Notes on interpreting quantum mechanics in a reflective process calculus

L.G. Meredith[1]

Managing Partner, Biosimilarity
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.

Everything, despite it's intrinsic shape, turns into a nail to be hit with this hammer. For example, even 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]. Recently, however, the theory of computation has matured to the point where we have candidates for theories of dynamics that offer very different perspectives on reasoning about dynamical systems and situations. It is time to see just how good they are by addressing the dynamics of interest in the

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

physical sciences. Testing these candidates against very successful accounts of dynamical situations, like quantum mechanics, is going to give us some sense of maturity, quality and range of applicability of computational notions of dynamics.

To address this program we realize and carry out the calculations of quantum mechanics over a novel formal theory of dynamics, a formal theory of dynamics that corresponds to a theory of concurrent computation with *reflection* [**?**],[**?**]. It has the advantage that the underlying theory is already 'quantized', but supports analogues all of the continuuous operations. Strikingly, this underlying theory has recently been connected with a notion of metric [**?**] that coincides with the metric induced by the inner product.

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 [**?**]. Rather, we are developing a story that aligns with Wheeler's slogan: It from Bit [**?**]. To do this we will first provide an account of the theory of computation at play here. Then we will dive into a calculation-driven interpretation of the operations of quantum mechanics.

The reason we take this approach is that – until very recently ([**?**], [**?**])– there hasn't been an axiomatic account of quantum mechanics. As a result there has been no sharp delineation of the mathematical theory supporting interpretation of the physical theory and the physical theory, itself. So, ambient features of the maths are free to be exploited (or suppressed) without a real accounting of their physical relevance. There is no sharp statement "here's the physical theory" qua *theory* and "here's the mathematical interpretation" enabling a judgment of how faithful the interpretation is – apart from experimental observation. When there is an axiomatic account we can judge how well a given mathematical formalism supports an interpretation of the axioms, independent of experimentation. Likewise, we can judge how well we have captured our physical evidence and experience with our axiomatics, independent of any specific mathematical implementation, with accidental detail that may or may not have physical significance.

In lieu of a fully fleshed out and vetted axiomatic account of quantum mechanics, interpreting the operational notions in service of modeling physical systems will have to suffice. In other words, we are not in the business of providing a model of Hilbert spaces and operators. We are in the business of providing a model of quantum mechanics because we are motivated by testing our notions of dynamics against physical theory; and, the predictive calculations of the physical theory must serve as the best formulation – shy of a fully fleshed out axiomatic account – of the physical theory itself (as they have for scientific theories since time immemorial). Put another way, despite a whole-hearted commitment to an It-from-Bit ontology, we are firmly aligned with the shut-up-and-calculate camp as the best way to obtain results either from the physical perspective or as a quality assurance measure of our fledgling theory of dynamics.

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	execution-formula + 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. The definitions we develop put us in contact with equational constraints coming from the theory that 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.

Penultimately, we demonstrate that the notion of metric coming from the inner product coincides with the notion of metric available from the theory of bisimulation. This demonstration gives us the right to think of space as arising from behavior. Finally, we consider where we might go from the new vantage point we have obtained.

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 [**?**] [**?**] [**?**];
- 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

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*

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

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

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, $@\mathsf{Proc}$ 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, 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 8.

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.

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

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

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.

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

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

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 [4].

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

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

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

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

Definition 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* $P \rightarrow P'$ *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 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.*

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, $\overset{\vee}{x}$, when we want to emphasize the connection to the use of the name. Note that, by virtue of name equivalence, $@\overset{\vee}{x} \equiv_{\mathsf{N}} x$; 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,

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

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

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.

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

And contexts are likewise extended in the obvious manner.

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

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

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.

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

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

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

$$\textit{equivalently}$$

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

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. There are many ways to achieve this.

Remark 4. The reader is invited to verify that for all $x, y, z \in @\mathsf{Proc}$ and $P \in \mathsf{Proc}$

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

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

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

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

Tensor product We define a tensor product on processes by structural induction.

Tensor of sums First note that all summations, including $\mathsf{0}$ and sequence, can be written $\Sigma_i x_i.A_i + \Sigma_j x_j.C_j$, where we have grouped input-guarded processes together and output-guarded processes together.

