\cdot\text{stop}\)
\[\mapsto_{\delta} (\lambda\quad\text{stop})\cdot\text{stop}; (\lambda\quad\text{go})\cdot\text{go}; (\lambda\quad\text{go})\cdot\text{go}; (\lambda\quad\text{stop})\cdot\text{stop}\]
\[\mapsto_{\rho} [\text{stop} \ll \text{go}]\text{go}; [\text{go} \ll \text{go}]\text{go}; [\text{go} \ll \text{go}]\text{stop}\]
\[\mapsto_{\sigma + \text{GC}} \text{go}; \text{stop}\]
Examples

\[(\lambda \text{cons}(x, l) \cdot x) \cdot \text{cons}(1, \text{nil})\]
\[\mapsto_\rho [\text{cons}(x, l) \ll \text{cons}(1, \text{nil})]x\]
\[\mapsto_\sigma 1\]

\[(\lambda \text{cons}(x, l) \cdot x) \cdot \text{nil}\]
\[\mapsto_\rho [\text{cons}(x, l) \ll \text{nil}]x\]

\[(\lambda \bigcirc . \text{stop}; \lambda \bigcirc . \text{go}; \lambda \bigcirc . \text{go}; \lambda \bigcirc . \text{stop}) \cdot \bigcirc\]
\[\mapsto_\delta (\lambda \bigcirc . \text{stop}) \cdot \bigcirc; (\lambda \bigcirc . \text{go}) \cdot \bigcirc; (\lambda \bigcirc . \text{go}) \cdot \bigcirc; (\lambda \bigcirc . \text{stop}) \cdot \bigcirc\]
\[\mapsto_\rho [\bigcirc \ll \bigcirc] \text{stop}; [\bigcirc \ll \bigcirc] \text{go}; [\bigcirc \ll \bigcirc] \text{go}; [\bigcirc \ll \bigcirc] \text{stop}\]
\[\mapsto_\sigma + \text{GC} \quad \text{go; stop}\]

Theory on structures?
Why a denotational semantics

- **Operational semantics** focuses on computation
  Computation equivalence not always clear without the Church-Rosser property
  ⇒ Typical difficulty: *Prove that terms foo and bar are not convertible*
Why a denotational semantics

- **Operational semantics** focuses on computation
  Computation equivalence not always clear without the Church-Rosser property
  \[\Rightarrow\] Typical difficulty: *Prove that terms foo and bar are not convertible*

- **Denotational semantics** focuses on the input-output behaviour
  - Computations become transparent (convertible terms have the same denotation)
  - A clear notion of value \[\Rightarrow\] connect with mathematical intuitions
Why a denotational semantics

- **Operational semantics** focuses on computation
  Computation equivalence not always clear without the Church-Rosser property
  ⇒ Typical difficulty: *Prove that terms foo and bar are not convertible*

- **Denotational semantics** focuses on the input-output behaviour
  – Computations become transparent (convertible terms have the same denotation)
  – A clear notion of value ⇒ connect with mathematical intuitions

- **How it works:**
  – Find a suitable model $D$ for the given notion of computation
  – Define the interpretation function $\llbracket \cdot \rrbracket : \text{Terms} \rightarrow D$
Why a denotational semantics

- **Operational semantics** focuses on computation
  Computation equivalence not always clear without the Church-Rosser property
  ⇒ Typical difficulty: *Prove that terms *foo* and *bar* are not convertible*

- **Denotational semantics** focuses on the input-output behaviour
  – Computations become transparent (convertible terms have the same denotation)
  – A clear notion of value ⇒ connect with mathematical intuitions

- **How it works:**
  – Find a suitable model $D$ for the given notion of computation
  – Define the interpretation function $\llbracket \rrbracket : \text{Terms} \rightarrow D$
  – Prove the soundness property: $M_1 \overset{\text{comput.}}{=} M_2 \Rightarrow \llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$
Why a denotational semantics

- **Operational semantics** focuses on computation
  Computation equivalence not always clear without the Church-Rosser property
  ⇒ Typical difficulty: *Prove that terms* foo and bar *are not convertible*

- **Denotational semantics** focuses on the input-output behaviour
  – Computations become transparent (convertible terms have the same denotation)
  – A clear notion of value ⇒ connect with mathematical intuitions

