Theory and Formal Methods 1994: Proceedings of the Second Imperial College Department of Computing Workshop on Theory and Formal Methods, Imperial College Press, 1995.

AN INTERNAL LANGUAGE FOR INTERACTION CATEGORIES

ROY CROLE
Department of Mathematics and Computer Science,
University of Leicester
E-mail: rlc3@mcs.le.ac.uk

SIMON GAY RAJAGOPAL NAGARAJAN
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom
E-mail: {sjg3,rn4}@doc.ic.ac.uk

ABSTRACT

We use the techniques of categorical logic to obtain a formal connection between interaction categories ($*$-autonomous categories with certain additional structure) and typed process theories (simplified versions of certain typed process calculi). The connection is similar in nature to that between cartesian closed categories and the simply typed λ-calculus. Interaction categories are the natural models of process theories. Furthermore, from any process theory we can construct an interaction category which is universal among models of the theory, and from any interaction category we can construct a process theory; these constructions are mutually inverse.

1 Introduction

It has been known for some years that formal systems of various kinds correspond to certain flavours of categorical structure. The first person to observe this phenomenon seems to have been Lawvere, who formulated a connection between certain kinds of algebraic theories and categories with finite products. Since Lawvere's original insight, there has been much progress in understanding

the connections between category theory and formal systems. Expositions can be found in [6, 11, 12, 13, 14]; there are many other references—these few will be of particular interest to Computer Scientists.

The precise way in which such correspondences can be realised has been perfected during the last decade or so. In most cases the following rather general procedures can be formulated with great precision. Given a formal system (logic, type theory, *etc.*) a category-theoretic structure can be manufactured. Conversely, given a specified category-theoretic structure, this gives information from which to build a formal system. These two processes are mutually inverse. If one builds a formal system from the category, and then a category from the resulting formal system, this latter category will be equivalent to the original. A similar game can be played by starting with the formal system; in this case one obtains an equivalence of formal systems which will be given by mutually inverse theory translations.

In this paper we describe a particular instance of a correspondence between a variety of category (an interaction category [1, 3]) and a formal system for equational reasoning in which the individuals of the intensional object level equality judgements are *processes*. The formal system contains a number of constructs for building larger processes from elementary processes, using operations (such as cut, tensor and par) mainly derived from linear logic [9]. There are two fundamental object level judgements. First, processes-in-context specify a process together with a list of its communicating ports. Second, there are judgements of equality between processes. Apart from the fact that individuals are to be thought of as processes, the underlying principles of the work described here are identical to those of the correspondence between categories with finite products and algebraic theories.

2 An Equational Theory of Typed Processes

2.1 A Syntax for Typed Processes

In this section we introduce some syntax which will be used to write down an equational theory of typed processes. The syntax is a restricted version of the typed process calculus studied in [7, 8]. A general exposition of the techniques of categorical type theory can be found in [6]. We begin by defining a notion of signature for such a theory. A *signature* Sg (for a process theory) is specified by the following data:

- A collection of *ground types*. From this, the collection of *types* is specified by the grammar

$$\alpha ::= I \mid \bot \mid \gamma \mid \alpha \otimes \alpha \mid \alpha \parr \alpha \mid \alpha^{\bot} \mid \bigcirc\alpha$$

  in which γ is any ground type. We write *Type* for the collection of all types.

- A collection of *process symbols*, each of which has a *sorting* consisting of a finite list of types. This is written $P : \alpha_1, \ldots, \alpha_n$.

We now wish to define a collection of so-called raw processes, which will play an analogous role to the raw terms of a type theory. These raw processes will involve a number of variable binding operations, and to simplify the definitions of substitution and α-equivalence, we will employ the theory of arities and expressions as a metalanguage. Thus our raw processes will in fact arise as α-equivalence classes of expressions in a certain simply typed lambda calculus for which there is a standard definition of substitution. For us, such a theory of expressions will have two *ground-arities*, namely PROC and NAME. The *arities* are the simple types over the ground-arities, that is

$$a ::= \textsc{name} \mid \textsc{proc} \mid a \Rightarrow a.$$

The collection of constants for the metalanguage will consist of symbols which should be thought of as constructors for raw processes. Given a signature Sg, the *constants* and their arities are given by

- If P is a process symbol of arity n, there is a constant of arity $\textsc{name}^n \Rightarrow \textsc{proc}$,
- I of arity $\textsc{name}^2 \Rightarrow \textsc{proc}$,
- $\cdot$ of arity $(\textsc{name} \Rightarrow \textsc{proc}) \Rightarrow (\textsc{name} \Rightarrow \textsc{proc}) \Rightarrow \textsc{proc}$,
- $\otimes$ of arity $(\textsc{name} \Rightarrow \textsc{proc}) \Rightarrow (\textsc{name} \Rightarrow \textsc{proc}) \Rightarrow \textsc{name} \Rightarrow \textsc{proc}$,
- $\parr$ of arity $(\textsc{name} \Rightarrow \textsc{name} \Rightarrow \textsc{proc}) \Rightarrow \textsc{name} \Rightarrow \textsc{proc}$,
- unit of arity $\textsc{name} \Rightarrow \textsc{proc}$,
- bottom of arity $\textsc{name} \Rightarrow \textsc{proc} \Rightarrow \textsc{proc}$,
- $\bigcirc$ of arity $\textsc{proc} \Rightarrow \textsc{proc}$,
- Rec of arity $(\textsc{name} \Rightarrow \textsc{name} \Rightarrow \textsc{proc}) \Rightarrow (\textsc{name} \Rightarrow \textsc{name} \Rightarrow \textsc{proc}) \Rightarrow \textsc{name} \Rightarrow \textsc{name} \Rightarrow \textsc{proc}$.

This data can be used as a signature for a simply typed lambda calculus which will constitute the metalanguage of arities and expressions. We take a set Var^a of variables for each arity a, and then the raw *expressions* are given by

$$e ::= x^a \mid c \mid ee \mid x^a.e$$

with $x^a \in Var^a$ for any arity a, and c is any constant. Well formed expressions of arity a will be written $e : a$; they are formed by the usual rules of the simply typed lambda calculus [10]. There are of course sets $Exp^a \stackrel{\text{def}}{=} \{e \mid e : a\}$ of well-formed expressions of arity a. Multiple applications will be written $e_1(e_2, \ldots, e_m)$. There is the usual notion of α-equivalence $\equiv_\alpha \subset Exp^a \times Exp^a$ on expressions of arity a, free and bound variables, and the usual definition of substitution of an expression for free occurrences of a variable in another expression. We write $fv(e)$ for the free variables of the expression e.

