Notes on interpreting quantum mechanics in a reflective process calculus

L.G. Meredith[1]

Partner, Biosimilarity
505 N72nd St, Seattle, WA 98103, USA,
`lgreg.meredith@biosimilarity.com`

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

1 Introduction

In this draft of the material i am going to have to dispense with the usual writing conventions adopted in papers on these topics. i'm going to have adopt whatever tone i need at the time i'm writing up the calculations. Sometimes this may be very conversational; others it may be the barest mathematical grunts; others still it may be that i have lifted text from one of my other papers because the exposition of some point was better said there. i hope that my readers are not unduly put out by this decision. i'm not doing this to flout convention or be rebellious. i find these calculations very technically challenging. To keep everything going technically, something has to give; i have to let go of some cognitive burden. So, the academic writing style – with all of its trade-offs in terms of facilitating technical communication – is what i'm letting go of. Perhaps subsequent drafts can be tightened and polished, but for now, i'm going to speak as if we were sitting together in a coffee shop with a laptop, wifi and a pad of paper and a pencil.

So, here's what i have to say. We – you and i, comfortably ensconced in our coffee shop and well-equipped with our tools – can realize and carry out the calculations of quantum mechanics over a very different 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 we can show, by calculating together, coincides with the metric induced by the inner product.

There are a lot of reasons why you might be interested in seeing calculations of this form. Here's why i'm interested. For the past several centuries there has been no competitor to the "Newtonian" account of dynamics. As a result the predominant share of accounts of dynamical systems and situations have had to be formulated in terms of the Newtonian machinery. i view this as an intellectually dangerous position to occupy. Everything, despite it's intrinsic shape, turns into a nail to be hit with this hammer. Recently, however, the theory of computation has matured to the point where we have candidates for theories of dynamics that offer very different perspective on reasoning about dynamical systems and situations. Testing these candidates against very successful accounts of dynamical situations, like quantum mechanics, is going to give us some sense of how mature they are and some measure of the quality of these accounts of dynamics.

1.1 Summary of contributions and outline of paper

So, we're going to develop an interpretation of the operations of quantum mechanics normally interpreted by Hilbert spaces and operators. We're going 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 supressed) 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. Thus,

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.

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) [22] and Hoare's Communicating Sequential Processes (CSP) [13] [4] [15] [14], and continue through the development of the so-called mobile process calculi, e.g. Milner, Parrow and Walker's π-calculus [25], Cardelli and Caires's spatial logic [7] [6] [5], or Meredith and Radestock's reflective calculi [19] [20]. The process-calculus-based algebraical specification of processes has expanded its scope of applicability to include the specification, analysis, simulation and execution of processes in domains such as:

- telecommunications, networking, security and application level protocols [1] [2] [16] [17];
- programming language semantics and design [16] [12] [11] [32];
- webservices [16] [17] [18];
- and biological systems [8] [9] [28] [27].

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 [23], the process calculi are so organized that the behavior —the semantics— of a system may be composed from the behavior of its components [10]. 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 [26] [29] [30] [31]. In particular, *bisimulation* encapsulates an effective notion of process equivalence that has been used in applications as far-ranging as algorithmic games semantics [3] and the construction of model-checkers [5]. The essential notion can be stated in an intuitively recursive formulation: a *bisimulation* between two processes P and Q is an equivalence relation E relating P and Q such that: whatever action of P can be observed, taking it to a new state P', can be observed of Q, taking it to a new state Q', such that P' is related to Q' by E and vice versa. P and Q are *bisimilar* if there is some bisimulation relating them. Part of what makes this notion so robust and widely applicable is that it is parameterized in the actions observable of processes P and Q, thus providing a framework for a broad range of equivalences and up-to techniques [21] all governed by the same core principle [29] [30] [31].