Thus, we can define the tensor product of two summations, $N_1 \otimes N_2$, where

$$N_1 := \Sigma_i x_i.A_i + \Sigma_j x_j.C_j \qquad N_2 := \Sigma_{i'} y_{i'}.B_{i'} + \Sigma_{j'} y_{j'}.D_{j'}$$

as follows.

$$\Sigma_i x_i.A_i + \Sigma_j x_j.C_j \otimes \Sigma_{i'} y_{i'}.B_{i'} + \Sigma_{j'} y_{j'}.D_{j'}$$

$$:= \ \Sigma_i \Sigma_{i'} @ \overset{\vee}{x_i} \mid \overset{\vee}{y_{i'}} .(A_i \otimes B_{i'}) \mid \Sigma_{i'} \Sigma_i @ \overset{\vee}{y_{i'}} \mid \overset{\vee}{x_i} .(B_{i'} \otimes A_i)$$

$$\mid \ \Sigma_j \Sigma_{j'} @ \overset{\vee}{x_j} \mid \overset{\vee}{y_{j'}} .(A_j \otimes B_{j'}) \mid \Sigma_{j'} \Sigma_j @ \overset{\vee}{y_{j'}} \mid \overset{\vee}{x_j} .(B_{j'} \otimes A_j)$$

Remark 5. Do we need to x^L and y^R for this construction as well?

Tensor of parallel compositions Next, we distribute tensor over par.

$$P_1|P_2 \otimes Q_1|Q_2 := (P_1 \otimes Q_1)|(P_1 \otimes Q_2)|(P_2 \otimes Q_1)|(P_2 \otimes Q_2)$$

Tensor with dropped names We treat tensor of a process with a dropped name as parallel composition.

$$P \otimes *x := P|*x$$

Tensor of agents Finally, we need to define tensor on agents. Note that the definition of tensor on summations only tensors inputs with inputs and outputs with outputs. Thus, we only have to define the operation on "homogeneous" pairings.

$$\begin{aligned}
&(\overrightarrow{x})P \otimes (\overrightarrow{y})Q \\
:= &(x_0^L|y_0^R, \ldots, x_0^L|y_n^R, \ldots, x_m^L|y_0^R, \ldots, x_m^L|y_n^R)(P\{\overrightarrow{x}^L/\overrightarrow{x}\} \otimes Q\{\overrightarrow{y}^R/\overrightarrow{y}\}) \\
&\langle\!|\overrightarrow{P}|\!\rangle \otimes \langle\!|\overrightarrow{Q}|\!\rangle \\
:= &\langle\!|P_0 \otimes Q_0, \ldots, P_0 \otimes Q_n, \ldots, P_m \otimes Q_0, \ldots, P_m \otimes Q_n|\!\rangle
\end{aligned}$$

Remark 6. Observe that arities of tensored abstractions matches arities of tensored concretions if the original arities matched. Note also that the length of the arities corresponds to the increase in dimension we see in ordinary vector space tensor product.

Remark 7. Operationally, this definition distributes the tensor down to components "linked" by summation. Tensor over summation is intriguing in that it mixes names. Moreover, as a consequence of the way it mixes names we have the identities for all $x \in @\mathsf{Proc}$ and $P, Q \in \mathsf{Proc}$

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

$$P \otimes \mathsf{0} \equiv P$$

that the reader is invited to verify.

Annihilation

$$P^{\perp} := \{Q : \forall R.P|Q \to^* R \Rightarrow R \to^* \mathsf{0}\}$$

$$P^{\times} := \Sigma_{Q \in P^{\perp}} @Q?(y).(*y|Q)|\Sigma_{Q \in P^{\perp}} @Q\langle\!|\Box|\!\rangle$$

Discussion The reader will note that $P^\perp$ is a *set* of processes, while $P^\times$ is a *context*. We call the set $P^\perp$ the *annihilators* of P. The parallel composition of a process in the annihilators of P with P will result in a process, the state space of which has all paths eventually leading to $\mathsf{0}$. Execution may endure loops; but under reasonable conditions of fairness (naturally guaranteed under most notions of bisimulation) such a composite process cannot get stuck in such a loop and will, eventually pop out and terminate.

The context $P^\times$ is ready and willing to "take the P out of" the process to which it is applied. It will effectively transmit the code of the process to which it is applied to one of the annihilators and run the process against it.

Remark 8. Note that $P^{\times *}$ is the abstraction corresponding to context $P^\times$. We will set $P^\bullet := P^{\times *}$.

Evaluation We fix M a domain of fully abstract interpretation with an equality coincident with bisimulation. We take $[\![\cdot]\!] : \mathsf{Proc} \to M$ to be the map interpreting processes and $(\!|\cdot|\!) : \mathcal{M} \to Proc$ to be the map running the other way. Then we define

$$\int P := (\!|[\![P]\!]|\!)$$

Discussion There are many fully abstract interpretations of Milner's π-calculus. Any of them can be used as a basis for interpreting the reflective calculus here. Equipped with such a domain it is largely a matter of grinding through to check that the Yoneda construction for the normalization-by-evaluation program can be extended to this setting.

