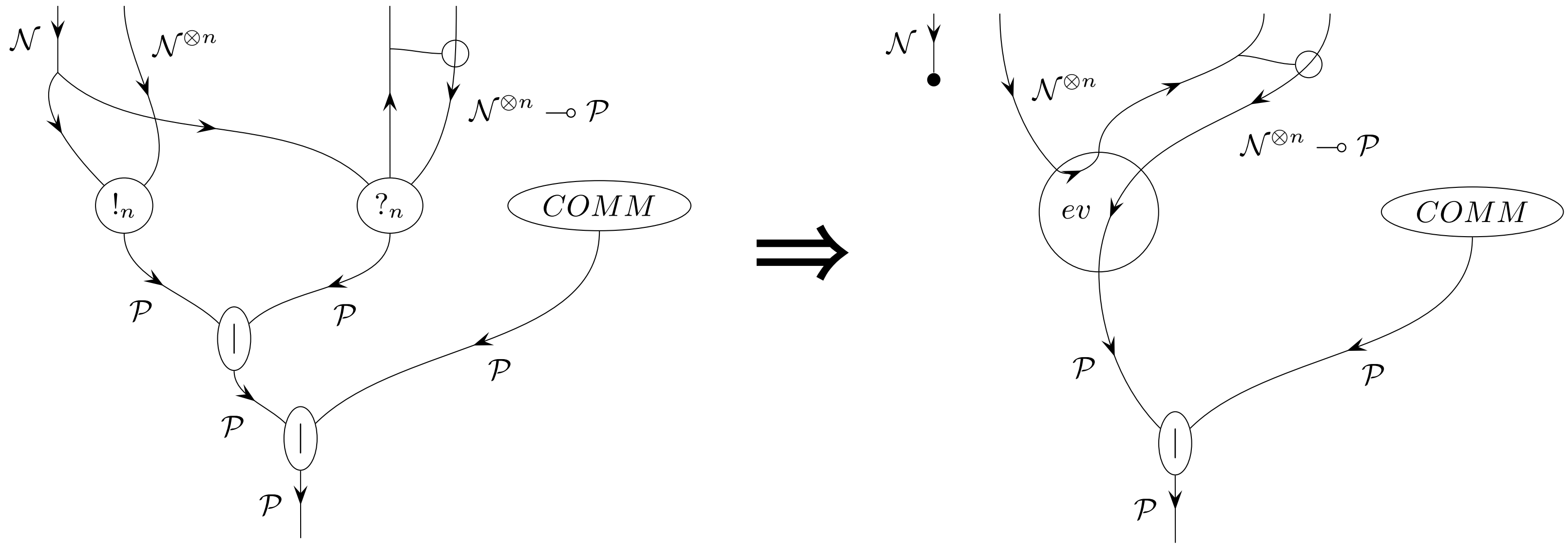
Higher category models of the π-calculus

The operational semantics and Curry-Howard



Mike Stay, L.G. Meredith

HDRA 2015

Higher category models of the π-calculus

Modern presentations of computational calculi generalize generators and relations style presentations of universal algebra

They are typically given in terms of

a grammar ≈ generators ≈ a free functor

a structural equivalence ≈ relations ≈ an algebra of a monad

bi-directional rewrites

an operational semantics no analog

uni-directional rewrites

Process calculi, like vector spaces, are two sorted algebraic structures

names	scalars
processes	vectors

dynamics internal: a relation on a single algebra

dynamics external: morphisms between algebras

Higher category models of the π-calculus

Syntax

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

Higher category models of the π-calculus

Free and bound names

$$\mathcal{FN}(0) \coloneqq \emptyset$$

$$\mathcal{FN}(x?(y_1, \ldots, y_n) \Rightarrow P) \coloneqq$$
$$\quad \{x\} \cup (\mathcal{FN}(P) \setminus \{y_1, \ldots y_n\})$$

$$\mathcal{FN}(x!(y_1, \ldots, y_n)) \coloneqq \{x, y_1, \ldots, y_n\}$$

$$\mathcal{FN}((\mathsf{new}\ x)P) \coloneqq \mathcal{FN}(P) \setminus \{x\}$$

$$\mathcal{FN}(P \mid Q) \coloneqq \mathcal{FN}(P) \cup \mathcal{FN}(Q)$$

Higher category models of the π-calculus

Structural congruence

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

$$(\mathsf{new}\ x)(\mathsf{new}\ x)P \equiv (\mathsf{new}\ x)P$$

$$(\mathsf{new}\ x)(\mathsf{new}\ y)P \equiv (\mathsf{new}\ y)(\mathsf{new}\ x)P$$