The set *RProc* of *raw processes* is defined to be the set of expressions of arity PROC up to α-equivalence, that is

$$RProc \stackrel{\text{def}}{=} Exp^{\text{PROC}} / \equiv_\alpha .$$

We will not distinguish notationally between an expression $P \in Exp^{\text{PROC}}$ and the α-equivalence in *RProc* which it represents. Note that substitution in the metalanguage gives us a function

$$RProc \times Var^{\text{NAME}} \times Var^{\text{NAME}} \to RProc$$

written $(P, x, y) \mapsto P[x/y]$. We shall refer to variables in the set Var^{NAME} as *names*. We use syntactic sugar to write down raw processes. For example, take $P, Q \in RProc$ and $x, y, z \in Var^{\text{NAME}}$. Then we have $\otimes(x.P, y.Q, z) \in RProc$, and this raw process will be written $P \otimes_z^{x,y} Q$.

A *process-in-context* consists of a raw process P and a *context* Γ, usually written $P \vdash \Gamma$, where Γ is a finite set of pairs $(x, \alpha) \in Var^{\text{NAME}} \times Types$ which will be written $x_1 : \alpha_1, \ldots, x_n : \alpha_n$. A *proved process* is a "well-formed" process-in-context; the collection of proved processes is generated by the rules in Figure 1. The proved process $P \vdash x_1 : \alpha_1, \ldots, x_n : \alpha_n$ should be thought of as a statement that the process P has an interface consisting of n ports of types $\alpha_1, \ldots, \alpha_n$, which are labelled for reference by the names $x_1, \ldots, x_n$.

Lemma 1 *The rules for generating proved processes are well-defined. If* $P \vdash x_1 : \alpha_1, \ldots, x_n : \alpha_n$ *is a proved process, then the names* x_i *are distinct and* $fv(P) \subseteq \{x_1, \ldots, x_n\}$.

Proof: To see that the rules are well defined one needs to observe that if different choices of α-representatives for raw processes in the hypotheses are

chosen, the same raw process in the conclusion results. But this is immediate from the congruence rules for α-equivalence.

The second part of the lemma is by rule induction. $\Box$

Lemma 2 *If $P \vdash \Gamma, x : A$ is a proved process and y does not occur in P then $P[y/x] \vdash \Gamma, y : A$ is also a proved process.*

Proof: By induction on the derivation of $P \vdash \Gamma, x : A$. $\Box$

2.2 Equational Theories

Our aim is to set up an equational theory of typed processes. To this effect, we shall define an *equation-in-context* to be a judgement of the form $P = Q \vdash \Gamma$ in which necessarily $P \vdash \Gamma$ and $Q \vdash \Gamma$ are proved processes. A *process theory* $\mathcal{T}h \stackrel{\text{def}}{=} (Sg, \mathcal{A}x)$ consists of a signature Sg together with a collection $\mathcal{A}x$ of equations-in-context. The *theorems* of the theory $\mathcal{T}h$ are equations-in-context which are generated by the rules in Figures 2, 3 and 4.

Lemma 3 *The rules for generating theorems are well-defined, that is, the conclusions are indeed equations-in-context.*

Proof: We need to show that if $P = Q \vdash \Gamma$ is a theorem, then both $P \vdash \Gamma$ and $Q \vdash \Gamma$ are proved processes. This is by induction on the proof of $P = Q \vdash \Gamma$. $\Box$

The rules for proving equality fall into several classes. First are the expected rules of equational reasoning and substitution. The structural equations express associativity and commutativity of $\otimes$, $\parr$ and Cut, and also associativities between these operators. These equations arise from the fact that a process configuration is a two- or three-dimensional structure which is being written down in a one-dimensional syntax. The cut elimination equations corresponds to removing a cut between ports of various types. It is important to note that proof net reduction steps (cut elimination steps), which correspond to β-reductions in the λ-calculus, become part of equality in our theory. The identity equations express functoriality of the type constructors. In addition to the rules in Figures 2, 3 and 4 there is a *congruence* rule, such as the following rule for cut, for each metalanguage constant.

Congruence for Cut

$$\frac{P = P' \vdash \Gamma, x : \alpha \qquad Q = Q' \vdash \Delta, x : \alpha^{\perp}}{P \;_{x}\; P' = Q \;_{x}\; Q' \vdash \Gamma, \Delta}$$

Process Symbols

$$\frac{P : \alpha_1, \ldots, \alpha_n}{P(x_1, \ldots, x_n) \vdash x_1 : \alpha_1, \ldots, x_n : \alpha_n} \quad \text{Process Symbol}$$

Structural Rules

$$\frac{}{I_{x,y} \vdash x : \alpha^{\perp}, y : \alpha} \quad \text{Axiom} \qquad \frac{P \vdash \Gamma, x : \alpha \quad Q \vdash x : \alpha^{\perp}, \Delta}{P \,\dot{x}\, Q \vdash \Gamma, \Delta} \quad \text{Cut}$$

Multiplicative Rules

$$\frac{P \vdash \Gamma, x : \alpha \quad Q \vdash y : \beta, \Delta}{P \otimes_z^{x,y} Q \vdash \Gamma, z : \alpha \otimes \beta, \Delta} \quad \text{Tensor} \qquad \frac{P \vdash \Gamma, x : \alpha, y : \beta}{\parr_z^{x,y}(P) \vdash \Gamma, z : \alpha \parr \beta} \quad \text{Par}$$

Unit Rules

$$\frac{}{\mathsf{unit}_x \vdash x : I} \quad \text{Unit} \qquad \frac{P \vdash \Gamma}{\mathsf{bottom}_x(P) \vdash \Gamma, x : \perp} \quad \text{Bottom}$$

Delay Rule

$$\frac{P \vdash \Gamma}{\bigcirc P \vdash \bigcirc \Gamma} \quad \text{Delay}$$

where $\bigcirc(x_1 : \alpha_1, \ldots, x_n : \alpha_n) = x_1 : \bigcirc\alpha_1, \ldots, x_n : \bigcirc\alpha_n$

Recursion Rule

$$\frac{P \vdash x : \alpha^{\perp}, y : \bigcirc\alpha \qquad Q \vdash u : \bigcirc\beta^{\perp}, v : \beta}{\mathsf{Rec}_{z,w}^{x,y,u,v}(P, Q) \vdash z : \alpha^{\perp}, w : \beta} \quad \text{Recursion}$$

Figure 1: Proved Processes Generated by a Process Signature.