- **How it works:**
  – Find a suitable model $D$ for the given notion of computation
  – Define the interpretation function $\llbracket \cdot \rrbracket : \text{Terms} \rightarrow D$
  – Prove the soundness property: $M_1 \overset{\text{comput.}}{=} M_2$ ⇒ $\llbracket M_1 \rrbracket = \llbracket M_2 \rrbracket$

  – Syntactic corollaries (Typically: *terms* foo and bar *are not convertible*)
Scott domains

- A Scott domain is a poset \((D, \leq)\) satisfying particular axioms.
  Namely: CPO + bottom element + bounded completeness + algebraicity

- Work with continuous functions (induced by Scott topology on \(D\))
- Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
  \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi
Scott domains

- A **Scott domain** is a poset \((D, \leq)\) satisfying particular axioms.
  - **Namely:** CPO + bottom element + bounded completeness + algebraicity
  - Work with **continuous functions** (induced by Scott topology on \(D\))
  - Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
    \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi

- **Intuition:** Each point of \(D\) represents some amount of information
Scott domains

- A Scott domain is a poset \((D, \leq)\) satisfying particular axioms.
  
  **Namely:** CPO + bottom element + bounded completeness + algebraicity

- Work with continuous functions (induced by Scott topology on \(D\))
- Domain equations such as \(D \approx (D \rightarrow D)\) have non-trivial solutions
  
  \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi

- **Intuition:** Each point of \(D\) represents some amount of information

  - Partiality: \(x \leq y \equiv x\) is less defined than \(y\)
  
  - In Comp. Sc.:
Scott domains

- A **Scott domain** is a poset \((D, \leq)\) satisfying particular axioms. 
  **Namely:** CPO + bottom element + bounded completeness + algebraicity
  - Work with **continuous functions** (induced by Scott topology on \(D\))
  - Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
    \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi

- **Intuition:** Each point of \(D\) represents some amount of information
  - Partiality: \(x \leq y \equiv x\) is less defined than \(y\)
  - In Comp. Sc.: Lack of (observable) information \(\equiv\) Non-termination
Scott domains

- A Scott domain is a poset \((D, \leq)\) satisfying particular axioms.
  
  **Namely:** CPO + bottom element + bounded completeness + algebraicity

- Work with **continuous functions** (induced by Scott topology on \(D\))
- Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
  
  \[ \Rightarrow \text{ Useful to interpret } \lambda\text{-calculi} \]

- **Intuition:** Each point of \(D\) represents some amount of information

  - Partiality: \(x \leq y \equiv x\) is less defined than \(y\)
  - In Comp. Sc.: Lack of (observable) information \(\equiv\) Non-termination
  - Bottom el\(^t\): \(\bot \equiv (\lambda x . xx)(\lambda x . xx) \equiv 10 \text{ GOTO 10, etc.} \)
Scott domains

- A Scott domain is a poset $(D, \leq)$ satisfying particular axioms.
  **Namely:** CPO + bottom element + bounded completeness + algebraicity

  - Work with continuous functions (induced by Scott topology on $D$)
  - Domain equations such as $D \approx (D \to D)$ have non-trivial solutions
    \Rightarrow Useful to interpret $\lambda$-calculi

- **Intuition:** Each point of $D$ represents some amount of information
  - Partiality: $x \leq y \equiv x$ is less defined than $y$
  - In Comp. Sc.: Lack of (observable) information $\equiv$ Non-termination
  - Bottom el$: \perp \equiv (\lambda x . xx)(\lambda x . xx) \equiv 10 \text{ GOTO } 10$, etc.
  - Monotonicity: Cannot extract information from non-termination (halting problem)
Scott domains

- A Scott domain is a poset \((D, \leq)\) satisfying particular axioms.