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

Process grammar

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

AGENT
$$A ::= (\boldsymbol{x})P \mid \langle\!|\boldsymbol{P}|\!\rangle$$

PROCESS
$$P, Q ::= N \mid P|Q \mid \urcorner x \ulcorner$$

NAME
$$x, y ::= \ulcorner P \urcorner$$

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

$$x?(\boldsymbol{y}).P := x.(\boldsymbol{y})P \qquad x\langle\!|\boldsymbol{P}|\!\rangle := x.\langle\!|\boldsymbol{P}|\!\rangle \qquad x!(y) := x\langle\!|\urcorner y \ulcorner|\!\rangle$$

$$\Pi_{i=0}^{n-1} P_i := P_0 | \ldots | P_{n-1}$$

Structural congruence

Free and bound names and alpha-equivalence. At the core of structural equivalence is alpha-equivalence which identifies process that are the same up to a change of variable. Formally, we recognize the distinction between free and bound names. The free names of a process, $\mathcal{FN}(P)$, may be calculated recursively as follows:

$$\mathcal{FN}(0) := \emptyset$$

$$\mathcal{FN}(x?(\boldsymbol{y}).P) := \{x\} \cup (\mathcal{FN}(P) \setminus \{\boldsymbol{y}\})$$

$$\mathcal{FN}(x\langle\!|\boldsymbol{P}|\!\rangle) := \{x\} \cup \{\boldsymbol{P}\}$$

$$\mathcal{FN}(P|Q) := \mathcal{FN}(P) \cup \mathcal{FN}(Q)$$

$$\mathcal{FN}(P + Q) := \mathcal{FN}(P) \cup \mathcal{FN}(Q)$$

$$\mathcal{FN}(\urcorner x \ulcorner) := \{x\}$$

The bound names of a process, $\mathcal{BN}(P)$, are those names occurring in P that are not free. For example, in $x?(y).0$, the name x is free, while y is bound.

Definition 1. *Then two processes, P, Q, are alpha-equivalent if $P = Q\{\boldsymbol{y}/\boldsymbol{x}\}$ for some $\boldsymbol{x} \in \mathcal{BN}(Q), \boldsymbol{y} \in \mathcal{BN}(P)$, where $Q\{\boldsymbol{y}/\boldsymbol{x}\}$ denotes the capture-avoiding substitution of $\boldsymbol{y}$ for $\boldsymbol{x}$ in Q.*

Definition 2. *The* structural congruence *[29] , $\equiv$, between processes is the least congruence containing alpha-equivalence, satisfying the abelian monoid laws (associativity, commutativity and* 0 *as identity) for parallel composition* | *and for summation* +*.*

2.2 Name equivalence

We take name equivalence, written $\equiv_N$, to be the smallest equivalence relation generated by the following rules.

$$\textsc{Quote-drop}\ \frac{}{\ulcorner \urcorner x \ulcorner \urcorner \equiv_N x} \qquad \frac{P \equiv Q}{\ulcorner P \urcorner \equiv_N \ulcorner Q \urcorner}\ \textsc{Struct-equiv}$$

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.

2.3 Substitution

We use $Proc$ for the set of processes, $\ulcorner Proc \urcorner$ for the set of names, and $\{\boldsymbol{y}/\boldsymbol{x}\}$ to denote partial maps, $s : \ulcorner Proc \urcorner \rightarrow \ulcorner Proc \urcorner$. A map, s lifts, uniquely, to a map on process terms, $\widehat{s} : Proc \rightarrow Proc$ by the following equations.

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

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

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

$$(x\langle\!| R |\!\rangle)\widehat{\{\ulcorner Q \urcorner / \ulcorner P \urcorner\}} := (x)\{\ulcorner Q \urcorner / \ulcorner P \urcorner\}\langle\!| R\widehat{\{\ulcorner Q \urcorner / \ulcorner P \urcorner\}} |\!\rangle$$

$$(\urcorner x \ulcorner)\widehat{\{\ulcorner Q \urcorner / \ulcorner P \urcorner\}} := \begin{cases} Q & x \equiv_N \ulcorner P \urcorner \\ \urcorner x \ulcorner & \textit{otherwise} \end{cases}$$

where

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

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

Remark 1. One consequence of these definitions is that $\forall P . \ulcorner P \urcorner \notin \mathcal{FN}(P)$.

2.4 Dynamic quote: an example

Anticipating something of what's to come, consider applying the substitution, $\widehat{\{u/z\}}$, to the following pair of processes, $w\langle\!|y!(z)|\!\rangle$ and $w[\ulcorner y!(z) \urcorner]$.

$$w\langle\!|y!(z)|\!\rangle\widehat{\{u/z\}} = w\langle\!|y!(u)|\!\rangle$$
$$w[\ulcorner y!(z) \urcorner]\widehat{\{u/z\}} = w[\ulcorner y!(z) \urcorner]$$

Because the body of the process between quotes is impervious to substitution, we get radically different answers. In fact, by examining the first process in an input context, e.g. $x?(z).w\langle\!|y!(z)|\!\rangle$, we see that the process under the lift operator may be shaped by prefixed inputs binding a name inside it. In this sense, the lift operator will be seen as a way to dynamically construct processes before reifying them as names.

