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 lazy 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
stay@google.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 $(P, |, 0)$ and $(P, +, 0)$ into 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.*

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

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

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* $P \to P'$ *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

We take our models in bicartesian closed 2-categories; by this we mean 2-categories that have all finite products, infinite coproducts, and are closed with respect to the product. The 2-category Cat of categories, functors, and natural transformations is the principal example. We denote the terminal object by 1; the internal hom by a lollipop $\multimap$; duplication and deletion of an object X by $\Delta_X : X \to X \times X$ and $\delta_X : X \to 1$, respectively; and the type of lists of an object X by $X^* = 1 + X + X \times X + X \times X \times X + \cdots$.

4. THE INTERPRETATION

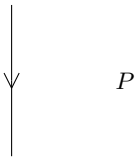
Given the BNF of a term calculus like that in section 2.1.1, we introduce an object in our 2-category for each parameter of the calculus. The pi calculus is parametric in a set of names and a set of processes, so we have objects N and P. We introduce 1-morphisms for each term constructor, a set of 1-morphisms for identifying reduction contexts, 2-morphisms for each reduction relation, and equations for structural equivalence.

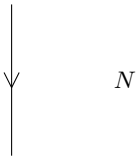
The theory of the pi calculus is the free such 2-category on

- an objects N for names,
- an object P for processes,
- a 1-morphism $0 \colon 1 \to P$,
- a 1-morphism $| \colon P \times P \to P$,
- a 1-morphism $? \colon N \times (N^* \multimap P) \to P$,
- a 1-morphism $! \colon N \times N^* \to P$,
- a 1-morphism $COMM \colon 1 \to P$,
- a 2-morphism *comm* encoding the COMM rule (see the diagram in section 4.1), and
- equations making $(P, |, 0)$ into a commutative monoid

TBD

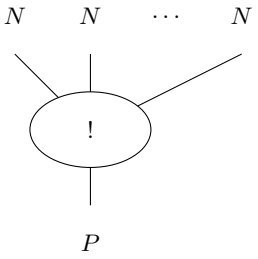
4.0.6 0-cells

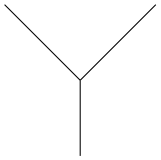


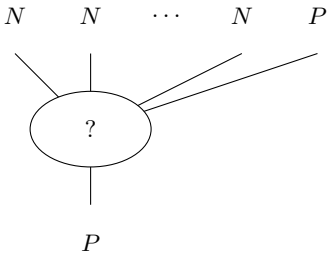


4.0.7 1-cells

P







N

4.1 2-cells

TBD

5. CONCLUSIONS AND FUTURE WORK

TBD

Acknowledgments.. TBD

6. REFERENCES