Equational Reasoning

$$\frac{P \vdash \Gamma}{P = P \vdash \Gamma} \qquad \frac{P = Q \vdash \Gamma}{Q = P \vdash \Gamma} \qquad \frac{P = Q \vdash \Gamma \qquad Q = R \vdash \Gamma}{P = R \vdash \Gamma}$$

Substitution

$$\frac{P = P' \vdash \Gamma, x : \alpha}{P[u/x] = P'[u/x] \vdash \Gamma, u : \alpha}$$

Commutativity of Cut

$$\frac{P \vdash \Gamma, x : \alpha \qquad Q \vdash x : \alpha^{\perp}, \Delta}{P \mathbin{\dot{x}} Q = Q \mathbin{\dot{x}} P \vdash \Gamma, \Delta}$$

Associativity of Tensor

$$\frac{P \vdash \Gamma, x : \alpha \qquad Q \vdash y : \beta, \Delta, u : \gamma \qquad R \vdash v : \delta, \Theta}{(P \otimes_z^{x,y} Q) \otimes_w^{u,v} R = P \otimes_z^{x,y} (Q \otimes_w^{u,v} R) \vdash \Gamma, z : \alpha \otimes \beta, \Delta, w : \gamma \otimes \delta, \Theta}$$

Associativity of Cut

$$\frac{P \vdash \Gamma, x : \alpha \qquad Q \vdash x : \alpha^{\perp}, \Delta, y : \beta \qquad R \vdash y : \beta^{\perp}, \Theta}{(P \mathbin{\dot{x}} Q) \mathbin{\dot{y}} R = P \mathbin{\dot{x}} (Q \mathbin{\dot{y}} R) \vdash \Gamma, \Delta, \Theta}$$

Associativity of Par

$$\frac{P \vdash \Gamma, x : \alpha, y : \beta, u : \gamma, v : \delta}{\parr_z^{x,y}(\parr_w^{u,v}(P)) = \parr_w^{u,v}(\parr_z^{x,y}(P)) \vdash \Gamma, z : \alpha \parr \beta, w : \gamma \parr \delta}$$

Structural Tensor-Cut

$$\frac{P \vdash \Gamma, x : \alpha \quad Q \vdash y : \beta, \Delta, u : \gamma \quad R \vdash u : \gamma^{\perp}, \Theta}{(P \otimes_z^{x,y} Q) \mathbin{\dot{u}} R = P \otimes_z^{x,y} (Q \mathbin{\dot{u}} R) \vdash \Gamma, z : \alpha \otimes \beta, \Delta, \Theta}$$

Figure 2: Theorems Generated by a Process Theory

Structural Par-Tensor

$$\frac{P \vdash \Gamma, x : \alpha, y : \beta, u : \gamma \qquad Q \vdash \Delta, v : \delta}{\parr_z^{x,y}(P) \otimes_w^{u,v} Q = \parr_z^{x,y}(P \otimes_w^{u,v} Q) \vdash \Gamma, \Delta, z : \alpha \parr \beta, w : \gamma \otimes \delta}$$

Structural Par-Cut

$$\frac{P \vdash \Gamma, x : \alpha, y : \beta, u : \gamma \qquad Q \vdash \Delta, u : \gamma^\perp}{\parr_z^{x,y}(P) \mathbin{\dot{u}} Q = \parr_z^{x,y}(P \mathbin{\dot{u}} Q) \vdash \Gamma, \Delta, z : \alpha \parr \beta}$$

Cut Elimination Equations

$$\frac{P \vdash \Gamma, x : \alpha \quad Q \vdash y : \beta, \Delta \quad R \vdash \Theta, x : \alpha^\perp, y : \beta^\perp}{(P \otimes_z^{x,y} Q) \mathbin{\dot{z}} \parr_z^{x,y}(R) = P \mathbin{\dot{x}} (R \mathbin{\dot{y}} Q) \vdash \Gamma, \Delta, \Theta}$$

$$\frac{P \vdash \Gamma, x : \alpha}{P \mathbin{\dot{x}} I_{x,y} = P[y/x] \vdash \Gamma, y : \alpha}$$

$$\frac{Q \vdash \Gamma}{\mathsf{bottom}_x(Q) \mathbin{\dot{x}} \mathsf{unit}_x = Q \vdash \Gamma}$$

$$\frac{Q \vdash \Gamma, x : A^\perp \qquad R \vdash y : A, \Delta}{(\circ Q) \mathbin{\dot{x}} (\circ R) = \circ(Q \mathbin{\dot{x}} R) \vdash \circ\Gamma, \circ\Delta}$$

Tensor-Par Identity

$$\frac{}{\parr_z^{x,y}(I_{x,u} \otimes_w^{u,v} I_{y,v}) = I_{z,w} \vdash z : \alpha^\perp \parr \beta^\perp, w : \alpha \otimes \beta}$$

Unit-Bottom Identity

$$\frac{}{\mathsf{bottom}_x(\mathsf{unit}_y) = I_{x,y} \vdash x : \perp, y : I}$$

Unit Delay Identity

$$\frac{}{\circ I_{x,y} = I_{x,y} \vdash x : A^\perp, y : A}$$

Figure 3: Theorems Generated by a Process Theory, continued

Recursion Equations

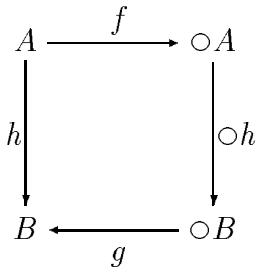
$$\frac{Q \vdash x : A^{\perp}, y : \bigcirc A \qquad R \vdash u : \bigcirc B^{\perp}, v : B}{Q[z/y] \mathbin{;}_{z} (\bigcirc \mathsf{Rec}_{z,w}^{x,y,u,v}(Q,R)) \mathbin{;}_{w} R[w/u] = \mathsf{Rec}_{x,v}^{p,y,u,q}(Q[p/x], R[q/v]) \vdash x : A^{\perp}, v : B}$$

$$\frac{Q[z/y] \mathbin{;}_{z} (\bigcirc S) \mathbin{;}_{w} R[w/u] = S[x/z][v/w]}{S = \mathsf{Rec}_{z,w}^{x,y,u,v}(Q,R) \vdash z : A^{\perp}, w : B}$$

Figure 4: Theorems Generated by a Process Theory, continued

3 Categorical Semantics

We now define the semantics of a process theory in a suitably structured category $\mathbb{C}$. One example of a suitable category is $\mathcal{SP}roc$ [1, 3], Abramsky's category of synchronous processes; another is $\mathcal{ASP}roc$ [2, 4], his category of asynchronous processes. Abstractly, $\mathbb{C}$ should be a $*$-autonomous category [5] with a monoidal endofunctor $\bigcirc$ which is self-dual and has the Unique Fixed Point Property (UFPP). The UFPP says that for any objects A and B of $\mathbb{C}$, and morphisms $f : A \to \bigcirc A$, $g : \bigcirc B \to B$, there is a unique morphism $h : A \to B$ such that this diagram commutes.



For the rest of this paper, we will call such a category an *interaction category*. We have not specified all of the structure of interaction categories such as $\mathcal{SP}roc$, but in the present paper it is sufficient to consider just the $\bigcirc$ functor and the UFPP.