  Namely:  CPO + bottom element + bounded completeness + algebraicity

- Work with continuous functions (induced by Scott topology on \(D\))
- Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
  \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi

- Intuition: Each point of \(D\) represents some amount of information

  - Partiality: \(x \leq y \equiv x\) is less defined than \(y\)
  - In Comp. Sc.: Lack of (observable) information \(\equiv\) Non-termination
  - Bottom el\(^t\): \(\bot \equiv (\lambda x . xx)(\lambda x . xx) \equiv 10\) GOTO 10, etc.
  - Monotonicity: Cannot extract information from non-termination (halting problem)
  - Finiteness: A finite piece of output is produced by only a finite piece of input
Scott domains

- A Scott domain is a poset \((D, \leq)\) satisfying particular axioms.
  - **Namely:** CPO + bottom element + bounded completeness + algebraicity
  - Work with continuous functions (induced by Scott topology on \(D\))
  - Domain equations such as \(D \approx (D \to D)\) have non-trivial solutions
    \(\Rightarrow\) Useful to interpret \(\lambda\)-calculi

- **Intuition:** Each point of \(D\) represents some amount of information
  - Partiality: \(x \leq y \equiv x\) is less defined than \(y\)
  - In Comp. Sc.: Lack of (observable) information \(\equiv\) Non-termination
  - Bottom el\(\uparrow\): \(\bot \equiv (\lambda x.xx)(\lambda x.xx) \equiv 10\ \text{GOTO}\ 10,\ \text{etc.}\)
  - Monotonicity: Cannot extract information from non-termination (halting problem)
  - Finiteness: A finite piece of output is produced by only a finite piece of input
  - Continuity: \(\equiv\) Monotonicity + Finiteness [i.e. commutation with directed limits]
Rho-models

A $\rho$-model is a Scott-domain $D$ equipped with:

**Beta-rule** Two (Scott-continuous) functions:

- $\text{lam} : (D \rightarrow D) \rightarrow D$
- $\text{app} : D \rightarrow (D \rightarrow D)$

s.t. $\text{app} \circ \text{lam} = \text{id}_{D \rightarrow D}$

and $\text{lam} \circ \text{app} \leq \text{id}_D$
Rho-models

A $\rho$-model is a Scott-domain $D$ equipped with:

- **Beta-rule** Two (Scott-continuous) functions:
  \[
  \text{lam} : (D \to D) \to D \quad \text{app} : D \to (D \to D) \quad \text{s.t.} \quad \text{app} \circ \text{lam} = \text{id}_{D \to D} \quad \text{lam} \circ \text{app} \leq \text{id}_D
  \]

- **Pattern-matching** For each constructor $c$ of arity $n$, two functions:
  \[
  c_* : D^n \to D \quad c^* : D \to \text{opt}(D^n) \quad \text{s.t.} \quad c^* \circ c_* = (\vec{w} \mapsto \text{some}(\vec{w}))
  \]
Rho-models

A \( \rho \)-model is a Scott-domain \( D \) equipped with:

- **Beta-rule** Two (Scott-continuous) functions:
  
  \[
  \text{lam} : (D \rightarrow D) \rightarrow D \\
  \text{app} : D \rightarrow (D \rightarrow D)
  \]

  such that
  
  \[
  \text{app} \circ \text{lam} = \text{id}_{D \rightarrow D} \\
  \text{lam} \circ \text{app} \leq \text{id}_D
  \]

- **Pattern-matching** For each constructor \( c \) of arity \( n \), two functions:

  \[
  c^* : D^n \rightarrow D \\
  c^* : D \rightarrow \text{opt}(D^n)
  \]

  such that
  
  \[
  c^* \circ c^* = (\vec{w} \mapsto \text{some}(\vec{w}))
  \]

- **Errors** For each pattern \( P \) of arity \( n \), a function:

  \[
  \text{error}_P : D \times (D^n \rightarrow D) \rightarrow D
  \]

  (no axioms)
**Rho-models**

A $\rho$-model is a Scott-domain $D$ equipped with:

- **Beta-rule** Two (Scott-continuous) functions:
  
  \[
  \begin{align*}
  \text{lam} & : (D \to D) \to D \\
  \text{app} & : D \to (D \to D)
  \end{align*}
  \]
  
  s.t.
  
  \[
  \text{app} \circ \text{lam} = \text{id}_{D \to D}
  \]
  \[
  \text{lam} \circ \text{app} \leq \text{id}_D
  \]

- **Pattern-matching** For each constructor $c$ of arity $n$, two functions:
  
  \[
  \begin{align*}
  c^* & : D^n \to D \\
  c^* & : D \to \text{opt}(D^n)
  \end{align*}
  \]
  
  s.t.
  