$$((\mathsf{new}\ x)P) \mid Q \equiv (\mathsf{new}\ x)(P \mid Q)$$

Higher category models of the π-calculus

Operational semantics

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

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

$$\frac{P \to P'}{(\mathsf{new}\ x)P \to (\mathsf{new}\ x)P'} \quad (\textsc{New})$$

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

Higher category models of the π-calculus

Bisimulation

$$\frac{}{x!(\vec{y}) \downarrow x}$$

$$\frac{P \downarrow x \;\; or \;\; Q \downarrow x}{P \mid Q \downarrow x}$$

DEFINITION 2.1.2. *An* barbed bisimulation, *is a symmetric binary relation* $\mathcal{S}$ *between agents such that* $P\ \mathcal{S}\ Q$ *implies:*

1. *If* $P \rightarrow P'$ *then* $Q \rightarrow Q'$ *and* $P'\ \mathcal{S}\ Q'$.
2. *If* $P \downarrow x$, *then* $Q \downarrow x$.

Higher category models of the π-calculus

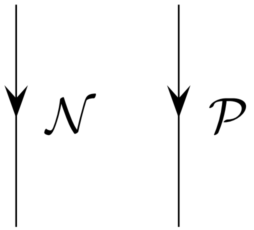
Bisimulation

P is barbed bisimilar to Q, written $P \stackrel{\cdot}{\approx} Q$, if $P \mathcal{S} Q$ for some barbed bisimulation $\mathcal{S}$.

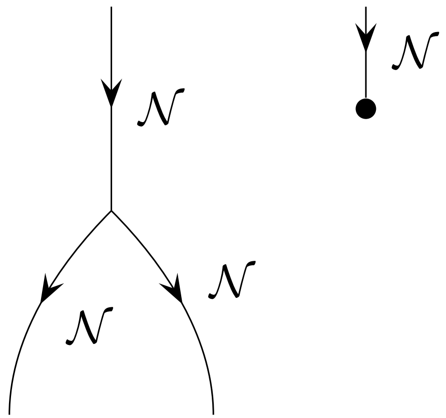
Higher category models of the π-calculus

The categorical machinery

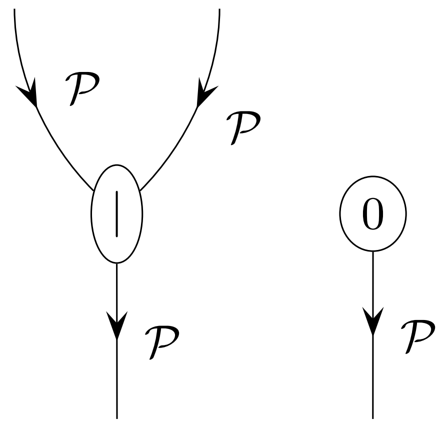
objects $\mathcal{N}$ for names and $\mathcal{P}$ for processes



1-morphisms $\Delta\colon \mathcal{N} \to \mathcal{N} \otimes \mathcal{N}$ and $\delta\colon \mathcal{N} \to I$



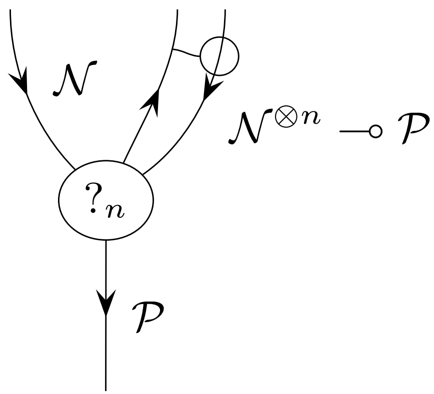
1-morphisms $|\colon \mathcal{P} \otimes \mathcal{P} \to \mathcal{P}$ and $0\colon I \to \mathcal{P}$,

