The rho calculus 20 years on

L.G. Meredith[1]

CEO, F1R3FLY.io 9336 California Ave SW, Seattle, WA 98103, USA,
lgreg.meredith@gmail.com

Abstract. We discuss developments and advances in the rho calculus 20 years after its inception.

1 Introduction

This December will find my submission of the first paper about the rho calculus to be 20 years in the rear view mirror. A lot has happened since then. Apart from production implementations of scalable decentralized asset management systems based on the calculus, a lot of work has been done on the rho calculus. Indeed, just this May (2024) i discovered a conservative extension of the calculus that fills a hole in the account of reflection that has long bothered me. And, last April (2023) i found a fuzzy version of the rho calculus that is just waiting for someone to take further. In 2018 i discovered a variant of the calculus (which i dubbed the *space calculus*) that makes it a computational analog (in a certain precise sense) of solutions of the Einstein field equations.

Thankfully, i am not the only one who has found the rho calculus interesting enough to think about. Stay and Williams studied a variant they called the π-rho calculus as an example of a system that could be typed in their native types construction. And Stian Lybech showed that rho calculus is more expressive than the the π-calculus. The list goes on.

The aim of this note is to collect, and summarize in one place, developments over the last 20 years in the study of the rho calculus. i suspect that there are more researchers than i know about who are interested in the questions raised by the calculus. Hopefully, this note can serve that community, and possibly spark wider interest due to the discoveries to date.

1.1 Summary of contributions and outline of paper

More specifically, this note will develop the following.

- the modern, programming-language-friendly syntax for the calculus;
- a handful of useful syntactic sugar coatings that make the calculus a suitable semantic framework for a realistic protocol-oriented programming language, including, but not limited to:
  - adding first class values;
  - adding unforgeable names (the π-rho calculus);
  - adding syntax for joins and sequences and choice;
- a procedurally reflective rho calculus;

– the fuzzy variant of rho;
– multi-level agency mechanisms in rho;
– the space calculus;
– probabilistic and quantum variations of rho;
– revisiting the encodings of π into rho and vice versa.

But, we will also touch briefly on the techniques used to represent reflective computation in the rho calculus and demonstrate that they are not bound to this specific setting. In particular, they can be applied the the λ-calculus and to set theory, yielding variations of these theories that have very useful features.

Additionally, we will look at relating the rho calculus to physical intuition. There has been a great deal of work on using mobile process calculi to describe network protocols and concurrent algorithms. It is not too much of an abstraction to see how these techniques apply to physical processes such as chemical reactions, signaling regimes within and between cells. But the rho calculus can also be used to model ordinary classical physics problems. We illustrate some of these.

Finally, we will conclude with some thoughts about the foundations of mathematics and its relationship to agency. As a preview, i believe that set theory (and even to some extent category theory) are theories of *data structures*. Implicitly, the community has believed that (only) mathematicians have agency and their formalisms merely carry information content between these agents. But the advances in artificial intelligence (AI) tell a different story. To some extent AI has brought their formalisms to life and is on the verge of giving them agency. As a result, i want to argue that a proper foundation of mathematics needs to include a notion of agency and that the ingredients of a foundational account of agency are present in the mobile process calculi, generally, and a pragmatically foundational account is present in the rho calculus, specifically.

2 Concurrent process calculi and spatial logics

In the last thirty years the process calculi have matured into a remarkably powerful analytic tool for reasoning about concurrent and distributed systems. Process-calculus-based algebraical specification of processes began with Milner's Calculus for Communicating Systems (CCS) [20] and Hoare's Communicating Sequential Processes (CSP) [13], and continue through the development of the so-called mobile process calculi, e.g. Milner, Parrow and Walker's π-calculus [23], [24], Cardelli and Caires's spatial logic [4] [6] [7], or Meredith and Radestock's reflective calculi [18] [17]. The process-calculus-based algebraical specification of processes has expanded its scope of applicability to include the specification, analysis, simulation and execution of processes in domains such as:

– telecommunications, networking, security and application level protocols [1] [2] [14] [15];
– programming language semantics and design [14] [11] [10] [31];
– webservices [14] [15] [16];
– blockchain [19]

– and biological systems [8] [9] [26] [25].

Among the many reasons for the continued success of this approach are two central points. First, the process algebras provide a compositional approach to the specification, analysis and execution of concurrent and distributed systems. Owing to Milner's original insights into computation as interaction [21], the process calculi are so organized that the behavior —the semantics— of a system may be composed from the behavior of its components. This means that specifications can be constructed in terms of components —without a global view of the system— and assembled into increasingly complete descriptions.

The second central point is that process algebras have a potent proof principle, yielding a wide range of effective and novel proof techniques [29] [27] [28]. In particular, *bisimulation* encapsulates an effective notion of process equivalence that has been used in applications as far-ranging as algorithmic games semantics [3] and the construction of model-checkers [5]. The essential notion can be stated in an intuitively recursive formulation: a *bisimulation* between two processes P and Q is an equivalence relation E relating P and Q such that: whatever action of P can be observed, taking it to a new state P', can be observed of Q, taking it to a new state Q', such that P' is related to Q' by E and vice versa. P and Q are *bisimilar* if there is some bisimulation relating them. Part of what makes this notion so robust and widely applicable is that it is parameterized in the actions observable of processes P and Q, thus providing a framework for a broad range of equivalences and up-to techniques [29] all governed by the same core principle [28].

2.1 The syntax and semantics of the notation system

We now summarize a technical presentation of the calculus that embodies our theory of dynamics. The typical presentation of such a calculus follows the style of giving generators and relations on them. The grammar, below, describing term constructors, freely generates the set of processes, Proc. This set is then quotiented by a relation known as structural congruence and it is over this set that the notion of dynamics is expressed. This presentation is essentially that of [18] with the addition of polyadicity and summation. For readability we have relegated some of the technical subtleties to an appendix.

Notational interlude when it is clear that some expression t is a sequence (such as a list or a vector), and a is an object that might be meaningfully and safely prefixed to that sequence then we write $a : t$ for the sequence with a prefixed (aka "consed") to t. We write $t(i)$ for the ith element of t.