  \[
  c^* \circ c^* = (\vec{w} \mapsto \text{some}(\vec{w}))
  \]

- **Errors** For each pattern $P$ of arity $n$, a function:
  
  \[
  \text{error}_P : D \times (D^n \to D) \to D
  \]
  
  (no axioms)

- **Structures** A function:
  
  \[
  \text{merge} : D \times D \to D
  \]
  
  s.t.
  
  \[
  \text{app}(\text{merge}(v_1, v_2), w) = \\
  \text{merge}(\text{app}(v_1, w), \text{app}(v_2, w))
  \]
Additional axioms

Extensions of the basic $\rho$-theory require extra axioms:

- Extensionality:
  
  \[ (\eta\text{-rule}) \quad \text{lam} \circ \text{app} = \text{id}_D \]
Additional axioms

Extensions of the basic $\rho$-theory require extra axioms:

- **Extensionality:**
  
  $\eta$-rule: \( \text{lam} \circ \text{app} = \text{id}_D \)

- **ACI theory** (or any combination of the axioms $A$, $C$, $I$):
  
  - (Associativity) \( \text{merge}(\text{merge}(v_1, v_2), v_3) = \text{merge}(v_1, \text{merge}(v_2, v_3)) \)
  
  - (Commutativity) \( \text{merge}(v_1, v_2) = \text{merge}(v_2, v_1) \)
  
  - (Idempotence) \( \text{merge}(v, v) = v \)
**Additional axioms**

Extensions of the basic $\rho$-theory require extra axioms:

- **Extensionality:**
  
  \[
  \text{(}\eta\text{-rule)} \quad \text{lam} \circ \text{app} = \text{id}_D
  \]

- **ACI theory** (or any combination of the axioms A, C, I):
  
  \[
  \begin{align*}
  \text{(Associativity)} & \quad \text{merge(merge}(v_1, v_2), v_3) = \text{merge}(v_1, \text{merge}(v_2, v_3)) \\
  \text{(Commutativity)} & \quad \text{merge}(v_1, v_2) = \text{merge}(v_2, v_1) \\
  \text{(Idempotence)} & \quad \text{merge}(v, v) = v
  \end{align*}
  \]

- **Constructor discrimination**:
  
  for all $c \neq c'$
  
  \[
  c^* \circ c'^* = (_\mapsto \text{none})
  \]
Compiling pattern matching

Compositionality of pattern-matching

The interpretation of constructors (of arity $n$) provided by any $\rho$-model

$$c_* : D^n \to D, \quad c^* : D \to \text{opt}(D^n) \quad \text{s.t.} \quad c^* \circ c_* = (\vec{w} \mapsto \text{some}(\vec{w}))$$

is easily extended to all ML-style patterns $P$ (of arity $n$):

$$P_* : D^n \to D, \quad P^* : D \to \text{opt}(D^n) \quad \text{s.t.} \quad P^* \circ P_* = (\vec{w} \mapsto \text{some}(\vec{w}))$$
Compiling pattern matching

Compositionality of pattern-matching

The interpretation of constructors (of arity $n$) provided by any $\rho$-model

$$c_* : D^n \rightarrow D, \quad c^* : D \rightarrow \text{opt}(D^n) \quad \text{s.t.} \quad c^* \circ c_* = (\overrightarrow{w} \mapsto \text{some}(\overrightarrow{w}))$$

is easily extended to all ML-style patterns $P$ (of arity $n$):

$$P_* : D^n \rightarrow D, \quad P^* : D \rightarrow \text{opt}(D^n) \quad \text{s.t.} \quad P^* \circ P_* = (\overrightarrow{w} \mapsto \text{some}(\overrightarrow{w}))$$

The matching function $\text{match}_P : D \times (D^n \rightarrow D) \rightarrow D$ is defined by

$$\text{match}_P(v, f) = \text{case}_\text{opt} \; P^*(v) \text{ with }$$

$$| \text{some}(\overrightarrow{w}) \mapsto f(\overrightarrow{w}) \quad | \text{none} \mapsto \text{error}_P(v, f)$$

(where $\text{case}_\text{opt}$ is the destruction operation of values of type $\text{opt}(D^n)$)
The interpretation function

Valuations  A valuation is a function $\rho : V \to D$ ($V =$ set of all variables)
The set of all valuations $D^V$ is a Scott-domain (i.e. infinite cartesian product).
The interpretation function

Valuations A valuation is a function $\rho : \mathcal{V} \rightarrow D$ ($\mathcal{V}$ = set of all variables)
The set of all valuations $D^\mathcal{V}$ is a Scott-domain (i.e. infinite cartesian product).