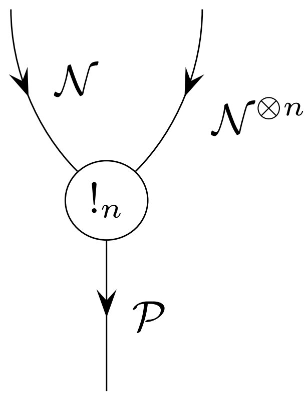


Higher category models of the π-calculus

The categorical machinery

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

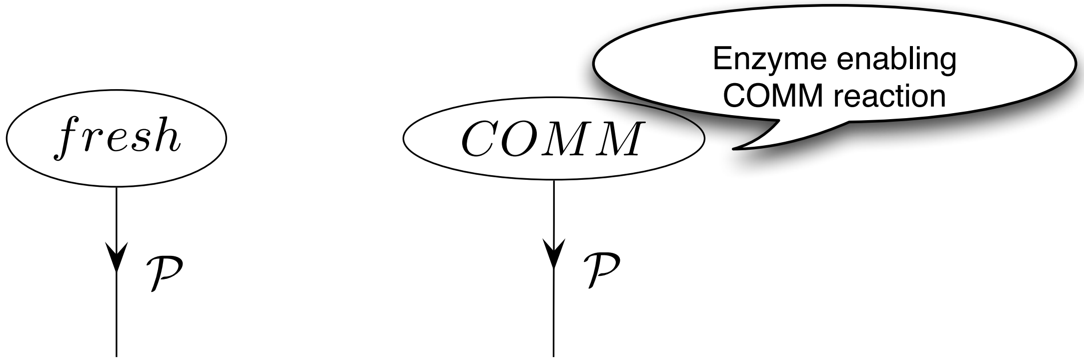




Higher category models of the π-calculus

The categorical machinery

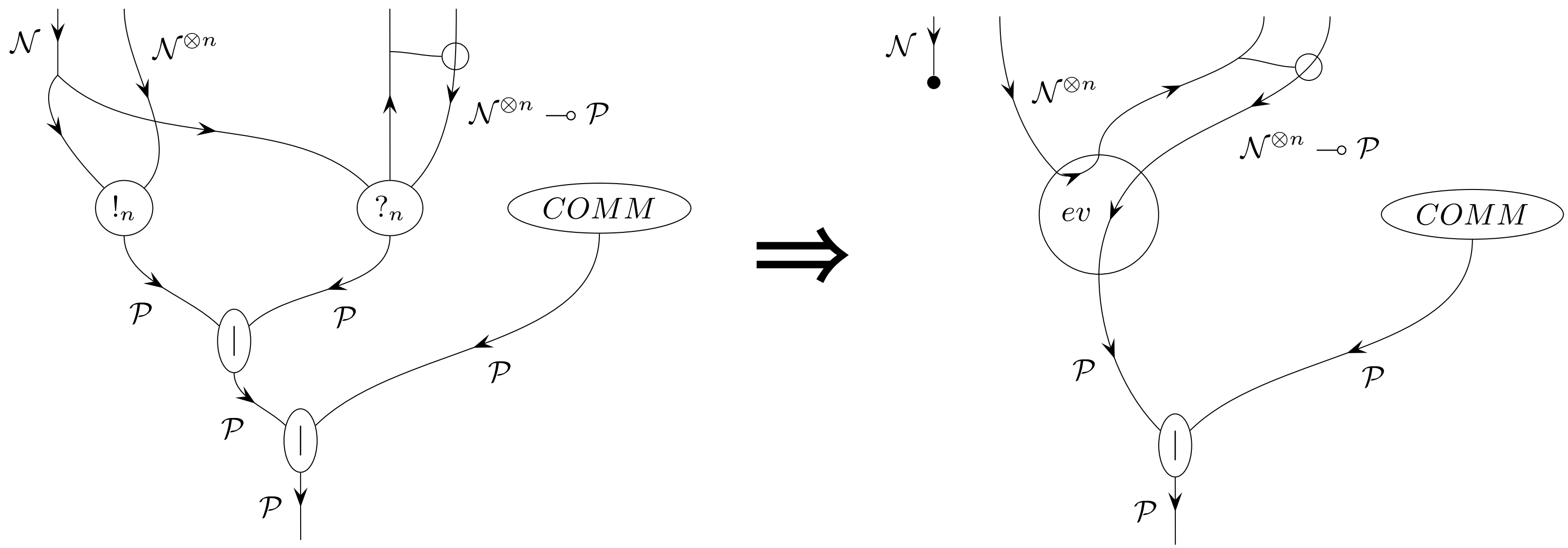
1-morphisms $fresh\colon I \to \mathcal{P}$ and $COMM\colon I \to \mathcal{P}$



