Representing operational semantics with enriched Lawvere theories

Michael Stay [1] **and L. G. Meredith** [2]

1 **Pyrofex Corp., Kirkland, WA, USA**
stay@pyrofex.net
2 **RChain Cooperative**
greg@rchain.coop

Abstract

1998 ACM Subject Classification Dummy classification – please refer to http://www.acm.org/about/class/ccs98-html

Keywords and phrases Dummy keyword – please provide 1–5 keywords

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

[[Reference our MFPS paper, say the approach of section 7 is very different from that one and Yoshida's combinators.]]

2 Previous work

There's a long history and an enormous body of work on modeling term rewriting and operational semantics with various notions of category enriched over category-like structures. Here's a sampling. Seely [**?**] suggested using 2-categories for modeling the [[what kind?]] semantics of lambda calculus; [[does he have relations between 2-cells? Certainly confluent.]]. Barney Hilken [[The simply typed 2λ-calculus]], Tom Hirschowitz [[CARTESIAN CLOSED 2-CATEGORIES AND PERMUTATION EQUIVALENCE IN HIGHER-ORDER REWRITING]]. Stell [**?**] considered sesquicategories for term rewriting [[2-morphism involve substitutions of variables; we eliminate variables entirely]]. Lüth and Ghani [[Monads and Modular Term Rewriting]] use poset-enriched categories.

[[In each case above, one sentence about what they did and how their work differs from ours.]]

3 Gph-enriched categories

Here we review some standard definitions and results in enriched category theory; see [3], [9], [4], and [12] for more details.

A **directed multigraph with self loops**, hereafter **graph**, consists of a set E of edges, a set V of vertices, two functions $s, t\colon E \to V$ picking out the source and target of each edge, and a function $a\colon V \to E$ such that $s \circ a$ and $t \circ a$ are both the identity on V—that is, a equips each vertex in V with a chosen self loop. There are no constraints on E, V, s, or t, so a graph may have infinitely many vertices, infinitely many edges between any pair of vertices. A **graph homomorphism** from (E, V, s, t, a) to (E', V', s', t', a') is a pair of functions $(\epsilon\colon E \to E', \upsilon\colon V \to V')$ such that $\upsilon \circ s = s' \circ \epsilon$ and $\upsilon \circ t = t' \circ \epsilon$.

Gph is the category of graphs and graph homomorphisms. Gph has finite products: the terminal graph is the graph with one vertex and one loop, while the product of two graphs $(E, V, s, t, a) \times (E', V', s', t', a')$ is $(E \times E', V \times V', s \times s', t \times t', a \times a')$.

A **Gph-enriched category** is a category where each hom set $\hom(x, y)$ is equipped with a set $E_{x,y}$ and functions $s_{x,y}, t_{x,y} \colon E_{x,y} \to \hom(x, y)$; that is, each hom set is thought of as a set of vertices and is equipped with a set of edges making it into a graph. A Gph-enriched category has finite products if the underlying category does.

Any category is trivially Gph-enriched by taking the all the sets of edges to be empty. The category Gph is nontrivially Gph-enriched in multiple ways. Gph is a topos, and therefore cartesian closed, and therefore enriched over itself. Given two graph homomorphisms $F, F' \colon (E, V, s, t, a) \to (E', V', s', t', a')$, a **graph transformation** assigns to each vertex v in V an edge e' in E' such that $s'(e') = F(v)$ and $t'(e') = F'(v)$. Given any two graphs G and G', there is an exponential graph G'^G whose vertices are graph homomorphisms between them and whose edges are graph transformations.

A **Gph-enriched functor** between two Gph-enriched categories C, D is a functor between the underlying categories such that the graph structure on each hom set is preserved, *i.e.* the functions between hom sets are graph homomorphisms between the hom graphs.

Let S be a finite set, FinSet be a skeleton of the category of finite sets and functions between them, and FinSet/S be the category of functions into S and commuting triangles. A **multisorted Gph-enriched Lawvere theory**, hereafter **Gph-theory** is a Gph-enriched category with finite products Th equipped with a finite set S of **sorts** and a Gph-enriched functor $\theta \colon \mathrm{FinSet}^{\mathrm{op}}/S \to \mathrm{Th}$ that preserves products strictly. Any Gph-theory has an underlying multisorted Lawvere theory given by forgetting the edges of each hom graph.