Remark 9. The reader is invited to verify that $\int (P^\times[P]) = 0$, and equivalently $(\nu\ x) \int P^\bullet\langle x\rangle | x(\!|P|\!) = 0$.

4.2 Quantum mechanics

What is the quantum mechanical notion of continuation? Imagine the following experimental set-up. Alice, our intrepid quantum investigator, prepares a state by performing some operation on some initial state. Then she performs some measurement to obtain an observation. Using the information of the observation, she selects a new initial state, operation and measurement and repeats the steps above. She iterates this procedure until she obtains some desired observation. What is the expression of this procedure in the language of quantum mechanics?

Figure 4.2 gives a schematic description of such an iterated experimental procedure. The question is how do we write down this iterated procedure without stepping outside the language of quantum mechanics? Note that accounts of famous composite quantum experiments, like the Stern-Gerlach experiment leave the language of quantum mechanics to describe the iterated experiment.

We ask this question for many reasons, but one of them is to help set up the exegesis of our interpretation. In our framework *everything* is a computation, both the quantum

```
E ::=
let S = U|L⟩ in (* prepare state*)
let m = ⟨M|S⟩² in (* take measurement *)
match m with (* use m to decide next experiment *)
v0 -> E
| ...
| vN -> E
| vExit -> m (* return observation *)
```

Fig. 1. Iterated experiment schematic

operations and processes (classical or quantum) that invoke those operations. There is no need to step out of the conceptual (and more pragmatically, the computational) framework to describe these kinds of experiments. More to the point, the framework we are proposing is – like the hybrid functional language employed in the schema – *compositional*: experiments, computations are built out of experiments and computations. This is of enormous pragmatic value if we are to build and reason about systems of significant scale.

Remark 10. It is also worth noting in this connection that this schema is the core of a wide range of recursive functions. Further, this connection to calculations of fixpoints makes it a close neighbor of search techniques like natural selection and the scientific method. This is a theme to which we will return, for quantum information seems very *unlife-like* in it's uncloneable, undeleteable nature.

Returning the matter of the computational interpretation, our interpretation will take the form of a map, written $[\![-]\!](-)$, from expressions in Dirac notation to expressions in our target reflective calculus. The map takes an *ancillary* argument, a channel along which to communicate results to subsequent computations. This is how we communicate, for example, the results of taking a measurement to a subsequent step in an experiment.

Interpretation Table 4.2 gives the core operational correspondences. It is meant as an intuitive guide.

quantum mechanics	process calculus
scalar	$x := @P$
state vector	$\|P\rangle := P$
dual	$\|P\rangle^* := \langle P^\times \| := @P^\times[-]$
matrix	$\Sigma_\alpha \|P_\alpha\rangle x_\alpha \langle Q_\alpha\|$
vector addition	$\|P\rangle + \|Q\rangle := \|P\|Q\rangle$
tensor product	$\|P\rangle \otimes \|Q\rangle := \|P \otimes Q\rangle$
inner product	$\langle P\|Q\rangle := @\int P^\times[Q]$

Table 2. QM - operational definitions

Discussion The process algebraic view of a state is called, ironically, a process, and that is what we map vectors to in our interpretation. It has long been noted in the process algebraic community that names play a role somewhat similar to scalars in a vector space. What is unique about the reflective calculus, and makes it suitable for an interpretation of this form is that with the structure of names reflecting the structure of processes we can both make this similarity in a precision instrument; and find a notion of *dual* to a state.

If we posit names as scalars, then in perfect analogy with vector spaces a dual is a map from processes to names. We actually have two candidates for this interpretation: nominal contexts, $@M$, and their corresponding abstraction, $@M^*$. The goal of supporting a notion of continuation selects the latter of the two for our interpretation.

Taking these as the basis of the interpretation together with the algebraic identities required by the Dirac notation more or less fixes the definitions of the rest of the operations. Among the interesting particularities, the definition of inner product finds near perfect mirroring of the Feyman interpretation.

STATES
$$[\![\mid P\rangle]\!](c) = c?(l,r).r(\!|P|\!)$$

EVENTS
$$[\![\langle P \mid]\!](c) = (x)c?(l,r).l(\!|P^\bullet\langle x\rangle|\!)$$

VECTOR ADDITION
$$[\![\mid P\rangle + \mid Q\rangle]\!](c) = [\![\mid P|Q\rangle]\!](c)$$

TENSOR PRODUCT
$$[\![\mid P\rangle \otimes \mid Q\rangle]\!](c) = [\![\mid P \otimes Q\rangle]\!](c)$$

INNER PRODUCT
$$[\![\langle P \mid Q\rangle]\!](c) = (\nu\ x)c(\!| \int P^\bullet\langle x\rangle|x(\!|Q|\!)|\!)$$

