A Simple Categorical Calculus of Interacting Processes

Chad Nester[a,1] Niels Voorneveld[b,2]

[a] *University of Tartu*
Tartu, Estonia

[b] *Cybernetica AS*
Tallinn, Estonia

Abstract

We present a calculus that models a simple sort of process interaction. Our calculus consists of a collection of terms together with a rewrite relation, parameterised by an arbitrary multicategory whose morphisms we understand as non-interactive processes. We show that our calculus is confluent and terminating, and that terms modulo the induced convertibility relation form a virtual double category. We relate our calculus to the free cornering of a monoidal category, which is a double-categorical model of process interaction that is similar in spirit to the calculus presented herein. Precisely, we construct a functor from the virtual double category given by our calculus into the underlying virtual double category of the free cornering of the free monoidal category on the multicategory of non-interacting processes. If we think of the terms of our calculus as programs and the rewriting system as an operational semantics for these programs, this functor gives a sound denotational semantics for our calculus in terms of the free cornering.

Keywords: Category Theory, Message Passing, Term Rewriting, Programming Language Semantics, Virtual Double Categories

1 Introduction

This paper concerns the categorical semantics of interaction. Specifically, we give a calculus of interacting processes and investigate its categorical structure. Our calculus is parameterised by a multicategory $\mathcal{M}$, the morphisms of which we think of as non-interactive processes. Our calculus augments these non-interactive processes with the ability to interact via a simple message-passing mechanism. More concretely, our calculus consists of a collection $T(\mathcal{M})$ of terms presented as a sequent calculus together with a rewrite relation $\to$ on those terms. This rewrite relation is confluent and terminating. When considered modulo the induced convertibility relation, the terms of our calculus form a virtual double category $[\mathcal{M}]$.

Our calculus is similar in spirit to the free cornering $\ulcorner \mathbb{A} \urcorner$ of a monoidal category $\mathbb{A}$ [13,15]. Both model the same sort of interaction, and are constructed from a pre-existing collection of non-interacting processes, being the monoidal category $\mathbb{A}$ in the case of the free cornering. Given this, it is not too surprising that the two formalisms are also related in a technical sense. The free cornering of a monoidal category is a strict double category, and there is a close relationship between strict double categories and virtual

⋆ Chad Nester was supported by the Estonian Research Council grant PRG2764. Niels Voorneveld was supported by Estonian Research Council grant No. PRG1780.

[1] Email: nester@ut.ee
[2] Email: niels.voorneveld@cyber.ee

double categories, analogous to the relationship between strict monoidal categories and multicategories. Specifically, there are adjunctions:

$$\mathsf{Mul} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathsf{Mon} \qquad\qquad \mathsf{VDC} \underset{U}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathsf{Dbl}$$

where Mul, Mon, VDC, and Dbl are the category of multicategories, strict monoidal categories, virtual double categories, and strict double categories, respectively. In fact, these two adjunctions are instances of the same general theorem concerning notions of generalised multicategory that arise from cartesian monads [11].

For any multicategory $\mathcal{M}$, there is a functor of virtual double categories:

$$[\![-]\!] : [\mathcal{M}] \to U\left(\ulcorner F(\mathcal{M}) \urcorner\right)$$

This is to say that our term calculus admits a sound denotational semantics, where terms are interpreted as cells of the free cornering of $F(\mathcal{M})$. The primary difference between the calculus presented here and the free cornering construction is that our calculus is *dynamic*, being presented as a term rewriting system, while the free cornering is *static*, in the sense that it is presented in terms of equations.

1.1 Organisation

In Section 2 we recall the free cornering of a monoidal category. This is done before anything else so that the string diagrams for the free cornering, which we find useful in conveying the intuition behind our term calculus, are available in the rest of our development. In Section 3 we construct the terms $T(\mathcal{M})$ and rewrite relation $\to$ that comprise our calculus, and show that $\to$ is confluent and terminating. We also give the interpretation $[\![-]\!]$ of terms as cells of the free cornering and show that it is coherent with respect to our rewrite relation, in the sense that if $t \overset{*}{\leftrightarrow} s$ then $[\![t]\!] = [\![s]\!]$. In Section 4 we show that terms modulo the convertibility relation $\overset{*}{\leftrightarrow}$ form a virtual double category, and that $[\![-]\!]$ extends to a functor of virtual double categories in the manner described above. We conclude in Section 5.

1.2 Prerequisites

The reader is assumed to have some familiarity with category theory, including adjunctions, monoidal categories, multicategories, and the basic ideas of categorical logic (see e.g., [12,10,17]). Familiarity with double categories and virtual double categories (see e.g., [11,8]) is also effectively necessary. Some knowledge of term rewriting is also required. We will use notions like confluence, termination, and local confluence together with a few classical results and techniques in the theory of term rewriting without introduction or comment. An excellent reference is [1]. We imagine that some level of awareness concerning the field of programming language semantics would be helpful, but such awareness is not, strictly speaking, necessary to follow the development herein.

1.3 Related Work

Our work is inspired by Cockett and Pastro's logic of message passing [6] and Wadler's work on the semantics of session types [18]. Other significant antecedents along these lines include the work of Caires and Pfenning [4], Honda [9], and Bellin and Scott [2].

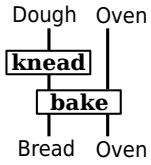
The calculus presented here grew out of our attempts to provide a more "operational" account of the free cornering of a monoidal category, which was introduced by Nester [13,15] and has been the subject of a series of papers exploring its use as a categorical model of process interaction [14,3,16].

2 The Free Cornering of A Monoidal Category

In this section we recapitulate the construction of the free cornering of a monoidal category and its interpretation as a theory of interacting processes. The free cornering is a single-object strict double

category, and the reader who is unfamiliar with double categories may wish to consult e.g., [11,16].

Morphisms in monoidal categories often admit interpretations as resource-transforming processes (see e.g., [7]). The objects of the category in question are interpreted as collections of resources, with the unit I denoting the empty collection and $A \otimes B$ denoting collection consisting of the resources of both A and B. Then a morphism $f : A \to B$ is interpreted as a process that transforms the resources of A into those of B, composition is interpreted as sequencing, and the tensor product allows independent processes to occur simultaneously. For example if we have morphisms **bake** : $\mathsf{Dough} \otimes \mathsf{Oven} \to \mathsf{Bread} \otimes \mathsf{Oven}$ and **knead** : $\mathsf{Dough} \to \mathsf{Dough}$ representing the processes of baking and kneading dough, respectively, then the composite morphism $(\mathbf{knead} \otimes 1_{\mathsf{Oven}}); \mathbf{bake}$ represents the combined process of kneading and then baking a given unit of dough. It is often helpful to picture such processes using string diagrams, as in:



(1)

The purpose of this section is to recall the *free cornering* of a monoidal category, which is a strict single-object double category that models a simple sort of process interaction. If morphisms of a given monoidal category model processes of some kind, then cells of the free cornering model interacting processes of that kind. Interaction in the free cornering is governed by simple protocol types, constructed as follows:

Definition 2.1 Let X be a set. We define the monoid $X^{\circ\bullet}$ of *X-valued exchanges* to be the free monoid on the set of polarised elements of X, as in $X^{\circ\bullet} = (X \times \{\circ, \bullet\})^*$. Explicitly, elements of $X^{\circ\bullet}$ are sequences $A_1^{p_1} \cdots A_n^{p_n}$ with $A_i \in X$ and $p_i \in \{\circ, \bullet\}$ for each $1 \leq i \leq n$. We write λ to indicate the empty sequence, and write either UW or U, W to indicate the concatenation of sequences U and W.

If elements of X can be understood as resources, then elements of $X^{\circ\bullet}$ can be understood as simple interaction protocols. Each protocol has two participants, one on the left and one on the right. The protocol A° demands that the left participant send the right participant an instance of the resource A, and dually $A^\bullet$ demands that the right participant send the left participant an instance of A. The protocol UW demands that the participants carry out the protocol U, and then carry out the protocol W. The protocol λ is the empty protocol, which demands nothing and is finished immediately. For example, suppose that $A, B \in X$, that the left participant is called "Alice", and that the right participant is called "Bob". In this case, to carry out $A^\circ B^\bullet A^\bullet$, first Alice must send Bob an instance of A, then Bob must send Alice an instance of B, and then Bob must send Alice an instance of A.

Definition 2.2 Let $\mathbb{A}$ be a strict monoidal category. The *free cornering of* $\mathbb{A}$, written $\ulcorner\mathbb{A}\urcorner$, is the strict double category with a single object $\ulcorner\mathbb{A}\urcorner_0 = \{*\}$, with horizontal edge monoid $\ulcorner\mathbb{A}\urcorner_H = (\mathbb{A}_0, \otimes, I)$ given by the object monoid of $\mathbb{A}$, with vertical edge monoid given by $\ulcorner\mathbb{A}\urcorner_V = \mathbb{A}_0^{\circ\bullet}$ given by the monoid of $\mathbb{A}_0$-valued exchanges (Definition 2.1), and with cells constructed according to the following rules:

$$\frac{f \in \mathbb{A}(A,B)}{\ulcorner f \urcorner : \left(\lambda \begin{smallmatrix} A \\ B \end{smallmatrix} \lambda\right)} \qquad \frac{A \in \mathbb{A}_0}{A^\urcorner : \left(A^\circ \begin{smallmatrix} I \\ A \end{smallmatrix} \lambda\right)} \qquad \frac{A \in \mathbb{A}_0}{A_\llcorner : \left(\lambda \begin{smallmatrix} A \\ I \end{smallmatrix} A^\circ\right)} \qquad \frac{A \in \mathbb{A}_0}{A^\ulcorner : \left(\lambda \begin{smallmatrix} I \\ A \end{smallmatrix} A^\bullet\right)} \qquad \frac{A \in \mathbb{A}_0}{A_\lrcorner : \left(A^\bullet \begin{smallmatrix} A \\ I \end{smallmatrix} \lambda\right)}$$

$$\frac{A \in \mathbb{A}_0}{1_A : \left(\lambda \begin{smallmatrix} A \\ A \end{smallmatrix} \lambda\right)} \qquad \frac{a : \left(U \begin{smallmatrix} A \\ B \end{smallmatrix} W\right) \qquad b : \left(U' \begin{smallmatrix} B \\ C \end{smallmatrix} W'\right)}{a \cdot b : \left(UU' \begin{smallmatrix} A \\ C \end{smallmatrix} WW'\right)} \qquad \frac{U \in \mathbb{A}_0^{\circ\bullet}}{id_U : \left(U \begin{smallmatrix} I \\ I \end{smallmatrix} U\right)} \qquad \frac{a : \left(U \begin{smallmatrix} A \\ B \end{smallmatrix} W\right) \qquad b : \left(W \begin{smallmatrix} A' \\ B' \end{smallmatrix} V\right)}{a \mid b : \left(U \begin{smallmatrix} A \otimes A' \\ B \otimes B' \end{smallmatrix} V\right)}$$