A **model** of a Gph-theory Th is a Gph-enriched functor from Th to Gph that preserves products up to natural isomorphism. A **homomorphism of models** is a braided Gph-enriched natural transformation between the functors. Let FPGphCat be the 2-category of small Gph-enriched categories with finite products, product-preserving Gph-functors, and braided Gph-natural transformations. The forgetful functor $U \colon \mathrm{FPGphCat}[\mathrm{Th}, \mathrm{Gph}] \to \mathrm{Gph}$ that picks out the underlying graph of a model has a left adjoint that picks out the free model on a graph.

Gph-enriched categories are part of a spectrum of 2-category-like structures. A strict 2-category is a category enriched over Cat with its usual product. Sesquicategories are categories enriched over Cat with the "funny" tensor product [**?**]; a sesquicategory can be thought of as a 2-category where the interchange law doesn't hold. A Gph-enriched category can be thought of as a sesquicategory where 2-morphisms (now edges) can't be composed. Any strict 2-category has an underlying sesquicategory, and any sesquicategory has an underlying Gph-enriched category. These forgetful functors have left adjoints.

4 Gph-theories as models of computation

Lawvere theories and their generalizations are categories with infinitely many objects and morphisms, but most theories of interest are finitely generated. A presentation of the underlying multisorted Lawvere theory of a finitely-generated Gph-theory is a signature for a term calculus, consisting of a set of sorts, a set of term constructors, and a set of equations, while the edges in the hom graphs of the theory encode the reduction relation.

Here is a presentation of the SKI combinator calculus as a Gph-theory:

- one sort T, for terms

- term constructors

$$\begin{aligned} S &: 1 \to T \\ K &: 1 \to T \\ I &: 1 \to T \\ (-\ -) &: T^2 \to T \end{aligned}$$

- structural congruence (no equations)
- rewrites

$$\begin{aligned} \sigma &: (((S\ x)\ y)\ z) \Rightarrow ((x\ z)\ (y\ z)) \\ \kappa &: ((K\ y)\ z) \Rightarrow y \\ \iota &: (I\ z) \Rightarrow z \end{aligned}$$

where in the rewrites we have used expressions like $((K\ y)\ z)$ as shorthand for

$$T \times T \xrightarrow{\text{left}^{-1}} 1 \times T \times T \xrightarrow{K \times T \times T} T \times T \times T \xrightarrow{(-\ -) \times T} T \times T \xrightarrow{(-\ -)} T.$$

A model M of this Gph-theory in Gph picks out a graph $M(T)$ of terms and rewrites. It picks out three special vertices S, K, and I of $M(T)$; it equips $M(T)$ with a graph homomorphism from $M(T)^2$ to $M(T)$ that says for every pair of vertices (u, v), there is a vertex $(u\ v)$, and similarly for edges; and it equips $M(T)$ with graph transformations asserting the existence of an edge out of a reducible expression to the term it reduces to.

That this Gph-theory captures the operational semantics of the SKI calculus is almost definitional: there is an edge between distinct vertices in the free model on the empty graph if and only if the source vertex is reducible to the target vertex in a single step.

It is straightforward to verify that Gph-theories suffice to capture the operational semantics of any calculus where every context is a reduction context. This restriction on reduction contexts is a consequence of the fact that models map term constructors to graph homomorphisms: given a model M, a graph homomorphism $F\colon M(T) \to M(T)$, and an edge $e\colon t_1 \to t_2$, there is necessarily an edge $F(e)\colon F(t_1) \to F(t_2)$.

5 Gph-theory for SKI with the weak head normal form evaluation strategy

In modern programming languages, many contexts are *not* reduction contexts. In Haskell, for instance, there are no reductions under a lambda abstraction: even if t_1 reduces to t_2 as a program, the term $\backslash x \to t_1$ does not reduce to $\backslash x \to t_2$.

Gph-theories can still capture the operational semantics of calculi with restrictions on reduction contexts by introducing term constructors that explicitly mark the reduction contexts. For example, suppose that we want an evaluation strategy for the SKI calculus that only reduces the leftmost combinator when it has been applied to sufficiently many arguments, *i.e.* we want the *weak head normal form*; we can accomplish this by introducing a term constructor $R\colon T \to T$ that explicitly marks the reduction contexts. We then add a structural congruence rule for propagating the context and modify the existing reduction rules to apply only to marked contexts.