Finally equipped with these standard features we can present the dynamics of the calculus.

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.

$$\textsc{Comm}\quad \frac{x_{src} \equiv_N x_{trgt} \qquad \boldsymbol{y} \cap \boldsymbol{v} = \emptyset \qquad |\boldsymbol{y}| = |\boldsymbol{z}|}{R_L + x_{trgt}?(\boldsymbol{y})P \mid x_{src}\langle\!|\boldsymbol{Q}|\!\rangle + R_R \rightarrow P\{\ulcorner \boldsymbol{Q} \urcorner / \boldsymbol{y}\}}$$

$$\textsc{Par}\quad \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \textsc{Equiv}\quad \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}$$

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

3 Replication

As mentioned before, it is known that replication (and hence recursion) can be implemented in a higher-order process algebra [29]. As our first example of calculation with the machinery thus far presented we give the construction explicitly in the ρ-calculus.

$$\begin{aligned} D_x &:= x?(y).(x!(y)|\urcorner y \ulcorner) \\ !_x P &:= x\langle\!| D_x | P |\!\rangle | D_x \end{aligned}$$

$$\begin{array}{ll} !_x P & \\ = & x\langle\!|(x?(y).(x!(y)|\urcorner y \ulcorner))|P|\!\rangle | x?(y).(x!(y)|\urcorner y \ulcorner) \\ \rightarrow & (x!(y)|\urcorner y \ulcorner)\{\ulcorner(x?(y).(\urcorner y \ulcorner | x!(y)))|P\urcorner / y\} \\ = & x!(\ulcorner(x?(y).(x!(y)|\urcorner y \ulcorner))|P\urcorner)|(x?(y).(x!(y)|\urcorner y \ulcorner))|P \\ \rightarrow & \ldots \\ \rightarrow^* & P|P|\ldots \end{array}$$

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.

$$!u?(v).P := x\llbracket u?(v).(D(x)|P)\rrbracket|D(x)$$

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 [24].

Definition 3. *An* observation relation*, $\downarrow_{\mathcal{N}}$, over a set of names, $\mathcal{N}$, is the smallest relation satisfying the rules below.*

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

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

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

Definition 4. *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 $P \stackrel{\bullet}{\approx}_{\mathcal{N}} Q$, 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 like 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 M to a process P, written $M[P]$, is tantamount to filling the hole in M 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.

SUMMATION
$$M_M, M_N ::= \Box \mid x.M_A \mid M_M + M_N$$

AGENT
$$M_A ::= (\boldsymbol{x})M_P \mid \langle\!|P_0, \ldots, M_P, \ldots, P_N|\!\rangle$$

PROCESS
$$M_P ::= M_N \mid P|M_P$$

Definition 5 (contextual application). *Given a context M, and process P, we define the* contextual application, *$M[P] := M\{P/\Box\}$. That is, the contextual application of M to P is the substitution of P for $\Box$ in M.*

Contextual duality Note that contexts extend the quotation operation to a family of operations from processes to names. Given a context, M, we can define a new map, $\ulcorner M \urcorner$ by $\ulcorner M \urcorner[P] := \ulcorner M[P] \urcorner$. 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 M we can construct M_x^* such that

$$M_x^* | x \langle\!| P |\!\rangle \rightarrow M[P]$$

namely,

$$M_x^* := x?(u).M[\urcorner u \ulcorner]$$

3.1 Additional notation

It will sometimes be convenient to denote the process a name quotes. We already have the notation $x = \ulcorner P \urcorner$, but 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, $\ulcorner \overset{\vee}{x} \urcorner \equiv_N x$; so, the notation is consistent with previous definitions.

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 [19]. This notation overloads to vectors of names: $\boldsymbol{x}^\pi := (x_i^\pi \mid 0 \leq i < |\boldsymbol{x}|)$ where $\pi \in \{L, R\}$.