Process grammar

PROCESS	NAME
$P, Q ::= 0 \mid \mathsf{for}(\overrightarrow{y} \leftarrow x)P \mid x!(\overrightarrow{Q}) \mid *x \mid P\|Q$	$x, y ::= @P$

Note that $\overrightarrow{x}$ (resp. $\overrightarrow{P}$) denotes a vector of names (resp. processes) of length $|\overrightarrow{x}|$ (resp. $|\overrightarrow{P}|$). We adopt the following useful abbreviations.

$$\Pi\,\overrightarrow{P} := \Pi_{i=1}^{|\overrightarrow{P}|} P_i := P_1|\ldots|P_{|\overrightarrow{P}|}$$

Definition 1. Free and bound names *The calculation of the free names of a process, P, denoted $\mathsf{FN}(P)$ is given recursively by*

$$\mathsf{FN}(0) = \emptyset \qquad \mathsf{FN}(\mathsf{for}(\overrightarrow{y} \leftarrow x)(P)) = \{x\} \cup \mathsf{FN}(P) \setminus \{\overrightarrow{y}\} \qquad \mathsf{FN}(x!(\overrightarrow{P})) = \{x\} \cup \mathsf{FN}(\overrightarrow{P})$$

$$\mathsf{FN}(P|Q) = \mathsf{FN}(P) \cup \mathsf{FN}(Q) \qquad\qquad \mathsf{FN}(x) = \{x\}$$

where $\{\overrightarrow{x}\} := \{x_1, \ldots, x_{|\overrightarrow{x}|}\}$ and $\mathsf{FN}(\overrightarrow{P}) := \bigcup \mathsf{FN}(P_i)$.

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 $\mathsf{N}(P)$.

2.2 Substitution

We use Proc for the set of processes, $@\mathsf{Proc}$ for the set of names, and $\{\overrightarrow{y}/\overrightarrow{x}\}$ to denote partial maps, $s : @\mathsf{Proc} \to @\mathsf{Proc}$. A map, s lifts, uniquely, to a map on process terms, $\widehat{s} : \mathsf{Proc} \to \mathsf{Proc}$. Historically, it is convention to use σ to range over lifted subsitutions, $\widehat{s}$, to write the application of a substitution, σ to a process, P, with the substitution on the right, $P\sigma$, and the application of a substitution, s, to a name, x, using standard function application notation, $s(x)$. In this instance we choose not to swim against the tides of history. Thus,

Definition 2. *given $x = @P'$, $u = @Q'$, $s = \{u/x\}$ we define the lifting of s to $\widehat{s}$ (written below as σ) recursively by the following equations.*

$$0\sigma := 0$$

$$(P|Q)\sigma := P\sigma|Q\sigma$$

$$(\mathsf{for}(\overrightarrow{y} \leftarrow v)P)\sigma := \mathsf{for}(\overrightarrow{z} \leftarrow \sigma(v))((P\{\widehat{\overrightarrow{z}/\overrightarrow{y}}\})\sigma)$$

$$(x!(Q))\sigma := \sigma(x)!(Q\sigma)$$

$$(*y)\sigma := \begin{cases} Q' & y \equiv_{\mathsf{N}} x \\ *y & otherwise \end{cases}$$

where

$$\{\widehat{@Q/@P}\}(x) = \{@Q/@P\}(x) = \begin{cases} @Q & x \equiv_{\mathsf{N}} @P \\ x & otherwise \end{cases}$$

and z is chosen distinct from $@P$, $@Q$, the free names in Q, and all the names in R. Our α-equivalence will be built in the standard way from this substitution.

Definition 3. *Then two processes, P, Q, are alpha-equivalent if $P = Q\{\overrightarrow{y}/\overrightarrow{x}\}$ for some $\overrightarrow{x} \in \mathsf{BN}(Q), \overrightarrow{y} \in \mathsf{BN}(P)$, where $Q\{\overrightarrow{y}/\overrightarrow{x}\}$ denotes the capture-avoiding substitution of $\overrightarrow{y}$ for $\overrightarrow{x}$ in Q.*

Definition 4. *The* structural congruence $\equiv$ *between processes [30] is the least congruence containing alpha-equivalence and satisfying the commutative monoid laws (associativity, commutativity and $\mathsf{0}$ as identity) for parallel composition $|$.*

Definition 5. *The* name equivalence $\equiv_{\mathsf{N}}$ *is the least congruence satisfying these equations*

$$\textsc{Quote-drop}\quad @{*x} \equiv_{\mathsf{N}} x \qquad\qquad \textsc{Struct-equiv}\quad \frac{P \equiv Q}{@P \equiv_{\mathsf{N}} @Q}$$

The astute reader will have noticed that the mutual recursion of names and processes imposes a mutual recursion on alpha-equivalence and structural equivalence via name-equivalence. Fortunately, all of this works out pleasantly and we may calculate in the natural way, free of concern. The reader interested in the details is referred to the appendix **??**.

Remark 1. One particularly useful consequence of these definitions is that $\forall P.@P \notin \mathsf{FN}(P)$. It gives us a succinct way to construct a name that is distinct from all the names in P and hence fresh in the context of P. For those readers familiar with the work of Pitts and Gabbay, this consequence allows the system to completely obviate the need for a fresh operator, and likewise provides a canonical approach to the semantics of freshness.

Structural complexity Because all processes ground out in $\mathsf{0}$, we can define a notion of structural complexity via a recursive function.

Definition 6. *The structural complexity of a process (resp. name) is defined recursively via*

$$\begin{aligned} \#(\mathsf{0}) &:= 0 \\ \#(\mathsf{for}(y \leftarrow x)P) &:= \#(x) + \#(P) + 1 \\ \#(P) &:= \#(x) + \#(P) + 1 \\ \#(P|Q) &:= \#(P) + \#(Q) + 1 \\ \#(*x) &:= \#(x) + 1 \\ \#(@P) &:= \#(P) + 1 \end{aligned}$$

Example 1. Thus $\#(\mathsf{for}(\mathsf{0} \leftarrow @\mathsf{0})\mathsf{0}) = \#(@\mathsf{0} + 0 + 1) = 1 + 1 = 2$

Building on the structural complexity of a term, we can define the structural complexity of a substitution and the cost of applying a substitution, respectively.

Definition 7.

$$\#(\{y/x\} : \sigma) := \#(y) + \#(x) + \#(\sigma)$$
$$\#(P\{y/x\} : \sigma) := \#(\{y/x\}) * \mathsf{occurs}(P, x) + \#(P\sigma)$$

where $\mathsf{occurs}(P, x)$ *is the number of occurrences of* x *in* P*.*

Equipped with the structural features of the term language we can present the dynamics of the calculus.