We also define two operations, $\hat{}$ and $\bar{}$, on morphisms such that if $f : A \to B$ and $g : I \to A \otimes B$, then $\bar{f} : I \to A^{\perp} \otimes B$ and $\hat{g} : A^{\perp} \to B$ satisfying $\hat{\bar{f}} = f$ and $\bar{\hat{g}} = g$. These can be defined using canonical isomorphisms arising from the $*$-autonomous structure. For example, $\bar{f} \stackrel{\text{def}}{=} \Lambda(\mathsf{unitl} \mathbin{;} f)$. These operations will be useful for moving types back and forth across arrows.

Let $F : \mathbb{C} \to \mathbb{D}$ be a functor between two interaction categories. We call F an *interaction functor* if it has the following properties:

- F is a *strict* symmetric monoidal functor (which means being a symmetric monoidal functor with the following isomorphism):

$$F(A \otimes B) \;\cong\; FA \otimes FB$$

- F should have

$$F(A^{\perp}) \;\cong\; (FA)^{\perp},$$

which, along with the previous isomorphism, implies

$$F(A \parr B) \;\cong\; FA \parr FB.$$

- F should preserve the closed structure, i.e. if f is the morphism defined by

$$F(A^{\perp} \parr B) \otimes FA \xrightarrow{\sim} F((A^{\perp} \parr B) \otimes A) \xrightarrow{F(\mathsf{Ap}_{A,B})} FB$$

then $\Lambda(f) : F(A^{\perp} \parr B) \to (FA)^{\perp} \parr FB$ should be an isomorphism, where $\mathsf{Ap}_{A,B} : (A^{\perp} \parr B) \otimes A \to B$ in $\mathbb{C}$.

- F preserves $\bigcirc$:

$$F(\bigcirc(A)) \;\cong\; \bigcirc(FA).$$

- F should preserve the UFPP, i.e. if h is the morphism defined by applying the UFPP to f and g in $\mathbb{C}$ then Fh is the morphism determined by applying the UFPP to Ff and Fg in $\mathbb{D}$.

Now we are ready to give a categorical semantics for our process theory. A *structure* in $\mathbb{C}$ for a process signature Sg is specified by the following data:

- For each ground type γ of Sg, an object $[\![\gamma]\!]$ of $\mathbb{C}$. The operation $\gamma \mapsto [\![\gamma]\!]$ is then extended inductively to all types by

$$\begin{aligned}
[\![I]\!] &\stackrel{\text{def}}{=} I \\
[\![\bot]\!] &\stackrel{\text{def}}{=} \bot \\
[\![\alpha^{\perp}]\!] &\stackrel{\text{def}}{=} [\![\alpha]\!]^{\perp} \\
[\![\alpha \otimes \beta]\!] &\stackrel{\text{def}}{=} [\![\alpha]\!] \otimes [\![\beta]\!] \\
[\![\alpha \parr \beta]\!] &\stackrel{\text{def}}{=} [\![\alpha]\!] \parr [\![\beta]\!] \\
[\![\bigcirc\alpha]\!] &\stackrel{\text{def}}{=} \bigcirc[\![\alpha]\!].
\end{aligned}$$

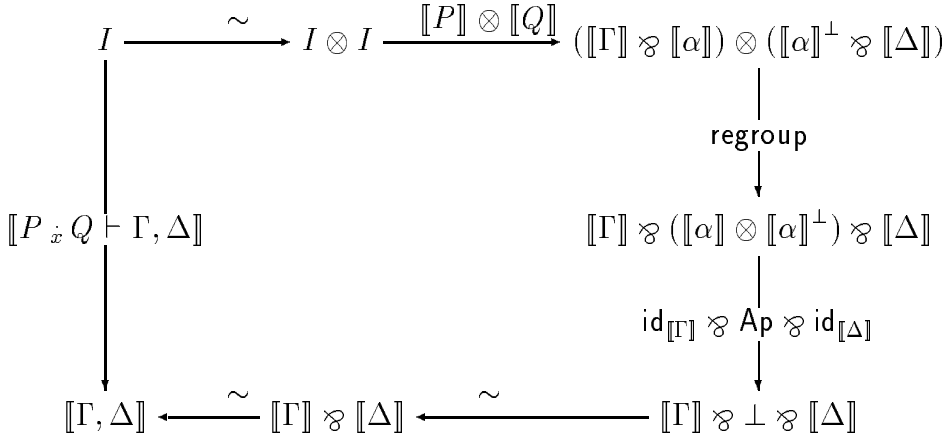
- For each process symbol $P : \alpha_1, \ldots, \alpha_n$ a morphism $[\![P]\!] : I \to [\![\alpha_1]\!] \parr \ldots \parr [\![\alpha_n]\!]$ in $\mathbb{C}$.

We then define, for each proved process $Q \vdash \Gamma$ generated by Sg, a morphism $[\![Q \vdash \Gamma]\!] : I \to [\![\Gamma]\!]$ in $\mathbb{C}$, where, if $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$, then $[\![\Gamma]\!] \stackrel{\text{def}}{=} [\![\alpha_1]\!] \parr \ldots \parr [\![\alpha_n]\!]$. The definition of $[\![Q \vdash \Gamma]\!]$ is by induction on the derivation of $Q \vdash \Gamma$.

Process Symbol If $P : \alpha_1 \ldots \alpha_n, [\![P(x_1 \ldots x_n) \vdash x_1 : \alpha_1, \ldots, x_n : \alpha_n]\!] = [\![P]\!]$.

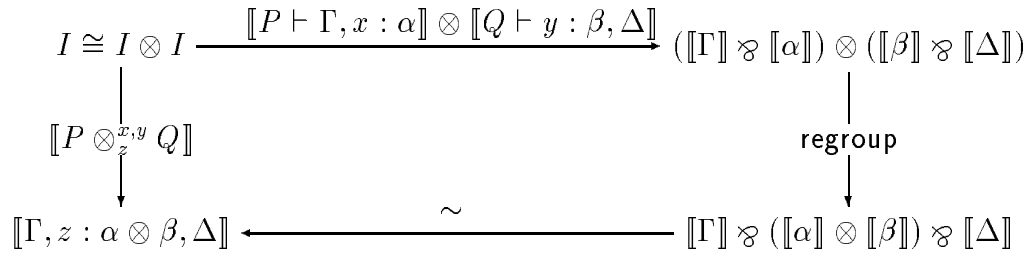
Axiom $[\![I_{x,y} \vdash x : \alpha^\perp, y : \alpha]\!] \stackrel{\text{def}}{=} \Lambda(\mathsf{unitl}_{[\![\alpha]\!]}) : I \to [\![\alpha]\!]^\perp \parr [\![\alpha]\!]$ where $\mathsf{unitl}_{[\![\alpha]\!]} : I \otimes [\![\alpha]\!] \to [\![\alpha]\!]$.