Higher category models of the π-calculus

The categorical machinery

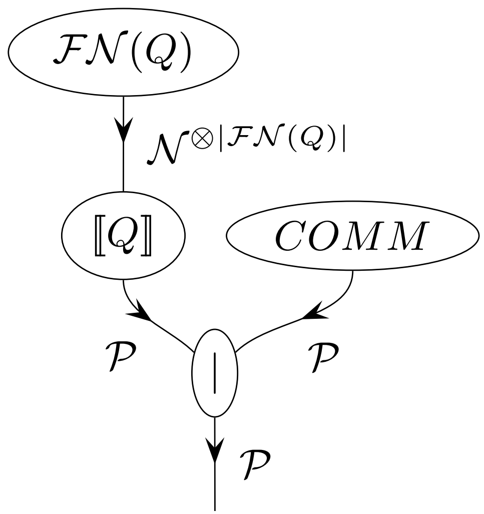
a 2-morphism $comm_n$ encoding the COMM rule
for each natural number $n \geq 0$.



Higher category models of the π-calculus

The semantics

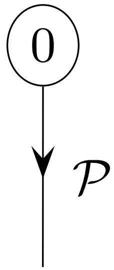
$[\![Q]\!]_{top} \coloneqq$



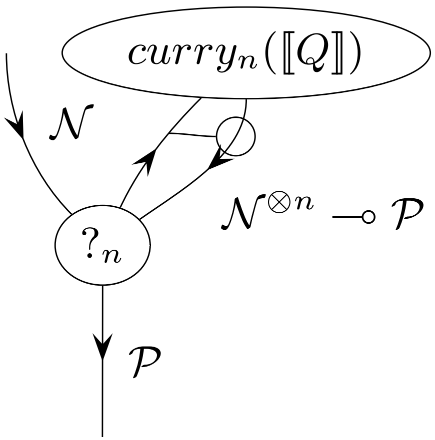
Higher category models of the π-calculus

The semantics

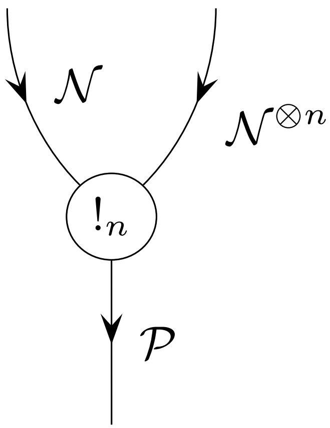
$[\![0]\!] \coloneqq$



$[\![x?(y_1, \ldots, y_n) \Rightarrow Q]\!] \coloneqq$



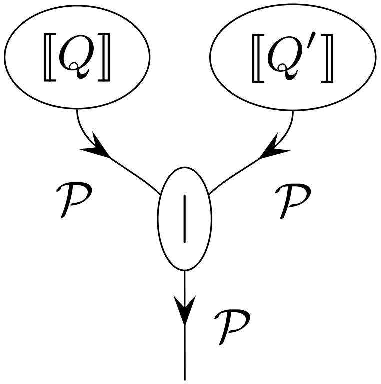
$[\![x!(y_1, \ldots, y_n)]\!] \coloneqq$



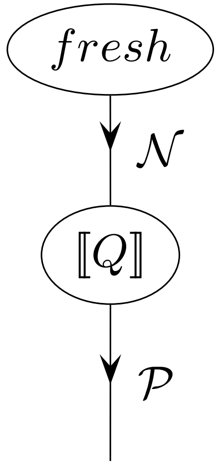
Higher category models of the π-calculus