Interpretation By induction on $M$ we set:

$$\llbracket x \rrbracket_\rho = \rho(x)$$
$$\llbracket c \rrbracket_\rho = c_\ast$$
The interpretation function

**Valuations**  A valuation is a function $\rho : \mathcal{V} \rightarrow D$ ($\mathcal{V} =$ set of all variables)

The set of all valuations $D^\mathcal{V}$ is a Scott-domain  (i.e. infinite cartesian product).

**Interpretation**  By induction on $M$ we set:

$$[x]_\rho = \rho(x)$$
$$[c]_\rho = \text{lam}_n(\text{curry}_n(c_*))$$
The interpretation function

Valuations  A valuation is a function $\rho : \mathcal{V} \to D$ ($\mathcal{V} =$ set of all variables).

The set of all valuations $D^\mathcal{V}$ is a Scott-domain (i.e. infinite cartesian product).

Interpretation  By induction on $M$ we set:

\[
\begin{align*}
[x]_\rho &= \rho(x) \\
[c]_\rho &= \text{lam}_n(\text{curry}_n(c_*)) \\
[[P\bar{x} \ll N]M]_\rho &= \text{match}_P([[N]_\rho, \bar{w} \mapsto [[M]_{(\rho;\bar{x} \mapsto \bar{w})}}]]
\end{align*}
\]
The interpretation function

**Valuations**  A valuation is a function \( \rho : \mathcal{V} \rightarrow D \)  (\( \mathcal{V} = \text{set of all variables} \))

The set of all valuations \( D^\mathcal{V} \) is a Scott-domain  (i.e. infinite cartesian product).

**Interpretation**  By induction on \( M \) we set:

\[
\begin{align*}
[x]_\rho &= \rho(x) \\
[c]_\rho &= \text{lam}_n(\text{curry}_n(c_*)) \\
[[P\bar{x} \ll N]M]_\rho &= \text{match}_P([N]_\rho, (\vec{w} \mapsto [M]_{(\rho;\bar{x} \leftarrow \vec{w})})) \\
[\lambda P\bar{x}. M]_\rho &= \text{lam}(v \mapsto \text{match}_P(v, (\vec{w} \mapsto [M]_{(\rho;\bar{x} \leftarrow \vec{w})})))
\end{align*}
\]
The interpretation function

**Valuations** A valuation is a function $\rho : V \to D$ ($V =$ set of all variables)

The set of all valuations $D^V$ is a Scott-domain (i.e. infinite cartesian product).

**Interpretation** By induction on $M$ we set:

\[
\begin{align*}
[x]_\rho &= \rho(x) \\
[c]_\rho &= \text{lam}_n(\text{curry}_n(c_*)) \\
[[P \tilde{x} \ll N] M]_\rho &= \text{match}_P([N]_\rho, (\tilde{w} \mapsto [M]_{(\rho; \tilde{x} \leftarrow \tilde{w})})) \\
[\lambda P \tilde{x} . M]_\rho &= \text{lam}(v \mapsto \text{match}_P(v, (\tilde{w} \mapsto [M]_{(\rho; \tilde{x} \leftarrow \tilde{w})}))) \\
[MN]_\rho &= \text{app} [M]_\rho [N]_\rho \\
[M; N]_\rho &= \text{merge}([M]_\rho, [N]_\rho)
\end{align*}
\]

The denotation $[[M]_\rho \in D$ continuously depends on the valuation $\rho \in D^V$. 

The interpretation function

**Valuations**  A valuation is a function $\rho : \mathcal{V} \rightarrow D$  ($\mathcal{V} =$ set of all variables)

The set of all valuations $D^\mathcal{V}$ is a Scott-domain  (i.e. infinite cartesian product).