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

4 Interpretation of QM

4.1 Supporting definitions

Multiplication

$$\ulcorner Q \urcorner \cdot \ulcorner R \urcorner := \ulcorner Q|R \urcorner$$

$$\ulcorner Q \urcorner \cdot P := P\{\ulcorner Q|R \urcorner / \ulcorner R \urcorner : \ulcorner R \urcorner \in \mathcal{FN}(P)\}$$

Discussion The first line needs little explanation. The second line 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 $x, y, z \in \ulcorner Proc \urcorner$ and $P \in Proc$

$$x \cdot \ulcorner 0 \urcorner \equiv x \qquad x \cdot y \equiv y \cdot x \qquad x \cdot (y \cdot z) \equiv (x \cdot y) \cdot z$$

$$\ulcorner 0 \urcorner \cdot P \equiv P$$

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

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

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

Tensor of sums First note that all summations, including 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.

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

$$:= \Sigma_i \Sigma_{i'} \ulcorner \overset{\vee}{x}_i \mid \overset{\vee}{y}_{i'} \urcorner.(A_i \otimes B_{i'}) \mid \Sigma_{i'} \Sigma_i \ulcorner \overset{\vee}{y}_{i'} \mid \overset{\vee}{x}_i \urcorner.(B_{i'} \otimes A_i)$$

$$\mid \Sigma_j \Sigma_{j'} \ulcorner \overset{\vee}{x}_j \mid \overset{\vee}{y}_{j'} \urcorner.(A_j \otimes B_{j'}) \mid \Sigma_{j'} \Sigma_j \ulcorner \overset{\vee}{y}_{j'} \mid \overset{\vee}{x}_j \urcorner.(B_{j'} \otimes A_j)$$

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.

$$P \otimes \urcorner x \ulcorner := P | \urcorner x \ulcorner$$

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

$$(\boldsymbol{x})P \otimes (\boldsymbol{y})Q$$

$$:= (x_0^L|y_0^R, \ldots, x_0^L|y_n^R, \ldots, x_m^L|y_0^R, \ldots, x_m^L|y_n^R)(P\{\boldsymbol{x}^L/\boldsymbol{x}\} \otimes Q\{\boldsymbol{y}^R/\boldsymbol{y}\})$$

$$\langle\!|\boldsymbol{P}|\!\rangle \otimes \langle\!|\boldsymbol{Q}|\!\rangle$$

$$:= \langle\!| P_0 \otimes Q_0, \ldots, P_0 \otimes Q_n, \ldots, P_m \otimes Q_0, \ldots, P_m \otimes Q_n |\!\rangle$$

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 $x \in \ulcorner Proc \urcorner$ and $P, Q \in Proc$

$$(x \cdot P) \otimes Q \equiv x \cdot (P \otimes Q) \equiv P \otimes (x \cdot Q) \qquad P \otimes 0 \equiv P$$

that the reader is invited to verify.

Annihilation

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

$$P^{\underline{\perp}} := \Sigma_{Q \in P^{\perp}} \ulcorner Q \urcorner ?(y).(\urcorner y \ulcorner | Q) | \Sigma_{Q \in P^{\perp}} \ulcorner Q \urcorner \langle\!| \Box |\!\rangle$$

Discussion The reader will note that $P^{\perp}$ is a *set* of processes, while $P^{\underline{\perp}}$ 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 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 $P^{\underline{\perp}}$ 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.

Evaluation We fix M a domain of fully abstract interpretation with an equality coincident with bisimulation. We take $[\![\cdot]\!] : Proc \to M$ 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 8. The reader is invited to verify that $\int(P^{\underline{\perp}}[P]) = 0$.

4.2 Quantum mechanics

Table 4.2 gives the core operational definitions

quantum mechanics	process calculus
scalar	$x := \ulcorner P \urcorner$
state vector	$\mid P\rangle := P$
dual	$\mid P\rangle^* := \langle P^{\perp} \mid := \ulcorner P^{\perp} \urcorner[-]$
matrix	$\Sigma_\alpha \mid P_\alpha\rangle x_\alpha \langle Q_\alpha \mid$
vector addition	$\mid P\rangle + \mid Q\rangle := \mid P \mid Q\rangle$
tensor product	$\mid P\rangle \otimes \mid Q\rangle := \mid P \otimes Q\rangle$
inner product	$\langle P \mid Q\rangle := \ulcorner \int P^{\perp}[Q] \urcorner$

Table 2. QM - operational definitions

where

$$\mid P\rangle\langle Q \mid := \mid P\rangle \ulcorner 0 \urcorner \langle Q \mid \qquad \mid P\rangle x \langle Q \mid := (\mid P\rangle, x, \langle Q \mid)$$

$$(\mid P\rangle x \langle Q \mid)(\mid R\rangle) := x \cdot \langle Q \mid R\rangle \cdot \mid P\rangle$$

$$(\mid P\rangle x \langle Q \mid)(\langle R \mid) := x \cdot \langle R \mid P\rangle \cdot \langle Q \mid$$

Discussion As promised: vectors (aka states) are represented as processes; duals as contextual duals; inner product definition should be compared with standard inner product definition for ....