The semantics

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



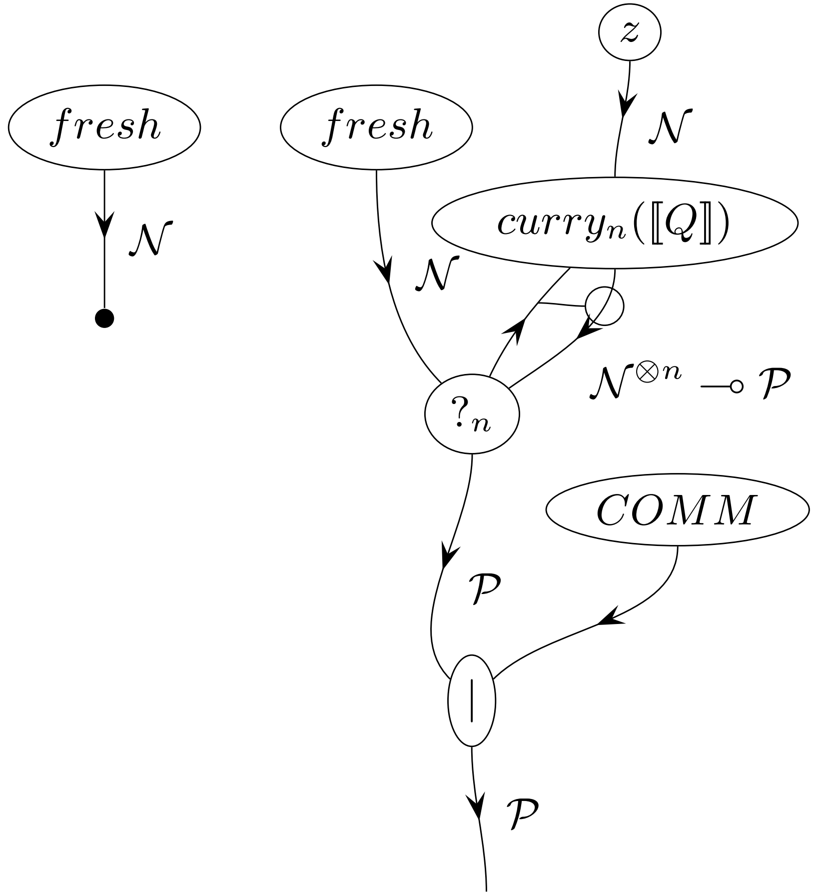
$[\![(\mathsf{new}\ x)Q]\!] :=$



Higher category models of the π-calculus

The semantics

$[\![(\mathsf{new}\ y)(\mathsf{new}\ x)x?(y_1, \ldots, y_n) \Rightarrow Q]\!]_{top}$



Higher category models of the π-calculus

The semantics

$$[\![x!(y_1, \ldots, y_n)]\!] \Downarrow [\![x]\!]$$

$$[\![P]\!] \Downarrow [\![x]\!] \text{ or } [\![Q]\!] \Downarrow [\![x]\!] \text{ implies } [\![P]\!] \mid [\![Q]\!] \Downarrow [\![x]\!]$$

Taken together these two notions provide an immediate lifting of the syntactic notion of bisimulation to a corresponding semantic notion, which we write, $\approx$.

Higher category models of the π-calculus

The semantics

THEOREM 4.1.1 (FULL ABSTRACTION).

$$P \stackrel{\cdot}{\approx} Q \quad \iff \quad [\![P]\!] \approx [\![Q]\!]$$

Higher category models of the π-calculus

Conclusions and future work

Mellies and Zeilberger types

Explicit computational resources

True concurrency

```
P,Q ::= 0                                   { }

    a![ v1, …, vn ]                         [| a |]( m ) ![ [| v1 |]( m ), …, [| vn |]( m ) ]

    a?( x1, …, xn )P                        for( [ x1, …, xn ] <- [| a |]( m ) ){
                                                [| P |]( m )( x1, …, xn )
                                            }

    P | Q                                    spawn{ [| P |]( m ) };spawn{ [| Q |]( m ) }

    (new a)P                                { val q = new Queue(); [| P |]( m[ a <- q ] ) }

    ( def X( x1, …, xn ) = P )[v1, …, vn]    object X {
                                                def apply( x1, …, xn ) = {
                                                   [| P |]( m ) ( x1, …, xn )
                                                }
                                             }

    X[v1, …, vn]                            X( [| v1 |]( m ), …, [| vn |]( m ) )


        [| - |]( - ) : ( π-calculus, Map[Symbol,Queue] ) -> Scala
```