**Interpretation**  By induction on $M$ we set:

\[
\begin{align*}
[x]_\rho &= \rho(x) \\
[c]_\rho &= \text{lam}_n(\text{curry}_n(c_*)) \\
[[P \bar{x} \ll N]M]_\rho &= \text{match}_P([[N]_\rho, (\bar{w} \mapsto [[M]_\rho(\bar{x} \leftarrow \bar{w}))]) \\
[\lambda P \bar{x}. M]_\rho &= \text{lam}(v \mapsto \text{match}_P(v, (\bar{w} \mapsto [[M]_\rho(\bar{x} \leftarrow \bar{w}))})) \\
[MN]_\rho &= \text{app}([[M]_\rho] [N]_\rho) \\
[M; N]_\rho &= \text{merge}([[M]_\rho], [N]_\rho)
\end{align*}
\]

The denotation $[[M]_\rho \in D$ continuously depends on the valuation $\rho \in D^\mathcal{V}$.

**Closed case**  If $M$ is closed, we write $[[M]] = [[M]_\rho$  (does not depend on $\rho$)
The soundness property

Write: \[ D \models M_1 = M_2 \iff \forall \rho \in D^\forall \ [M_1]_\rho = [M_2]_\rho \]
The soundness property

Write: \[ D \models M_1 = M_2 \iff \forall \rho \in D^y \ [M_1]_\rho^D = [M_2]_\rho^D \]

Lemma (Soundness) — In all the \( \rho \)-models \( D \):

\[ M_1 \rho = \delta \implies D \models M_1 = M_2 \]
The soundness property

Write: \[ D \models M_1 = M_2 \equiv \forall \rho \in D^\gamma \ [M_1]_\rho^D = [M_2]_\rho^D \]

Lemma (Soundness) — In all the \( \rho \)-models \( D \):

\[ M_1 = M_2_{\rho\sigma\delta} \Rightarrow D \models M_1 = M_2 \]

Lemma (Soundness w.r.t. ACI) — In all the \( \rho \)-models \( D \) where ‘merge’ is ACI:

\[ M_1 = M_2_{\rho\sigma\delta_{\text{ACI}}} \Rightarrow D \models M_1 = M_2 \]
The soundness property

Write: \[ D \models M_1 = M_2 \Leftrightarrow \forall \rho \in D^V \ [M_1]_\rho^D = [M_2]_\rho^D \]

Lemma (Soundness) — In all the \( \rho \)-models \( D \):

\[
M_1 =_{\rho\sigma\delta} M_2 \Rightarrow D \models M_1 = M_2
\]

Lemma (Soundness w.r.t. ACI) — In all the \( \rho \)-models \( D \) where ‘merge’ is ACI:

\[
M_1 =_{\rho\sigma\delta}^{\text{ACI}} M_2 \Rightarrow D \models M_1 = M_2
\]

Remarks:

– Soundness still holds for any combination of A, C, I and/or the \( \eta \)-reduction rule (provided we restrict to the corresponding notion of \( \rho \)-model).
The soundness property

Write:

\[ D \models M_1 = M_2 \iff \forall \rho \in D^V \ [M_1]^D_{\rho} = [M_2]^D_{\rho} \]

Lemma (Soundness) — In all the \( \rho \)-models \( D \):

\[ M_1 = M_2 \rho \sigma \delta \Rightarrow D \models M_1 = M_2 \]

Lemma (Soundness w.r.t. ACI) — In all the \( \rho \)-models \( D \) where ‘merge’ is ACI:

\[ M_1 = M_2 \rho \sigma \delta \_{\text{ACI}} \Rightarrow D \models M_1 = M_2 \]

Remarks:

– Soundness still holds for any combination of A, C, I and/or the \( \eta \)-reduction rule (provided we restrict to the corresponding notion of \( \rho \)-model).
– Proofs do not depend on any kind of Church-Rosser/confluence property.
A fundamental particular case: $D^\infty$

Let $D^\infty$ be the ‘historical’ non-trivial solution of: $D^\infty \simeq (D^\infty \to D^\infty)$

and set: $\text{merge}(v_1, v_2) := \sup\{v_1, v_2\}$ [Works since $D^\infty$ has a top-element]

