Notes on interpreting quantum mechanics in a reflective process calculus

L.G. Meredith[1]

Managing Partner, RTech Unchained 9336 California Ave SW, Seattle, WA 98103, USA,
lgreg.meredith@gmail.com

Abstract. We sketch an interpretation of operations of quantum mechanics in terms of operations in a reflective process calculus.

1 Introduction

For the past several centuries there has been no serious competitor to the "Newtonian" account of dynamics in the physical sciences. By the Newtonian account of dynamics we means formulations guided by two principles:

- the form of physical laws are given as laws of motion plus initial state;
- the laws of motion are stated in terms of the mathematical apparati of the integral and derivative and the principles of applications of those apparati to modeling dynamical systems, such as the Lagrangian and Hamiltonian [**?**].

In recent years several distinguished physicists have begun to question the scope and applicability of the first principle. Notably David Deutsch and his collaborator Chiara Marletto have begun to ask whether laws of physics might be better formulated as counterfactuals. Their motivations are informed by considerations going as far back as John Wheeler's suggestions that physics might derive from information processing, it arising from bit, as it were. As Deutsch and Marletto point out, the challenge in attempting to carry out such a program is that in modern physics information is *emergent*, and not a quantity on par with properties like position or momentum.

A similar, but perhaps more pointed critique can levied against the second principal. It is common place to render versions of the laws of physics in a programming language. From predictive models for particle colliders to physics engines for games this activity has been standard practice for decades. Yet, no one has rendered even the most basic of programming models in the laws of physics *considered as an abstract language*. Even quantum computation introduces abstractions like quantum gates or the ZX-calculus. The standard model is not seen as a programming paradigm. Programming paradigms, on the other hand, are seen as just the right tool for rendering the full range of dynamical abstractions from the Lagrangian and Hamiltonian to perturbative techniques into a form where we can gain concrete computational insights. Could it be that computation is better suited as the language of dynamics, including physical dynamics, and not just the dynamics of information?

We might even go so far as to speculate whether it might be intellectually dangerous to limit our view of dynamics to one abstraction. Several examples in the history of physics suggest it is appropriate to worry about this. Feynman diagrams introduced a new abstraction that dramatically expanded the scope of what could be calculated and hence grasped. At their inception, and for years after, well known physicists dismissed them as mere syntax, not physics. Likewise, as Coecke and Kissinger point out, it took *60 years* from the discovery of quantum mechanics to the formulation of quantum teleportation. Yet, when rendered in string diagrams, the phenomenon is dead obvious and would have been spotted by a high school student performing routine calculations. Both developments share the same underlying drive: to formulate a clear computational framework that expands what is concretely amenable to calculation. They suggest an intriguing idea: what if syntax is an integral part of physics? Indeed of all science.

It is from this perspective that we seek to explore new developments in computing as a potential syntax for physical dynamics. In particular, these new theories of computing offer new ways of thinking about how computation, and by extension the physical world, is organized. They re-imagine computation as interaction. Instead of (active) functions acting on (passive) data, resulting in new, (transformed) data – or (active) forces acting on (passive) particles – these models view computation as the *patterns of interaction* amongst *autonomous* agents. The view distributes agency throughout the elements comprising a computation. Further, these interactive models embody principles not found in traditional accounts of physical dynamics. Among these, perhaps the most important, is *compositionality*.

This principle is innocent enough. It asks that our models of system dynamics be built of the models of the dynamics of their subsystems. This requirement has two important aspects: one, that we can compose models to get new models of increasing complexity; and, two, that we can decompose more complex models into models of reduced complexity. But, it is not just about composing models. It must be the case that the composition of models faithfully represents the composition of the systems they represent. Likewise, if a model does decompose, it must correspond to a natural decomposition of the system it represents. This is not a property enjoyed, for example, by differential equation models of stoichiometry. If we have a model of the reactions in beaker A, and a model of the reactions in beaker B, we cannot simply combine the equations of the model of A with the equations of beaker B to get a model of the system obtained by pouring the contents of A and B into a third beaker. In fact, differential models often fail this particular test.

Challenges to classical accounts of physics as deep as General Relativity [**?**] and Quantum mechanics [**?**] are still *essentially* couched in the methods and procedures derived from the Newtonian integral and derivative [1]. Meanwhile, the very successful program of the ZX-calculus emphasizes compositionality to great benefit. The current work is an attempt to push that particular envelope by encoding the conceptual core of quantum mechanics into a calculus for concurrent computing called the rho-calculus.