2.3 Operational semantics

Finally, we introduce the computational dynamics. What marks these algebras as distinct from other more traditionally studied algebraic structures, e.g. vector spaces or polynomial rings, is the manner in which dynamics is captured. In traditional structures, dynamics is typically expressed through morphisms between such structures, as in linear maps between vector spaces or morphisms between rings. In algebras associated with the semantics of computation, the dynamics is expressed as part of the algebraic structure itself, through a reduction reduction relation typically denoted by $\rightarrow$. Below, we give a recursive presentation of this relation for the calculus used in the encoding.

$$\text{COMM}\ \frac{x_t \equiv_{\mathsf{N}} x_s, \quad |\overrightarrow{y}| = |\overrightarrow{Q}|}{\mathsf{for}(\overrightarrow{y} \leftarrow x_t)P \mid x_s!(\overrightarrow{Q}) \rightarrow P\{\overrightarrow{@Q}/\overrightarrow{y}\}} \qquad \text{PAR}\ \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q}$$

$$\text{EQUIV}\ \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}$$

We write $P \rightarrow$ if $\exists Q$ such that $P \rightarrow Q$ and $P \not\rightarrow$, otherwise.

Definition 8. COMM*-events: when* $P \rightarrow P'$ *we can record the information justifying this reduction step with an expression of the form* $\mathsf{COMM}_{(P,P')}(x_t, x_s, \sigma)$*, leaving off the subscript* $_{(P,P')}$ *when the context makes the source* (P) *and target* (P') *states clear. Likewise, if* $c = \mathsf{COMM}_{(P,P')}(x_t, x_s, \sigma)$*, we write* $\mathsf{src}(c) = P$*,* $\mathsf{trgt}(c) = P'$*,* $\mathsf{src}_{\mathsf{N}}(c) = x_s$*, and* $\mathsf{trgt}_{\mathsf{N}}(c) = x_t$ *and* $P \xrightarrow{c} P'$*.*

We use $\alpha, \beta, ...$ to range over reduction *paths*. That is, if

$$P_1 \overset{\mathsf{COMM}(x_{t_1}, x_{s_1}, \sigma_1)}{\rightarrow} P_2 \overset{\mathsf{COMM}(x_{t_2}, x_{s_2}, \sigma_2)}{\rightarrow} \cdots \overset{\mathsf{COMM}(x_{t_{n-1}}, x_{s_{n-1}}, \sigma_{n-1})}{\rightarrow} P_n,$$

then we may write

$$\alpha = \mathsf{COMM}(x_{t_1}, x_{s_1}, \sigma_1) : \mathsf{COMM}(x_{t_2}, x_{s_2}, \sigma_2) : \ldots : \mathsf{COMM}(x_{t_{n-1}}, x_{s_{n-1}}, \sigma_{n-1}).$$

Definition 9. *The set of paths between* P *and* Q*, written* $\mathsf{paths}(P, Q)$ *is defined by* $\mathsf{paths}(P, Q) := \{c : \alpha \ : P \xrightarrow{c} P', \alpha \in \mathsf{paths}(P', Q)\}$ *and* ϵ *denotes the empty path.*

2.4 Dynamic quote: an example

Anticipating something of what's to come, let $z = @P$, $u = @Q$, and $x = @y!(*z)$. Now consider applying the substitution, $\widehat{\{u/z\}}$, to the following pair of processes, $w!(y!(*z))$ and $w!(*x) = w!(*@y!(*z))$.

$$w!(y!(*z))\widehat{\{u/z\}} = w!(y!(Q))$$
$$w!(*x)\widehat{\{u/z\}} = w!(*x)$$

The body of the quoted process, $@y!(*z)$, is impervious to substitution, thus we get radically different answers. In fact, by examining the first process in an input context, e.g. $\mathsf{for}(z \leftarrow x)w!(y!(*z))$, we see that the process under the output operator may be shaped by prefixed inputs binding a name inside it. In this sense, the combination of input prefix binding and output operators will be seen as a way to dynamically construct processes before reifying them as names.

3 Replication

As mentioned before, it is known that replication (and hence recursion) can be implemented in a higher-order process algebra [30]. As our first example of calculation with the machinery thus far presented we give the construction explicitly in the ρ-calculus.

$$D_x := \mathsf{for}(y \leftarrow x)(x!(y)|*y)$$
$$!_x P := x!(D_x|P)|D_x$$

$$\begin{aligned}
&!_x P \\
&= \quad x!((\mathsf{for}(y \leftarrow x)(x!(y)|*y))|P)|\mathsf{for}(y \leftarrow x)(x!(y)|*y) \\
&\to \quad (x!(y)|*y)\{@(\mathsf{for}(y \leftarrow x)(*y|x!(y)))|P/y\} \\
&= x!(@(\mathsf{for}(y \leftarrow x)(x!(y)|*y))|P)|(\mathsf{for}(y \leftarrow x)(x!(y)|*y))|P \\
&\to \quad \ldots \\
&\to^* \quad P|P|\ldots
\end{aligned}$$

Of course, this encoding, as an implementation, runs away, unfolding $!P$ eagerly. A lazier and more implementable replication operator, restricted to input-guarded processes, may be obtained as follows.

$$!\mathsf{for}(v \leftarrow u)P := x!(\mathsf{for}(v \leftarrow u)(D(x)|P))|D(x)$$

Remark 2. Note that the lazier definition still does not deal with summation or mixed summation (i.e. sums over input and output). The reader is invited to construct definitions of replication that deal with these features.

Further, the definitions are parameterized in a name, x. Can you, gentle reader, make a definition that eliminates this parameter and guarantees no accidental interaction between the replication machinery and the process being replicated – i.e. no accidental sharing of names used by the process to get its work done and the name(s) used by the replication to effect copying. This latter revision of the definition of replication is crucial to obtaining the expected identity $!!P \sim !P$.

Remark 3. The reader familiar with the lambda calculus will have noticed the similarity between D and the paradoxical combinator.

[Ed. note: the existence of this seems to suggest we have to be more restrictive on the set of processes and names we admit if we are to support no-cloning.]

Bisimulation The computational dynamics gives rise to another kind of equivalence, the equivalence of computational behavior. As previously mentioned this is typically captured *via* some form of bisimulation.