Cut For the Cut rule, the interpretation of $P \stackrel{x}{,} Q \vdash \Gamma, \Delta$ is defined by



where regroup is a canonical morphism in any $*$-autonomous category.

Tensor The interpretation of $\otimes_z^{x,y}(P, Q) \vdash \Gamma, z : \alpha \otimes \beta, \Delta$ is defined by



Par Applying the Par rule does not change the interpretation of a process: $[\![\parr_z^{x,y}(P) \vdash \Gamma, z : \alpha \parr \beta]\!] = [\![P \vdash \Gamma, x : \alpha, y : \beta]\!]$.

Unit $[\![\mathsf{unit}_x \vdash x : I]\!] = \mathsf{id}_I : I \to I$.

Bottom $[\![\mathsf{bottom}_x(P) \vdash \Gamma, x : \bot]\!]$ is defined by

$$\begin{array}{ccc} I & \xrightarrow{\quad [\![P \vdash \Gamma]\!] \quad} & [\![\Gamma]\!] \\ \Big\downarrow [\![\mathsf{bottom}_x(P) \vdash \Gamma, x : \bot]\!] & & \Big\downarrow \sim \\ [\![\Gamma, x : \bot]\!] & = & [\![\Gamma]\!] \parr \bot \end{array}$$

Delay The interpretation of $\bigcirc Q \vdash x_1 : \bigcirc\alpha_1, \ldots, x_n : \bigcirc\alpha_n$ is defined by

$$\begin{array}{ccccc} I & \xrightarrow{\quad \mathsf{monunit} \quad} & \bigcirc I & \xrightarrow{\quad \bigcirc[\![Q]\!] \quad} & \bigcirc([\![A_1]\!] \parr \ldots \parr [\![A_n]\!]) \\ \Big\downarrow [\![\bigcirc Q]\!] & & & & \Big\downarrow \sim \\ \bigcirc[\![A_1]\!] \parr \ldots \parr \bigcirc[\![A_n]\!] & & = & & [\![x_1 : \bigcirc A_1, \ldots, x_n : \bigcirc A_n]\!] \end{array}$$

Recursion From the hypotheses of the rule we have

$$I \xrightarrow{[\![P \vdash x : \alpha^\bot, y : \bigcirc\alpha]\!]} [\![\alpha]\!]^\bot \parr \bigcirc[\![\alpha]\!]$$

and

$$I \xrightarrow{[\![Q \vdash u : \bigcirc\beta^\bot, v : \beta]\!]} \bigcirc[\![\beta]\!]^\bot \parr [\![\beta]\!]$$

From these morphisms, we can canonically construct $f : [\![\alpha]\!] \to \bigcirc[\![\alpha]\!]$ and $g : \bigcirc[\![\beta]\!] \to [\![\beta]\!]$. Applying the UFPP gives $h : [\![\alpha]\!] \to [\![\beta]\!]$. Then

$$[\![\mathsf{Rec}^{x,y,u,v}_{z,w}(P, Q) \vdash z : \alpha^\bot, w : \beta]\!] \stackrel{\text{def}}{=} \Lambda(\mathsf{unitl} \,;\, h) : I \to [\![\alpha]\!]^\bot \parr [\![\beta]\!].$$

We can now define the notion of a model of a process theory in an interaction category. First, we define *satisfaction* of equations-in-context. If we are given a process theory $\mathcal{T}h$ and a structure $\mathbf{M}$ of the underlying Sg in an interaction category $\mathbb{C}$, then $\mathbf{M}$ *satisfies* an equation-in-context $P = Q \vdash \Gamma$ if $[\![P \vdash \Gamma]\!] = [\![Q \vdash \Gamma]\!]$ in $\mathbb{C}$. A *model* of $\mathcal{T}h$ in $\mathbb{C}$ is a structure $\mathbf{M}$ for Sg in $\mathbb{C}$ which satisfies the axioms in $\mathcal{A}x$.

Proposition 4 *If a structure* $\mathbf{M}$ *for* Sg *in* $\mathbb{C}$ *satisfies the axioms in* $\mathcal{A}x$*, then it satisfies the theorems of* $\mathcal{T}h$*.*

Proof: By induction on the rules for generating theorems. □

4 Classifying Category of a Typed Process Theory

4.1 Definition of the Classifying Category

Let $\mathcal{T}h$ be a process theory, $\mathbb{C}$ and $\mathcal{D}$ interaction categories, $\mathbf{M}$ a model of $\mathcal{T}h$ in $\mathbb{C}$ and $F : \mathbb{C} \to \mathcal{D}$ an interaction functor. Then there is a model $F_*\mathbf{M}$ of $\mathcal{T}h$ in $\mathcal{D}$, defined as follows.

On ground types γ of the signature Sg of $\mathcal{T}h$, $[\![\gamma]\!]_{F_*\mathbf{M}} \stackrel{\text{def}}{=} F([\![\gamma]\!]_{\mathbf{M}})$. Now note that for each type α there is a canonical isomorphism $[\![\alpha]\!]_{F_*\mathbf{M}} \cong F([\![\alpha]\!]_{\mathbf{M}})$. If $P : \alpha_1, \ldots \alpha_n$ is a process symbol of $\mathcal{T}h$ then these canonical isomorphisms are used to define $[\![P]\!]_{F_*\mathbf{M}}$.

$$\begin{array}{ccccc}
I & \xrightarrow{\sim} & FI & \xrightarrow{F[\![P]\!]_{\mathbf{M}}} & F([\![\alpha_1]\!]_{\mathbf{M}} \otimes \ldots \otimes [\![\alpha_n]\!]_{\mathbf{M}}) \\
\downarrow{\scriptstyle [\![P]\!]_{F_*\mathbf{M}}} & & & & \downarrow{\scriptstyle \sim} \\
[\![\alpha_1]\!]_{F_*\mathbf{M}} \otimes \ldots \otimes [\![\alpha_n]\!]_{F_*\mathbf{M}} & & \xleftarrow{\sim} & & F[\![\alpha_1]\!]_{\mathbf{M}} \otimes \ldots \otimes F[\![\alpha_n]\!]_{\mathbf{M}}
\end{array}$$

Given a process theory $\mathcal{T}h$, the *classifying category* of the theory is an interaction category $Cl(\mathcal{T}h)$ with the following universal property. For any other interaction category $\mathbb{C}$ and model $\mathbf{M}$ of $\mathcal{T}h$ in $\mathbb{C}$, there exists a unique (up to isomorphism) interaction functor $F : Cl(\mathcal{T}h) \to \mathbb{C}$ such that

$$\begin{array}{ccc}
\mathcal{T}h & \xrightarrow{\mathbf{M}} & \mathcal{C} \\
\downarrow{\scriptstyle \mathbf{G}} & \nearrow{\scriptstyle \exists! F} & \\
Cl(\mathcal{T}h) & &
\end{array}$$

commutes, by which we mean that $F_*\mathbf{G} = \mathbf{M}$.

