# Logic, resource, and reflection

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

[1] Pyrofex Corp.
stay@pyrofex.net
[2] RChain Cooperative
lgreg.meredith@rchain.coop

**Abstract.** We present an algorithm for deriving a spatial-behavioral type system and term assignment from a formal presentation of a interactive computational calculus. This turns out to identify a species of category which we offer an axiomatic characterization of interaction categories.

## 1 Introduction and motivation

In groundbreaking work, Abramsky, Gay, and Nagarajan put forward the idea of interaction categories to give a categorical framework for interactive models of computing. Indeed, interactive models of computation are relatively new and offer distinct insights because they include the computational environment as part of the computation. Examples include the $\pi$-calculus, in fact, all the mobile process calculi, as well as the lambda calculus. However, interaction categories fail to say axiomatically what interaction is.

We put forward a simple, intuitive axiomatic characterization of interaction and use this to derive not only a logic but a proof theory and term assignment algorithm for all systems satisfying these axioms. The resulting structure identifies a species of category and we offer this as an axiomatic characterization of interaction categories.

### 1.1 Intuitions

The key idea in this construction is to use both the evaluation context of Meredith and Stay, and computational reflection, similar to what is found in Meredith and Radestock, to build a proof theory with a cut elimination that corresponds exactly with the notion of computation embodied in any interactive rewrite system.

The requirement that the rewrite system is interactive means the left hand side of every rewrite rule will necessarily be a term constructor, say $\mathsf{K}$, taking at least two terms. That is, when viewed as a piece of syntax it is at least a tuple of terms. Then, by controlling evaluation with an evaluation context we can force a distinction between a tensor and a cut, where the cut forms the redex with an evaluation context supplied, and the tensor is denied the evaluation context,

effectively rendering it a data structure. Of course, the data is inaccessible until there is a means to unpack it, and a par term is constructed as the principal means to extract the data from the tensor.

The use of the term constructor that forms a redex as a data structure is the essential use of reflection. The lack of a reduction context gives us the ability to suspend computation. Having suspended it, we reify computation as data, and use par to unpack the data making up the computation. Completing the circle, the tensor-par cut rule constitutes the means to reflect computational data back into actual computations.

It turns out that denying the evaluation context is not the only way to suspend computation. We can also form contexts. In this case, we only admit 1-holed contexts, ranged over by $\chi$. With contexts we can define a notion of rely-guarantee, two notions actually (indicated by the operations $(- \rhd -)$, and $(- \lhd -)$), as a redex term constructor might not be commutative. .

$$\tau \rhd_{\mathsf{K}} \tau' \triangleq \{u \quad | \quad \exists t.u = !\mathsf{K}[]r(t,[]), \forall u' : \tau.(\exists \rho : u@u' \to v) \Rightarrow v : \tau'\}$$
$$\tau' \lhd_{\mathsf{K}} \tau \triangleq \{t \quad | \quad \exists u.t = \mathsf{K}[]l([],u)!, \forall t' : \tau.(\exists \rho : t@t' \to v) \Rightarrow v : \tau'\}$$

In the sequel we drop the subscript on the triangles if it's understood from context. We use the notation $\chi@t$ to mean the term formed by substituting $t$ for hole in $\chi$. Thus, $\mathsf{K}(t,[])@u = \mathsf{K}(t,[])[u/[]] = \mathsf{K}(t,u)$ and similarly, $\mathsf{K}([],u)@t = \mathsf{K}([],u)[t/[]] = \mathsf{K}(t,u)$

To give some examples, the comm rule of rho-calculus is given by

$$\mathsf{for}(y \leftarrow x)P \quad | \quad x!(Q) \to P\{@Q/y\}$$

In this case K is parallel composition, and the revised, resource constrained comm rule, looks like

$$R|\mathsf{for}(y \leftarrow x)P \quad | \quad x!(Q) \to P\{@Q/y\}$$

In the lambda calculus, $\beta$-reduction is given by

$$(\lambda x.M)N \to M\{N/x\}$$

$\mathsf{K}$ is application, and the revised, resource constrained $\beta$-reduction is given by

$$R((\lambda x.M)N) \to M\{N/x\}$$

In what follows we will focus on calculi that don't employ binding operators and so-called nominal phenomena. This is not to say that we can't handle nominal phenomena, just that the content is already complex enough and we want to focus on the core ideas.

### 1.2 Related work

In many ways this work is inspired by and derives from domain theory in logical form (DTLF). In much the same way that DTLF takes as input a notion of computation encoded in a domain theory and produces a logical system encoding the same notion of computation, but from the logical view, this is an algorithm that takes as input a notion of computation encoded as a Lawvere theory and produces a new Lawvere theory that is resource constrained and enjoys a type system that guarantees a range of security and liveness properties, while at the same time having the tightest possible conformance to the original notion of computation. In particular, the resulting type theory is also a logic enjoying a notion of cut-elimination and proof normalization that matches exactly with the operational semantics of the original theory. As such, it automatically enjoys a Curry-Howard correspondence.

Other inspiration and guidance come from the two major branches in the development of behavioral types. One branch derives from the spatial logics of Caires, et al, while the other derives from the session types of Wadler, et al. The former is predominantly model theory based, while the latter is predominantly proof theory based. In the spatial logics branch the negation is largely set-based, while in the session types work the negation is related to the reduction of terms. This construction brings both branches together and offers both model theory and proof theory that enjoys a full abstraction-like property.