The notion we use in this paper is derived from weak barbed bisimulation [22].

Definition 10. *An* observation relation*, $\downarrow_{\mathcal{N}}$, over a set of names, $\mathcal{N}$, is the smallest relation satisfying the rules below.*

$$\frac{y \in \mathcal{N},\ x \equiv_{\mathsf{N}} y}{x!(v) \downarrow_{\mathcal{N}} x} \qquad (\textsc{Out-barb})$$

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

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

Definition 11. *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 $\mathcal{S}_{\mathcal{N}}$.

Contexts One of the principle advantages of computational calculi from the λ-calculus to the π-calculus is a well-defined notion of context, contextual-equivalence and a correlation between contextual-equivalence and notions of bisimulation. The notion of context allows the decomposition of a process into (sub-)process and its syntactic environment, its context. Thus, a context may be thought of as a process with a "hole" (written $\Box$) in

it. The application of a context K to a process P, written $K[P]$, is tantamount to filling the hole in K with P. In this paper we do not need the full weight of this theory, but do make use of the notion of context in the proof the main theorem.

$$\begin{array}{l}\text{CONTEXT}\\ K ::= \Box \mid \mathsf{for}(\overrightarrow{y} \leftarrow x)K \mid x!(\overrightarrow{P}, K, \overrightarrow{Q}) \mid K|P\end{array}$$

Definition 12 (contextual application). *Given a context K, and process P, we define the* contextual application*, $K[P] := K\{P/\Box\}$. That is, the contextual application of K to P is the substitution of P for $\Box$ in K.*

Remark 4. Note that we can extend the definition of free and bound names to contexts.

Contextual duality Note that contexts extend the quotation operation to a family of operations from processes to names. Given a context, K, we can define a *nominal context*, $@M$ by $@K[P] := @(K[P])$. To foreshadow what is to come we observe that these operations enjoy a duality with processes very much like the duality between vectors and maps from vectors to scalars.

Further, because the calculus is essentially higher-order, we have a correspondence between contexts and processes. More specifically, given a name x and a context K we can construct K^*_x such that

$$K^*_x|x!(P) \to K[P]$$

namely,

$$K^*_x := \mathsf{for}(y \leftarrow x)K[*y]$$

This correspondence mirrors the usual correspondence between vectors and duals, where given a vector v we can produce its dual $w \mapsto v \cdot w$ taking a vector w to the dot product $v \cdot w$.

3.1 Additional notation

We achieve some notational compression with the following convention

$$\mathsf{for}(y_1 \leftarrow x_1;\ \ldots; y_n \leftarrow x_n)P := \mathsf{for}(y_1 \leftarrow x_1)\mathsf{for}(y_2 \leftarrow x_2)\ldots\mathsf{for}(y_n \leftarrow x_n)P$$

Even though we already have the notation $x = @P$, allowing us to pick out the process a name quotes, it will be convenient to introduce an alternate notation, $\dot{x}$, when we want to emphasize the connection to the use of the name. Note that, by virtue of name equivalence, $@\dot{x} \equiv_{\mathsf{N}} x$; so, the notation is consistent with previous definitions.

Further, because names have structure it is possible to effect substitutions on the basis of that structure. This means we need to upgrade our notation for substitutions, which we accomplish by adapting comprehension notation. Thus,

$$P\{y/x : x \in S\}$$

is interpreted to mean the process derived from P by replacing (in a capture-avoiding manner) each occurrence of x in S by y. For example,

$$P\{@(\dot{x} \mid \dot{x})/x : x \in \mathsf{FN}(P)\}$$

will replace each (occurrence) of a free name x in P by $@(\dot{x} \mid \dot{x})$.

When the context makes it clear that the substitution is over all of $\mathsf{FN}(P)$ we drop the predicate, writing $P\{f(x)\}$ instead of

$$P\{f(x)/x : x \in \mathsf{FN}(P)\}$$

Of course, dual to operators for expanding substitutions we need operators for contracting them.

$$\{y/x : x \in S\} \setminus S' := \{y/x : x \in S \setminus S'\}$$

Also, we will avail ourselves of the notation x^L and x^R to denote injections of a name into disjoint copies of the name space. There are numerous ways to accomplish this. One example can be found in [18]. This notation overloads to vectors of names: $\overrightarrow{x}^\pi := (x_i^\pi \ : \ 0 \leq i < |\overrightarrow{x}|)$ where $\pi \in \{L, R\}$.

We also use $P^\Box := P|\Box$.

In [18] an interpretation of the new operator is given. It turns out that there are several possible interpretations all enjoying the requisite algebraic properties of the operator (see [22]). We will therefore make liberal use of $(\mathsf{new}\ \overrightarrow{x})P$.

3.2 Extensions to the calculus

Values While it is standard in calculi such as the λ-calculus to define a variety of common values such as the natural numbers and booleans in terms of Church-numeral style encodings, it is equally common to simply embed values directly into the calculus. Not being higher-order, this presents some challenges for the π-calculus, but for the rho-calculus, everything works out very nicely if we treat values, e.g. the naturals, the booleans, the reals, etc as processes. This choice means we can meaninfully write expressions like $x!(\mathsf{5})$ or $u!(\mathsf{true})$, and in the context $\mathsf{for}(y \leftarrow x)P[*y]|x!(\mathsf{5})$ the value $\mathsf{5}$ will be substituted into P. Indeed, since operations like addition, multiplication, etc. can also be defined in terms

of processes, it is meaningful to write expressions like $\mathsf{5}|+|\mathsf{1}$, and be confident that this expression will reduce to a process representing $\mathsf{6}$. Thus, we can also use standard mathematical expressions, such as $\mathsf{5}+\mathsf{1}$, as processes, and know that these will evaluate to their expected values. Further, when combined with for-comprehensions, we can write algebraic expressions, such as $\mathsf{for}(y \leftarrow x)\mathsf{5}{+}{*}y$, and in contexts like $(\mathsf{for}(y \leftarrow x)\mathsf{5}{+}{*}y)|x!(\mathsf{1})$ this will evaluate as expected, producing the process (aka value) $\mathsf{6}$.

