Denotational Semantics for Concurrency

Steffen Lösch

Supervised by Prof Glynn Winskel, University of Cambridge

18. June 2011
Why Concurrent Denotational Semantics?

Concurrency is still kind of a mess... A functioning denotational semantics can give better understanding, exchange of ideas, and additional semantic tools.
Why Concurrent Denotational Semantics?

Concurrency is still kind of a mess...
Why Concurrent Denotational Semantics?

Concurrency is still kind of a mess...

A functioning denotational semantics can give

- better understanding
- exchange of ideas
- additional semantic tools
What is the "meaning" of a process?

\[ M = \{ \emptyset, a, ab \} \]

\[ M' = \{ \emptyset, a, ab \} \]

Typed Processes: \( M : A \)

The type \( A \) is the "shape" of computations paths \( M \) can perform.

Abstractly: \( A \) is a pre-order \( \rightarrow \) path order

\[ M \in \hat{A} \]

Domain theory: \( \hat{A} \) is a complete lattice
Modelling

What is the "meaning" of a process?

$$\llbracket M \rrbracket = \text{the set of computation paths of } M$$
Modelling

What is the "meaning" of a process?

$[M] = \text{the set of computation paths of } M$

Examples:

- $[a.b.0] = \{\emptyset, a, ab\}$
What is the "meaning" of a process?

$[[M]] = \text{the set of computation paths of } M$

Examples:

- $[[a.b.0]] = \{\emptyset, a, ab\}$
- $[[a.b.0 + a.0]] = \{\emptyset, a, ab\}$
What is the "meaning" of a process?

$[M] = \text{the set of computation paths of } M$

Examples:
- $[a.b.0] = \{\emptyset, a, ab\}$
- $[a.b.0 + a.0] = \{\emptyset, a, ab\}$

Typed Processes: $M : A$
What is the "meaning" of a process?

\([M] = \) the set of computation paths of \(M\)

Examples:

- \([a.b.0] = \{\emptyset, a, ab\}\)
- \([a.b.0 + a.0] = \{\emptyset, a, ab\}\)

Typed Processes: \(M : A\)

The type \(A\) is the "shape" of computations paths \(M\) can perform.

- Abstractly: \(A\) is a pre-order \(\rightarrow\) path order
Modelling

What is the "meaning" of a process?

$[M] = \text{the set of computation paths of } M$

Examples:

- $[a.b.0] = \{\emptyset, a, ab\}$
- $[a.b.0 + a.0] = \{\emptyset, a, ab\}$

Typed Processes: $M : A$

The type $A$ is the "shape" of computations paths $M$ can perform.

- Abstractly: $A$ is a pre-order $\rightarrow$ path order
- $[M] \in \hat{A}$ where $\hat{A} = \{X\downarrow \mid X \subseteq A\} \rightarrow \text{path set}$
What is the "meaning" of a process?

\[[M]\] = the set of computation paths of \(M\)

Examples:
- \(\llbracket a.b.0 \rrbracket = \{\emptyset, a, ab\}\)
- \(\llbracket a.b.0 + a.0 \rrbracket = \{\emptyset, a, ab\}\)

Typed Processes: \(M : \mathbb{A}\)

The type \(\mathbb{A}\) is the "shape" of computations paths \(M\) can perform.

- Abstractly: \(\mathbb{A}\) is a pre-order \(\rightarrow\) path order
- \(\llbracket M \rrbracket \in \hat{\mathbb{A}}\) where \(\hat{\mathbb{A}} = \{X \downarrow | X \subseteq \mathbb{A}\}\) \(\rightarrow\) path set
- Domain theory: \(\hat{\mathbb{A}}\) is a complete lattice
Some Categories

Define the categories:

- **Pre** = Path orders $A$, $B$ and monotone functions $A \rightarrow B$
- **Lin** = Path orders $A$, $B$ and join-preserving functions $\hat{A} \rightarrow \hat{B}$
- **Cts** = Path orders $A$, $B$ and directed-join-preserving functions $\hat{A} \rightarrow \hat{B}$

Rich categorical structure:

- Adjunctions $\text{Pre}(A, \hat{B}) \sim \text{Lin}(A, B)$ and $\text{Lin}(\!A, B) \sim \text{Cts}(A, B)$

Lin is a categorical model of linear logic and Cts is cartesian closed

Hom-sets of Lin and Cts have the structure of commutative monoids

Use the categorical structure for defining a programming language!

