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

Michael 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. It's not only due to lack of space that we focus on a minimal presentation of the calculus. One of the goals has been to provide a modular semantics to address a range of different calculi and modeling options. For example, a significant bifurcation occurs in the treatment of names with Milner's original calculus hiding all internal structure of names, while the ρ-calculus variant provides a reflective version in which names are the codes or processes. The semantics presented here can be seen as a framework capable of providing a categorical interpretation of both variants.

Of particular interest to theoreticians and implementers, the semantics shines light on a key difference between the categorical and computational machinery it interprets. The latter is intrinsically lazy in the sense that all contexts where rewrites can apply must be explicitly spelled out (cf the context rules), while the former is intrinsically eager; in fact, one of the contributions of the paper is the delineation of an explicit control mechanism to prevent unwanted rewrites that would otherwise create an insurmountable divergence between the two formalisms. It is worth noting that even when modeling such fundamental calculi as the lazy lambda calculus, attempting to use higher categorical notion to capture a more operational semantics is still faced with this challenge. Fortunately, the techniques developed here extend to that setting, as we sketch at the end.

1.0.1 Related work

Montanari, et al have considered double category models of the π-calculus.

1.0.2 Organization of the rest of the paper

In the remainder of the paper we present the core fragment of the calculus we model followed by a manifest of the categorical equipment needed to faithfully model it. Then we give the semantics function an sketch a proof that the interpretation is fully abstract.

2. THE CALCULUS

Some examples of process expressions.

2.1 Our running process calculus

2.1.1 Syntax

$$
\begin{array}{lr}
P ::= 0 & \text{stopped process} \\
\quad \mid x?(y_1, \ldots, y_n) \Rightarrow P & \text{input} \\
\quad \mid x!(y_1, \ldots, y_n) & \text{output} \\
\quad \mid P \mid Q & \text{parallel}
\end{array}
$$

2.1.2 Free and bound names

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

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 $(P, |, 0)$ into commutative monoids.

2.1.4 Operational Semantics

$$\frac{|\vec{y}| = |\vec{z}|}{x?(\vec{y}) \Rightarrow P \mid x!(\vec{z}) \to P\{@\vec{z}/\vec{y}\}} \quad (\textsc{Comm})$$

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.*

$$\frac{x \in \mathcal{N}}{x!(\vec{y}) \downarrow_\mathcal{N} x} \quad (\textsc{Out-barb})$$

$$\frac{P \downarrow_\mathcal{N} x \text{ or } Q \downarrow_\mathcal{N} x}{P \mid Q \downarrow_\mathcal{N} x} \quad (\textsc{Par-barb})$$

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 2-categories with an underlying symmetric monoidal closed category; the 2-categories Cat (categories, functors, and natural transformations) and Rel (sets, relations, and implications) are examples. We denote the monoidal unit object by I, the tensor product by $\otimes$, the nth tensor power of an object X by $X^{\otimes n}$, and the internal hom by a lollipop $\multimap$.

4. THE INTERPRETATION

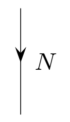
Given the abstract syntax of a term calculus like that in section 2.1.1, we introduce an object in our 2-category for each parameter of the calculus. We introduce 1-morphisms for each term constructor, 2-morphisms for each reduction relation, and equations for structural equivalence; we also add 1-morphisms to mark contexts in which reductions may occur.

The π-calculus is parametric in a set of names and a set of processes, so we have objects N and P. Since names can be reused in the π-calculus, we also add 1-morphisms and equations to make N be a cocommutative comonoid. We denote comultiplication by $\Delta\colon N \to N \otimes N$ and counit by $\delta\colon N \to I$. If the underlying category is cartesian, I is the terminal object, N is a comonoid in a unique way, and Δ and δ are duplication and deletion, respectively.

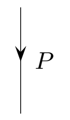
In the π-calculus, all reductions occur at the topmost context, so we have one unary morphism from P to P. There are some benefits to constructing the top context marker out of the existing binary morphism $|\colon P \otimes P \to P$ and a unary morphism $COMM\colon I \to P$; we'll talk about some of the benefits in the conclusion.

The theory of the π-calculus is the free symmetric monoidal closed 2-category on

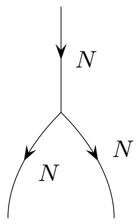
- an objects N for names,



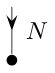
- an object P for processes,



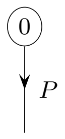
- a 1-morphism $\Delta\colon N \to N \otimes N$,



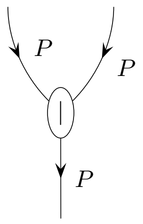
- a 1-morphism $\delta\colon N \to I$,



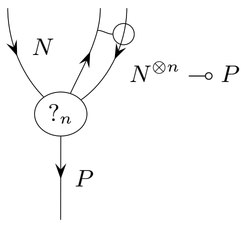
- a 1-morphism $0\colon I \to P$,



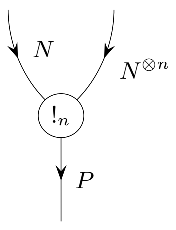
- a 1-morphism $|\colon P \otimes P \to P$,



