# A calculus from outer space!

Lucius Gregory Meredith[1]

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

**Abstract.** We describe a discrete calculus built from the intuitions informing the Einstein field equations.

## 1 Introduction

One of the notable things about the Einstein field equations [7] is their mutual recursion. The stress energy tensor describing the distribution of matter and energy informs the metric tensor, which in turn informs the stress energy tensor. Any red blooded computer scientist sees this and immediately wants to express it as a couple of mutually recursive functions and/or data structures.

Meanwhile, in computer science a revolution in the notion of location has happened and all but gone unnoticed. On the one hand, Huet's zipper [1] generalizes the intuitions of the Dedekind cut for all differentiable data structures. Specifically, for a data structure described by a differentiable functor, $T$, an element of the data type $\partial\, T \times T$ describes a location in an element of $T$ [3].

On the other, Milner's $\pi$-calculus transforms the $\lambda$-calculus by reifying the site of interaction as a channel, and introducing parallel composition [6]. This change simultaneously gives computations autonomy and a means of finding each other. Meredith's rho-calculus takes Milner's insight a step further [5], making the reified site of interaction into a fully first class entity with a complexity of structure equivalent to computations.

A question worth exploring is whether these two notions of location can be combined, yielding a site of interaction corresponding to a location in a computation. The thought is that such a reconciliation of the two ideas would allow for a computational calculus enjoying something like the intuitions underlying the dynamics in Einstein's field equations.

### 1.1 Summary of contributions and outline of paper

**Summary of contributions** We present a calculus, dubbed the space calculus (from outer space!) whose primitives are channels built from locations in computations reified as data structures. This gives us the ability to model a variety of space-like phenomena.

**Outline of paper** TBD

## 2 The calculus from outer space

### 2.1 Syntax and semantics

We now give a technical presentation of the calculus. 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, $\mathsf{Proc}$. 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.

#### Process grammar

PROCESS
$$P, Q ::= 0 \mid \mathsf{U}(x) \mid \mathsf{for}(y \leftarrow x)P \mid x!(Q) \mid P|Q \mid *x \mid \mathsf{COMM}(K)$$

NAME
$$x, y ::= @\langle K, P\rangle$$

CONTEXT
$$K ::= \square \mid \mathsf{for}(y \leftarrow x)K \mid x!(K) \mid P|K$$

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

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

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

$$\mathsf{FN}(\mathsf{COMM}(K)) = \mathsf{FN}(K) \qquad \mathsf{FN}(\square) = \emptyset \qquad \mathsf{FN}(\mathsf{for}(y \leftarrow x)K) = \{x\} \cup \mathsf{FN}(K) \setminus \{y\}$$

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

*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 $\mathsf{Proc}$ 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 $\sigma$ to range over lifted subsitutions, $\widehat{s}$, to write the application of a substitution, $\sigma$ to a process, $P$, with the substitution on the right, $P\sigma$, and the application of a substitution, $s$, to a name, $x$, using standard function application notation, $s(x)$. In this instance we choose not to swim against the tides of history. Thus,

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

$$0\sigma := 0$$

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

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

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

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

*where*

$$\widehat{\{w/v\}}(x) = \{w/v\}(x) = \begin{cases} w & x \equiv_{\mathsf{N}} v \\ x & otherwise \end{cases}$$

and $z$ is fresh for $P$. Our $\alpha$-equivalence will be built in the standard way from this substitution.

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

**Definition 4.** *The* structural congruence $\equiv$ *between processes [8] is the least congruence containing alpha-equivalence and satisfying the commutative monoid laws (associativity, commutativity and $\mathsf{0}$ as identity) for parallel composition $|$.*

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

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

The astute reader will have noticed that the mutual recursion of names and processes imposes a mutual recursion on alpha-equivalence and structural equivalence via name-equivalence. Fortunately, all of this works out pleasantly and we may calculate in the natural way, free of concern. The reader interested in the details is referred to [5].

*Remark 1.* One particularly useful consequence of these definitions is that $\forall K, P. @\langle K, P \rangle \notin \mathsf{FN}(P)$. 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.

### 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.

$$\textsc{Comm}\quad \frac{x_t \equiv_{\mathsf{N}} x_s}{\mathsf{COMM}(K) \mid \mathsf{for}(y \leftarrow x_t)P \mid x_s!(Q) \rightarrow P\{@\langle K, Q\rangle/y\}}$$

$$\textsc{Catalyze}\quad \mathsf{U}(x) \mid *@\langle K, Q\rangle \rightarrow \mathsf{COMM}(K) \mid x!(Q)$$

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

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

### 2.4 Dynamic quote: an example

TBD

## 3 Replication

TBD

**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 [6].

**Definition 6.** *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 7.** *An* $\mathcal{N}$-barbed bisimulation *over a set of names,* $\mathcal{N}$*, is a symmetric binary relation* $\mathcal{S}_{\mathcal{N}}$ *between agents such that* $P \; \mathcal{S}_{\mathcal{N}} Q$ *implies:*

1. *If* $P \to P'$ *then* $Q \Rightarrow Q'$ *and* $P' \; \mathcal{S}_{\mathcal{N}} Q'$.
2. *If* $P \downarrow_{\mathcal{N}} x$*, then* $Q \Downarrow_{\mathcal{N}} x$.

$P$ *is* $\mathcal{N}$*-barbed bisimilar to* $Q$*, written* $P \dot{\approx}_{\mathcal{N}} Q$*, if* $P \; \mathcal{S}_{\mathcal{N}} Q$ *for some* $\mathcal{N}$*-barbed bisimulation* $\mathcal{S}_{\mathcal{N}}$.

## 4 Main theorem

TBD

## 5 Conclusions and future work

TBD

*Acknowledgments.* The author wishes to thank Iceland Air.

# References


1. Gérard P. Huet. The zipper. *J. Funct. Program.*, 7(5):549–554, 1997.
2. Barbara Liskov. Keynote address - data abstraction and hierarchy. In *OOPSLA Addendum*, pages 17–34. ACM, 1987.
3. Conor McBride. Clowns to the left of me, jokers to the right (pearl): dissecting data structures. In *POPL*, pages 287–295. ACM, 2008.
4. L. Gregory Meredith and Matthias Radestock. Namespace logic: A logic for a reflective higher-order calculus. In *TGC* [5], pages 353–369.
5. L. Gregory Meredith and Matthias Radestock. A reflective higher-order calculus. *Electr. Notes Theor. Comput. Sci.*, 141(5):49–67, 2005.
6. Robin Milner. The polyadic $\pi$-calculus: A tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
7. C. W. Misner, K. S. Thorne, and J. A. Wheeler. *Gravitation.* San Francisco: W.H. Freeman and Co., 1973, 1973.
8. Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001.