Remark 9. Assuming $\int(P^{\perp}[P]) = 0$, the reader is invited to verify that $(\mid P\rangle x \langle P \mid)(\mid P\rangle) = x \cdot \mid P\rangle$.

Remark 10. The reader is invited to verify that $\langle P \mid Q\rangle$ could equally well have been written $\ulcorner \int \overset{\vee}{x} \urcorner$ where $x = \langle P^{\perp} \mid (Q)$.

One of the motivations for this remark is that there is another way to factor these operations. We could package up evaluation in the dual:

$$\mid P\rangle^* := \langle \int P^{\perp} \mid := \ulcorner \int P^{\perp} \urcorner[-]$$

and then have inner product defined by

$$\langle P \mid Q\rangle := \langle P \mid (Q)$$

Hopefully, experience with the calculations will provide guidance on the best factoring.

Remark 11. Assuming $\int(P^{\perp}[P]) = 0$, the reader is invited to verify that $\forall P, Q.(\,|\,0\rangle\langle Q\,|\,)(\,|\,0\rangle) = \,|\,0\rangle$ and dually $(\,|\,P\rangle\langle 0\,|\,)(\langle 0\,|\,) = \langle 0\,|\,$.

Remark 12. 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 characterizes.

For now, we will simply bookmark the notion with $\overline{x}$.

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

$$(\Sigma_{\alpha} \mid P_{\alpha}\rangle x_{\alpha}\langle Q_{\alpha}\mid)^{\dagger} = \Sigma_{\alpha} \mid (Q_{\alpha}^{\perp})^{*}\rangle \overline{x}_{\alpha}\langle P_{\alpha}^{\perp}\mid$$

But, $(Q_{\alpha}^{\perp})^{*}$ requires a name along which to communicate the process to achieve the context application.

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 (an asynchronous version of) 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 13. The attentive reader may be a bit concerned. Certainly, the collection S, K and I is a finite set of combinators. 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 $[\![[\![\langle P \mid Q \rangle]\!]_\phi]\!] = [\![\ulcorner D \urcorner_\phi(P,Q)]\!]$ where

Definition 6 (metric).

$$\Delta(P,Q) := \{\phi \mid (P \models \phi \;\&\; Q \not\models \phi) \vee (P \not\models \phi \;\&\; Q \models \phi)\}$$

$$D_\phi(P,Q) := \bigvee_{\phi \in \Delta(P,Q)} \phi \qquad\qquad \ulcorner D \urcorner_\phi(P,Q) := \ulcorner \bigvee_{\phi \in \Delta(P,Q)} \phi \urcorner$$

$$D(P,Q) := min\{\#(\phi) \mid \phi \in \Delta(P,Q)\}$$

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
$\phi, \psi ::= \mathit{true} \mid \neg\phi \mid \phi \& \psi$	$\mid \ 0 \ \mid \phi \vert \psi$

NOMINAL CONNECTIVES	BEHAVIORAL CONNECTIVES	FIXPT CONNECTIVES
$\mid \urcorner b \ulcorner \mid \forall n : \psi \, . \, \phi$	$\mid \langle a?\boldsymbol{b}\rangle\phi \mid a\langle\!\lvert \boldsymbol{\phi} \rvert\!\rangle$	$\mid \mathsf{rec}\ X \, . \, \phi \mid X$

PATTERNS	LITERALS
$a ::= \ulcorner \phi \urcorner \mid b$	$b ::= \ulcorner P \urcorner \mid n$

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 [?], we give the semantics in terms of sets of processes (and names). We need the notion of a valuation $v : \mathcal{V} \to \wp(Proc)$, and use the notation $v\{\mathcal{S}/X\}$ to mean