Cells are subject to a number of equations. First, equations concerning the cells $\ulcorner f \urcorner$:

$$\ulcorner f \urcorner \cdot \ulcorner g \urcorner = \ulcorner fg \urcorner \qquad \ulcorner f \urcorner \mid \ulcorner g \urcorner = \ulcorner f \otimes g \urcorner \qquad 1_A = \ulcorner 1_A \urcorner$$

second, the *yanking equations*:

$$A_{\llcorner} \mid A^{\urcorner} = 1_A \qquad A^{\ulcorner} \mid A_{\lrcorner} = 1_A \qquad A^{\urcorner} \cdot A_{\llcorner} = id_{A^\circ} \qquad A_{\lrcorner} \cdot A^{\ulcorner} = id_{A^\bullet}$$

and finally equations ensuring that the axioms of a double category are satisfied:

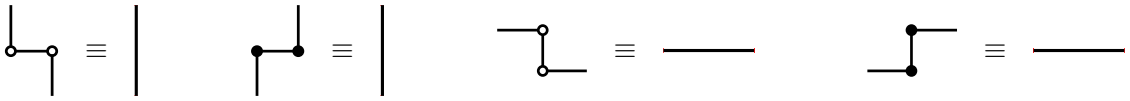
$$1_A \cdot a = a = a \cdot 1_B \qquad (a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad id_U \mid a = a = a \mid id_W \qquad (a \mid b) \mid c = a \mid (b \mid c)$$

$$(a \mid b) \cdot (c \mid d) = (a \cdot c) \mid (b \cdot d) \qquad id_\lambda = 1_I \qquad id_U \cdot id_W = id_{UW}$$

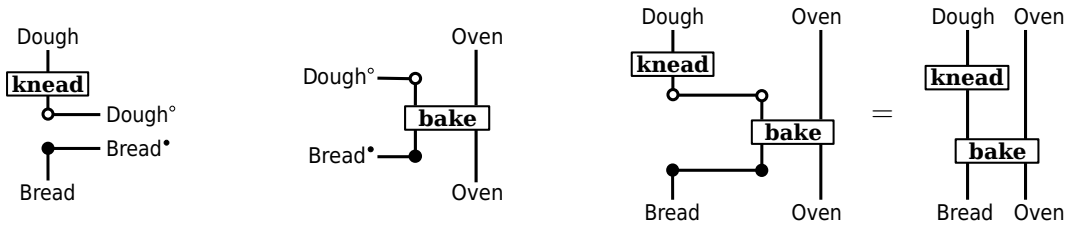
Note in particular that $1_{A\otimes B} = \ulcorner 1_{A\otimes B} \urcorner = \ulcorner 1_A \otimes 1_B \urcorner = \ulcorner 1_A \urcorner \mid \ulcorner 1_B \urcorner = 1_A \otimes 1_B$ already holds.

If a morphism of $\mathbb{A}$ can be understood as a process that transforms the resources indicated by its domain into those indicated by its codomain, then a cell of $\ulcorner \mathbb{A} \urcorner$ can be understood as a process that transforms the resources indicated by its top boundary into those indicated by its bottom boundary by interacting with other such processes in the manner indicated by its left and right boundary. The cells $\ulcorner f \urcorner$ and the equations concerning them serve to include the processes represented by $\mathbb{A}$ in the new setting in a coherent manner. The *corner cells* $A^{\urcorner}$,$A_{\llcorner}$,$A^{\ulcorner}$, and $A_{\lrcorner}$ allow interacting processes to exchange resources. It is helpful to think of the corner cells in terms of string diagrams, as in:

For example $A_{\llcorner}$ is the process that transforms an instance of A into nothing by giving it away along the right boundary, and $A^{\urcorner}$ is the process that transforms nothing into an instance of A by receiving that instance of A along the left boundary. The yanking equations tell us that being exchanged in this manner has no effect on the resources involved. The name "yanking equations" comes from their string-diagrammatic representation, as in:



For example, our procedure for baking bread (1) can be decomposed in the free cornering of the ambient monoidal category into the two cells pictured below left. Composing these cells horizontally recovers the image under $\ulcorner - \urcorner$ of the original morphism, pictured below on the right.



3 The Term Calculus

In this section we introduce the terms and rewrite relation that make up our calculus. Our development takes place with respect to a fixed multicategory $\mathcal{M}$, whose morphisms we understand as (non-interactive) processes. The purpose of the calculus is to allow such processes to interact by exchanging resources along a left and right boundary, much as in the free cornering of a monoidal category. The major difference between the calculus presented here and the free cornering is that our calculus is *directed*, with the relationship between terms given by a confluent and terminating rewrite relation. The free cornering is *undirected*, with the relationship between terms given by equations. Put another way, while the free cornering is a *static* model of process interaction, the term calculus presented here is a *dynamic* model of process interaction.

3.1 Sequent Calculus for Multicategories

Our terms will be presented as a sequent calculus, and in light of the fact that the construction is parameterised by our fixed multicategory $\mathcal{M}$ it will be convenient to use the sequent calculus notation for multicategories, which we recall now. Let Σ be a (multi-sorted) signature in which operation symbols are typed as in $f : A_1,..,A_n \vdash B$ where $A_1,..,A_n, B$ are generating sorts. Then a *term* over Σ is a sequent that is derivable via the following inference rules:

$$\frac{}{x : A \vdash x : A}\ \textsc{var} \qquad\qquad \frac{(\Gamma_i \vdash t_i : A_i)_{i=1}^n \qquad (f : A_1,..,A_n \vdash B) \in \Sigma}{\Gamma_1,..,\Gamma_n \vdash f(t_1,..,t_n) : B}\ \textsc{op}$$

Crucially, for a sequent $\Gamma \vdash t : B$ to be considered well-formed, the context Γ must not contain any repeated variables. This means that, for example, when we write $\Gamma_1,..,\Gamma_n$ it is implied that the variables in the Γ_i are disjoint. Terms over Σ form a multicategory, with identity morphisms given by the VAR rule and with composition given by substitution. More precisely, the composition operation is given by the following admissible inference rule:

$$\frac{(\Gamma_i \vdash t_i : A_i)_{i=1}^n \qquad x_1{:}A_1,..,x_n{:}A_n \vdash t : B}{\Gamma_1,..,\Gamma_n \vdash t[t_1,..,t_n/x_1,..,x_n] : B}\ \textsc{comp}$$

That this satisfies the equations of a multicategory follows from certain elementary properties of substitution. For example, for any $\Gamma \vdash t : B$ the right-unitality law ($1_B \circ f = f$) holds as in:

$$(\Gamma \vdash x[t/x] : B) = (\Gamma \vdash t : B)$$

The multicategory of terms over a signature is in fact the *free* multicategory over that signature, in the sense that this construction gives the left adjoint of an adjunction between a category of signatures and the category of multicategories. The right adjoint maps a multicategory $\mathcal{M}$ to the $\mathcal{M}_0$-sorted signature $\Sigma_{\mathcal{M}}$ with an operation symbol $f : A_1,..,A_n \vdash B$ for each $f \in \mathcal{M}(A_1,..,A_n;B)$. The counit of the adjunction gives a morphism from the multicategory of terms over this signature into $\mathcal{M}$, and quotienting the terms by the equations that hold in the image of this morphism yields a sequent calculus presentation of $\mathcal{M}$. To understand the effect of these extra equations, suppose $f \in \mathcal{M}(A,A;B)$, $g \in \mathcal{M}(C;A)$ and consider the following derivations:

$$\frac{\dfrac{}{x_1 : C \vdash x_1 : C}\ \textsc{var} \qquad \dfrac{}{x_2 : C \vdash x_2 : C}\ \textsc{var} \qquad ((f \circ (g,g)) : C, C \vdash B) \in \Sigma_{\mathcal{M}}}{x_1{:}C, x_2{:}C \vdash (f \circ (g,g))(x_1,x_2) : B}\ \textsc{op}$$

$$\frac{\dfrac{\dfrac{}{x_1 : C \vdash x_1 : C} \qquad (g : C \vdash A) \in \Sigma_{\mathcal{M}}}{x_1 : C \vdash g(x_1) : A}\ \textsc{op} \qquad \dfrac{\dfrac{}{x_2 : C \vdash x_2 : C} \qquad (g : C \vdash A) \in \Sigma_{\mathcal{M}}}{x_2 : C \vdash g(x_2) : A}\ \textsc{op} \qquad (f : A, A \vdash B) \in \Sigma_{\mathcal{M}}}{x_1{:}C, x_2{:}C \vdash f(g(x_1), g(x_2)) : B}\ \textsc{op}$$

These both represent the same morphism of $\mathcal{M}$ in the sense that they are equal in the image of the counit, but are not equal in the free multicategory over the underlying signature $\Sigma_{\mathcal{M}}$ of $\mathcal{M}$. To obtain a sequent calculus presentation of $\mathcal{M}$, we must reconcile these two ways of representing composition (and identities) in $\mathcal{M}$, which is achieved by quotienting our derivations in the manner described above.