With these conventions in place it is useful to reduce the proliferation of $*$'s, by adopting a pattern-matching convention. Thus, we write $\mathsf{for}(@v \leftarrow x)P$ to denote binding v to the value passed and not the *name* of the value. Hence, we may write $\mathsf{for}(@v \leftarrow x)\mathsf{5}+\mathsf{v}$ without any loss of clarity, confident that this translates unambiguously into the formal calculus presented above. We achieve even greater compression and a more familiar notation if we also adopt the notation

$$\mathsf{let}\ x = v\ \mathsf{in}\ P := (\mathsf{new}\ u)(\mathsf{for}(@x \leftarrow u)P)|u!(v)$$

and generalized to nested expressions via

$$\mathsf{let}\ x_1 = v_1;\ \ldots; x_n = v_n\ \mathsf{in}\ P := (\mathsf{new}\ \overrightarrow{u})\mathsf{for}(x_1 \leftarrow u_1;\ \ldots; x_n \leftarrow u_n)P|\Pi u_i!(v_i)$$

Mixed summation The presentation given so far is often referred to as the polyadic, asynchronous version of the rho-calculus. And all of the syntactic sugar is just that: sugar. Values and let expressions can be desugared back down to the original calculus. However, the current investigation is made simpler if we expand to a version of the calculus that includes mixed summation, that is non-deterministic choice over both guarded input (for-comprehension) and output. Although there is an encoding of the calculus with mixed summation to the asynchronous polyadic calculus, it is not par-preserving. That is, if $[\![-]\!]_{async} : \mathsf{MixSumProc} \to \mathsf{Proc}$ is a mapping from the rho-calculus with mixed summation to the asynchronous polyadic rho-calulus, then it cannot be the case that

$$[\![P|Q]\!] = [\![P]\!]|[\![Q]\!]$$

for any such encoding. For good measure we throw in synchronous communication, but it is the mixed summation that constitutes the real jump in expressive power. We call this out because if we are going to relate quantum computing to concurrent computing, it is important to track the points where there are significant increases in expressive power of our target language.

Because Milner's presentation of the polyadic π-calculus with mixed summation is so parsimonious we use it as a template for a similar version of the rho-calculus.

SUMMATION
$$M, N ::= \mathsf{0} \mid x.A \mid M + N$$

AGENT
$$A ::= (\overrightarrow{x})P \mid [\overrightarrow{P}]Q$$

PROCESS
$$P, Q ::= M \mid P|Q \mid {*}x$$

NAME
$$x ::= @P$$

In this presentation we adopt the syntactic conventions

$$\mathsf{for}(\overrightarrow{y} \leftarrow x)P := x.(\overrightarrow{y})P \qquad\qquad x!(\overrightarrow{Q});P := x.[\overrightarrow{Q}]P$$

The structural equivalence is modified thusly.

Definition 13. *The* structural congruence $\equiv$ *between processes is the least congruence containing alpha-equivalence and satisfying the commutative monoid laws (associativity, commutativity and* $\mathsf{0}$ *as identity) for parallel composition* $|$ *and summation* $+$*.*

The COMM rule is modified to incorporate non-deterministic choice.

$$\text{COMM}\quad \frac{x_t \equiv_{\mathsf{N}} x_s, \quad |\overrightarrow{y}| = |\overrightarrow{Q}|}{(R_1 + \mathsf{for}(\overrightarrow{y} \leftarrow x_t)P) \mid (x_s!(\overrightarrow{Q}).P' + R_2) \rightarrow P\{\overrightarrow{@Q}/\overrightarrow{y}\}|P'}$$

And contexts are likewise extended in the obvious manner.

$$\text{SUMMATION-CONTEXT}\quad K_M ::= \Box \mid x.K_A \mid K_M + M \qquad\qquad \text{AGENT-CONTEXT}\quad K_A ::= (\overrightarrow{x})K_P \mid [\overrightarrow{P}, K_P, \overrightarrow{P'}]Q \mid [\overrightarrow{P}]K_P$$

$$\text{PROCESS-CONTEXT}\quad K_P ::= K_M \mid P|K_P$$

The reader can check that all the notational conventions, such as

$$\mathsf{for}(y_1 \leftarrow x_1;\ \ldots; y_n \leftarrow x_n)P$$

$$\mathsf{let}\ x_1 = v_1;\ \ldots; x_n = v_n\ \mathsf{in}\ P$$

adopted above still make sense for the calculus extended with mixed summation.

Annihilation Another important variation has to do with the rewrite rules. There is a recursive version of the COMM-rule that aligns with intuitions about the recursive nature of behavior in compositionally defined agents. The maths make it clearer than the English. First, we define what it means for two processes to annihilate each other.

Definition 14. *Annihilation: Processes* P *and* Q *are said to annihilate one another, written* $P \perp Q$*, just when* $\forall R.P|Q \rightarrow^* R \Rightarrow R \rightarrow^* \mathsf{0}$*.*

Thus, when $P \perp Q$, all rewrites out of $P|Q$ eventually lead to $\mathsf{0}$. Evidently, $P \perp Q \iff Q \perp P$, and $\mathsf{0} \perp \mathsf{0}$. Naturally, we can extend annihilation to names: $x \perp y \iff \dot{x} \perp \dot{y}$.

Annihilation affords a new version of the COMM-rule:

COMM

$$\frac{x_t \perp x_s, \quad |\overrightarrow{y}| = |\overrightarrow{Q}|}{(R_1 + \mathsf{for}(\overrightarrow{y} \leftarrow x_t)P) \mid (x_s!(\overrightarrow{Q}).P' + R_2) \to P\{\overrightarrow{@Q}/\overrightarrow{y}\}|P'}$$

All annihilation-based reduction happens in terms of reductions that happen at a lower degree of quotation, and grounds out in the fact that $\mathsf{0}\perp\mathsf{0}$ and thus $@\mathsf{0}\perp@\mathsf{0}$.

Example 2. For example, let $P_1 := \mathsf{for}(@\mathsf{0} \leftarrow @\mathsf{0})\mathsf{0}|@\mathsf{0}!(\mathsf{0})$. Then $P_1 \to \mathsf{0}$ because $\mathsf{0}\perp\mathsf{0}$. Suppose now that we set $x_0^-, x_0^+ := @\mathsf{0}$. Then, we can write P_1 as $\mathsf{for}(x_0^- \leftarrow x_0^-)\mathsf{0}|x_0^+!(\mathsf{0})$. Now, set

$$x_1^- := @(\mathsf{for}(x_0^- \leftarrow x_0^-)\mathsf{0}) \qquad\qquad x_1^+ := @(x_0^+!(\mathsf{0}))$$

Then define $P_2 := \mathsf{for}(x_1^- \leftarrow x_1^-)\mathsf{0}|x_1^+!(\mathsf{0})$. Then $P_2 \to \mathsf{0}$ because $x_1^-\perp x_1^+$, and hence $\mathsf{for}(x_1^- \leftarrow x_1^-)\mathsf{0}\perp x_1^+!(\mathsf{0})$.