MATRIX
$$[\![\mid P\rangle x\langle Q \mid]\!](c) =$$
$$(u)(\nu\ lr)c!(l,r).(l?(e).(\nu\ y)[\![\langle *e \mid Q\rangle]\!](x)|x?(z).x?(a).c(\!|(P^\bullet\sigma(z,a))\langle u\rangle|\!)|y!(x)$$
$$+r?(e).(\nu\ x)[\![\langle P \mid *e\rangle]\!](x)|x?(z).x?(a).c(\!|Q\sigma(z,a)|\!)|x!(x_\alpha))$$

MATRIX APPLICATION
$$[\![(\mid P\rangle x\langle Q \mid)(\mid S\rangle)]\!](c) = (\nu\ c'u)[\![\mid P\rangle x\langle Q \mid]\!](c')\langle u\rangle|[\![\mid S\rangle]\!](c')|c'?(a).c!(a)$$
$$[\![(\mid P\rangle x\langle Q \mid)(\langle S \mid)]\!](c) = (\nu\ c'u)[\![\mid P\rangle x\langle Q \mid]\!](c')\langle u\rangle|[\![\langle S \mid]\!](c')|c'?(a).c!(a)$$

where

$$P\sigma(z,a) := P\{z \cdot a \cdot r/r : r \in \mathsf{FN}(P)\}$$

Remark 11. The reader is invited to verify that

$$[\![\langle P \mid Q\rangle]\!](c)$$

$$\dot{\approx}$$

$$(\nu\ x)(\nu c'lr)[\![\langle P \mid]\!](c')\langle x\rangle \mid [\![\mid Q\rangle]\!](c')|\ c'!(l,r).l?(p).r?(q).c(\!| \int *p|x!(q)|\!)$$

This provides a (more) compositional definition of inner product. It also illustrates an important point of the computational interpretation. We have a notion of equivalence providing a crucial proof method: bisimulation.

Remark 12. Assuming $\int(P^{\times}[P]) = 0$, the reader is invited to verify that $(\mid P\rangle x\langle P \mid)(\mid P\rangle) = x \cdot \mid P\rangle$.

Remark 13. Assuming $\int(P^{\times}[P]) = 0$, the reader is invited to verify that $\forall P, Q.(\mid 0\rangle\langle Q \mid)(\mid 0\rangle) = \mid 0\rangle$ and dually $(\mid P\rangle\langle 0 \mid)(\langle 0 \mid) = \langle 0 \mid$.

Interpreting continuations As promised, we can combine these interpretations with standard semantics for conditionals and continuations to provide an interpretation of the iterated experiment.

$[\![$ **let** S $= U \mid L\rangle$ **in**
let m $= \langle M \mid S\rangle^2$ **in**
match m **with**
v_0 -> $\mathcal{E}$
| ...
| v_N -> $\mathcal{E}$
| v_{Exit} -> m $]\!](c)$
=
$(\nu\ c')[\![\langle M \mid U \mid L\rangle\rangle]\!](c')$
$|c'?(m).m!(m)|\Sigma_{i=0}^{N} v_i?(m).[\![\mathcal{E}]\!] + v_{Exit}?(m).c!(m)$

A quick tally Already the interpretation is beginning to show signs of promise. First of all, it is no more notationally cumbersome than the notation used in QM calculations. Beyond syntax, we have a new proof principle in hand and the ability to reason about more complex experimental situations than is directly calculable in ordinary quantum mechanics.

Adjointness We need to give a definition of $(\cdot)^{\dagger}$ for matrices. The obvious candidate definition is

$$[\![(\mid P\rangle x\langle Q \mid)^{\dagger}]\!](c) = (\nu\ u)[\![\mid Q^{\bullet}\langle u\rangle\rangle \overline{x} \langle P^{\bullet}\langle u\rangle \mid]\!](c)$$

Remark 14. i'm a little worried that i don't (yet) have proper support for complex conjugacy. But, the observation above may give us a clue. According to Abramsky, it must be the case that the scalars are iso to the homset of the identity for the tensor – which the observation above (13) characterizes. For now, we will simply bookmark the notion with $\overline{x}$.

Basis for a basis If processes label states and "addition" of states (a.k.a. vector addition) is interpreted as parallel composition, what corresponds to notions of linear independence and basis? Here, we recall that Yoshida has developed a set of *combinators* for an asynchronous verison of Milner's π-calculus [**?**]. These are a finite set of processes such any process can be expressed as parallel composition of these combinators together with liberal uses of the new operator and replication. We can simply give a translation of these into the present calculus and have reasonable expectation that the property carries over. That is, that the resultant set allows to express all processes via parallel composition. Note, however, that there is no new operator or replication in this calculus. As a result, we expect that the corresponding set is actually infinite. That is, we expect that the space is actually infinite dimensional.