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

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

$$\begin{aligned} [\![-]\!](-) &: PForm \times [\mathcal{V} \to \wp(Proc)] \to \wp(Proc) \\ (\!| - |\!)(-) &: QForm \times [\mathcal{V} \to \wp(Proc)] \to \wp(\ulcorner Proc \urcorner) \end{aligned}$$

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

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

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

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,

$$\phi \Rightarrow \psi \triangleq \neg(\phi \& \neg\psi)$$
$$\phi \vee \psi \triangleq \neg(\neg\phi \& \neg\psi)$$

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

An example

Controlling access to namespaces Suppose that $\ulcorner \phi \urcorner$ 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

$$\langle \ulcorner \phi \urcorner ? b \rangle \; true \, \& \neg \langle \ulcorner \neg\phi \urcorner ? b \rangle true$$

which simply says the the process is currently able to take input from a name in the namespace $\ulcorner \phi \urcorner$ 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 $\ulcorner \phi \urcorner$ throughout the lifetime of its behavior [1]

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

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 $\ulcorner \phi \urcorner$ without the need for the additional overhead of the watchdog machinery.

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

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

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

Definition 8 (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.

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 the physics.

Quantum continuations In light of this last observation i want to point out that the predominant account of quantum mechanics is missing a key aspect of a truly compositional story of the physical situation. 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 means that after the superposition was collapsed the entire experimental set up remained in superposition. While QM offers a means of writing this down it doesn't quite line up well with the well-trodden formulation of computation and continuation that we see so succinctly expressed in Milner's calculi. This suggests that there might be advantages to this account of dynamics waiting to be explored.

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. Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *POPL*, pages 33–44, 2002.
2. Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 3(298):387–415, 2003.
3. Samson Abramsky. Algorithmic game semantics and static analysis. In *SAS*, page 1, 2005.
4. S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. Assoc. Comput. Mach.*, 31(3):560–599, 1984.
5. Luís Caires. Behavioral and spatial observations in a logic for the pi-calculus. In *FoSSaCS*, pages 72–89, 2004.
6. Luís Caires and Luca Cardelli. A spatial logic for concurrency (part i). *Inf. Comput.*, 186(2):194–235, 2003.
7. Luís Caires and Luca Cardelli. A spatial logic for concurrency - ii. *Theor. Comput. Sci.*, 322(3):517–565, 2004.
8. Luca Cardelli. Brane calculi. In *CMSB*, pages 257–278, 2004.
9. Vincent Danos and Cosimo Laneve. Core formal molecular biology. In Pierpaolo Degano, editor, *ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2003.
10. Wan Fokkink, Rob van Glabbeek, and Paulien de Wind. Compositionality of Hennessy-Milner logic through structural operational semantics. In *Fundamentals of computation theory*, volume 2751 of *Lecture Notes in Comput. Sci.*, pages 412–422. Springer, Berlin, 2003.
11. Cédric Fournet, Fabrice Le Fessant, Luc Maranget, and Alan Schmitt. Jocaml: A language for concurrent distributed and mobile programming. In *Advanced Functional Programming*, pages 129–158, 2002.
12. 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.
13. C. A. R. Hoare. *Communicating sequential processes*. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985. With a foreword by Edsger W. Dijkstra.
14. C. A. R. Hoare. Notes on communicating sequential systems. In *Control flow and data flow: concepts of distributed programming (Marktoberdorf, 1984)*, volume 14 of *NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci.*, pages 123–204. Springer, Berlin, 1985.
15. C. A. R. Hoare. Algebraic specifications and proofs for communicating sequential processes. In *Logic of programming and calculi of discrete design (Marktoberdorf, 1986)*, volume 36 of *NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci.*, pages 277–300. Springer, Berlin, 1987.
16. Allen L. Brown Jr., Cosimo Laneve, and L. Gregory Meredith. Piduce: A process calculus with native xml datatypes. In Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro, editors, *EPEW/WS-FM*, volume 3670 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2005.
17. Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *FoSSaCS*, pages 282–298, 2005.
18. Greg Meredith and Steve Bjorg. Contracts and types. *Commun. ACM*, 46(10):41–47, 2003.