More generally, set

$$x_i^- := @(\mathsf{for}(x_{i-1}^- \leftarrow x_{i-1}^-)\mathsf{0}) \qquad\qquad x_i^+ := @x_{i-1}^+!(\mathsf{0})$$

$$P_i := \mathsf{for}(x_{i-1}^- \leftarrow @x_{i-1}^-)\mathsf{0}|@x_{i-1}^+!(\mathsf{0})$$

Then $P_i \to \mathsf{0}$ and hence $x_i^-\perp x_i^+$.

This allows for a measure of reduction complexity.

Definition 15. *Define the complexity of a* COMM*-event* $c = \mathsf{COMM}_{(P,P')}(x_t, x_s, \sigma)$ *in terms of the recursive function*

$$\begin{aligned} \#(c) &:= \Sigma_{\alpha \in \mathsf{paths}(\dot{x_t}|\dot{x_s},\mathsf{0})} \#(P', \sigma) + \#(\alpha) \\ \#(c : \alpha) &:= \#(c) + \#(\alpha) \\ \#(\epsilon) &:= 0 \end{aligned}$$

Note that we can relativize annihilation.

Definition 16. *Relative annihilation: Processes* P *and* Q *are said to annihilate to* R*, written* $R \vdash P\perp Q$*, just when* $\forall R'.P|Q \to^* R' \Rightarrow R' \to^* R$*.*

Clearly, when such an R exists for P and Q, it is unique. Thus, we can define a partial function, $(- \odot -) : \mathsf{Proc} \times \mathsf{Proc} \to \mathsf{Proc}_\perp$ via

$$P \odot Q := \begin{cases} R & \exists R.R \vdash P\perp Q \\ \perp & otherwise \end{cases}$$

We can think of $\odot$ as a form of *evaluation* for the confluent, terminating fragment of Proc[1].

[1] Indeed, confluent and terminating computations are probably as good a candidate, as any, for the notion of *data*.

4 Interpretation of QM

4.1 Supporting definitions

To provide our interpretation of quantum mechanics we need to develop a number of supporting definitions. As the reader familiar with process algebraic systems can readily verify, these definitions make *essential* use of the reflective operations and as such identify this calculus as uniquely suited to this particular task.

Among these operations we find a notion of *multiplication* of names that interacts well with a notion of *tensor product* of processes. Even more intriguingly, we find a notion of a *dual* to a process in the form of maps from processes to names. While notions of composite names have been investigated in the process algebraic literature, it is the fact that names reflect process structure that enables the collection of duals to enjoy an algebraic structure dual to the collection of processes (i.e. there are operations available to duals that reflect the operations on processes). Moreover, it is this structure that enables an effective definition of inner product.

Multiplication

$$@P \cdot @Q := @(P|Q) \qquad \textit{equivalently} \qquad x \cdot y := @(\dot{x} \mid \dot{y})$$

$$@Q \cdot P := P\{@(Q|R)/@R : @R \in \mathsf{FN}(P)\}$$

$$\textit{equivalently}$$

$$x \cdot P := P\{@(\dot{x} \mid \dot{z})/z : z \in \mathsf{FN}(P)\}$$

Discussion The first equation needs little explanation; the second says that each free name of the process is replaced with the multiplication of that name by the scalar. Multiplication of a scalar (name) by a state (process) results in a process all the names of which have been 'moved over' by parallel composition with the process the scalar quotes. There is a subtlety that the bound names have to be manipulated so that multiplied names aren't accidentally captured. Since there are many ways to achieve this we simply demand that multiplication not accidentally capture free names.

Remark 5. The reader is invited to verify that for all $x, y, z \in @\mathsf{Proc}$ and $P \in \mathsf{Proc}$

$$x \cdot @0 \equiv x \qquad x \cdot y \equiv y \cdot x \qquad x \cdot (y \cdot z) \equiv (x \cdot y) \cdot z$$

$$@0 \cdot P \equiv P$$

$$x \cdot (y \cdot P) \equiv (x \cdot y) \cdot P$$

$$x \cdot (P|Q) \equiv (x \cdot P)|(x \cdot Q)$$

$$x \cdot (P+Q) = (x \cdot P)+(x \cdot Q)$$

Contexts and duality As mentioned previously, contexts are going to play in the role of duals to vectors. Or in Dirac's nomenclature, K plays bra to P's ket. As such, we will want something that acts like an inner product, $K \cdot P$. Note that if names are scalars, then we expect that $K \cdot P$ to yield a name.

$$K \cdot P := @((K[P])\sigma_{\otimes}(K[\mathsf{0}], P))$$

where

$$\sigma_{\otimes}(P, Q) := \{w_1 \cdot x/x, w_2 \cdot y/y : x \in \mathsf{FN}(P), y \in \mathsf{FN}(Q), w_1 = \mathsf{NF}(Q), w_2 = \mathsf{NF}(P)\}$$
$$\mathsf{NF}(P) := @\Pi_{u \in \mathsf{FN}(P)}\ \dot{u}$$

The definition may not seem intuitive at first. However, all that's happening is guarding against unwarranted interaction, as we will see shortly. In the meantime let us formally specify the duality between contexts and processes.

$$P^{\perp} := P|\Box \qquad\qquad K^{\perp} := K[0]$$

We can check that for all P and for contexts of the form $K = P|\Box$ for some P the operation $(-)^{\perp}$ is involutive.

$$(P^{\perp})^{\perp} = (P|\Box)^{\perp} = P|0 = P \qquad\qquad (K^{\perp})^{\perp} = (K[0])^{\perp} = (P)^{\perp} = P|\Box$$

More generally, if K is not of the form $P|\Box$ then define

$$K^{\perp} := K[*(@K[0])]$$

Using the observation in remark () we note that $K^{\perp}$ has a *unique* decomposition, because $K[0]$ cannot include a name of the form $@K[0]$. Thus, we can provide a definition for $P^{\perp}$ that replaces $*(@K[0])$ with $\Box$, i.e.

$$P^{\perp} := P\{\Box/*(@K[0])\}$$

The reader can check that this is involutive for all P and K. We shall primarily be dealing with contexts of the form $P|\Box$, and so will typically use the simpler definition.