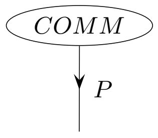
- a 1-morphism $?_n\colon N \otimes (N^{\otimes n} \multimap P) \to P$ for each natural number $n \geq 0$,



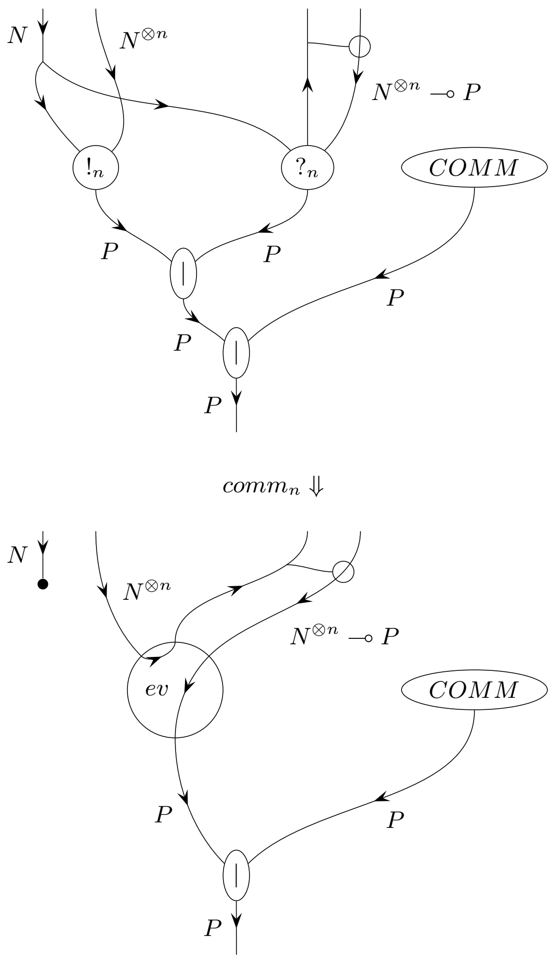
- a 1-morphism $!_n\colon N \otimes N^{\otimes n} \to P$ for each natural number $n \geq 0$,



- a 1-morphism $COMM\colon I \to P$,

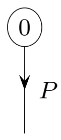


- equations making $(P, |, 0)$ into a commutative monoid,
- equations making (N, Δ, δ) into a cocommutative comonoid, and
- a 2-morphism $comm_n$ encoding the COMM rule for each natural number $n \geq 0$.

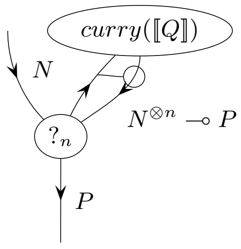


4.1 Semantics

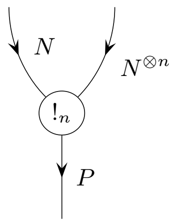
$[\![0]\!] :=$



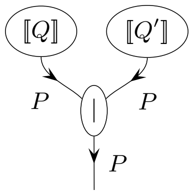
$[\![x?(y_1, \dots, y_n) \Rightarrow Q]\!] :=$



$[\![x!(y_1, \dots, y_n)]\!] :=$

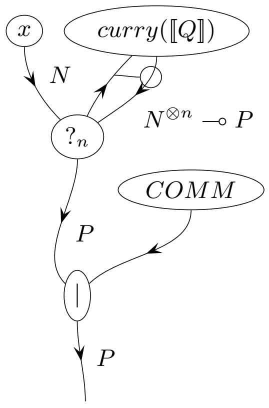


$[\![Q \mid Q']\!] :=$



$[\![Q]\!]_{top} := ([\![Q]\!]|COMM) \circ \mathcal{FN}(Q)$

The last point deserves a bit more explanation. At the top level, all remaining inputs are wired up to the corresponding free names and the result is placed in parallel with COMM. For example, suppose that the top level term is $x?(y_1, \dots, y_n) \Rightarrow Q$ and that only $y_1, \dots, y_n$ are free in Q; then only x is free in $x?(y_1, \dots, y_n) \Rightarrow Q$ and the interpretation is



4.1.1 Full abstraction

THEOREM 4.1.1 (FULL ABSTRACTION). $P \overset{\cdot}{\approx} Q \iff [\![P]\!] = [\![Q]\!]$

5. CONCLUSIONS AND FUTURE WORK

To model lazy lambda calculus, we use two unary contexts T and U, mark the topmost context with T, and use two rules:

- $T(App(l, r)) \Rightarrow U(App(T(l), r))$ for moving T to the leftmost subterm, and
- $U(App(T(Lam(x, M)), N) \Rightarrow T(M\{N/x\})$ for $\beta-$reduction.

TBD

Acknowledgments.. TBD

6. REFERENCES