It follows that we may reason about morphisms in any multicategory by means of a sequent calculus, interpreting composition as substitution and identities as variables. Going forward, we will write $\Gamma \vdash v : A$ to indicate a morphism of our fixed multicategory $\mathcal{M}$. For a more detailed account of the connection between multicategories and sequent calculus see e.g., [17].

3.2 Terms

We are now ready to introduce the terms of our calculus:

$$\frac{\Gamma \vdash v : A}{\Gamma \Vdash [v] : (\lambda, A, \lambda)}\ \text{SEQ} \qquad \frac{(\Gamma_i \Vdash t_i : (U_{i-1}, A_i, U_i))_{i=1}^n \qquad x_1{:}A_1,..,x_n{:}A_n \Vdash t : (V, B, W)}{\Gamma_1,..,\Gamma_n \Vdash \mathsf{let}\ x_1,..,x_n \downarrow (t_1 \mid \cdots \mid t_n)\ \mathsf{in}\ t : (U_0 V, B, U_n W)}\ \text{LET}$$

$$\frac{\Delta \Vdash t : (U, B, W) \qquad \Gamma \vdash v : A}{\Delta, \Gamma \Vdash \mathsf{putR}(v, t) : (U, B, A^\circ W)}\ \text{PUTR} \qquad \frac{x : A, \Gamma \Vdash t : (U, B, W)}{\Gamma \Vdash \mathsf{getL}(x.t) : (A^\circ U, B, W)}\ \text{GETL}$$

$$\frac{\Gamma, x : A \Vdash t : (U, B, W)}{\Gamma \Vdash \mathsf{getR}(x.t) : (U, B, A^\bullet W)}\ \text{GETR} \qquad \frac{\Gamma \vdash v : A \qquad \Delta \Vdash t : (U, B, W)}{\Gamma, \Delta \Vdash \mathsf{putL}(v, t) : (A^\bullet U, B, W)}\ \text{PUTL}$$

Fig. 1. Term formation rules for $T(\mathcal{M})$.

Definition 3.1 The collection of *terms over* $\mathcal{M}$, written $T(\mathcal{M})$, is constructed according to the inference rules of Figure 1. It is important to note that we retain the convention that contexts Γ must not contain repeated variables.

In what follows terms t are implied to be part of the ultimate conclusion $\Gamma \Vdash t : (U, B, W)$ of some derivation. We will often write the term t to stand for its derivation, and will often speak of the two interchangeably. This is, in some sense, the point of terms: to be a sort of shorthand for the accompanying derivation. For example, while our rewrite relation is specified on terms, it should be understood as rewriting the associated derivations. We introduce a bit of useful terminology:

Definition 3.2 We say that a term is:

- *neutral*, or sometimes a *value*, if it is of the form $[v]$.
- *left-facing* if it is of the form $\mathsf{putL}(v, t)$ or $\mathsf{getL}(x.t)$.
- *right-facing* if it is of the form $\mathsf{putR}(v, t)$ or $\mathsf{getR}(x.t)$.
- a *let-binding* if it is of the form $\mathsf{let}\ x_1,..,x_n \downarrow (t_1 \mid \cdots \mid t_n)\ \mathsf{in}\ t$.

If morphisms of $\mathcal{M}$ admit interpretation as resource-transforming processes, then terms of $T(\mathcal{M})$ can be understood as *interacting* resource-transforming processes, much as in the free cornering of a monoidal category. In general, a derivable sequent $\Gamma \Vdash t : (U, B, W)$ is understood as a process that transforms the resources indicated by Γ into those indicated by B by interacting with other such processes in the manner indicated by U and W. More precisely, neutral terms $[v]$ perform no interaction, and represent the same process that v does in $\mathcal{M}$. The term $\mathsf{putR}(v, t)$ represents the process in which we perform v on some inputs, send the result out along the right boundary, and proceed as in t. The term $\mathsf{getR}(x.t)$ represents the process in which we wait for some input along the right boundary, and proceed as in t once we have it. The left-facing terms are interpreted similarly. Finally, the term $\mathsf{let}\ x_1,..,x_n \downarrow (t_1 \mid \cdots \mid t_n)\ \mathsf{in}\ t'$ is the process in which we perform the processes indicated by the t_i and, once they are all finished, use the resulting values as the input to the process t'. Crucially, the processes indicated by the t_i may interact as they are performed by exchanging resources along their left and right boundaries via the get and put constructors.

This is similar to the way in which cells of the free cornering can be understood as interacting processes, raising the question of how the two formalisms relate. We answer this question by giving an interpretation $[\![-]\!]$ of terms in $T(\mathcal{M})$ as cells of the free cornering of the free monoidal category $F(\mathcal{M})$ on $\mathcal{M}$. The object monoid of $F(\mathcal{M})$ is the free monoid on $\mathcal{M}_0$, and a morphism of $F(\mathcal{M})(\Gamma_1,..,\Gamma_n; A_1,..,A_n)$ is a sequence $(f_1,..,f_n)$ of morphisms of $\mathcal{M}$ where $f_i \in \mathcal{M}(\Gamma_i; A_i)$ for each $i \in \{1,..,n\}$. Composition is defined in terms of composition in $\mathcal{M}$, which works because there is at most one way to compose suitably typed sequences $(f_1,..,f_n)$ and $(g_1,..,g_m)$. Identities are given by sequences of identity morphisms, and the monoidal structure is given by concatenation of sequences with the empty sequence serving as the unit. For more details on the adjunction relating multicategories and strict monoidal categories see e.g., [11].

Specifically, we define:

Definition 3.3 Let $x_1 : A_1,..,x_n : A_n \Vdash t : (U, B, W)$ be a term of $T(\mathcal{M})$. Then the cell $[\![t]\!] : ({}_U{}^{A_1,..,A_n}_{\ B} W)$

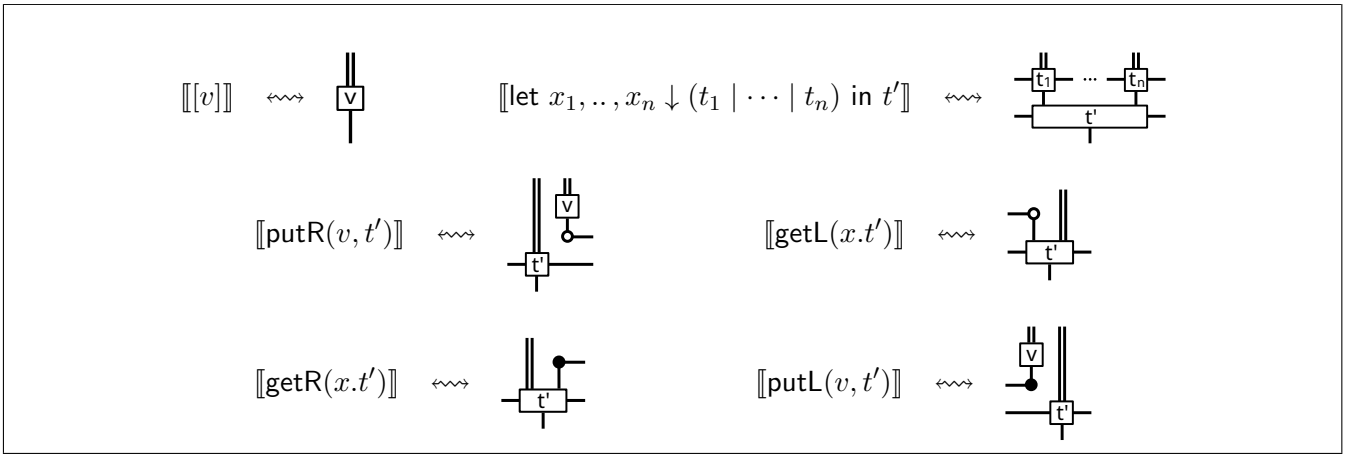


Fig. 2. Interpretation of terms in $T(\mathcal{M})$ as cells of $\ulcorner F(\mathcal{M})\urcorner$.

of $\ulcorner F(\mathcal{M})\urcorner$ is defined as in:

$$[\![t]\!] = \begin{cases} \ulcorner (v) \urcorner & \text{if } t = [v] \\ ([\![t_1]\!] \mid \cdots \mid [\![t_m]\!]) \cdot [\![t']\!] & \text{if } t = \mathsf{let}\ y_1, .., y_m \downarrow (t_1 \mid \cdots \mid t_m)\ \mathsf{in}\ t' \\ (\ulcorner (v) \urcorner \cdot A_{\lrcorner}) \mid 1_\Delta) \cdot [\![t']\!] & \text{if } t = \mathsf{putL}(v, t') \\ (A^{\urcorner} \mid 1_\Gamma) \cdot [\![t']\!] & \text{if } t = \mathsf{getL}(x.t') \\ (1_\Gamma \mid A^{\ulcorner}) \cdot [\![t']\!] & \text{if } t = \mathsf{getR}(x.t') \\ (1_\Delta \mid (\ulcorner (v) \urcorner \cdot A_{\llcorner})) \cdot [\![t']\!] & \text{if } t = \mathsf{putR}(v, t') \end{cases}$$

This can, alternatively, be expressed in terms of string diagrams as in Figure 2. We find these diagrams to be a helpful tool in developing one's intuition about the term calculus presented herein.