4.2 Syntactic Construction of $Cl(\mathcal{T}h)$ and its Generic Model

Given a typed process theory $\mathcal{T}h$ over a signature Sg there is a syntactic interaction category $Cl(\mathcal{T}h)$, defined as follows.

- The objects of $Cl(\mathcal{T}h)$ are the types of Sg.
- A morphism $A \to B$ in $Cl(\mathcal{T}h)$ is an equivalence class of proved processes

$$Q \quad \vdash \quad x : A^{\perp}, y : B$$

where $Q \vdash x : A^\perp, y : B$ and $R \vdash u : A^\perp, v : B$ are equivalent if and only if $Q = R[x/u, y/v] \vdash x : A^\perp, y : B$ is a theorem of $\mathcal{T}h$. The equivalence class of $Q \vdash x : A^\perp, y : B$ is denoted by $(Q \vdash x : A^\perp, y : B)$.

- The identity morphism on A is $(I_{x,y} \vdash x : A^\perp, y : A) : A \to A$.
- Composition is defined by $(Q \vdash x : A^\perp, y : B)\,;(R \vdash u : B^\perp, v : C) = (Q \fatsemi_y R[y/u, v'/v] \vdash x : A^\perp, v' : C) : A \to C$. where v' is a fresh name.

Lemma 5 *Composition is well defined.*

Proof: Suppose we have $Q' \vdash x' : A^\perp, y' : B$ and $R' \vdash u' : B^\perp, v' : C$ with $Q = Q'[x/x'][y/y'] \vdash x : A^\perp, y : B$ and $R = R'[u/u'][v/v'] \vdash u : B^\perp, v : C$ provable.

Then we need to prove $Q \fatsemi_y R[y/u] = (Q' \fatsemi_y R'[y'/u'])[x/x'][v/v'] \vdash x : A^\perp, v : C$. The following calculation establishes this, using observations about substitution and which names occur in which processes.

$$\begin{array}{rcl}
\text{RHS} & \equiv_\alpha & (Q'[z/y'] \fatsemi_z R'[y'/u', v'''/v'][z/y'])[x/x', v''/v'''] \\
& = & (Q'[z/y'] \fatsemi_z R'[z/u', v'''/v'])[x/x', v''/v'''] \\
& = & Q'[z/y', x/x'] \fatsemi_z R'[z/u', v''/v'] \\
& \equiv_\alpha & Q'[y/y', x/x'] \fatsemi_y R'[y/u', v''/v] \\
& = & Q'[y/y', x/x'] \fatsemi_y R'[u/u', v/v'][y/u, v''/v] \\
& = & Q \fatsemi_y R[y/u, v''/v] \\
& = & \text{LHS}
\end{array}$$

□

Proposition 6 *$Cl(\mathcal{T}h)$ is a category.*

Proof: This essentially follows from the cut-elimination equation for the identity axiom, and the associativity of cut. The formal proof uses these equations and some manipulation of substitutions. □

There is a $*$-autonomous structure on $Cl(\mathcal{T}h)$. The bifunctor $\otimes$ is defined as follows.

- On objects, by the syntactic $\otimes$ on types.
- On morphisms, if $(Q \vdash x : A^\perp, y : C) : A \to C$ and $(R \vdash u : B^\perp, v : D) : B \to D$ then

  $(Q) \otimes (R) \stackrel{\text{def}}{=} (\parr_z^{x,u}(Q \otimes_w^{y,v} R) \vdash z : A^\perp \parr B^\perp, w : C \otimes D) : A \otimes B \to C \otimes D.$

Lemma 7 *$\otimes$ is well defined on morphisms.*

Proof: Follows from properties of substitutions. □

Proposition 8 $\otimes$ *is a bifunctor.*

Proof: Follows from the Tensor-Par cut elimination equation and the Tensor-Par identity equation. $\Box$

The canonical isomorphisms describing the symmetric monoidal structure are defined as follows.

- $\mathsf{unitl} : I \otimes A \to A$ is $(\parr_u^{z,x}(\mathsf{bottom}_z(I_{x,y})) \vdash u : \bot \parr A^\bot, y : A)$.
- $\mathsf{unitr} : A \otimes I \to A$ is $(\parr_u^{x,z}(\mathsf{bottom}_z(I_{x,y})) \vdash u : A^\bot \parr \bot, y : A)$.
- $\mathsf{unitl}^{-1} : A \to I \otimes A$ is $(\mathsf{unit}_z \otimes_u^{z,y} I_{x,y} \vdash x : A^\bot, u : I \otimes A)$.
- $\mathsf{unitr}^{-1} : A \to A \otimes I$ is $(I_{x,y} \otimes_u^{y,z} \mathsf{unit}_z \vdash x : A^\bot, u : A \otimes I)$.
- $\mathsf{symm} : A \otimes B \to B \otimes A$ is $(\parr_w^{x,u}(I_{x,y} \otimes_z^{v,y} I_{u,v}) \vdash w : A^\bot \parr B^\bot, z : B \otimes A)$.
- $\mathsf{assoc} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$ is $(\parr_s^{u,r}(\parr_r^{v,w}((I_{u,x} \otimes_p^{x,y} I_{v,y}) \otimes_q^{p,z} I_{w,z})) \vdash s : A^\bot \parr (B^\bot \parr C^\bot), q : (A \otimes B) \otimes C)$.
- $\mathsf{assoc}^{-1} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ is $(\parr_s^{r,w}(\parr_r^{u,v}(I_{u,x} \otimes_q^{x,p}(I_{v,y} \otimes_p^{y,z} I_{w,z}))) \vdash s : (A^\bot \parr B^\bot) \parr C^\bot, q : A \otimes (B \otimes C))$.

All the necessary equations among the canonical isomorphisms, including the coherence conditions and naturality, follow from the cut elimination and identity equations.

The definition of negation is as follows.

- On objects, by the syntactic $(-)^\bot$.
- On morphisms, $(-)^\bot$ of $(Q \vdash x : A^\bot, y : B) : A \to B$ is $(Q \vdash y : B^{\bot\bot}, x : A^\bot) : B^\bot \to A^\bot$. Note that this is exactly the same process, but its ports are taken in the opposite order.

For the closed structure:

- Given $f : A \otimes B \to C$, we need $\Lambda(f) : A \to B^\bot \parr C$. If f is $(Q \vdash x : A^\bot \parr B^\bot, y : C) : A \otimes B \to C$ then $\Lambda(f)$ is $(\parr_p^{w,y}((I_{u,v} \otimes_x^{v,z} I_{w,z}) \,_x\, Q) \vdash u : A^\bot, p : B^\bot \parr C) : A \to B^\bot \parr C$.
- We need $\mathsf{Ap} : A \otimes (A^\bot \parr B) \to B$. It is $(\parr_w^{x,z}(I_{x,y} \otimes_z^{y,u} I_{u,v}) \vdash w : A^\bot \parr (A \otimes B^\bot), v : B : A \otimes (A^\bot \parr B) \to B$.