Remark 15. The reader familiar with the lambda calculus may reasonably object: certainly, the collection S, K and I is a finite set of combinators [1]. Shouldn't we expect to see a finite set of combinators for an effectively equivalent system? i am very sympathetic to this critique and feel it warrants full attention. On the other hand, i also have in mind the following analogy. The natural numbers, as a monoid under addition, has exactly 1 generator, while the natural numbers, as a monoid under multiplication, has countably many generators (the primes). We observe that the application of the lambda calculus is much less resource sensitive than the parallel composition of the π-calculus. Could it be the case that we have an analogy of the form

$$m + n : MN :: m * n : M|N$$

giving a similar blow up in the set of "primes"? This is such a wonderful thought that, even if it's not true, i think it's worth writing down.

5 Stern-Gerlach again

This is where we have some real fun. Stern-Gerlach was one of a host of experiments being developed around the first third of the previous century that gave rise to quantum mechanics. Modeling Stern-Gerlach is seen as one of the tests of quantum mechanical theory.

Here's the Wikipedia link describing the experimental set up and relevance to the development of quantum mechanics:

Stern-Gerlach on Wikipedia

5.1 Modeling the experiment

TBD

6 Space from behavior

We now give the main theorem of the paper.

Theorem 1 (main). *The metric induced by the inner product coincides with the metric induced by bisimulation.*

Proof sketch The metric induced by bisimulation (when we put in the definition) will be the quote of the smallest witness of the smallest distinguishing formulae. The inner product quotes the effect of calculating the difference (what's left after you whack P against Q and let them run). These two notions coincide.

To make this statement precise enough to prove, we have a number of obligations to discharge. First of all, we have to give the metric induced by bisimulation. To do this, we need to give the Hennessy-Milner construction for the process calculus defined above. This logic enjoys the usual property that processes are bisimilar if and only if there is no distinguishing formula. We will take the distance between processes to be the smallest distinguishing formula. Thus, we have to give a measure of the size of the formula of this logic, which we write $\#(\phi)$.

With these definitions in hand we show that the quotation of the smallest witness of the smallest formulae is "bisimilar" to the name computed by the inner product. Notice that up this point, the calculation we are calling inner product has yet to be proved to provide a metric. Establishing the correspondence to the HML-induced metric is actually what gives us the right to think of the inner product calculation as a metric.

Note that it is possible to give an alternative definition following Caires' notion of a characteristic formula. If $[\![\phi]\!]$ denotes the set of processes satisfying ϕ and $[\![P]\!]_\phi$ denotes the characteristic formula of a process, then we can show that $[\![[\![\langle P \,|\, Q\rangle]\!]_\phi]\!] = [\![@D_\phi(P,Q)]\!]$ where

Definition 9 (metric).

$$\Delta(P,Q) := \{\phi \ : \ (P \models \phi \ \& \ Q \not\models \phi) \vee (P \not\models \phi \ \& \ Q \models \phi)\} \qquad D_\phi(P,Q) := \bigvee_{\phi \in \Delta(P,Q)} \phi$$

$$@D_\phi(P,Q) := @ \bigvee_{\phi \in \Delta(P,Q)} \phi \qquad D(P,Q) := min\{\#(\phi) \ : \ \phi \in \Delta(P,Q)\}$$

So, without further ado, we give you the HML construction for a reflective calculus.

6.1 Namespace logic: a Hennessy-Milner logic of reflection

Namespace logic resides in the subfamily of Hennessy-Milner logics discovered by Caires and Cardelli and known as spatial logics [**?**]. Thus, as is seen below, in addition to the action modalities, we also find formulae for *separation*, corresponding, at the logical level, to the structural content of the parallel operator at the level of the calculus. Likewise, we have quantification over names.

In this connection, however, we find an interesting difference between spatial logics investigated heretofore and this one. As in the calculus, we find no need for an operator corresponding to the ν construction. However, revelation in spatial logic, is a structural notion [**?**]. It detects the *declaration* of a new name. No such information is available in the reflective calculus or in namespace logic. The calculus and the logic can arrange that names are used in a manner consistent with their being declared as new in the π-calculus, but it cannot detect the declaration itself. Seen from this perspective, revelation is a somewhat remarkable observation, as it seems to be about detecting the programmer's intent.