- one sort T, for terms

- term constructors

$$\begin{aligned} S &: 1 \to T \\ K &: 1 \to T \\ I &: 1 \to T \\ (- \; -) &: T^2 \to T \\ R &: T \to T \end{aligned}$$

- structural congruence

$$R(x\ y) = (Rx\ y)$$

- rewrites

$$\begin{aligned} \sigma &: (((RS\ x)\ y)\ z) \Rightarrow ((Rx\ z)\ (y\ z)) \\ \kappa &: ((RK\ y)\ z) \Rightarrow Ry \\ \iota &: (RI\ z) \Rightarrow Rz \end{aligned}$$

▶ **Theorem 1.** *Let t be a term in which R does not appear. Then Rt reduces to Rt', where t' is the weak head normal form of t.*

Proof. If we form the term Rt where t contains no uses of R, no reductions will ever take place in the right-hand argument of an application: the structural congruence and rewrite rules enforce that the R context can only move to the left term in an application, never the right. The result follows by induction on the number of steps to reach t'. ◀

[[Is it ever *not* possible to mark reduction contexts this way?]]

6 Explicit reduction contexts as gas

The Ethereum [**?**] and RChain [**?**] projects are building virtual machines on the blockchain. Both use the concept of a linear resource called "gas" (as in gasoline) that is consumed as the virtual machine executes. Gph-theories can capture the operational semantics of a calculus where reduction contexts are consumable, and thus play a role similar to that of gas [**?**].

- one sort T, for terms
- term constructors

$$\begin{aligned} S &: 1 \to T \\ K &: 1 \to T \\ I &: 1 \to T \\ (- \; -) &: T^2 \to T \\ R &: T \to T \end{aligned}$$

- structural congruence

$$R(x\ y) = (Rx\ y)$$

- rewrites

$$\begin{aligned} \sigma &: (((RS\ x)\ y)\ z) \Rightarrow ((x\ z)\ (y\ z)) \\ \kappa &: ((RK\ y)\ z) \Rightarrow y \\ \iota &: (RI\ z) \Rightarrow z \end{aligned}$$

▶ **Theorem 2.** *Let t be a term in which R does not appear; let t' be the weak head normal form of t; let m be the number of steps by which Rt reduces to Rt' in the calculus of section 5; and let $n \geq m$. Then in this calculus, $R^n t$ reduces to $R^{n-m}t'$ in m steps.*

Proof. As before, if we form the term Rt where t contains no uses of R, no reductions will ever take place in the right-hand argument of an application. Each application of the reduction rules reduces the number of Rs by one, and structural equivalence preserves the number of Rs. The result follows by induction on the number of steps to reach t'. ◀

7 Gph-theory for a pi calculus variant

Gph-theories can capture the operational semantics of concurrent calculi as well as serial calculi like SKI above.

Meredith and Radestock [7] describe a reflective higher-order variant of pi calculus we call the RHO calculus. Rather than the usual replication and new operators, they have quoting and unquoting operators. Quoting turns a process into a name and unquoting does the opposite; freshness of names is obtained using a type discipline. They prove that there is a faithful embedding of the monadic asynchronous pi calculus into the RHO calculus.

7.1 The RHO calculus

7.1.1 Syntax.

$$
\begin{array}{rcl}
P, Q & ::= & 0 \\
& | & \text{for}(y \leftarrow x)P \\
& | & x!P \\
& | & P \mid Q \\
& | & *x \\
\\
x, y & ::= & \&P
\end{array}
$$

7.1.2 Free and bound names.

$$
\begin{array}{rl}
FN(0) & = \emptyset \\
FN(\text{for}(y \leftarrow x)P) & = \{x\} \cup (FN(P) \backslash \{y\}) \\
FN(x!P) & = \{x\} \cup FN(P)
\end{array}
\qquad
\begin{array}{rl}
FN(P|Q) & = FN(P) \cup FN(Q) \\
FN(*x) & = \{x\}
\end{array}
$$

7.1.3 Structural congruence.

Structural (process) congruence is the smallest equivalence relation $\equiv$ containing α-equivalence and making $(|, 0)$ into a commutative monoid.

