Keywords
higher category theory, concurrency, message-passing, types, Curry-Howard

ABSTRACT
We present an approach to modeling computational calculi using higher category theory. While the paper focuses on applications to the mobile process calculi, and more specifically, the π-calculus, because they provide unique challenges for categorical models, the approach extends smoothly to a variety of other computational calculi, including important milestones such as the λ-calculus. One of the key contributions is a method of restricting rewrites to specific contexts inspired by catalysis in chemical reactions.

Submission to arXiv

Higher category models of mobile process calculi

Mike Stay
Google
metaweta@gmail.com

L.G. Meredith
Biosimilarity, LLC
lgreg.meredith@biosimilarity.com

1. INTRODUCTION

One of the major distinctions in programming language semantics has been the division between denotational and operational semantics. In the former computations are interpreted as mathematical objects which – more often than not – completely unfold the computational dynamics, and are thus infinitary in form. In the latter computations are interpreted in terms of rules operating on finite syntactic structure. Historically, categorical semantics for programming languages, even variations such as games semantics which capture much more of the intensional structure of computations, are distinctly denotational in flavor. Meanwhile, operational semantics continues to dominate in the presentation of calculi underlying programming languages used in practice.

Motivated, in part, by the desire to make a closer connection between theory and practice, the approach taken here investigates a direct categorical interpretation of the operational presentation of the π-calculus.

1.0.1 Organization of the rest of the paper

TBD

2. THE CALCULUS

Some examples of process expressions.

2.1 Our running process calculus

2.1.1 Syntax

$$
\begin{array}{ll}
M, N ::= 0 & \text{stopped process} \\
\quad \mid x?(y_1, \ldots, y_N) \Rightarrow P & \text{input} \\
\quad \mid x!(y_1, \ldots, y_N) & \text{output} \\
\quad \mid M{+}N & \text{choice} \\
P, Q ::= M & \text{include IO processes} \\
\quad \mid P \mid Q & \text{parallel}
\end{array}
$$

2.1.2 Free and bound names

$$
\begin{aligned}
&\mathcal{FN}(0) \coloneqq \emptyset \\
&\mathcal{FN}(x?(y_1, \ldots, y_N) \Rightarrow P) \coloneqq \\
&\quad \{x\} \cup (\mathcal{FN}(P) \setminus \{y_1, \ldots y_N\}) \\
&\mathcal{FN}(x!(y_1, \ldots, y_N)) \coloneqq \{x, y_1, \ldots, y_N\} \\
&\mathcal{FN}(P \mid Q) \coloneqq \mathcal{FN}(P) \cup \mathcal{FN}(Q)
\end{aligned}
$$

An occurrence of x in a process P is *bound* if it is not free. The set of names occurring in a process (bound or free) is denoted by $\mathcal{N}(P)$.

2.1.3 Structural congruence

The *structural congruence* of processes, noted $\equiv$, is the least congruence containing α-equivalence, $\equiv_\alpha$, making $(P, \mid, 0)$ and $(P, +0)$ commutative monoids.

2.1.4 Operational Semantics

$$
\frac{|\vec{y}| = |\vec{z}|}{P_1 + x_0?(\vec{y}) \Rightarrow P \mid x_1!(\vec{z}) + P_2 \to P\{@\vec{z}/\vec{y}\}} \quad (\textsc{Comm})
$$

In addition, we have the following context rules:

$$
\frac{P \to P'}{P \mid Q \to P' \mid Q} \quad (\textsc{Par})
$$

$$
\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q} \quad (\textsc{Equiv})
$$

2.1.5 Bisimulation

DEFINITION 2.1.1. *An* observation relation, $\downarrow_{\mathcal{N}}$, *over a set of names,* $\mathcal{N}$, *is the smallest relation satisfying the rules below.*

$$\frac{x \in \mathcal{N}}{x!(\vec{y}) \downarrow_{\mathcal{N}} x} \quad \textit{(OUT-BARB)}$$

$$\frac{P \downarrow_{\mathcal{N}} x \text{ or } Q \downarrow_{\mathcal{N}} x}{P \mid Q \downarrow_{\mathcal{N}} x} \quad \textit{(PAR-BARB)}$$

We write $P \Downarrow_{\mathcal{N}} x$ *if there is* Q *such that* $P \Rightarrow Q$ *and* $Q \downarrow_{\mathcal{N}} x$.