BOOLEAN CONNECTIVES	SPATIAL CONNECTIVES	NOMINAL CONNECTIVES
$\phi, \psi ::= true \mid \neg\phi \mid \phi\&\psi$	$\mid \mathsf{0} \mid \phi \mid \psi$	$\mid \ulcorner b \urcorner \mid \forall n : \psi \,.\, \phi$

BEHAVIORAL CONNECTIVES	FIXPT CONNECTIVES	PATTERNS	LITERALS
$\mid \langle a? \overrightarrow{b} \rangle \phi \mid a(\!\mid \overrightarrow{\phi} \mid\!)$	$\mid \mathsf{rec}\ X \,.\, \phi \mid X$	$a ::= @\phi \mid b$	$b ::= @P \mid n$

We let $PForm$ denote the set of formulae generated by the ϕ-production, $QForm$ denote the set of formulae generated by the a-production and $\mathcal{V}$ denote the set of propositional variables used in the rec production.

Inspired by Caires' presentation of spatial logic [2], we give the semantics in terms of sets of processes (and names). We need the notion of a valuation $v : \mathcal{V} \to \wp(\mathsf{Proc})$, and use the notation $v\{\mathcal{S}/X\}$ to mean

$$v\{\mathcal{S}/X\}(Y) = \begin{cases} S & Y = X \\ v(Y) & otherwise \end{cases}$$

The meaning of formulae is given in terms of two mutually recursive functions,

$$[\![-]\!](-) : PForm \times [\mathcal{V} \to \wp(\mathsf{Proc})] \to \wp(\mathsf{Proc})$$
$$(\!|-|\!)(-) : QForm \times [\mathcal{V} \to \wp(\mathsf{Proc})] \to \wp(@\mathsf{Proc})$$

taking a formula of the appropriate type and a valuation, and returning a set of processes or a set of names, respectively.

$$
\begin{aligned}
[\![true]\!](v) &= \mathsf{Proc} \\
[\![\mathsf{0}]\!](v) &= \{P : P{\equiv}\mathsf{0}\} \\
[\![\neg\phi]\!](v) &= \mathsf{Proc}/[\![\phi]\!](v) \\
[\![\phi\&\psi]\!](v) &= [\![\phi]\!](v) \cap [\![\psi]\!](v) \\
[\![\phi|\psi]\!](v) &= \{P : \exists P_0, P_1.P{\equiv}P_0|P_1,\ P_0 \in [\![\phi]\!](v),\ P_1 \in [\![\psi]\!](v)\} \\
[\![\ulcorner b \urcorner]\!](v) &= \{P : \exists Q, P'.P{\equiv}Q|{*}x,\ x \in (\!|b|\!)(v)\} \\
[\![a\langle\!|\phi|\!\rangle]\!](v) &= \{P : \exists x, P'.P{\equiv}x!(P'),\ x \in (\!|a|\!)(v),\ P' \in [\![\phi]\!](v)\} \\
[\![\langle a?b\rangle\phi]\!](v) &= \{P : \exists x, P'.P{\equiv}\mathsf{for}(y \leftarrow x)P', x \in (\!|a|\!)(v), \\
&\qquad \forall c.\exists z.P'\{z/y\} \in [\![\phi\{c/b\}]\!](v)\} \\
[\![\mathsf{rec}\ X\ .\ \phi]\!](v) &= \cup\{\mathcal{S} \subseteq \mathsf{Proc} : \mathcal{S} \subseteq [\![\phi]\!](v\{\mathcal{S}/X\})\} \\
[\![\forall n : \psi\ .\ \phi]\!](v) &= \cap_{x \in (\!|@\psi|\!)(v)} [\![\phi\{x/n\}]\!](v) \\
(\!|@\phi|\!)(v) &= \{x : x{\equiv}_{\mathsf{N}}@P, P \in [\![\phi]\!](v)\} \\
(\!|@P|\!)(v) &= \{x : x{\equiv}_{\mathsf{N}}@P\}
\end{aligned}
$$

We say P witnesses ϕ (resp., x witnesses $@\phi$), written $P \models \phi$ (resp., $x \models @\phi$) just when $\forall v.P \in [\![\phi]\!](v)$ (resp., $\forall v.x \in [\![@\phi]\!](v)$).

Theorem 2 (Equivalence). $P \dot{\approx} Q \Leftrightarrow \forall\phi.P \models \phi \Leftrightarrow Q \models \phi$.

The proof employs an adaptation of the standard strategy. As noted in the introduction, this theorem means that there is no algorithm guaranteeing that a check for the witness relation will terminate.

Syntactic sugar In the examples below, we freely employ the usual DeMorgan-based syntactic sugar. For example,

$$
\begin{aligned}
\phi \Rightarrow \psi &\triangleq \neg(\phi\&\neg\psi) \\
\phi \vee \psi &\triangleq \neg(\neg\phi\&\neg\psi)
\end{aligned}
$$

Also, when quantification ranges over all of Proc, as in $\forall n : @\ true\ .\phi$, we omit the typing for the quantification variable, writing $\forall n\ .\ \phi$.