The discussion would not be complete without mentioning blockchain. While the idea of resource limiting computation certainly has a long and illustrious history, inspired by catalysts in chemical and biochemical reactions, it has immediate practical application in blockchain platforms such as RChain and Ethereum. Here, the programming model is throttled by the existence of a resource. The cost of obtaining that resource constitutes a prophylactic against denial-of-service attacks. Specifically, an attacker attempting to deploy computations that run forever or take up infinite storage must pay for the corresponding infinite resources necessary to enable the infinite computations.

### 1.3 Notation

We use

- $\Gamma, \Delta, \Xi$ to range over type contexts;
- $\triangleright, \triangleleft$ for left and right residuation;
- $(-)_\perp$ and ${}_\perp(-)$ for left and right negations;
- $!(-)$ and $(-)!$ for left and right abstractions;
- $\rho$ to range over rewrite rules;
- $\chi$ to range over contexts;
- $\parr, \otimes, \sqcap, \sqcup$ for logical connectives derived from $K$; $\sqcap$ serial, $\otimes$ parallel
- both double and single sided $\vdash$ for typing judgments;
- $\Rightarrow$ for meta level logical implication.

## 2 Construction

To illustrate the general construction we apply it to the simplest possible interactive rewrite theory, consisting of a single sort, $T$, for term, a single constant, $C$, and a single term constructor $K$, along with left and right identities for $K$ (which are, of course, constants). It has a single rewrite rule, $\rho$.

We add a resource, $R$, which can be thought of as either an evaluation context, or as a constant. In the resourced theory, the rewrite rule only applies in an evaluation context.

Then we add a notion of 1-holed context. We add a new sort, $\chi_T$, then a constant for the hole, together with left and right maps for placing the hole in either side of $K$, along with left and right maps $!(-)$ and $(-)!$ essentially for treating contexts as abstractions. We add a map $@(-,-)$ for substituting a term into a context marked as an abstraction.

It is important to note that we do not allow the rewrite rule to operate on contexts. Thus, there are two ways to suspend computation: withholding a resource, or forming an abstraction. Input to the algorithm It is to be understood that the initial theory is provided as input to the system. As mentioned earlier, this is an algorithm that takes as input a notion of computation encoded as a Lawvere theory and produces a new Lawvere theory that is resource constrained and enjoys a type system that guarantees a range of security and liveness properties.

We encode term calculi as Lawvere theories by starting with the theory of reflexive directed multigraphs:

- one sort $T$
- function symbols $s, t : T \to T$
- equations $ss = ts = s$, $st = tt = t$

To this theory, we add function symbols for each term constructor and reduction rule. We also add equations for the source and target of each reduction rule. For example, we can model the SKI combinator calculus by adding

- $S, K, I : 1 \to T$
- $(--) : T^2 \to T$
- $\sigma : T^3 \to T$
- $\kappa : T^2 \to T$
- $\iota : T \to T$
- $s\sigma(x, y, z) = (((Sx)y)z$
- $t\sigma(x, y, z) = ((xz)(yz))$
- $s\kappa(x, y) = ((Kx)y)$
- $t\kappa(x, y) = x$
- $s\iota(x) = (Ix)$
- $t\iota(x) = x$

We'll use the following symbols to talk about a generic term calculus with at least one binary term constructor and one rewrite rule that uses the term constructor as its source.

- $C, l_K, r_K : 1 -> T$
- $K : T^2 \to T$
- $\rho : T^2 \to T$
- $K(l_K, u) = u$
- $K(t, r_K) = t$
- $s(\rho(t, u)) = K(t, u)$
- $t(\rho(t, u)) = f(t, u)$

Note that K may not be commutative, so we ask for either a left identity, lK, or a right identity, rK, or if we have both they must coincide.

We proceed in stages, producing first a resourced version of the theory that requires an evaluation context to run the rewrite rule, then producing a contextualized version of the theory.

## 3 Main theorems

In this section, we prove a cut-elimination theorem.

### 3.1 Cut-elimination

Every proof that uses a cut can be transformed into a proof that does not use a cut. Compilation from contextualized resource theory back to original theory. In *lambda*-calculus and $\pi$-calculus, $\mathsf{K}$ is not directly interpreted as application or parallel, respectively. Instead, it turns into a data type so we can extract the info using a par term introduced in the contextualized resource theory. In $\rho$, there's no need because it's reflective. But either way, the term calculi are powerful enough to support the embedding of the new terms back into the original.

It remains to show that the embedding is full and faithful.

## 4 Conclusion and future work

The aim of this construction is to generate a logic and its proof theory from a notion of computation expressed as a graph-enriched Lawvere theory. The logic respects both a Curry-Howard correspondence, as well as realizability. Additionally, it enjoys a cut-elimination property, and proof normalization corresponds precisely to the operational semantics of the notion of computation provided in the graph-enriched Lawvere theory.

We have not addressed nominal aspects of rewrite systems such as the $\lambda$-calculus or the $\pi$-calculus. We note that there are two very promising directions. One is captured in Clouston's work on nominal Lawvere theories. The other is expressed in Fiore's work on HOAS. It also bears mentioning that we can generate the names used by such a nominal theory using reflection, ala Meredith and Radestock.

Also, in the spirit of Meredith and Radestock, we can seek to minimize risk taken on in the theory in the form of "outside influences."

## References

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