7.1.4 Name equivalence.

Name equivalence is the smallest equivalence relation $\equiv_N$ on names such that

$$\frac{}{\&*x \equiv_N x} \text{ and } \frac{P \equiv Q}{\&P \equiv_N \&Q}.$$

7.1.5 Substitution.

Syntactic substitution:

$$\begin{aligned}
(0)\{\&Q/\&P\} &= 0 \\
(\mathrm{for}(y \leftarrow x)R)\{\&Q/\&P\} &= \mathrm{for}(z \leftarrow (x\{\&Q/\&P\}))(R\{z/y\}\{\&Q/\&P\}) \\
(x!R)\{\&Q/\&P\} &= (x\{\&Q/\&P\})!(R\{\&Q/\&P\}) \\
(R|S)\{\&Q/\&P\} &= (R\{\&Q/\&P\}) \mid (S\{\&Q/\&P\}) \\
(*x)\{\&Q/\&P\} &= \begin{cases} *\&Q & \text{when } x \equiv_N \&Q \\ *x & \text{otherwise,} \end{cases}
\end{aligned}$$

where

$$x\{\&Q/\&P\} = \begin{cases} \&Q & \text{if } x \equiv_N \&P \\ x & \text{otherwise} \end{cases}$$

and, in the rule for input, z is chosen to be distinct from $\&P, \&Q$, the free names in Q, and all the names in R.

Semantic substitution, for use in α-equivalence:

$$(*x)\{\&Q/\&P\} = \begin{cases} Q & \text{when } x \equiv_N \&Q \\ *x & \text{otherwise} \end{cases}$$

7.1.6 Reduction rules.

We use $\to$ to denote single-step reduction.

$$\frac{x_0 \equiv_N x_1}{x_0!Q \mid \mathrm{for}(y \leftarrow x_1)P \quad \to \quad P\{\&Q/y\}}$$

$$\frac{P \to P'}{P \mid Q \quad \to \quad P' \mid Q}$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}$$

7.2 RHO combinators

We can define an embedding $[\![-]\!]$ of closed RHO calculus terms into a set of RHO combinators. We follow Milner [**?**] in thinking of an input-prefixed process $\mathrm{for}(x \leftarrow y)P$ as consisting of two parts: the first names the channel y on which the process is listening, while the second describes the continuation $\lambda x.P$ in terms of an abstracted name. The right hand side of the communication rule, in effect, applies the continuation to the name to be substituted. Since the only bound names in the RHO calculus come from input prefixing, we can completely eliminate bound names by using abstraction elimination on the continuation. Like the weak head normal form SKI calculus above, this combinator calculus uses a linear resource C to reify reduction contexts.

A Gph-theory for the operational semantics of these combinators has:

- one sort T, for terms
- term constructors

$$\begin{array}{rl} C & : 1 \to T \\ 0 & : 1 \to T \\ | & : 1 \to T \\ \mathrm{for} & : 1 \to T \\ ! & : 1 \to T \\ \& & : 1 \to T \end{array} \qquad \begin{array}{rl} * & : 1 \to T \\ S & : 1 \to T \\ K & : 1 \to T \\ I & : 1 \to T \\ () & : T \times T \to T \end{array}$$

- structural congruence rules

$$\begin{array}{rll} ((|\ 0)\ P) & = P & \text{unit law} \\ ((|\ ((|\ P)\ Q))\ R) & = ((|\ P)\ ((|\ Q)\ R) & \text{associativity} \\ ((|\ P)\ Q) & = ((|\ Q)\ P) & \text{commutativity} \end{array}$$

- reduction rules

$$\begin{array}{ll} \sigma\colon (((S\ P)\ Q)\ R) \Rightarrow ((P\ R)\ (Q\ R)) & \text{action of } S \\ \kappa\colon ((K\ P)\ Q) \Rightarrow P & \text{action of } K \\ \iota\colon (I\ P) \Rightarrow P & \text{action of } I \\ \xi\colon ((|\ C)\ ((|\ ((\text{for}\ (\&\ P))\ Q))\ ((!\ (\&\ P))\ R))) \Rightarrow ((|\ C)\ (Q\ (\&\ R))) & \text{communication} \\ \epsilon\colon ((|\ C)\ (*\ (\&\ P))) \Rightarrow ((|\ C)\ P) & \text{evaluation} \end{array}$$

7.3 Embedding

We define an interpretation function $[\![-]\!]$ from RHO calculus terms into RHO combinators by

$$\begin{array}{rl} [\![0]\!] & = 0 \\ [\![\text{for}(x \leftarrow \&P)Q]\!] & = ((for\ (\&\ [\![P]\!]))\ [\![Q]\!]_x) \\ [\![\&P!Q]\!] & = ((!\ (\&\ [\![P]\!]))\ [\![Q]\!]) \\ [\![P|Q]\!] & = ((|\ [\![P]\!])\ [\![Q]\!]) \\ [\![*\&P]\!] & = (*\ (\&\ [\![P]\!])) \end{array}$$

where $[\![-]\!]_x$ eliminates the free name x :