Next, we define a substitution operation, called *value substitution*, on terms of $T(\mathcal{M})$. Value substitution is ultimately resolved in terms of composition in $\mathcal{M}$, via the admissible COMP rule in the associated sequent calculus (Section 3.1). It is convenient to define value substitution by manipulating derivations explicitly, from which perspective we are introducing an admissible inference rule in the sequent calculus presenting $T(\mathcal{M})$. The admissible rule is:

$$\frac{(\Gamma_i \vdash v_i : A_i)_{i=1}^n \qquad x_1{:}A_1,..,x_n{:}A_n \Vdash t : (U, B, W)}{\Gamma_1,..,\Gamma_n \Vdash t[v_1, .., v_n/x_1, .., x_n] : (U, B, W)}\ \textsc{vsub}$$

To show that VSUB is admissible is equivalently to define the operation of value substitution. We do so by induction on the form of t. The cases are as follows: If $t = [u]$, we define:

$$\frac{(\Gamma_i \vdash v_i : A_i)_{i=1}^n \qquad \dfrac{x_1{:}A_1,..,x_n{:}A_n \vdash u : B}{x_1{:}A_1,..,x_n{:}A_n \Vdash [u] : (\lambda, B, \lambda)}\ \textsc{seq}}{\Gamma_1,..,\Gamma_n \Vdash [u][v_1,..,v_n/x_1,..,x_n] : (\lambda, B, \lambda)}\ \textsc{vsub}$$

$$=$$

$$\frac{\dfrac{(\Gamma_i \vdash v_i : A_i)_{i=1}^n \qquad x_1{:}A_1,..,x_n{:}A_n \vdash u : B}{\Gamma_1,..,\Gamma_n \vdash u[v_1,..,v_n/x_1,..,x_n] : B}\ \textsc{comp}}{\Gamma_1,..,\Gamma_n \Vdash [u[v_1,..,v_n/x_1,..,x_n]] : (\lambda, B, \lambda)}\ \textsc{seq}$$

If $t = \mathsf{let}\ y_1,..,y_m \downarrow (t_1 \mid \cdots \mid t_m)\ \mathsf{in}\ r$, we define:

$$\dfrac{((\Gamma^i_j \vdash v^i_j : A^i_j)^{k_j}_{i=1})^m_{j=1} \qquad \dfrac{(x^1_j{:}A^1_j,..,x^{k_j}_j{:}A^{k_j}_j \Vdash t_j : (U_{j-1},B_j,U_j))^m_{j=1} \qquad y_1{:}B_1,..,y_m{:}B_m \Vdash r : (U,B,W)}{x^1_1{:}B^1_1,..,x^{k_m}_m{:}B^{k_m}_m \Vdash \mathsf{let}\ y_1,..,y_m \downarrow (t_1 \mid \cdots \mid t_m)\ \mathsf{in}\ r : (U_0U,B,U_mW)}\ \textsc{let}}{\Gamma^1_1,..,\Gamma^{k_m}_m \Vdash (\mathsf{let}\ y_1,..,y_m \downarrow (t_1 \mid \cdots \mid t_m)\ \mathsf{in}\ r)[v^1_1,..,v^{k_m}_m/x^1_1,..,x^{k_m}_m] : (U_0U,B,U_mW)}\ \textsc{vsub}$$

$$=$$

$$\dfrac{\left(\dfrac{(\Gamma^i_j \vdash v^i_j)^{k_j}_{i=1} \qquad x^1_j{:}A^1_j,..,x^{k_j}_j{:}A^{k_j}_j \Vdash t_j : (U_{j-1},B_j,U_j)}{\Gamma^1_j,..,\Gamma^{k_j}_j \Vdash t_j[v^1_j,..,v^{k_j}_j/x^1_j,..,x^{k_j}_j] : (U_{j-1},B_j,U_j)}\textsc{vsub}\right)^m_{j=1} \qquad y_1{:}B_1,..,y_m{:}B_m \Vdash r : (U,B,W)}{\Gamma^1_1,..,\Gamma^{k_m}_m \Vdash \mathsf{let}\ y_1,..,y_m \downarrow (t_1[v^1_1,..,v^{k_1}_1/x^1_1,..,x^{k_1}_1] \mid \cdots \mid t_m[v^1_m,..,v^{k_m}_m/x^1_m,..,x^{k_m}_m])\ \mathsf{in}\ r : (U_0U,B,U_mW)}\ \textsc{let}$$

If $t = \mathsf{putR}(v,s)$ we define:

$$\dfrac{(\Gamma_i \vdash v_i : B_i)^n_{i=1} \qquad (\Delta_j \vdash u_j : A_j)^m_{j=1} \qquad \dfrac{x_1{:}B_1,..,x_n{:}B_n \Vdash s : (U,B,W) \qquad y_1{:}A_1,..,y_m{:}A_m \vdash v : A}{x_1{:}B_1,..,x_n{:}B_n, y_1{:}A_1,..,y_m{:}A_m \Vdash \mathsf{putR}(v,s) : (U,B,A^\circ W)}\ \textsc{putr}}{\Gamma_1,..,\Gamma_n,\Delta_1,..,\Delta_m \Vdash \mathsf{putR}(v,s)[v_1,..,v_n,u_1,..,u_m/x_1,..,x_n,y_1,..,y_m] : (U,B,A^\circ W)}\ \textsc{vsub}$$

$$=$$

$$\dfrac{\dfrac{(\Gamma_i \vdash v_i : B_i)^n_{i=1} \qquad x_1{:}B_1,..,x_n{:}B_n \Vdash s : (U,B,W)}{\Gamma_1,..,\Gamma_n \Vdash s[v_1,..,v_n/x_1,..,x_n] : (U,B,W)}\ \textsc{vsub} \qquad \dfrac{(\Delta_j \vdash u_j : A_j)^m_{j=1} \qquad y_1{:}A_1,..,y_m{:}A_m \vdash v : A}{\Delta_1,..,\Delta_m \vdash v[u_1,..,u_m/y_1,..,y_m] : A}\ \textsc{comp}}{\Gamma_1,..,\Gamma_n,\Delta_1,..,\Delta_m \Vdash \mathsf{putR}(v[u_1,..,u_n/y_1,..,y_n], s[v_1,..,v_m/x_1,..,x_m]) : (U,B,A^\circ W)}\ \textsc{putr}$$

The case for $t = \mathsf{putL}(v,s)$ is similar. If $t = \mathsf{getR}(x.s)$ we define:

$$\dfrac{(\Gamma_i \vdash v_i : A_i)^n_{i=1} \qquad \dfrac{x_1{:}A_1,..,x_n{:}A_n, x{:}A \Vdash s : (U,B,W)}{x_1{:}A_1,..,x_n{:}A_n \Vdash \mathsf{getR}(x.s) : (U,B,A^\bullet W)}\ \textsc{getr}}{\Gamma_1,..,\Gamma_n \Vdash \mathsf{getR}(x.s)[v_1,..,v_n/x_1,..,x_n] : (U,B,A^\bullet W)}\ \textsc{vsub}$$

$$=$$

$$\dfrac{\dfrac{(\Gamma_i \vdash v_i : A_i)^n_{i=1} \qquad \overline{x : A \vdash x : A}\ \textsc{id} \qquad x_1{:}A_1,..,x_n{:}A_n, x{:}A \Vdash s : (U,B,W)}{\Gamma_1,..,\Gamma_n, x{:}A \Vdash s[v_1,..,v_n,x/x_1,..,x_n,x] : (U,B,W)}\ \textsc{vsub}}{\Gamma_1,..,\Gamma_n \Vdash \mathsf{getR}(x.s[v_1,..,v_n,x/x_1,..,x_n,x]) : (U,B,A^\bullet W)}\ \textsc{getr}$$

The case for $t = \mathsf{getL}(x.s)$ is similar, and it follows that VSUB is admissible.

We will sometimes omit variables that are to be substituted for themselves from the substitution list, so that for example $t[x_1,..,v,..,x_n/x_1,..,x,..,x_n] = t[v/x]$ and $t[v_1,..,v_n,x/x_1,..,x_n,x] = t[v_1,..,v_n/x_1,..,x_n]$. We may also abbreviate lists as in $\overline{v} = v_1,..,v_n$ when the meaning of $\overline{v}$ is clear in context, so that for example we may write $t[\overline{v}/\overline{x}] = t[v_1,..,v_n/x_1,..,x_n]$. Then a shorter definition of value substitution is:

$$t[\overline{v}/\overline{x}] = \begin{cases} [u[\overline{v}/\overline{x}]] & \text{if } t = [u] \\ \mathsf{putR}(u[\overline{v_2}/\overline{x_2}], s[\overline{v_1}/\overline{x_1}]) & \text{if } t = \mathsf{putR}(u,s) \text{ and } [\overline{v}/\overline{x}] = [\overline{v_1},\overline{v_2}/\overline{x_1},\overline{x_2}] \\ \mathsf{getL}(y.s[\overline{v}/\overline{x}]) & \text{if } t = \mathsf{getL}(y.s) \\ \mathsf{putL}(u[\overline{v_1}/\overline{x_1}], s[\overline{v_2}/\overline{x_2}]) & \text{if } t = \mathsf{putR}(u,x) \text{ and } [\overline{v}/\overline{x}] = [\overline{v_1},\overline{v_2}/\overline{x_1},\overline{x_2}] \\ \mathsf{getR}(y.s[\overline{v}/\overline{x}]) & \text{if } t = \mathsf{getR}(y.s) \\ \mathsf{let}\ y_1,..,y_m \downarrow (t_1[\overline{v_1}/\overline{x_1}] \mid \cdots \mid t_n[\overline{v_n}/\overline{x_n}])\ \mathsf{in}\ s & \text{if } t = \mathsf{let}\ y_1,..,y_m \downarrow (t_1 \mid \cdots \mid t_n)\ \mathsf{in}\ s \\ & \text{and } [\overline{v}/\overline{x}] = [\overline{v_1},..,\overline{v_n}/\overline{x_1},..,\overline{x_n}] \end{cases}$$

There will only ever be one well-defined way to split up the variables $\overline{x}$ in the cases that require it, as can be seen in the more explicit definition of value substitution in terms of derivations.