An example

Controlling access to namespaces Suppose that $@\phi$ describes some namespace, i.e. some collection of names. We can insist that a process restrict its next input to names in that namespace by insisting that it witness the formula

$$\langle@\phi?b\rangle\ true\ \&\neg\langle@\neg\phi?b\rangle true$$

which simply says the the process is currently able to take input from a name in the namespace $@\phi$ and is not capable of input on any name not in that namespace. In a similar manner, we can limit a server to serving only inputs in $@\phi$ throughout the lifetime of its behavior [2]

$$\mathsf{rec}\ X\ .\ \langle @\phi ?b\rangle X \& \neg\langle @\neg\phi ?b\rangle true$$

This formula is reminiscent of the functionality of a firewall, except that it is a *static* check. A process witnessing this formula will behave as though it were behind a firewall admitting only access to the ports in $@\phi$ without the need for the additional overhead of the watchdog machinery.

6.2 The size of a formula

We give a recursive definition of the size of a formula, written $\#(\phi)$, in terms of a measure of the set of processes it denotes.

Definition 10 (measure). *For the time being we simply demand a Lebesgue measure, written μ_{Proc} on the set* Proc*, and define*

$$\#(\phi) := \mu_{\mathsf{Proc}}([\![\phi]\!])$$

Definition 11 (witness). *TBD*

7 Conclusions and future work

Testing physical space You, gentle reader, may wonder why of all the theorems to be proved given this set up we pick the one above. In some sense it's hardly central to quantum mechanics. We see it as central in the sense that it firmly establishes a notion of physical space arising from a notion of the equivalence of behavior. Relating bisimulation to a metric is a big step forward, but one is faced with interpreting the relationship of that metric space to something more physical. Quantum mechanical notions of "physical" space are still far from intuitive, but by relating this idea of distance as testing to calculations that predict physical circumstances we are making a not insignificant step forward toward an understanding of the physical space we inhabit as essentially dynamic.

Effectivity and simulation One of the observations we have yet to make is that the entire program spelled out here is effective. We have built various interpreters for the reflective calculus at work in this interpretation. In principle, then, we can simulate quantum mechanics on a computer. The place where the simulation may lose fidelity is the infinitely branching summation for the annihilator.

[2] Of course, this formula also says the server never goes down, either – or at least is always willing to take such input...;-)

In this connection i also want to point out that the evaluation style calculation of the inner product puts the non-determinism of the summation right at the heart of measurement. This suggests that Milner's original reduction-based formulation of the dynamics of his calculi in terms of sums was not just notationally suggestive of a notion of measure-and-continue but captured some significant part of a physical intuition.

Quantum continuations In light of this last observation i want to point out that the connection to the observation regarding iterated experiments and continuations made in section 4.2. An account of continuation is necessary to provide a truly compositional story of physical processes involving quantum operations. As noted, in a real lab, when a measurement is made the observation can be made to feed into another device that then makes another measurement conditioned on the results of the first. This is not a mere nicety; it is a pragmatic requirement of a theory that provides a computational basis for the design of large-scale processes (i.e., consisting of many steps, or having a high degree of branching, or enjoying some other complexity characteristic) involving quantum operations. It shows up in descriptions of quantum cryptographic protocols [**?**] [**?**] and computational processes that mix classical and quantum computations [**?**] [**?**]. This suggests that there might be advantages to a more uniform account of dynamics that encompasses both quantum and computational paradigms.

Quantum logic In this connection, we also note that by virtue of having the Hennessy-Milner construction, we can pull the construction through the interpretation of QM. This gives us a natural candidate for a quantum logic that enjoys an extremely tight connection with it's domain of interpretation, making the construction much less ad hoc (rather it is the image of functor!).

Quantum probabiity i have questions about the basis of the interpretation of inner product as probability amplitude. In particular, using which axiomatization of probability theory does the notion of probability amplitude earn the right to be so dubbed? In other words, where is the proof that the operation for calculating a probability amplitude (and then squaring) satisfies the axioms of what it means to calculate a probability? Even if such a proof exists (i have yet to find it in the literature), i wonder if it might not be possible to turn things on their heads. Can we view the calculation of the probability amplitude as an axiomatization of probability? If so, then the definition we give for calculating probability amplitude may provide the basis for an *effective* theory of probability.

Quantum vs "biological" information Finally, i want to conclude with a more philosophical observation. At a recent workshop in which QM was a predominant topic i noticed something about quantum information. The speaker was giving a riveting discussion of axiomatic QM and showing how properties of "no cloning" and "no deleting" emerged as consequences of the axiomatization. Theorems of this form are necessary to give us a sense of confidence that our axioms characterize the physical theory. What struck me, though, was that if quantum information is neither erasable nor replicable it is markedly different from *life*. Two of the things we know about life is that

– it ends;
– to gain some measure of persistence, to transcend it's finitude it is imminently copyable.