Notice that $x?(y) \Rightarrow P$ has no barb. Indeed, in RHO-calculus as well as other asynchronous calculi, an observer has no direct means to detect if a sent message has been received or not.

DEFINITION 2.1.2. *An* $\mathcal{N}$-barbed bisimulation *over a set of names,* $\mathcal{N}$, *is a symmetric binary relation* $\mathcal{S}_{\mathcal{N}}$ *between agents such that* $P \mathcal{S}_{\mathcal{N}} Q$ *implies:*

1. *If* $P \rightarrow P'$ *then* $Q \Rightarrow Q'$ *and* $P' \mathcal{S}_{\mathcal{N}} Q'$.
2. *If* $P \downarrow_{\mathcal{N}} x$, *then* $Q \Downarrow_{\mathcal{N}} x$.

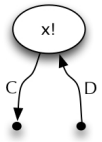
P *is* $\mathcal{N}$*-barbed bisimilar to* Q, *written* $P \dot{\approx}_{\mathcal{N}} Q$, *if* $P \mathcal{S}_{\mathcal{N}} Q$ *for some* $\mathcal{N}$*-barbed bisimulation* $\mathcal{S}_{\mathcal{N}}$.

3. CATEGORICAL MACHINERY

TBD

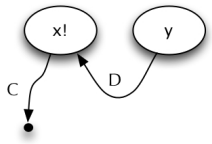
4. THE INTERPRETATION

TBD



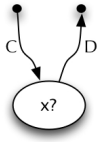
Interpreting names a morphisms, x : J -> D* ⊠ C

Figure 1: Interpretation of output



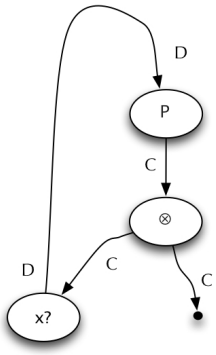
That means we can interpret output, x!(y), as connecting a source to the input of the morphism.

Figure 2: Interpretation of output - again



And the adjoint morphism, x : D* ⊠ C -> J, corresponds to input

Figure 3: Interpretation of input



This provides the interpretation of x?(y)P.

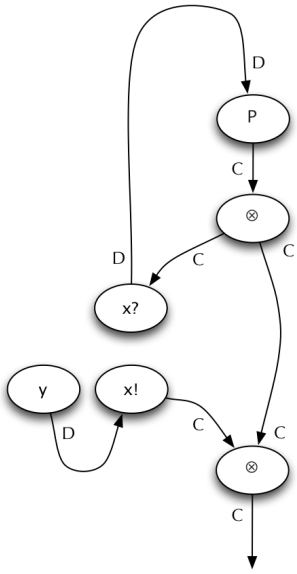
Figure 4: Interpretation of input guarded process

5. CONCLUSIONS AND FUTURE WORK

TBD

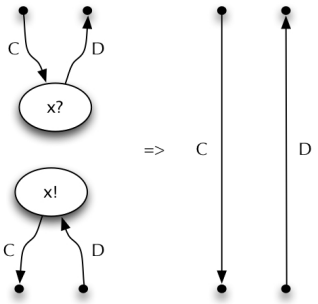
Acknowledgments.. TBD

6. REFERENCES



This provides the interpretation of x?(z)P | x!(y).

Figure 5: Interpretation of basic π-calculus redex



The co-unit of the adjunction provides a mechanism for synchronization and data flow.

Figure 6: Interpretation of adjunction-based rewrite template