[R0] $\mathsf{let}\ x_1,..,x_n \downarrow ([v_1] \mid \cdots \mid [v_n])\ \mathsf{in}\ t \to t[v_1,..,v_n/x_1,..,x_n]$

[R1] $\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{putR}(v,t) \mid \mathsf{getL}(y.s) \mid \overline{b})\ \mathsf{in}\ r \to \mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid t \mid s[v/y] \mid \overline{b})\ \mathsf{in}\ r$

[R2] $\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{getR}(y.t) \mid \mathsf{putL}(v,s) \mid \overline{b})\ \mathsf{in}\ r \to \mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid t[v/y] \mid s \mid \overline{b})\ \mathsf{in}\ r$

[R3] $\mathsf{let}\ x_1,..,x_n \downarrow (\mathsf{putL}(v,t) \mid \overline{a})\ \mathsf{in}\ r \to \mathsf{putL}(v, \mathsf{let}\ x_1,..,x_n \downarrow (t \mid \overline{a})\ \mathsf{in}\ r)$

[R4] $\mathsf{let}\ x_1,..,x_n \downarrow (\mathsf{getL}(y.t) \mid \overline{a})\ \mathsf{in}\ r \to \mathsf{getL}(y.\mathsf{let}\ x_1,..,x_n \downarrow (t \mid \overline{a})\ \mathsf{in}\ r)$

[R5] $\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{putR}(v,t))\ \mathsf{in}\ r \to \mathsf{putR}(v, \mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid t)\ \mathsf{in}\ r)$

[R6] $\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{getR}(y.t))\ \mathsf{in}\ r \to \mathsf{getR}(y.\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid t)\ \mathsf{in}\ r)$

[R7] $\mathsf{putR}(v, \mathsf{putL}(w,t)) \to \mathsf{putL}(w, \mathsf{putR}(v,t))$

[R8] $\mathsf{putR}(v, \mathsf{getL}(x.t)) \to \mathsf{getL}(x.\mathsf{putR}(v,t))$ when x does not occur in v

[R9] $\mathsf{getR}(x.\mathsf{putL}(v,t)) \to \mathsf{putL}(v, \mathsf{getR}(x.t))$ when x does not occur in v

[R10] $\mathsf{getR}(x.\mathsf{getL}(y.t)) \to \mathsf{getL}(y.\mathsf{getR}(x.t))$

Fig. 3. Generating rewrites for $\to$.

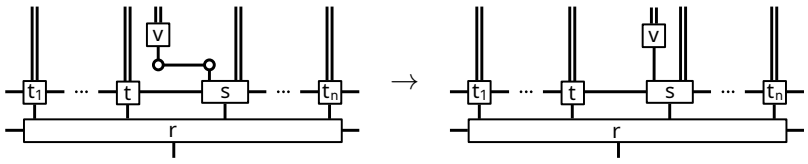
3.3 Rewrite Rules

We proceed to give a term rewrite relation on $T(\mathcal{M})$.

Definition 3.4 Let $\to$ be the least congruence on terms of $T(\mathcal{M})$ generated by the rewrites of Figure 3.

Adopting the usual notation for term rewriting relations, we write $\xrightarrow{*}$ for the reflexive transitive closure of $\to$, write $\overset{*}{\leftrightarrow}$ for the symmetric closure of $\xrightarrow{*}$, and write $\leftarrow$ and $\xleftarrow{*}$ for the converse of $\to$ and reflexive transitive closure of $\leftarrow$, respectively. Intuitively, rule **[R0]** says that when all of the inputs to a let-binding are neutral we may substitute them into the term in which they are bound via our value substitution operation VSUB. Rules **[R1]** and **[R2]** say that when matching left-facing and right-facing terms occur next to each other in a let-binding they may interact, and it is through these two rules that processes interact in our calculus. Rules **[R3-R6]** allow "outward-facing" gets and puts in the input sequence of a let-binding to be moved "above" the let-binding, and rules **[R7-R10]** allow us to commute left-facing gets and puts past right-facing ones. It may be helpful to conceptually separate **[R0-R6]**, which perform useful computational work, from **[R7-R10]**, which serve mainly to ensure that $\to$ is confluent. We note that the restriction on the use of **[R8]** and **[R9]** is not strictly necessary: if the reduct is well-formed then typing constraints ensure that x cannot occur in v anyway.

We note that rules **[R1]** and **[R2]** become instances of the yanking equations (Definition 2.2) under our interpretation (Definition 3.3). For example, **[R1]** becomes:



Together, the terms of Definition 3.1 and rewrites of Definition 3.4 constitute a *calculus*. We give a few simple examples demonstrating the way in which this calculus models process interaction:

Example 3.5 Say $\mathcal{M}$ has objects including **Pants**, **Shirt**, **Clothes**, **Pattern** and **Thread**, and has morphisms including:

$$\mathbf{cut} \in \mathcal{M}(\mathbf{Fabric}; \mathbf{Pattern}) \qquad \mathbf{sew} \in \mathcal{M}(\mathbf{Pattern}, \mathbf{Thread}; \mathbf{Shirt})$$

$$\mathbf{pack} \in \mathcal{M}(\mathbf{Pants}, \mathbf{Shirt}; \mathbf{Clothes})$$

Then let A be the derivation:

$$\dfrac{\dfrac{\dfrac{}{p : \mathbf{Pants} \vdash p : \mathbf{Pants}}\,\text{VAR}}{p : \mathbf{Pants} \Vdash [p] : (\lambda, \mathbf{Pants}, \lambda)}\,\text{SEQ} \qquad \dfrac{f : \mathbf{Fabric} \vdash \mathbf{cut}(f) : \mathbf{Pattern}}{f : \mathbf{Fabric} \Vdash [\mathbf{cut}(f)] : (\lambda, \mathbf{Pattern}, \lambda)}\,\text{SEQ}}{p : \mathbf{Pants}, f : \mathbf{Fabric} \Vdash \mathsf{putR}(\mathbf{cut}(f), [p]) : (\lambda, \mathbf{Pants}, \mathbf{Pattern}^\circ)}\,\text{PUTR}$$

and let B be the derivation:

$$\dfrac{\dfrac{a : \mathbf{Pattern}, t : \mathbf{Thread} \vdash \mathbf{sew}(a, t) : \mathbf{Shirt}}{a : \mathbf{Pattern}, t : \mathbf{Thread} \Vdash [\mathbf{sew}(a, t)] : (\lambda, \mathbf{Shirt}, \lambda)}\,\text{SEQ}}{t : \mathbf{Thread} \Vdash \mathsf{getL}(a.[\mathbf{sew}(a, t)]) : (\mathbf{Pattern}^\circ, \mathbf{Shirt}, \lambda)}\,\text{GETL}$$

Then we may derive:

$$\dfrac{A \qquad B \qquad \dfrac{x : \mathbf{Pants}, y : \mathbf{Shirt} \vdash \mathbf{pack}(x, y) : \mathbf{Clothes}}{x : \mathbf{Pants}, y : \mathbf{Shirt} \Vdash [\mathbf{pack}(x, y)] : (\lambda, \mathbf{Clothes}, \lambda)}\,\text{SEQ}}{p : \mathbf{Pants}, f : \mathbf{Fabric}, t : \mathbf{Thread} \Vdash \mathsf{let}\ x, y \downarrow (\mathsf{putR}(\mathbf{cut}(f), [p]) \mid \mathsf{getL}(a.[\mathbf{sew}(a, t)]))\ \mathsf{in}\ [\mathbf{pack}(x, y)] : (\lambda, \mathbf{Clothes}, \lambda)}\,\text{LET}$$

Which rewrites to normal form as in:

$$\begin{aligned}
&\mathsf{let}\ x, y \downarrow (\mathsf{putR}(\mathbf{cut}(f), [p]) \mid \mathsf{getL}(a.[\mathbf{sew}(a, t)]))\ \mathsf{in}\ [\mathbf{pack}(x, y)] \\
&\to \mathsf{let}\ x, y \downarrow ([p] \mid [\mathbf{sew}(a, t)][\mathbf{cut}(f)/a])\ \mathsf{in}\ [\mathbf{pack}(x, y)] \\
&= \mathsf{let}\ x, y \downarrow ([p] \mid [\mathbf{sew}(\mathbf{cut}(f), t)])\ \mathsf{in}\ [\mathbf{pack}(x, y)] \\
&\to [\mathbf{pack}(x, y)][p, \mathbf{sew}(\mathbf{cut}(f), t)/x, y] \\
&= [\mathbf{pack}(p, \mathbf{sew}(\mathbf{cut}(f), t))]
\end{aligned}$$

In this way, subterms of a message passing term may exchange resources during evaluation.

In the second example, we show how a get can be moved "above" a let.