- cartesian closed category $\iff$ lambda-calculus
- Cts $\iff$ HOPLA

S. Lösch (Univ. Cambridge)
Denotational Semantics for Concurrency 18.06.11 4 / 7
Some Categories

Define the categories:

- **Pre** = Path orders $A, B$ and monotone functions $A \to B$
- **Lin** = Path orders $A, B$ and join-preserving functions $\hat{A} \to \hat{B}$
- **Cts** = Path orders $A, B$ and directed-join-preserving functions $\hat{A} \to \hat{B}$
Some Categories

Define the categories:

- **Pre** = Path orders $A, B$ and monotone functions $A \rightarrow B$
- **Lin** = Path orders $A, B$ and join-preserving functions $\hat{A} \rightarrow \hat{B}$
- **Cts** = Path orders $A, B$ and directed-join-preserving functions $\hat{A} \rightarrow \hat{B}$

Rich categorical structure:
Some Categories

Define the categories:

- **Pre** = Path orders $A, B$ and monotone functions $A \rightarrow B$
- **Lin** = Path orders $A, B$ and join-preserving functions $\hat{A} \rightarrow \hat{B}$
- **Cts** = Path orders $A, B$ and directed-join-preserving functions $\hat{A} \rightarrow \hat{B}$

Rich categorical structure:

- Adjunctions $Pre(A, \hat{B}) \cong Lin(A, B)$ and $Lin(!A, B) \cong Cts(A, B)$
Some Categories

Define the categories:

- **Pre** = Path orders $\mathbb{A}, \mathbb{B}$ and monotone functions $\mathbb{A} \to \mathbb{B}$
- **Lin** = Path orders $\mathbb{A}, \mathbb{B}$ and join-preserving functions $\hat{\mathbb{A}} \to \hat{\mathbb{B}}$
- **Cts** = Path orders $\mathbb{A}, \mathbb{B}$ and directed-join-preserving functions $\hat{\mathbb{A}} \to \hat{\mathbb{B}}$

Rich categorical structure:

- Adjunctions $\text{Pre}(\mathbb{A}, \hat{\mathbb{B}}) \cong \text{Lin}(\mathbb{A}, \mathbb{B})$ and $\text{Lin}(\hat{\mathbb{A}}, \mathbb{B}) \cong \text{Cts}(\mathbb{A}, \mathbb{B})$
- **Lin** is a categorical model of linear logic and **Cts** is cartesian closed
Some Categories

Define the categories:

- **Pre** = Path orders $\mathbb{A}, \mathbb{B}$ and monotone functions $\mathbb{A} \to \mathbb{B}$
- **Lin** = Path orders $\mathbb{A}, \mathbb{B}$ and join-preserving functions $\hat{\mathbb{A}} \to \hat{\mathbb{B}}$
- **Cts** = Path orders $\mathbb{A}, \mathbb{B}$ and directed-join-preserving functions $\hat{\mathbb{A}} \to \hat{\mathbb{B}}$

Rich categorical structure:

- Adjunctions $\text{Pre}(\mathbb{A}, \hat{\mathbb{B}}) \cong \text{Lin}(\mathbb{A}, \mathbb{B})$ and $\text{Lin}(!\mathbb{A}, \mathbb{B}) \cong \text{Cts}(\mathbb{A}, \mathbb{B})$
- **Lin** is a categorical model of linear logic and **Cts** is cartesian closed
- Hom-sets of **Lin** and **Cts** have the structure of commutative monoids
Some Categories

Define the categories:

- **Pre** = Path orders $A, B$ and monotone functions $A \to B$
- **Lin** = Path orders $A, B$ and join-preserving functions $\hat{A} \to \hat{B}$
- **Cts** = Path orders $A, B$ and directed-join-preserving functions $\hat{A} \to \hat{B}$

Rich categorical structure:

- Adjunctions $\text{Pre}(A, \hat{B}) \cong \text{Lin}(A, B)$ and $\text{Lin}(!A, B) \cong \text{Cts}(A, B)$
- **Lin** is a categorical model of linear logic and **Cts** is cartesian closed
- Hom-sets of **Lin** and **Cts** have the structure of commutative monoids

Use the categorical structure for defining a programming language!
Some Categories

Define the categories:

- \textbf{Pre} = Path orders \( \mathcal{A}, \mathcal{B} \) and monotone functions \( \mathcal{A} \to \mathcal{B} \)
- \textbf{Lin} = Path orders \( \mathcal{A}, \mathcal{B} \) and join-preserving functions \( \hat{\mathcal{A}} \to \hat{\mathcal{B}} \)
- \textbf{Cts} = Path orders \( \mathcal{A}, \mathcal{B} \) and directed-join-preserving functions \( \hat{\mathcal{A}} \to \hat{\mathcal{B}} \)

Rich categorical structure:

- Adjunctions \( \text{Pre}(\mathcal{A}, \hat{\mathcal{B}}) \cong \text{Lin}(\mathcal{A}, \mathcal{B}) \) and \( \text{Lin}(!\mathcal{A}, \mathcal{B}) \cong \text{Cts}(\mathcal{A}, \mathcal{B}) \)
- \textbf{Lin} is a categorical model of linear logic and \textbf{Cts} is cartesian closed
- Hom-sets of \textbf{Lin} and \textbf{Cts} have the structure of commutative monoids

Use the categorical structure for defining a programming language!

\[
\begin{align*}
\text{cartesian closed category} & \iff \text{lambda-calculus} \\
\textbf{Cts} & \iff \text{HOPLA}
\end{align*}
\]
Types:
\[ A ::= A_1 \rightarrow A_2 | \bigoplus_{i \in I} A_i | ! A | \mu \vec{A}.\vec{A} \]

Terms:
\[ M ::= x | \text{rec } x.t | \lambda x.M | M_1 M_2 | l : M | \pi l M | ! M | [M_1 > ! x \Rightarrow M_2] \]

Has an operational semantics
Denotational semantics is fully abstract
Can express CCS, higher order CCS and ambient calculus with public names

S. Lösch (Univ. Cambridge)
Denotational Semantics for Concurrency
18.06.11 5 / 7
Types:

\[ \mathbb{A} ::= \mathbb{A}_1 \rightarrow \mathbb{A}_2 \mid \bigoplus_{i \in I} \mathbb{A}_i \mid !\mathbb{A} \mid \mathbb{A} \mid \mu \vec{\mathbb{A}}.\vec{\mathbb{A}} \]
Types:

\[ \mathbf{A} ::= \mathbf{A}_1 \to \mathbf{A}_2 \mid \bigoplus_{i \in I} \mathbf{A}_i \mid !\mathbf{A} \mid \mathbf{A} \mid \mu \vec{A}. \vec{A} \]

Terms:

\[ \mathbf{M} ::= x \mid \text{rec } x.t \mid \lambda x.\mathbf{M} \mid \mathbf{M}_1 \mathbf{M}_2 \mid l : \mathbf{M} \mid \pi_l \mathbf{M} \mid !\mathbf{M} \mid [\mathbf{M}_1 >!x \Rightarrow \mathbf{M}_2] \]
Types:

\[ \mathcal{A} ::= \mathcal{A}_1 \rightarrow \mathcal{A}_2 \mid \bigoplus_{i \in I} \mathcal{A}_i \mid !\mathcal{A} \mid \mathcal{A} \mid \mu \vec{\mathcal{A}}.\vec{\mathcal{A}} \]

Terms:

\[ M ::= x \mid \text{rec } x.t \mid \lambda x.\mathcal{M} \mid M_1 M_2 \mid l : M \mid \pi_l M \mid !M \mid [M_1 >!x \Rightarrow M_2] \]

- Has an operational semantics
- Denotational semantics is fully abstract
- Can express CCS, higher order CCS and ambient calculus with public names
What am I doing?

Caveats of HOPLA:
- Lacks name generation construct (ν, M), so cannot express π-calculus or full ambient calculus.
- Denotational semantics does not reflect bisimulation.

Remedies:
- Nominal HOPLA: model everything with Pitts' nominal sets and gain name generation by that.
- Generalize by switching to a presheaf semantics: 
  \[ \hat{A} = \text{presheaves over } A = (\mathcal{A}^{op}, \text{Set}) \]

Questions for the resulting language:
- Is it fully abstract?
- How expressive is it?
- What can we learn about concurrency/logic/computing with it?
Caveats of HOPLA:

- Lacks name generation construct ($\nu a.M$), so cannot express $\pi$-calculus or full ambient calculus
- Denotational semantics does not reflect bisimulation
What am I doing?

Caveats of HOPLA:
- Lacks name generation construct \((\nu a. M)\), so cannot express \(\pi\)-calculus or full ambient calculus
- Denotational semantics does not reflect bisimulation

Remedies?
- Nominal HOPLA: model everything with Pitts’ nominal sets and gain name generation by that
- Generalize by switching to a presheaf semantics: \(\hat{\mathcal{A}} = \text{presheaves over } \mathcal{A} = ([\mathcal{A}^{op}, \text{Set}])\)
What am I doing?

Caveats of HOPLA:
- Lacks name generation construct ($\nu a.M$), so cannot express $\pi$-calculus or full ambient calculus
- Denotational semantics does not reflect bisimulation

Remedies?
- Nominal HOPLA: model everything with Pitts’ nominal sets and gain name generation by that
- Generalize by switching to a presheaf semantics: $\mathcal{A} = \text{presheaves over } \mathcal{A} = ([\mathcal{A}^{op}, \text{Set}])$

Questions for the resulting language:
- Is it fully abstract?
- How expressive is it?
- What can we learn about concurrency/logic/computing with it?
Thank you for your attention.

Don’t hesitate to ask further questions.