$$\begin{array}{rll} [\![P]\!]_x & = (K\ [\![P]\!]) \text{ where } x \text{ is not free in } P & \\ [\![\text{for}(y \leftarrow \&P)Q]\!]_x & = \begin{cases} ((S\ ((S\ (Kfor))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ [\![[\![Q]\!]_y]\!]_x) & \text{when } y \not\equiv_N x \\ ((S\ ((S\ (Kfor))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ [\![Q]\!]_y) & \text{when } y \equiv_N x \end{cases} & \\ [\![\&P!Q]\!]_x & = ((S\ ((S\ (K\ !))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ [\![Q]\!]_x) & \\ [\![P|Q]\!]_x & = ((S\ ((S\ (K\ |))\ [\![P]\!]_x))\ [\![Q]\!]_x) & \\ [\![*\&P]\!] & = \begin{cases} ((S\ (K\ *))\ I) & \text{when } \&P \equiv_N x \\ ((S\ (K\ *))\ ((S(K\ \&))\ [\![P]\!]_x) & \text{otherwise.} \end{cases} & \end{array}$$

7.4 Barbed bisimilarity

An **observation relation** $\downarrow_{\mathcal{N}}$ over a set of names $\mathcal{N}$ is the smallest relation satisfying

$$\frac{y \in \mathcal{N} \qquad x \equiv_N y}{x!P \downarrow_{\mathcal{N}} x} \quad \text{and} \quad \frac{P \downarrow_{\mathcal{N}} x \text{ or } Q \downarrow_{\mathcal{N}} x}{P \mid Q \downarrow_{\mathcal{N}} x}$$

for the RHO calculus or

$$\frac{y \in \mathcal{N} \qquad x \equiv_N y}{((!\ x)\ P) \downarrow_{\mathcal{N}} x} \quad \text{and} \quad \frac{P \downarrow_{\mathcal{N}} x \text{ or } Q \downarrow_{\mathcal{N}} x}{((|\ P)\ Q) \downarrow_{\mathcal{N}} x}\ .$$

for the RHO combinators.

We denote eventual reduction by $\rightarrow^*$ and write $P \downarrow^*_{\mathcal{N}} x$ if there exists a process Q such that $P \rightarrow^* Q$ and $Q \downarrow_{\mathcal{N}} x$.

An **$\mathcal{N}$-barbed bisimulation** over a set of names $\mathcal{N}$ is a symmetric binary relation $S_{\mathcal{N}}$ between agents such that $P\, S_{\mathcal{N}}\, Q$ implies

1. if $P \rightarrow P'$ then $Q \rightarrow^* Q'$ and $P\, S_{\mathcal{N}}\, Q'$, and
2. if $P \downarrow_{\mathcal{N}} x$, then $Q \downarrow^*_{\mathcal{N}} x$.

P is $\mathcal{N}$-barbed bisimilar to Q, written $P \approx Q$, if $P\, S_{\mathcal{N}}\, Q$ for some $\mathcal{N}$-barbed bisimulation $S_{\mathcal{N}}$.

7.5 Faithfulness

▶ **Theorem 3.** $P \sim_{calc} Q \iff ((|\ C)\ [\![P]\!]) \sim_{comb} ((|\ C)\ [\![Q]\!])$.