Example 3.6 Say $\mathcal{M}$ has objects including **Person**, **Money**, **Beans**, **Water**, **Coffee**, and **Ready**, and has morphisms including:

$$\mathbf{brew} \in \mathcal{M}(\mathbf{Money}, \mathbf{Beans}, \mathbf{Water}; \mathbf{Coffee}) \qquad \mathbf{drink} \in \mathcal{M}(\mathbf{Person}, \mathbf{Coffee}; \mathbf{Person})$$

$$\mathbf{done} \in \mathcal{M}(; \mathbf{Ready})$$

Let A be the term modeling a person paying for a coffee at a coffee machine, and then drinking it:

$$p : \mathbf{Person}, m : \mathbf{Money} \Vdash \mathsf{putR}(m, \mathsf{getR}(c.[\mathbf{drink}(p, c)])) : (\lambda, \mathbf{Person}, \mathbf{Money}^\circ \mathbf{Coffee}^\bullet)$$

and let B be the term modeling the coffee machine, which gets water from the mains:

$$b : \mathbf{Beans} \Vdash \mathsf{getL}(n, \mathsf{getR}(w.\mathsf{putL}(\mathbf{brew}(n, b, w), \mathbf{done}()))) : (\mathbf{Money}^\circ \mathbf{Coffee}^\bullet, \mathbf{Ready}, \mathbf{Water}^\bullet)$$

Consider the term $\mathsf{let}\ x, y \downarrow (A \mid B)\ \mathsf{in}\ C$ for some well-typed continuation term C. Then we have the following reductions:

$$\begin{aligned}
&\mathsf{let}\ x, y \downarrow (A \mid B)\ \mathsf{in}\ C \\
&= \mathsf{let}\ x, y \downarrow (\mathsf{putR}(m, \mathsf{getR}(c.[\mathbf{drink}(p, c)])) \mid \mathsf{getL}(n, \mathsf{getR}(w.\mathsf{putL}(\mathbf{brew}(n, b, w), \mathbf{done}()))))\ \mathsf{in}\ C \\
&\to \mathsf{let}\ x, y \downarrow (\mathsf{getR}(c.[\mathbf{drink}(p, c)]) \mid \mathsf{getR}(w.\mathsf{putL}(\mathbf{brew}(m, b, w), \mathbf{done}())))\ \mathsf{in}\ C \\
&\to \mathsf{getR}(w.\mathsf{let}\ x, y \downarrow (\mathsf{getR}(c.[\mathbf{drink}(p, c)]) \mid \mathsf{putL}(\mathbf{brew}(m, b, w), \mathbf{done}()))\ \mathsf{in}\ C) \\
&\to \mathsf{getR}(w.\mathsf{let}\ x, y \downarrow ([\mathbf{drink}(p, \mathbf{brew}(m, b, w))] \mid \mathbf{done}())\ \mathsf{in}\ C) \\
&\to \mathsf{getR}(w.C[\mathbf{drink}(p, \mathbf{brew}(m, b, w)), \mathbf{done}()/x, y])
\end{aligned}$$

We show that our interpretation $[\![-]\!]$ of terms (Definition 3.3) is coherent with respect to the convertibility relation $\overset{*}{\leftrightarrow}$ induced by $\rightarrow$. It is convenient to do this in two parts. First, we have:

Lemma 3.7 $[\![\mathsf{let}\ x_1,..,x_n \downarrow ([v_1] \mid \cdots \mid [v_n])\ \mathsf{in}\ t]\!] = [\![t[v_1,..,v_n/x_1,..,x_n]]\!]$ *whenever this makes sense.*

And then using Lemma 3.7 it is straightforward to obtain:

Lemma 3.8 *If* $t \overset{*}{\leftrightarrow} t'$ *then* $[\![t]\!] = [\![t']\!]$.

Lemma 3.8 will be important later on, when we show that $[\![-]\!]$ gives a functor of virtual double categories. In particular, our terms $T(\mathcal{M})$ will form a virtual double category modulo $\overset{*}{\leftrightarrow}$, and Lemma 3.8 is required for $[\![-]\!]$ to be a well-defined function from $\overset{*}{\leftrightarrow}$-equivalence classes to cells of the free cornering.

3.4 Termination and Confluence

We proceed to show that $\rightarrow$ is terminating and confluent. Termination is straightforward: we assign each term a size and show that each of our generating rewrites decreases this size. The size is given as in:

Definition 3.9 We assign to each term t a *size* $\#(t) \in \mathbb{N}$ as follows:

$$\#(t) = \begin{cases} 2 & \text{if } t = [v] \\ 2 \cdot \#(t_1) \cdots \#(t_n) \cdot \#(s) & \text{if } t = \mathsf{let}\ x_1,..,x_n \downarrow (t_1 \mid \cdots \mid t_n)\ \mathsf{in}\ s \\ 2 + (2 \cdot \#(s)) & \text{if } t = \mathsf{putL}(v,s) \text{ or } t = \mathsf{getL}(x.s) \\ 1 + (2 \cdot \#(s)) & \text{if } t = \mathsf{putR}(v,s) \text{ or } t = \mathsf{getR}(x.s) \end{cases}$$

Next, we require an auxiliary lemma concerning value substitution:

Lemma 3.10 $\#(t) = \#(t[v_1,..,v_n/x_1,..,x_n])$

From here, we easily obtain that reduction decreases the size of the term involved:

Lemma 3.11 *If* $t \rightarrow t'$ *then* $\#(t) > \#(t')$.

We may now record, since $\rightarrow$ reduces the size and this cannot be done infinitely many times, that:

Corollary 3.12 *The rewrite relation* $\rightarrow$ *is terminating*

Our next goal will be to show that $\rightarrow$ is confluent. The following lemma is helpful:

Lemma 3.13 *Any let-binding reduces.*

We note that Lemma 3.13 may be viewed as a kind of cut-elimination theorem for the sequent calculus that presents $T(\mathcal{M})$ (Definition 3.1). From the perspective of our interpretation of the calculus in terms of interacting processes, we can think of this in terms of "deadlock freedom": no term in normal form may contain a let-binding, which means that any internally resolvable interaction in a term of $T(\mathcal{M})$ is in fact resolved by reducing the term in question to normal form.

Our proof of confluence relies on a bit of technical machinery concerning terms in context. We define:

Definition 3.14 For each $U, V, W \in {\mathcal{M}_0}^{\circ\bullet}$, each $B \in \mathcal{M}_0$, and each $\Gamma, \Delta \in \mathcal{M}_0^*$ the *term contexts*

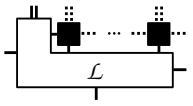
$$\mathcal{L} : \{U \underset{\Delta}{\blacksquare} \blacksquare\} \rightsquigarrow \{V \overset{\Gamma,}{\underset{B}{\blacksquare}} \blacksquare W\}$$

are constructed according to the following inference rules:

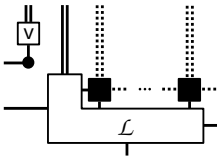
$$\frac{(\Gamma_i \Vdash a_i : (U_{i-1}, A_i, U_i))_{i=1}^k \qquad x_1{:}A_1,..,x_n{:}A_n \Vdash r : (V,B,W) \qquad k<n}{\mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_k \mid \blacksquare)\ \mathsf{in}\ r : \left\{ U_k \begin{smallmatrix} \blacksquare \\ A_{n-k},..,A_n \end{smallmatrix} \blacksquare \right\} \rightsquigarrow \left\{ U_0 V \begin{smallmatrix} \Gamma_1,..,\Gamma_k,\blacksquare \\ B \end{smallmatrix} \blacksquare W \right\}}$$

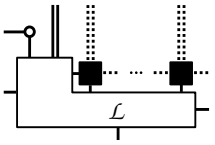
$$\frac{\Sigma \vdash_{\mathcal{M}} v : A \qquad \mathcal{L} : \left\{ U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare \right\} \rightsquigarrow \left\{ V \begin{smallmatrix} \Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W \right\}}{\mathsf{putL}(v,\mathcal{L}) : \left\{ U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare \right\} \rightsquigarrow \left\{ A^{\bullet} V \begin{smallmatrix} \Sigma,\Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W \right\}} \qquad \frac{\mathcal{L} : \left\{ U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare \right\} \rightsquigarrow \left\{ V \begin{smallmatrix} A,\Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W \right\}}{\mathsf{getL}(x.\mathcal{L}) : \left\{ U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare \right\} \rightsquigarrow \left\{ A^{\circ} V \begin{smallmatrix} \Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W \right\}}$$

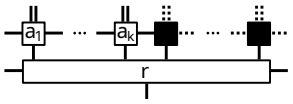
Contexts $\mathcal{L}$ may be pictured as in:

$\mathcal{L}$ ⟺ 

in which case the individual inference rules for constructing term contexts are pictured as in:

$\mathsf{putL}(v,\mathcal{L})$ ⟺ 

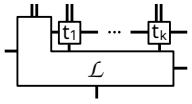
$\mathsf{getL}(x.\mathcal{L})$ ⟺ 

$\mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_k \mid \blacksquare)\ \mathsf{in}\ r$ ⟺ 

We obtain terms from term contexts by supplying a sequence of terms of the appropriate type. Specifically, if $\mathcal{L} : \{U \begin{smallmatrix} \blacksquare \\ A_1,..,A_k \end{smallmatrix} \blacksquare\} \rightsquigarrow \{V \begin{smallmatrix} \Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W\}$ is a term context, then whenever $(\Gamma_i \Vdash t_i : (U_{i-1}, A_i, U_i))_{i=1}^k$ we have $\Gamma, \Gamma_1,.., \Gamma_k \Vdash \mathcal{L}[t_1,..,t_k] : (V, B, U_k W)$ with $\mathcal{L}[-]$ defined as in:

$$\mathcal{L}[t_1,..,t_k] = \begin{cases} \mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_m \mid t_1 \mid \cdots \mid t_k)\ \mathsf{in}\ r & \text{if } \mathcal{L} = \mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_m \mid \blacksquare)\ \mathsf{in}\ r \\ \mathsf{putL}(v, \mathcal{L}'[t_1,..,t_k]) & \text{if } \mathcal{L} = \mathsf{putL}(v,\mathcal{L}') \\ \mathsf{getL}(x.\mathcal{L}'[t_1,..,t_k]) & \text{if } \mathcal{L} = \mathsf{getL}(x.\mathcal{L}') \end{cases}$$

Graphically:

$\mathcal{L}[t_1,..,t_k]$ ⟺ 

Write $s(\mathcal{L})$ to indicate the let-bound sequence of terms in $\mathcal{L}$. Explicitly:

$$s(\mathcal{L}) = \begin{cases} a_1,..,a_k & \text{if } \mathcal{L} = \mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_k \mid \blacksquare)\ \mathsf{in}\ r \\ s(\mathcal{L}') & \text{if } \mathcal{L} = \mathsf{putL}(v,\mathcal{L}') \text{ or } \mathcal{L} = \mathsf{getL}(x.\mathcal{L}') \end{cases}$$