19. L. Gregory Meredith and Matthias Radestock. A reflective higher-order calculus. *Electr. Notes Theor. Comput. Sci.*, 141(5):49–67, 2005.
20. L.G. Meredith and Matthias Radestock. A reflective higher-order calculus. In Mirko Viroli, editor, *ETAPS 2005 Satellites*. Springer-Verlag, 2005.
21. R. Milner and D. Sangiorgi. Techniques of weak bisimulation up-to, 1992.
22. Robin Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.
23. Robin Milner. Elements of interaction - turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.
24. Robin Milner. The polyadic π-calculus: A tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
25. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. I, II. *Inform. and Comput.*, 100(1):1–40, 41–77, 1992.
26. Robin Milner and Davide Sangiorgi. Barbed bisimulation. In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.
27. Corrado Priami, Aviv Regev, Ehud Y. Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.
28. Amitai Regev and Ehud Y. Shapiro. Cells as computation. In Corrado Priami, editor, *CMSB*, volume 2602 of *Lecture Notes in Computer Science*, pages 1–3. Springer, 2003.
29. David Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
30. Davide Sangiorgi. On the proof method for bisimulation (extended abstract). In Jirí Wiedermann and Petr Hájek, editors, *MFCS*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer, 1995.
31. Davide Sangiorgi. Bisimulation in higher-order process calculi. *Information and Computation*, 131:141–178, 1996.
32. Pawel T. Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In *ASA/MA*, pages 2–12, 1999.

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 $x?(z).w\langle\!|y!(z)|\!\rangle$ and $x?(v).w\langle\!|y!(v)|\!\rangle$, for instance, requires calculating a substitution, e.g.

$x?(v).w\langle\!|y!(v)|\!\rangle\{z/v\}$. 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 $\equiv_N$, to be the smallest equivalence relation generated by the following rules.

$$\textsc{Quote-drop}\ \frac{}{\ulcorner\urcorner x\ulcorner\urcorner \equiv_N x} \qquad\qquad \frac{P \equiv Q}{\ulcorner P\urcorner \equiv_N \ulcorner Q\urcorner}\ \textsc{Struct-equiv}$$

8.2 Syntactic substitution

Now we build the substitution used by α-equivalence. We use $Proc$ for the set of processes, $\ulcorner Proc\urcorner$ for the set of names, and $\{\boldsymbol{y}/\boldsymbol{x}\}$ to denote partial maps, $s : \ulcorner Proc\urcorner \to \ulcorner Proc\urcorner$. A map, s lifts, uniquely, to a map on process terms, $\widehat{s} : Proc \to Proc$ by the following equations.

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

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

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

$$(x\langle\!| R |\!\rangle)\{\widehat{\ulcorner Q\urcorner / \ulcorner P\urcorner}\} := (x)\{\ulcorner Q\urcorner / \ulcorner P\urcorner\}\langle\!| R\{\widehat{\ulcorner Q\urcorner / \ulcorner P\urcorner}\} |\!\rangle$$

$$(\urcorner x \ulcorner)\{\widehat{\ulcorner Q\urcorner / \ulcorner P\urcorner}\} := \begin{cases} \urcorner\ulcorner Q \urcorner\ulcorner & x \equiv_N \ulcorner P \urcorner \\ \urcorner x \ulcorner & \mathit{otherwise} \end{cases}$$

where

$$(x)\{\ulcorner Q\urcorner / \ulcorner P\urcorner\} = \begin{cases} \ulcorner Q \urcorner & x \equiv_N \ulcorner P \urcorner \\ x & \mathit{otherwise} \end{cases}$$

and z is chosen distinct from $\ulcorner P\urcorner$, $\ulcorner Q\urcorner$, 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 $\equiv_N$ (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.

$$\begin{aligned} \#(\ulcorner P\urcorner) &= 1 + \#(P) \\ \#(P) &= \begin{cases} max\{\#(x) : x \in \mathcal{N}(P)\} & \mathcal{N}(P) \neq \emptyset \\ 0 & \mathit{otherwise} \end{cases} \end{aligned}$$

The grammar ensures that $\#(\ulcorner P\urcorner)$ is bounded. Then the termination of $\equiv_N$ (respectively, $\equiv$, $\equiv_\alpha$) is an easy induction on quote depth.

Remark 14. Note that by a related piece of reasoning we can see that $\forall P. \ulcorner P\urcorner \notin \mathcal{FN}(P)$.

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.

$$(\urcorner x \ulcorner)\{\widehat{\ulcorner Q \urcorner / \ulcorner P \urcorner}\} = \begin{cases} Q & x \equiv_N \ulcorner P \urcorner \\ \urcorner x \ulcorner & \mathit{otherwise} \end{cases}$$

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 $\widehat{\{y/x\}}$.