Following a similar line of reasoning we can define

$$P^{\perp} \cdot x := (x \cdot P)^{\perp}$$

Our definitions allow us to verify:

$$P^{\perp} \cdot Q := @(P|Q)\{w_1 \cdot x/x, w_2 \cdot y/y : x \in \mathsf{FN}(P), y \in \mathsf{FN}(Q), w_1 = \mathsf{NF}(Q), w_2 = \mathsf{NF}(P)\}$$

Outer product We can define an outer product of processes

$$P \otimes Q := (P|Q)\{w_1 \cdot x/x, w_2 \cdot y/y : x \in \mathsf{FN}(P), y \in \mathsf{FN}(Q), w_1 = \mathsf{NF}(Q), w_2 = \mathsf{NF}(P)\}$$

which is consistent with inner product and respects the usual identities

$$(u \cdot P) \otimes Q = P \otimes (u \cdot Q)$$

as the reader may quickly check.

$$\begin{aligned}
(x \cdot P) \otimes Q & \\
&= (u \cdot P|Q)\sigma_{\otimes}((u \cdot P), Q) \\
&= (u \cdot P)\sigma_{\otimes}((u \cdot P), Q)|Q\sigma_{\otimes}((u \cdot P), Q) \\
&= (u \cdot P)\{\mathsf{NF}(Q) \cdot u \cdot x\}|Q\{\mathsf{NF}(u \cdot P) \cdot y\} \\
&= P\{\mathsf{NF}(u \cdot Q) \cdot x\}|(u \cdot Q)\{\mathsf{NF}(P) \cdot (u \cdot y)\} \\
&= P\sigma_{\otimes}(P, (u \cdot Q))|(u \cdot Q)\sigma_{\otimes}(P, (u \cdot Q)) \\
&= (P|u \cdot Q)\sigma_{\otimes}(P, (u \cdot Q)) \\
&= P \otimes (u \cdot Q)
\end{aligned}$$

Superposition as summation In this interpretation superposition corresponds to summation. That is, if P and Q are going to represent states, then $P + Q$ represents the superposition of those states. In terms of the evolution of states, this choice is natural and intuitive. It also suggests a definition for addition of scalars.

$$x + y := @(\dot{x} + \dot{y})$$

Intriguingly, in an interleaving-style operational semantics processes correspond to the sum of all paths (sequences of actions), which would give the usual identities for the distribution of multiplication and addition.

Evaluation The technique underlying the definitions for multiplication can be extended to $\odot$ in the obvious manner, giving us the right to write expressions like $x \odot P$, etc.

Dirac notation Here we show the uncanny correspondence between the rho-calculus operators and the basic operators of quantum mechanics in Dirac style presentation and in general adopt a notation that emphasizes the roles each object is playing.

quantum mechanics	process calculus
$\|P\rangle$	P
$\langle P\|$	$P^\perp$
$\langle P\|Q\rangle$	$P^\perp \cdot Q$
$\|P\rangle \otimes \|Q\rangle$	$P \otimes Q$
$\|P\rangle + \|Q\rangle$	$P + Q$

Table 1. QM - process calculi correspondences

These definitions respect the well known identities. For example

$$\langle P|(x \cdot |Q\rangle) = (\langle P| \cdot x)|Q\rangle$$

which allows us to write $\langle P|x|Q\rangle$ unambiguously.

The Born rule as an interpretation of non-determinism in the evolution of states The COMM-rule clearly admits non-deterministic evolution of states. Specifically, there are two basic forms of non-determinism typically referred to as *races* in the classical interpretation of the rho-calculus. The first one is when there are more outputs than there are inputs, and the second is when there are more inputs than there are outputs. Symbolically,

$$\mathsf{for}(y_1 \leftarrow x)P_1|x!(Q)|\mathsf{for}(y_2 \leftarrow x)P_2 \qquad\qquad x!(Q_1)|\mathsf{for}(y \leftarrow x)P|x!(Q_2)$$

In the first case the two states reachable by reduction are $P_1\{@Q/y_1\}$ and $P_2\{@Q/y_2\}$; and in the second case the two states reachable by reduction are $P\{@Q_1/y\}$ and $P\{@Q_2/y\}$. In many implementations of the rho-calculus, such as in the RChain blockchain, the interpretation of this non-determinism is as an actual race. One of the communications beats the other, either as an asynchronous communication over a standard protocol (such as TCP), or one pair of threads are scheduled to work over the other pair, in a threading library inside an executing process.

In stochastic versions of these calculi the idea is to extend channels with *rates*. Thus,

$$\mathsf{for}(y \leftarrow x)P \mapsto \mathsf{for}(y \leftarrow (x, r))P \qquad\qquad x!(Q) \mapsto (x, r)!(Q)$$

The rates are used to determine a 1-norm probability distribution over the possible COMM events, at a given number of reduction steps from the starting expression.

There is nothing in principle that prevents using 2-norm probability distributions, where the rates are given by complex numbers and contribute probability amplitudes. That's what we develop next.

Continuous Time Markov Chains and their Quantum Variants

Gillespie's method Gillespie's method [12] allows for an interpretation of chemical equations with rates that affords a stochastic simulation when the species populations are relatively constrained (as they are in cells). The algorithm has been succesfully adapted to stochastic versions of the π-calculus. The basis for this adaptation is the intuition that chemical species, i.e. molecules, are processes, and thus each COMM event corresponds the execution of a reaction. To apply the method we must enrich the channel-based reaction of the π-calculus with reaction rates. For example, in the π-calculus process

$$\mathsf{for}(y_1 \leftarrow x_1)P_1 + \mathsf{for}(y_2 \leftarrow x_2)P_2$$

represents a molecule that can react on two channels, x_1, and x_2. Meanwhile,

$$x_1!(z_1);Q_1 \qquad\qquad x_2!(z_2);Q_2$$

represent molecules that can react on x_1 and x_2, respectively. In this context, a solution is a parallel composition of a number of copies of each of the different kinds of processes. Writing $[P]_N$ for N copies of P in parallel composition, i.e. $P|\cdots|P$, then the general form of a solution using the species of processes described above is

$$S = [x_1!(z_1);Q_1]_{M_1} \mid [\mathsf{for}(y_1 \leftarrow x_1)P_1 + \mathsf{for}(y_2 \leftarrow x_2)P_2]_{M_2} \mid [x_2!(z_2);Q_2]_{M_3}$$

If we fix

$$\mathsf{rate}(x_1) = r_1 \qquad\qquad \mathsf{rate}(x_2) = r_2$$

then the probability of S reducing via any interaction is $R = \Sigma r_i M_i M_{i+1}$. And the probability of interacting via x_i is $r_i M_i M_{i+1}/R$. Having specified our interpretation as a CMTC, we can apply Gillespie's algorithm to simulate how S evolves.