Note that $s(\mathcal{L})$ may be empty. In case it is not, we call $\mathcal{L}$ *nonempty*. A nonempty term context $\mathcal{L} : \{U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare\} \rightsquigarrow \{V \begin{smallmatrix} \Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W\}$ is said to be *effectful* in case $U \neq I$. An effectful term context $\mathcal{L}$ is said to be *active* in case $s(\mathcal{L}) = a_1,..,a_k$ where a_k is right-facing. Note further that for any nonempty $\mathcal{L} : \{U \begin{smallmatrix} \blacksquare \\ \Delta \end{smallmatrix} \blacksquare\} \rightsquigarrow \{V \begin{smallmatrix} \Gamma,\blacksquare \\ B \end{smallmatrix} \blacksquare W\}$ with $s(\mathcal{L}) = a_1,..,a_k, a$ where $a : (U' \begin{smallmatrix} \Sigma \\ C \end{smallmatrix} U)$ there exists another term context $\mathcal{L}' : \{U' \begin{smallmatrix} \blacksquare \\ C,\Delta \end{smallmatrix} \blacksquare\} \rightsquigarrow$

$\{V_B^{\Gamma,\Sigma,\blacksquare}\blacksquare W\}$ such that $\mathcal{L}[t_1,..,t_n] = \mathcal{L}'[a,t_1,..,t_n]$. In particular this means that for any active $\mathcal{L}$ with $s(\mathcal{L}) = a_1,..,a_k, \mathsf{putR}(v,t)$ there exists $\mathcal{L}'$ such that $\mathcal{L}[t_1,..,t_n] = \mathcal{L}'[\mathsf{putR}(v,t),t_1,..,t_n]$, and similarly if $s(\mathcal{L}) = a_1,..,a_k, \mathsf{getR}(x.t)$ there exists $\mathcal{L}'$ such that $\mathcal{L}[t_1,..,t_n] = \mathcal{L}'[\mathsf{getR}(x.t),t_1,..,t_n]$. We say that $\mathcal{L} \to \mathcal{L}'$ in case for all suitable $t_1,..,t_k$ we have $\mathcal{L}[t_1,..,t_k] \to \mathcal{L}'[t_1,..,t_k]$. Moreover, we define $\#(\mathcal{L}) \in \mathbb{N}$ as follows:

$$\#(\mathcal{L}) = \begin{cases} 2 \cdot \prod_{i=1}^{n}(\#(a_i)) \cdot \#(r) & \text{if } \mathcal{L} = \mathsf{let}\ x_1,..,x_n \downarrow (a_1 \mid \cdots \mid a_k \mid \blacksquare)\ \mathsf{in}\ r \\ 2 + (2 \cdot \#(\mathcal{L}')) & \text{if } \mathcal{L} = \mathsf{putL}(v,s) \text{ or } \mathcal{L} = \mathsf{getL}(x.\mathcal{L}') \end{cases}$$

Note that if $\mathcal{L} \to \mathcal{L}'$ then $\#(\mathcal{L}) > \#(\mathcal{L}')$, via a straightforward extension of the proof of Lemma 3.12.

We may now state the main lemma concerning term contexts:

Lemma 3.15 *Let $\mathcal{L}$ be an effectful term context, then there is an active term context $\mathcal{L}'$ such that $\mathcal{L} \xrightarrow{*} \mathcal{L}'$.*

We require one further technical lemma:

Lemma 3.16 *Let $\mathcal{L}$ be a term context. Then we have:*

(i) $\mathcal{L}[b_1,..,b_n, \mathsf{putR}(v,t)]$ *and* $\mathsf{putR}(v, \mathcal{L}[b_1,..,b_n,t])$ *are joinable.*

(ii) $\mathcal{L}[b_1,..,b_n, \mathsf{getR}(x.t)]$ *and* $\mathsf{getR}(x.\mathcal{L}[b_1,..,b_n,t])$ *are joinable.*

whenever these expressions make sense. Here when we say two terms t and t' of $T(\mathcal{M})$ are joinable *we mean that there exists some h such that $t \xrightarrow{*} h \xleftarrow{*} t'$.*

We know that $\to$ is terminating, so to show that it is confluent it suffices to show that it is locally confluent. This is proven by analysis of the possible critical pairs, which are all joinable. The term contexts play an important technical role in joining certain critical pairs.

Lemma 3.17 *$\to$ is locally confluent.*

We briefly discuss the proof of Lemma 3.17. There are twelve critical pairs which need to be resolved, which fall into three general groups:

- In the first group, an internal interaction with **[R1,R2]** comes into critical conflict with a commutation of gets and puts with **[R7-R10]**. This is exemplified by the following term:

  $$\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{putR}(v, \mathsf{putL}(u,t)) \mid \mathsf{getL}(x.s) \mid \overline{b})\ \mathsf{in}\ r$$

  to which we can apply **R1** and **R7**. This pair cannot be directly resolved, since there is a $\mathsf{putL}(u,-)$ in the way. To join this pair, consider the term context $\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \blacksquare)\ \mathsf{in}\ r$ and reduce this to an *active* term context. Such an active term context has an appropriate $\mathsf{getR}(-.-)$ in the right place to resolve the interfering $\mathsf{putL}(u,-)$ with an internal interaction **R2**. Once resolved, the pair can be joined.
- In the second type, we can move a get or put above a let term, both on the left with **[R3,R4]** and the right with **[R5,R6]**. This is exemplified by the following term:

  $$\mathsf{let}\ x_1,..,x_n \downarrow (\mathsf{putL}(v,t) \mid \overline{a} \mid \mathsf{putR}(u,s))\ \mathsf{in}\ r$$

  to which we can apply rule **R3** and **R5**. This pair can be resolved by commuting the puts using **R7**.
- In the third group, we can move a get or put above a let with **[R5,R6]**, but this comes in critical conflict with the commuting of gets and puts with **[R7-R10]**. This is exemplified by the term:

  $$\mathsf{let}\ x_1,..,x_n \downarrow (\overline{a} \mid \mathsf{putR}(v, \mathsf{putL}(u,t)))\ \mathsf{in}\ r$$

  to which we can apply rule **R5** and **R7**. Same as in the first group, we use term contexts to reduce the term to such a state in which we can resolve the $\mathsf{putL}(u,-)$ and join the terms.

Note that the rules **[R7-R10]** are necessary to resolve a critical pair group arising from rules **[R0-R6]**. The rules **[R7-R10]** themselves give rise to further critical pairs, which are resolved using term contexts.

Now, from Lemma 3.12 and Lemma 3.17 we have:

Theorem 3.18 *$\to$ is confluent.*

The reader may have noticed that our rewrite relation $\to$ and term contexts $\mathcal{L}$ are "left-biased". This is in service of confluence: we must pick which of the left-facing and right-facing puts gets and puts are to be resolved first when both are available. Of course, we could have made the opposite choice and chosen to work with a "right-biased" rewrite relation, in which case the appropriate analogue of term contexts is similarly dual. This also yields a confluent and terminating rewrite relation. While it does not really matter which system we choose, we must choose one, and have here chosen the left-biased version.

4 A Virtual Double Category of Terms

In this section we show that the terms of our calculus form a virtual double category when considered modulo the convertibility relation $\overset{*}{\leftrightarrow}$ induced by our term rewrite relation $\to$. This done, we show that our interpretation $[\![-]\!]$ of terms as cells of the free cornering defines a functor of virtual double categories. Our fixed multicategory $\mathcal{M}$ from Section 3 remains fixed in this section.

The data of our virtual double category of terms is as follows:

Definition 4.1 Define $[\mathcal{M}]$ to be the virtual double category that has one object, $*$, has horizontal morphisms $A : * \to *$ given by objects A of $\mathcal{M}$, and with the category of vertical morphisms given by the monoid $\mathcal{M}_0^{\circ\bullet}$, viewed as a category with one object. Cells $a : [\mathcal{M}](U \begin{smallmatrix} A_1,..,A_n \\ B \end{smallmatrix} W)$ are $\overset{*}{\leftrightarrow}$-equivalence classes of derivations of the form $x_1{:}A_1,..,x_n{:}A_n \Vdash a : (U, B, W)$ (i.e., terms of $T(\mathcal{M})$). Identity cells $1_A : [\mathcal{M}](I \begin{smallmatrix} A \\ A \end{smallmatrix} I)$ are given by $x : A \Vdash [x] : (\lambda, A, \lambda)$. Composition is defined by:

$$f \circ (g_1, .. , g_n) = \begin{cases} id_U(f) & \text{if } (g_1, .. , g_n) = (\,)_U \\ \mathsf{let}\ x_1, .. , x_n \downarrow (g_1 \mid \cdots \mid g_n)\ \mathsf{in}\ f & \text{otherwise} \end{cases}$$

where for $\Vdash f : (V, B, W)$ we define $\Vdash id_U(f) : (UV, B, UW)$ by induction on U as in:

$$id_U(f) = \begin{cases} f & \text{if } U = \lambda \\ \mathsf{getL}(x.\mathsf{putR}(x.id_{U'}(f))) & \text{if } U = A{\circ}U' \\ \mathsf{getR}(x.\mathsf{putL}(x.id_{U'}(f))) & \text{if } U = A^{\bullet}U' \end{cases}$$

We must show that the data of $[\mathcal{M}]$ satisfies the axioms of a virtual double category. To do this, we require a number of technical lemmas concerning the behaviour of cells of the form $id_U(t)$. These show that $id_U(t)$ behaves like a "horizontal identity" when considered modulo our rewrite relation. First, we have:

Lemma 4.2 *For all $U \in \mathcal{M}_0^{\circ\bullet}$, we have:*

$$\mathit{let}\ x_1,..,x_n \downarrow (id_U(t_1) \mid \cdots \mid id_U(t_n))\ \mathit{in}\ t \overset{*}{\leftrightarrow} id_U(\mathit{let}\ x_1,..,x_n \downarrow (t_1 \mid \cdots \mid t_n)\ \mathit{in}\ t)$$

for all terms $t, t_1, .. , t_n$ for which this makes sense.