[1] The covariant derivative still bears the imprimature of the original notion. Inner product in the wave-function formulation is still an integral

1.1 Summary of contributions and outline of paper

The plan is to develop an interpretation of the operations of quantum mechanics normally interpreted by Hilbert spaces and operators [**?**]. The primary novelty of the approach is to do this over a theory of computation. Note that this is very different than the usual quantum computation program which develops notions of computation over quantum mechanics [**?**]. In detail, we present a reflective process calculus. Then we develop intuitive correspondences between the notions available in this calculus and the usual physical notions supporting quantum mechanical calculations.

quantum mechanics	process calculus
scalar	name
state vector	process
dual	contextual duals
matrix	formal sums of process-context-dual pairs
orthogonality	process annihilation
inner product	parallel composition + quoting

Table 1. QM - process calculi correspondences

Then we tighten up these intuitions to operational definitions. We employ the Dirac notation as the best proxy we can find for an abstract syntax of the quantum mechanical notions that also makes contact with notations that working physicists understand. The definitions we develop put us in contact with equational constraints coming from the theory, and we demonstrate the definitions and calculations satisfy.

This puts us in a position to shut up and calculate for the Stern-Gerlach experimental set up, showing how these predictive calculations become calculations on processes in our theory of a reflective process calculus. The particular example is of interest because it is a composite experiment, consisting of iterated measurements. The framework developed here is particularly suited to reasoning about composite physical systems and situations.

Apart from the novelty of reflective apparatus of the rho-calculus, one must ask how it can span the gap between classical computation and quantum computation. The key idea is the abstraction of non-determinism. The calculus leaves to various models to supply a notion of non-determinism. Thus, the same framework maps to classical computing when non-determinism is mapped to a race-condition. It maps to stochastic models when non-determinism is governed by a 1-norm probability distribution. It maps to quantum computation with non-determinism is governed by a 2-norm probability distribution, thus allowing for negative probabilities which then set up interference in the causal evolution of a computation.

Another way to understand the approach is that rho-calculus terms plus structural equivalence give an algebra on states, much like Hilbert spaces gives an algebra on states. The rho-calculus terms only form a rig, however. The comm rule of the rho-calculus gives a way to calculate the evolution of states much like the Schroedinger equation allows on

to calculate an evolution of states. The mapping of non-determinism in terms of a 2-norm probability distribution plays in the same role as the Born rule.

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) [?] and Hoare's Communicating Sequential Processes (CSP) [?] [?] [?] [?], and continue through the development of the so-called mobile process calculi, e.g. Milner, Parrow and Walker's π-calculus [?], Cardelli and Caires's spatial logic [?] [?] [?], or Meredith and Radestock's reflective calculi [?] [?]. 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 [?] [?] [?] [?];
- programming language semantics and design [?] [3] [?] [?];
- webservices [?] [?] [?];
- blockchain [?]
- and biological systems [?] [?] [?] [?].

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 [?], 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 [?] [5] [?] [6]. In particular, *bisimulation* encapsulates an effective notion of process equivalence that has been used in applications as far-ranging as algorithmic games semantics [?] and the construction of model-checkers [?]. 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 [?] all governed by the same core principle [5] [?] [6].

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 [**?**] with the addition of polyadicity and summation. For readability we have relegated some of the technical subtleties to an appendix.

Process grammar

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

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*

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

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

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, @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, 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 2. *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 3. *The* structural congruence $\equiv$ *between processes [5] is the least congruence containing alpha-equivalence and satisfying the commutative monoid laws (associativity, commutativity and* $\mathsf{0}$ *as identity) for parallel composition* $|$.

Definition 4. *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.

Finally equipped with these standard features 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.

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 [5]. As our first example of calculation with the machinery thus far presented we give the construction explicitly in the ρ-calculus.

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

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

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 [4].

Definition 5. *An* observation relation, $\downarrow_{\mathcal{N}}$, *over a set of names,* $\mathcal{N}$, *is the smallest relation satisfying the rules below.*

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

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

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

Definition 6. *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 $\mathcal{S}_{\mathcal{N}}$.

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 7 (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 [**?**]. 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 [**?**] 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 [4]). 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 8. *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.

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

And contexts are likewise extended in the obvious manner.

SUMMATION-CONTEXT
$K_M ::= \Box \mid x.K_A \mid K_M + M$

AGENT-CONTEXT
$K_A ::= (\overrightarrow{x})K_P \mid [\overrightarrow{P}, K_P, \overrightarrow{P'}]Q \mid [\overrightarrow{P}]K_P$

PROCESS-CONTEXT
$K_P ::= K_M \mid P|K_P$

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.

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.

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

Superposition as summation In this interpretation superposition corresponds to summation. This 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.

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 2. 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.

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 Stern-Gerlach

TBD

7 Conclusion and future work

TBD

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. Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
2. Luís Caires. Behavioral and spatial observations in a logic for the pi-calculus. In *FoSSaCS*, pages 72–89, 2004.
3. 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.
4. Robin Milner. The polyadic π-calculus: A tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
5. David Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
6. Davide Sangiorgi. Bisimulation in higher-order process calculi. *Information and Computation*, 131:141–178, 1996.