All the necessary properties of abstraction and application follow from cut-elimination and structural equations.

We can define a unit delay functor $\circ$ on $Cl(\mathcal{T}h)$ as follows.

- On objects, we use the syntactic $\circ$.
- On morphisms, $\circ(Q \vdash x : A^\perp, y : B)$ is $(\circ Q \vdash x : \circ A^\perp, y : \circ B) : \circ A \to \circ B$.

Functoriality follows from the $\circ$ cut-elimination and identity equations.

Proposition 9 *$\circ$ has the UFPP.*

Proof: If $f : A \to \circ A$ is $(Q \vdash x : A^\perp, y : \circ A)$ and $g : \circ B \to B$ is $(R \vdash u : \circ B^\perp, v : B)$ then $h : A \to B$ is $(\mathsf{Rec}_{z,w}^{x,y,u,v}(Q, R) \vdash z : A^\perp, w : B)$.

The facts that $f\,;\circ h\,;g = h$ and h is the unique such morphism follow from the recursion equations. □

The canonical morphism $\alpha : I \to \circ I$ is $(\mathsf{bottom}_x(\circ\mathsf{unit}_y) \vdash x : \bot, y : \circ I)$.

The natural transformation (component) $\beta : \circ A \otimes \circ B \to \circ(A \otimes B)$ is

$$(\parr_w^{x,u}(\circ(I_{x,y} \otimes_z^{y,v} I_{u,v})) \vdash w : \circ A^\perp \parr \circ B^\perp, z : \circ(A \otimes B^\perp)).$$

These make $\circ$ a monoidal endofunctor on $Cl(\mathcal{T}h)$.

For $(\circ A)^\perp \cong \circ(A^\perp)$, the types are syntactically equivalent, so the isomorphism is just the identity on $\circ A$.

There is a structure $\mathbf{G}$ for the signature Sg of the theory $\mathcal{T}h$ in $Cl(\mathcal{T}h)$ defined by

- For a ground type γ of Sg,
$$[\![\gamma]\!] = \gamma.$$
- For a process symbol $P : \alpha_1 \ldots \alpha_n$ of Sg,
$$\begin{aligned}[\![P]\!] \;=\; & (\mathsf{bottom}_y(\parr_z^{x_1,\ldots,x_n}(P(x_1,\ldots,x_n))) \vdash y : \bot, z : \alpha_1 \parr \ldots \parr \alpha_n) : \\ & I \to [\![\alpha_1]\!] \parr \ldots \parr [\![\alpha_n]\!],\end{aligned}$$
where $\parr_z^{x_1,\ldots,x_n}$ is an abbreviation for $n-1$ applications of a binary $\parr$.

Proposition 10 **G** *satisfies the axioms of $\mathcal{T}h$ and so is a model of the theory in the category $Cl(\mathcal{T}h)$.*

Proof: If $Q = R \vdash \Gamma$ is an axiom of $\mathcal{T}h$, where $\Gamma = x_1 : A_1 \ldots x_n : A_n$, then

$$[\![Q \vdash \Gamma]\!] = (\mathsf{bottom}_y(\otimes_z^{x_1,\ldots,x_n}(Q)) \vdash y : \bot, z : A_1 \otimes \ldots \otimes A_n)$$

and

$$[\![R \vdash \Gamma]\!] = (\mathsf{bottom}_y(\otimes_z^{x_1,\ldots,x_n}(R)) \vdash y : \bot, z : A_1 \otimes \ldots \otimes A_n).$$

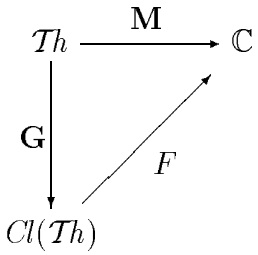
These morphisms are equal, because the congruence equations imply that

$$(\mathsf{bottom}_y(\otimes_z^{x_1,\ldots,x_n}(Q)) = (\mathsf{bottom}_y(\otimes_z^{x_1,\ldots,x_n}(R)) \vdash y : \bot, z : A_1 \otimes \ldots \otimes A_n)$$

is a theorem of $\mathcal{T}h$. □

We can now show that the category $Cl(\mathcal{T}h)$ and the model $\mathbf{G}$ of $\mathcal{T}h$ in $Cl(\mathcal{T}h)$ do indeed constitute a classifying category for the process theory.

Theorem 11 *If $\mathcal{T}h$ is a typed process theory and $\mathbf{M}$ is a model of $\mathcal{T}h$ in an interaction category $\mathbb{C}$, then there is a unique interaction functor $F : Cl(Th) \to \mathbb{C}$ such that the diagram*



commutes, that is, $F_*\mathbf{G} = \mathbf{M}$.

Proof:

We define the functor $F : Cl(\mathcal{T}h) \to \mathbb{C}$. On objects, $F(\alpha) \stackrel{\text{def}}{=} [\![\alpha]\!]_M$ and on a morphism $Q : \alpha \to \beta$ in $Cl(\mathcal{T}h)$, $F(Q) \stackrel{\text{def}}{=} \hat{Q} : [\![\alpha]\!]_M \to [\![\beta]\!]_M$.

If α is a type of $\mathcal{T}h$,

$$[\![\alpha]\!]_{F_*G} \stackrel{\text{def}}{=} F([\![\alpha]\!]_G) = F(\alpha) \stackrel{\text{def}}{=} [\![\alpha]\!]_M.$$

If $P : \alpha_1 \ldots \alpha_n$ is a process symbol of $\mathcal{T}h$,

$$\begin{aligned} [\![P]\!]_{F_*G} &\stackrel{\text{def}}{=} F([\![P]\!]_G) \\ &= \widehat{([\![P]\!]_G)} \\ &= [\![P]\!]_G ; \mathsf{parunitl} \\ &= [\![P]\!]_M ; \mathsf{parunitl}^{-1} ; \mathsf{parunitl} \\ &= [\![P]\!]_M. \end{aligned}$$

□

where $\mathsf{parunitl} : \bot \parr A \to A$ and $\mathsf{parunitl}^{-1} : A \to \bot \parr A$. We also use the fact that when $g : I \to \bot \parr B$, $\hat{g} : I \to B$ and hence $\hat{g} = g; \mathsf{parunitl}$.

F is well defined because $\mathbb{C}$ soundly models $\mathcal{T}h$. Functoriality is proved using the $*$-autonomous structure of $\mathbb{C}$. F is an interaction functor and preserves everything.