Second, we have:

Lemma 4.3 *We have:*

(i) $f \circ (.., \mathit{getR}(x.t), id_{A^{\bullet}U}(h_1),.., id_{A^{\bullet}U}(h_k), \mathit{putL}(v, s),..) \overset{*}{\to} f \circ (.., t[v/x], id_U(h_1),.., id_U(h_k), s,..)$

(ii) $f \circ (.., \mathit{putR}(v, t), id_{A^{\circ}U}(h_1),.., id_{A^{\circ}U}(h_k), \mathit{getL}(x.s),..) \overset{*}{\to} f \circ (.., t, id_U(h_1),.., id_U(h_k), s[v/x],..)$

whenever these expressions make sense.

Third and finally, we have:

Lemma 4.4 *We have:*

(i) $f \circ (id_{A^{\bullet}U}(h_1),.., id_{A^{\bullet}U}(h_k), \mathsf{putL}(v, t), g_1,.., g_n) \overset{*}{\to} \mathsf{putL}(v, f \circ (id_U(h_1),.., id_U(h_k), t, g_1,.., g_n))$

(ii) $f \circ (id_{A \circ U}(h_1),..,id_{A \circ U}(h_k), \mathsf{getL}(x.t), g_1,.., g_n) \xrightarrow{*} \mathsf{getL}(x.f \circ (id_U(h_1),..,id_U(h_k), t, g_1,.., g_n))$

(iii) $f \circ (g_1,.., g_n, \mathsf{putR}(v, t), id_{A \circ U}(h_1),.., id_{A \circ U}(h_k)) \xrightarrow{*} \mathsf{putR}(v, f \circ (g_1,.., g_n, t, id_U(h_1),.., id_U(h_k)))$

(iv) $f \circ (g_1,.., g_n, \mathsf{getR}(x.t), id_{A \bullet U}(h_1),.., id_{A \bullet U}(h_k)) \xrightarrow{*} \mathsf{getR}(x.f \circ (g_1,.., g_n, t, id_U(h_1),.., id_U(h_k)))$

whenever these expressions make sense.

Technical lemmas in hand, we proceed to show that composition in $[\mathcal{M}]$ is associative:

Lemma 4.5 *We have:*

$$f \circ (g_1 \circ (h_1^1,.., h_1^{m_1}),.., g_n \circ (h_n^1,.., h_n^{m_n})) \overset{*}{\leftrightarrow} (f \circ (g_1,.., g_n)) \circ (h_1^1,.., h_1^{m_1},.., h_n^1,.., h_n^{m_n})$$

whenever these expressions make sense.

This is shown by simultaneously reducing the term on the left and the right, dealing with any $g_i \circ () = id_U(g)$ using the technical lemmas.

It is easy to see that $[\mathcal{M}]$ satisfies the unitality axioms, and so we have:

Theorem 4.6 $[\mathcal{M}]$ *is a virtual double category.*

We proceed to show that our interpretation of terms of $T(\mathcal{M})$ as cells of $\ulcorner F(\mathcal{M}) \urcorner$ (Definition 3.3) defines a functor of virtual double categories from $[\mathcal{M}]$ into the underlying virtual double category of $\ulcorner F(\mathcal{M}) \urcorner$. If $\mathbb{X}$ is a single-object double category with horizontal edge monoid $(\mathbb{X}_H, \otimes, I)$ then the virtual double category $U(\mathbb{X})$ has cells $a : (U \overset{A_1,..,A_n}{B} W)$ given by cells $a : (U \overset{A_1 \otimes \cdots \otimes A_n}{B} W)$ of $\mathbb{X}$, with composition defined in terms of vertical and horizontal composition in $\mathbb{X}$ as in $f \circ (g_1,.., g_n) = (g_1 \mid \cdots \mid g_n) \cdot f$, and with identities given by vertical identities in $\mathbb{X}$ (see e.g., [11]).

We require one final technical lemma:

Lemma 4.7 *Let $U, W \in M_0^{\circ\bullet}$, and let t be a term. Then we have:*

(i) $[\![id_U(t)]\!] = id_U \cdot [\![t]\!]$

(ii) $id_U(id_W(t)) = id_{UW}(t)$

Finally, we have:

Theorem 4.8 *The interpretation $[\![-]\!]$ of terms defines a functor of virtual double categories:*

$$[\![-]\!] : [\mathcal{M}] \to U\left(\ulcorner F(\mathcal{M}) \urcorner\right)$$

5 Concluding Remarks

We have introduced a calculus of interacting processes, consisting of a collection of terms together with a term rewrite relation. The calculus is parameterised by an arbitrary multicategory of non-interacting processes. We have shown that the calculus is confluent and terminating, and that terms of the calculus form a virtual double category when considered modulo the induced convertibility relation.

We imagine two primary directions for future work. First, we believe, but cannot yet prove, that the functor of virtual double categories given in Theorem 4.8 is faithful. If we think of the rewrite relation of our calculus in terms of operational semantics, then this functor gives a sound denotational semantics of our notion of process interaction. For the interpretation functor to be faithful is for our denotational semantics to be *adequate* (see e.g., [5]). Adequacy is, essentially, the property that the denotational and operational semantics coincide in an appropriate sense. An adequate denotational semantics is much more useful than one that is merely sound, and as such we would like to be able to prove that the functor in question is faithful.

Second, process interaction in our calculus is governed by what are essentially session types. Viewed from this perspective, the calculus is missing certain features. For example, it is unable to express branching protocols. The free cornering construction has been extended with the ability to express

branching protocols and protocol iteration [16], and we imagine that the ideas developed in that setting could be applied to our calculus in order to increase its expressiveness. We hope that a more expressive version of the calculus presented here could be used to reason about e.g., cryptographic protocols.

References

[1] Baader, F. and T. Nipkow, *Term rewriting and all that*, Cambridge university press (1998).

[2] Bellin, G. and P. Scott, *On the pi-calculus and linear logic*, Theoretical Computer Science **135**, pages 11–65 (1994), ISSN 0304-3975.
https://doi.org/https://doi.org/10.1016/0304-3975(94)00104-9

[3] Boisseau, G., C. Nester and M. Román, *Cornering optics*, in: J. Master and M. Lewis, editors, Proceedings Fifth International Conference on *Applied Category Theory,* Glasgow, United Kingdom, 18-22 July 2022, volume 380 of *Electronic Proceedings in Theoretical Computer Science*, pages 97–110, Open Publishing Association (2023).
https://doi.org/10.4204/EPTCS.380.6

[4] Caires, L. and F. Pfenning, *Session types as intuitionistic linear propositions*, in: *International Conference on Concurrency Theory*, pages 222–236, Springer (2010).

[5] Cardone, F., *Games, Full Abstraction and Full Completeness*, in: E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*, Metaphysics Research Lab, Stanford University, Spring 2021 edition (2021).

[6] Cockett, J. and C. Pastro, *The Logic of Message-Passing*, Science of Computer Programming **74**, pages 498–533 (2009).
https://doi.org/10.1016/j.scico.2007.11.005

[7] Coecke, B., T. Fritz and R. Spekkens, *A Mathematical Theory of Resources*, Information and Computation **250**, pages 59–86 (2016).
https://doi.org/10.1016/j.ic.2016.02.008

[8] Cruttwell, G. S. and M. A. Shulman, *A unified framework for generalized multicategories*, Theory and Applications of Categories **24**, pages 580–655 (2010).

[9] Honda, K., *Types for dyadic interaction*, in: *CONCUR'93: 4th Intrenational Conference on Concurrency Theory Hildesheim, Germany, August 23–26, 1993 Proceedings 4*, pages 509–523, Springer (1993).

[10] Lambek, J. and P. Scott, *Introduction to Higher-Order Categorical Logic*, Cambridge University Press (1986).

[11] Leinster, T., *Higher Operads, Higher Categories*, Cambridge University Press (2004). Preprint version available at arXiv:math.CT/0305049.

[12] Mac Lane, S., *Categories for the Working Mathematician*, Springer (1971).
https://doi.org/10.1007/978-1-4757-4721-8

[13] Nester, C., *The Structure of Concurrent Process Histories*, in: *International Conference on Coordination Languages and Models*, pages 209–224, Springer (2021).
https://doi.org/10.1007/978-3-030-78142-2_13

[14] Nester, C., *Situated transition systems*, in: K. Kishida, editor, Proceedings of the Fourth International Conference on *Applied Category Theory,* Cambridge, United Kingdom, 12-16th July 2021, volume 372 of *Electronic Proceedings in Theoretical Computer Science*, pages 103–115, Open Publishing Association (2022).
https://doi.org/10.4204/EPTCS.372.8

[15] Nester, C., *Concurrent Process Histories and Resource Transducers*, Logical Methods in Computer Science **Volume 19, Issue 1** (2023).
https://doi.org/10.46298/lmcs-19(1:7)2023

[16] Nester, C. and N. Voorneveld, *Protocol Choice and Iteration for the Free Cornering*, Journal of Logical and Algebraic Methods in Programming **137**, page 100942 (2024), ISSN 2352-2208.
https://doi.org/https://doi.org/10.1016/j.jlamp.2023.100942

[17] Shulman, M., *Categorical logic from a categorical point of view*, Technical report, AARMS Summer School (2016).
https://mikeshulman.github.io/catlog/catlog.pdf

[18] Wadler, P., *Propositions as Sessions*, Journal of Functional Programming **24**, pages 384–418 (2014).
https://doi.org/10.1145/2364527.2364568