Both of these qualities are summarized succinctly in the aphorism: all flesh is grass. For me these two kinds of "information" – call them quantum and biological – are end points on a spectrum of strategies for persistence. At one end, we have those curious entities that enjoy uniqueness and permanence; at the other, we have those who in the face of a certain end and an uncertain present make a go of passing something on. To me one of the more remarkable aspects of the latter strategy is that in the presence of noise (and certain features of copying) we get a kind of dynamism, a chance for improvement against a given persistent condition.

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.

8 Appendix : reflection in more detail

8.1 Name equivalence

We now come to one of the first real subtleties of this calculus as compared to other process calculi. Both the calculation of the free names of a process and the determination of structural congruence between processes critically depend on being able to establish whether two names are equal. In the case of the calculation of the free names of an input-guarded process, for example, to remove the bound name we must determine whether it is in the set of free names of the continuation. Likewise, structural congruence includes α-equivalence. But, establishing α-equivalence between the processes $x?(z).w!(y!(z))$ and $x?(v).w!(y!(v))$, for instance, requires calculating a substitution, e.g. $x?(v).w!(y!(v))\{z/v\}$. But this calculation requires, in turn, being able to determine whether two names, in this case the name in the object position of the output, and the name being substituted for, are equal.

As will be seen, the equality on names involves structural equivalence on processes, which in turn involves alpha equivalence, which involves name equivalence. This is a subtle mutual recursion, but one that turns out to be well-founded. Before presenting the technical details, the reader may note that the grammar above enforces a strict alternation between quotes and process constructors. Each question about a process that involves a question about names may in turn involve a question about processes, but the names in the processes the next level down, as it were, are under fewer quotes. To put it another way, each 'recursive call' to name equivalence will involve one less level of quoting, ultimately bottoming out in the quoted zero process.

Let us assume that we have an account of (syntactic) substitution and α-equivalence upon which we can rely to formulate a notion of name equivalence, and then bootstrap our notions of substitution and α-equivalence from that. We take name equivalence, written $\equiv_{\mathsf{N}}$, to be the smallest equivalence relation generated by the following rules.

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

8.2 Syntactic substitution

Now we build the substitution used by α-equivalence. 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, $\widehat{s} : \mathsf{Proc} \to \mathsf{Proc}$ by the following equations.

$$(0)\{\widehat{@Q/@P}\} := 0$$

$$(R \mid S)\{\widehat{@Q/@P}\} := (R)\{\widehat{@Q/@P}\} \mid (S)\{\widehat{@Q/@P}\}$$

$$(x?(y).R)\{\widehat{@Q/@P}\} := (x)\{@Q/@P\}(z).((R\{\widehat{z/y}\})\{\widehat{@Q/@P}\})$$

$$(x!(R))\{\widehat{@Q/@P}\} := (x)\{@Q/@P\}!(R\{\widehat{@Q/@P}\})$$

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

where

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

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.

But, given these mutual recursions, the question is whether the calculation of $\equiv_{\mathsf{N}}$ (respectively, $\equiv$, $\equiv_\alpha$) terminates. To answer this question it suffices to formalize our intuitions regarding level of quotes, or quote depth, $\#(x)$, of a name x as follows.

$$\#(@P) = 1 + \#(P)$$
$$\#(P) = \begin{cases} max\{\#(x) : x \in \mathsf{N}(P)\} & \mathsf{N}(P) \neq \emptyset \\ 0 & otherwise \end{cases}$$

The grammar ensures that $\#(@P)$ is bounded. Then the termination of $\equiv_{\mathsf{N}}$ (respectively, $\equiv$, $\equiv_\alpha$) is an easy induction on quote depth.

Remark 16. Note that by a related piece of reasoning we can see that $\forall P.@P \notin \mathsf{FN}(P)$.

8.3 Semantic substitution

The substitution used in α-equivalence is really only a device to formally recognize that binding occurrences do not depend on the specific names. It is not the engine of computation. The proposal here is that while synchronization is the driver of that engine, the real

engine of computation is a semantic notion of substitution that recognizes that a dropped name is a request to run a process. Which process? Why the one whose code has been bound to the name being dropped. Formally, this amounts to a notion of substitution that differs from syntactic substitution in its application to a dropped name.

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

In the remainder of the paper we will refer to semantic and syntactic substitutions simply as substitutions and rely on context to distinguish which is meant. Similarly, we will abuse notation and write $\{y/x\}$ for $\widehat{\{y/x\}}$.