$error_P(v, f) := \bot$
A fundamental particular case: $D^\infty$

Let $D^\infty$ be the ‘historical’ non-trivial solution of: $D^\infty \approx (D^\infty \rightarrow D^\infty)$
and set: $\text{merge}(v_1, v_2) := \sup\{v_1, v_2\}$ [Works since $D^\infty$ has a top-element]
error$_P(v, f) := \bot$

Lemma. — $D^\infty$ is a $\rho$-model that satisfies the axioms $A$, $C$, $I$ and $\eta$
A fundamental particular case: $D^\infty$

Let $D^\infty$ be the ‘historical’ non-trivial solution of: $D^\infty \approx (D^\infty \to D^\infty)$ and set: $\text{merge}(v_1, v_2) := \sup\{v_1, v_2\}$ [Works since $D^\infty$ has a top-element]

$\text{error}_P(v, f) := \bot$

Lemma. — $D^\infty$ is a $\rho$-model that satisfies the axioms $A$, $C$, $I$ and $\eta$

Moreover, the following diagram commutes:

\[
\begin{array}{ccc}
\rho\text{-calculus} & \xrightarrow{[\cdot]} & D^\infty \\
\text{syntactic embedding} & \uparrow & \\
\lambda\text{-calculus} & \xleftarrow{\text{Scott’s } [\cdot]} & \\
\end{array}
\]
A fundamental particular case: \( D^\infty \)

Let \( D^\infty \) be the ‘historical’ non-trivial solution of: \( D^\infty \approx (D^\infty \to D^\infty) \)

and set:

\[
\text{merge}(v_1, v_2) := \sup\{v_1, v_2\} \quad \text{[Works since } D^\infty \text{ has a top-element]}
\]

\[
\text{error}_P(v, f) := \bot
\]

**Lemma.** — \( D^\infty \) is a \( \rho \)-model that satisfies the axioms \( A, C, I \) and \( \eta \)

Moreover, the following diagram commutes:

Since Scott’s interpretation is faithfull (i.e. injective) on \( \beta\eta \)-normal forms, we get:

**Corollary (Conservativity on \( \lambda \)-normal forms / Weak C.R.)** — \( \rho\sigma\delta\eta\text{ACI-theory of the } \rho \text{-calculus identifies no pair of distinct } \beta\eta \text{-normal terms of the } \lambda \text{-calculus.} \)
Discussion

- Compositionality of matching.
- Right distributivity rule.
- Notion of values.
- Weakness of the model
  - Management of errors.
  - Structures (cannot be destructed).
    ⇒ Monads.
Another perspective

- The construction $\lambda P_{\overline{x}} \cdot M_{\overline{x}}$ can be understood as $M \circ P^{-1}$

**The (hidden) dream of $\rho$-calculists:** allow the use of all terms as patterns...

⇒ Allow the inversion of arbitrary functions: $M^{-1} \equiv \lambda Mx \cdot x$
Another perspective

- The construction \( \lambda P \vec{x} \cdot M \vec{x} \) can be understood as \( M \circ P^{-1} \)

**The (hidden) dream of \( \rho \)-calculists:** allow the use of all terms as patterns...
⇒ Allow the inversion of arbitrary functions: \( M^{-1} \equiv \lambda M \vec{x} \cdot \vec{x} \)

- **Slogans:**

  1940’s  The \( \lambda \)-calculus: ‘Legalize (arbitrary) application’
  2000’s  The \( \rho \)-calculus: ‘Legalize (arbitrary) inversion’
Another perspective

- The construction $\lambda P \vec{x}. M \vec{x}$ can be understood as $M \circ P^{-1}$

**The (hidden) dream of $\rho$-calculists:** allow the use of all terms as patterns...

$\Rightarrow$ Allow the inversion of arbitrary functions: $M^{-1} \equiv \lambda M \vec{x} . \vec{x}$

- **Slogans:**

  1940’s  The $\lambda$-calculus: ‘Legalize (arbitrary) application’
  2000’s  The $\rho$-calculus: ‘Legalize (arbitrary) inversion’

- But **inversion** is fundamentally **anti-monotonic!**  
  [Think of $x \mapsto 1/x$, $f \mapsto f^{-1}$]

  $\Rightarrow$ The full dream of $\rho$-calculists will not be realised with Scott-style semantics

  $\Rightarrow$ But an exciting challenge for denotational semantics!