The same method works for the rho-calculus. Note that the method is oblivious to the structure of names, as it must be to interpret the π-calculus, which hides all structure of names. However, there is an intriguing option when using the annihilation-based version of reduction.

Quantum Continuous-Time Markov Chains Xu, et al [32] have developed an analog of Gillespie's method for quantum continuous time Markov chains. We follow their presentation closely.

Definition 17. *A quantum continuous Markov chain, $\mathcal{D}$, with a finite set S of classical states is a pair $(\mathcal{H}_{cq}, \mathcal{L})$, such that*

- $\mathcal{H}_{cq} := \mathcal{C} \times \mathcal{H}$ *where*
  - $\mathcal{C} := span(\{|s\rangle : s \in S\})$
  - $\mathcal{H}$ *a Hilbert space.*
- $\mathcal{L}$ *is a transition generator function given by a Hermitian operator* H *and a finite set of linear operator,* L_j*, on* $\mathcal{H}_{cq}$*.*

We need a lemma allowing us to convert a CTMC to a QCTMC.

Lemma 1. *A continuous-time Markov chain $\mathcal{C} = (S, \mathbf{Q})$ can be faithfully modeled by a QCMTC $(\widehat{\mathcal{C}} \otimes \mathcal{H}, \mathcal{L})$ where $\widehat{\mathcal{C}} := \{|s\rangle : s \in S\}$ and $\mathsf{dim}(\mathcal{H}) = 1$.*

5 From formalism to physical intuition and back

The guiding intuition is that information about a physical entity must be present at a name. To model the non-deterministic character of quantum mechanics the information and its variants are spread out over a collection of names, i.e. a namespace, $@\phi$. In symbols, we represent the possible states of a physical entity, like the spin of a particle, as expressions of the form

$$S = \Sigma_{u \in @\phi, s \in S} u!(Q_s)$$

We represent a test for a property as input at a name.

$$T = \Sigma_{u \in @\phi, c \in C} \mathsf{for}(v \leftarrow u) P_c$$

Then a measurement at a specific name, x, is given by

$$\langle T|x|S\rangle = x \cdot (T^{\perp}[S]) = x \cdot (T|S)$$

6 Conclusions and future work

One way to understand the rho-calculus is a study in the seams of the π-calculus. There are several holes in Milner's calculus:

- *the zero process*, $\mathsf{0}$, is an input to the theory; adding other elements here corresponds to supplying "builtin" processes;
- *names* are an input to the theory, which we discussed at length;
- *the source of non-determinism* is an input to the theory.

The results here can really be seen as an initial exploration of the a general theory of this third dependency.

Acknowledgments. The author wishes to thank Phil Scott for bringing the normalization-by-evaluation program to his attention. It turned out to be a key piece of the puzzle.

References

1. Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *POPL*, pages 33–44. ACM, 2002.
2. Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.
3. Samson Abramsky. Algorithmic game semantics and static analysis. In *SAS*, volume 3672 of *Lecture Notes in Computer Science*, page 1. Springer, 2005.
4. Luís Caires. Behavioral and spatial observations in a logic for the pi-calculus. In *FoSSaCS*, pages 72–89, 2004.
5. Luis Caires. Spatial logic model checker, Nov 2004.
6. Luís Caires and Luca Cardelli. A spatial logic for concurrency (part I). *Inf. Comput.*, 186(2):194–235, 2003.
7. Luís Caires and Luca Cardelli. A spatial logic for concurrency - II. *Theor. Comput. Sci.*, 322(3):517–565, 2004.
8. Luca Cardelli. Brane calculi. In *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.
9. Vincent Danos and Cosimo Laneve. Core formal molecular biology. In *ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2003.
10. Cédric Fournet, Fabrice Le Fessant, Luc Maranget, and Alan Schmitt. Jocaml: A language for concurrent distributed and mobile programming. In *Advanced Functional Programming*, volume 2638 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2002.
11. Cédric Fournet, Georges Gontheir, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, 1996.
12. Daniel T. Gillespie. Stochastic Simulation of Chemical Kinetics. *Annual Review of Physical Chemistry*, 58:35–55, May 2007.
13. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
14. Allen L. Brown Jr., Cosimo Laneve, and L. Gregory Meredith. Piduce: A process calculus with native XML datatypes. In *EPEW/WS-FM*, volume 3670 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2005.
15. Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 282–298. Springer, 2005.
16. Greg Meredith. Documents as processes: A unification of the entire web service stack. In *WISE*, pages 17–20. IEEE Computer Society, 2003.
17. L. Gregory Meredith and Matthias Radestock. Namespace logic: A logic for a reflective higher-order calculus. In *TGC* [18], pages 353–369.
18. L. Gregory Meredith and Matthias Radestock. A reflective higher-order calculus. *Electr. Notes Theor. Comput. Sci.*, 141(5):49–67, 2005.
19. Lucius Meredith, Jan 2017.
20. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
21. Robin Milner. Elements of interaction - turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.
22. Robin Milner. The polyadic π-calculus: A tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
23. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
24. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.
25. Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.
26. Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.
27. Davide Sangiorgi. Beyond bisimulation: The "up-to" techniques. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 2005.
28. Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):15:1–15:41, 2009.
29. Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.

30. Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.
31. Peter Sewell, Pawel T. Wojciechowski, and Asis Unyapoth. Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. *ACM Trans. Program. Lang. Syst.*, 32(4):12:1–12:63, 2010.
32. Ming Xu, Jingyi Mei, Ji Guan, and Nengkun Yu. Model checking quantum continuous-time markov chains. *CoRR*, abs/2105.00382, 2021.