5 Deriving a Theory from a Category

Given an interaction category $\mathbb{C}$ we can define a process theory $\mathcal{T}h(\mathbb{C}) = (Sg(\mathbb{C}), \mathcal{A}x(\mathbb{C}))$.

- The ground types of $Sg(\mathbb{C})$ are the objects of $\mathbb{C}$.
- We denote the syntactic type constructors in $\mathcal{T}h(\mathbb{C})$ by $\overline{\otimes}$, $\overline{\parr}$ and so on. Thus if A and B are objects of $\mathbb{C}$, $Sg(\mathbb{C})$ has a ground type $A \otimes B$ as well as a compound type $A \,\overline{\otimes}\, B$.
- For each morphism $P : I \to \alpha_1 \parr \ldots \parr \alpha_n$ in $\mathbb{C}$, $Sg(\mathbb{C})$ has a process symbol $P : \alpha_1 \ldots \alpha_n$.
- Additionally, for each type α of $Sg(\mathbb{C})$ there are process symbols $I_\alpha : \alpha^\bot, [\![\alpha]\!]$ and $J_\alpha : [\![\alpha]\!]^\bot, \alpha$ where $[\![-]\!]$ is defined below.

There is a canonical structure $\mathbf{M}$ of $Sg(\mathbb{C})$ in $\mathbb{C}$ defined by

$$\begin{aligned}
[\![\gamma]\!] &= \gamma \quad (\gamma \text{ any ground type of } Sg_{\mathbb{C}}) \\
[\![\alpha \,\overline{\otimes}\, \beta]\!] &= [\![\alpha]\!] \otimes [\![\beta]\!] \quad \textit{etc.} \\
[\![P]\!] &= P \\
[\![I]\!] &= \Lambda(\mathsf{unitl}_{[\![\alpha]\!]}) \\
[\![J]\!] &= \Lambda(\mathsf{unitl}_{[\![\alpha]\!]})
\end{aligned}$$

The axioms of $\mathcal{T}h(\mathbb{C})$ are the equations-in-context which are generated from $Sg(\mathbb{C})$ and which are satisfied by the canonical structure $\mathbf{M}$. Thus it is immediate that $\mathbf{M}$ is indeed a model of $\mathcal{T}h(\mathbb{C})$ in $\mathbb{C}$.

6 Categorical Logic Correspondence

Theorem 12 *There is an equivalence $Eq : Cl(\mathcal{T}h(\mathbb{C})) \simeq \mathbb{C}$ in which the functor Eq arises from the universal property of the generic model of $\mathcal{T}h(\mathbb{C})$ in $Cl(\mathcal{T}h)$ applied to the canonical model of $\mathcal{T}h(\mathbb{C})$ in $\mathbb{C}$.*

Proof: We define a functor $Eq^{-1} : \mathbb{C} \to Cl(\mathcal{T}h(\mathbb{C}))$ by setting $Eq^{-1}(A) \stackrel{\text{def}}{=} A$ on objects A of $\mathbb{C}$, and $Eq^{-1}(P) \stackrel{\text{def}}{=} (\bar{P} \vdash A^{\perp}, B)$: where $P : A \to B$ is a morphism of $\mathbb{C}$.

Note that $[\![\bar{P}]\!] : I \to A^{\perp} \otimes B$ in $\mathbb{C}$, and hence $\bar{P}$ is a process symbol in the signature $Sg(\mathbb{C})$.

It is a routine but lengthy exercise to verify that the functors Eq and Eq^{-1} give rise to an equivalence of interaction categories. □

7 Conclusions and Future Work

We have established a precise correspondence between interaction categories and a process theory in the same mould as the correspondence between, say, cartesian closed categories and typed λ-calculus.

The situation is rather simplified—made interesting and different from previous accounts by the $*$-autonomous structure and the unit delay functor. Future work would involve extending this work to cover the full process calculus, for example, as in [7, 8] and more of the categorical structure of Interaction Categories. We need to include a treatment of non-deterministic sum and prefixing constructions on the process calculus side and products and exponentials among the categorical aspects. Our account does not distinguish between synchrony and asynchrony, and such a general approach may not be possible once we incorporate additional constructs.

Acknowledgements

Roy Crole was funded by an EPSRC Postdoctoral Fellowship; Simon Gay by an EPSRC Studentship and the EU Basic Research Project 9102 (COORDINATION); and Rajagopal Nagarajan by the EU Basic Research Project 6454 (CONFER).

Lindsay Errington and Samson Abramsky made several useful comments on a draft of this paper.

References

[1] S. Abramsky, S. J. Gay, and R. Nagarajan. Interaction categories and foundations of typed concurrent programming. In M. Broy, editor, *Deductive Program Design: Proceedings of the 1994 Marktoberdorf International Summer School*,

NATO ASI Series F: Computer and Systems Sciences. Springer-Verlag, 1995. Also available as `theory/papers/Abramsky/marktoberdorf.ps.gz` via anonymous ftp to `theory.doc.ic.ac.uk`.

[2] S. Abramsky, S. J. Gay, and R. Nagarajan. Specification structures and propositions-as-types for concurrency. In G. Birtwistle and F. Moller, editors, *Logics for Concurrency: Structure vs. Automata—Proceedings of the VIIIth Banff Higher Order Workshop*, Lecture Notes in Computer Science. Springer-Verlag, 1995. To appear.

[3] S. Abramsky. Interaction Categories (Extended Abstract). In G. L. Burn, S. J. Gay, and M. D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, pages 57–70. Springer-Verlag Workshops in Computer Science, 1993.

[4] S. Abramsky. Interaction Categories and communicating sequential processes. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, pages 1–15. Prentice Hall International, 1994.

[5] M. Barr. $*$-*Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.

[6] R. L. Crole. *Categories for Types*. Cambridge University Press, 1994.

[7] S. J. Gay. *Linear Types for Communicating Processes*. PhD thesis, University of London, 1995. Available as `theory/papers/Gay/thesis.ps.gz` via anonymous ftp to `theory.doc.ic.ac.uk`.

[8] S. J. Gay and R. Nagarajan. A typed calculus of synchronous processes. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1995. To appear.

[9] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[10] J. R. Hindley and J. P. Seldin. *Introduction to combinators and λ-calculus*. Cambridge University Press, 1986.

[11] J. Lambek. From lambda calculus to cartesian closed categories. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–402. Academic Press, London, 1980.

[12] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics Vol. 7. Cambridge University Press, 1986.

[13] C. McLarty. *Elementary Categories, Elementary Toposes*, volume 21 of *Oxford Logic Guides*. Oxford University Press, 1991.

[14] A. M. Pitts. Polymorphism is set theoretic, constructively. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science, Proc. Edinburgh 1987*, volume 283 of *Lecture Notes in Computer Science*, pages 12–39. Springer-Verlag, Berlin, 1987.