. [[...]] ◀

8 Conclusion and future work

9 Appendix: abstraction elimination calculations

$$
\begin{aligned}
& ((K\ [\![P]\!])\ x) \\
= \quad & [\![P]\!]
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ ((S\ (K\ for))\ ((S\ (K\ \&))\ \ [\![P]\!]_x)))\ [\![[\![Q]\!]_y]\!]_x)\ x) \\
= \quad & ((((S\ (K\ for))\ ((S\ (K\ \&))\ \ [\![P]\!]_x))\ x)\ ([\![[\![Q]\!]_y]\!]_x\ x)) \\
= \quad & ((((K\ for)\ x)\ (((S\ (K\ \&))\ \ [\![P]\!]_x)\ x))\ [\![Q]\!]_y) \\
= \quad & ((for\ (((K\ \&)\ x)\ ([\![P]\!]_x\ x)))\ [\![Q]\!]_y) \\
= \quad & ((for\ (\&\ [\![P]\!]))\ [\![Q]\!]_y)
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ ((S\ (K\ for))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ [\![Q]\!]_y)\ x) \\
= \quad & ((((S\ (K\ for))\ ((S\ (K\ \&))\ [\![P]\!]_x))\ x)\ ([\![Q]\!]_y\ x)) \\
= \quad & ((((K\ for)\ x)\ (((S\ (K\ \&))\ [\![P]\!]_x)\ x))\ [\![Q]\!]\{x/y\}) \\
= \quad & ((for\ (((K\ \&)\ x)\ ([\![P]\!]_x\ x)))\ [\![Q]\!]\{x/y\}) \\
= \quad & ((for\ (\&\ [\![P]\!]))\ [\![Q]\!]\{x/y\})
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ ((S\ (K\ !))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ [\![Q]\!]_x)\ x) \\
= \quad & (((((S\ (K\ !))\ ((S\ (K\ \&))\ [\![P]\!]_x)))\ x)\ ([\![Q]\!]_x\ x)) \\
= \quad & ((((K\ !)\ x)\ (((S\ (K\ \&))\ [\![P]\!]_x)\ x))\ [\![Q]\!]) \\
= \quad & ((!\ (((K\ \&)\ x)\ ([\![P]\!]_x\ x)))\ [\![Q]\!]) \\
= \quad & ((!\ (\&\ [\![P]\!]))\ [\![Q]\!])
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ ((S\ (K\ |))\ [\![P]\!]_x))\ [\![Q]\!]_x)\ x) \\
= \quad & ((((S\ (K\ |))\ [\![P]\!]_x)\ x)\ ([\![Q]\!]_x\ x)) \\
= \quad & ((((K\ |)\ x)\ ([\![P]\!]_x\ x))\ [\![Q]\!]) \\
= \quad & ((|\ [\![P]\!])\ [\![Q]\!])
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ (K\ *))\ I)\ x) \\
= \quad & (((K\ *)\ x)\ (I\ x)) \\
= \quad & (*\ x)
\end{aligned}
$$

$$
\begin{aligned}
& (((S\ (K\ *))\ ((S\ (K\ \&))\ [\![P]\!]_x))\ x) \\
= \quad & (((K\ *)\ x)\ (((S\ (K\ \&))\ [\![P]\!]_x)\ x)) \\
= \quad & (*\ (((K\ \&)\ x)\ ([\![P]\!]_x\ x))) \\
= \quad & (*\ (\&\ [\![P]\!]))
\end{aligned}
$$

References

1 Ranald Clouston, *Nominal lawvere theories: A category theoretic account of equational theories with names*, J. Comput. Syst. Sci. **80** (2014), no. 6, 1067–1086.

2 Murdoch Gabbay and Andrew M. Pitts, *A new approach to abstract syntax with variable binding*, Formal Asp. Comput. **13** (2002), no. 3-5, 341–363.

3 John W. Gray, *Review: G. m. kelly, basic concepts of enriched category theory*, Bulletin of the American Mathematical Society **9** (1983), no. 1, 102–107.

4 Stephen Lack and Jirí Rosický, *Notions of lawvere theory*, Applied Categorical Structures **19** (2011), no. 1, 363–391.

5 Christoph Lüth and Neil Ghani, *Monads and modular term rewriting*, Category Theory and Computer Science, 7th International Conference, CTCS '97, Santa Margherita Ligure, Italy, September 4-6, 1997, Proceedings (Eugenio Moggi and Giuseppe Rosolini, eds.), Lecture Notes in Computer Science, vol. 1290, Springer, 1997, pp. 69–86.

6 Microsoft Corporation, *Microsoft biztalk server*, microsoft.com/biztalk/default.asp.

7 L. Gregory Meredith and Matthias Radestock, *A reflective higher-order calculus.*, Electr. Notes Theor. Comput. Sci. **141** (2005), no. 5, 49–67.

8 Robin Milner, *The polyadic π-calculus: A tutorial*, Logic and Algebra of Specification **Springer-Verlag** (1993).

9 John Power, *ENRICHED LAWVERE THEORIES Dedicated to Jim Lambek.*

10 David Sangiorgi and David Walker, *The π-calculus: A theory of mobile processes*, Cambridge University Press, 2001.

11 Moses Schönfinkel, *On the Building Blocks of Mathematical Logic*, From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931 (Jean van Heijenoort, ed.), iuniverse.com, 1924, pp. 355–366.

12 T Trimble, *Multisorted lawvere theories.*

13 Lucian Wischik, *Old names for nu*, Submitted for publication, 2004.

14 Nobuko Yoshida, *Minimality and separation results on asynchronous mobile processes - representability theorems by concurrent combinators*, Theor. Comput. Sci. **274** (2002), no. 1-2, 231–276.