Cartesian Closed Double Categories, their Lambda-Notation, and the Pi-Calculus

Roberto Bruni and Ugo Montanari
Computer Science Department
University of Pisa
Corso Italia 40, I-56125 Pisa, ITALY
{bruni,ugo}@di.unipi.it

Abstract

We introduce the notion of cartesian closed double category *to provide mobile calculi for communicating systems with specific semantic models: One dimension is dedicated to compose systems and the other to compose their computations and their observations. Also, inspired by the connection between simply typed λ-calculus and cartesian closed categories, we define a new typed framework, called* double λ-notation*, which is able to express the abstraction/application and pairing/projection operations in all dimensions. In this development, we take the categorical presentation as a guidance in the interpretation of the formalism. A case study of the π-calculus, where the double λ-notation straightforwardly handles name passing and creation, concludes the presentation.*

1. Introduction

The (untyped) λ-calculus was introduced by Church as a convenient formalism for developing the theory of computable functions. Since then, the λ-calculus in its various versions has played a central rôle in both logic and computer science.

However, the λ-calculus is not very well-suited for dealing with communicating systems, where observable interactions between components assume a key rôle. Instead, *process calculi* are designed having this interactive environment in mind. They are a widely used formalism for the description of concurrent and distributed systems. In particular, *mobile* process calculi are those equipped with constructs for passing names (i.e., channels) and for creating them. The π-calculus [15] is one of the most studied calculi for mobility, thanks to its ability of modelling systems with dynamic communication topologies (e.g., mobile telephones). Mobility can also be used for encoding higher-order capabilities. We take the π-calculus as a case study to illustrate our approach.

Process calculi come usually equipped either with a *reduction* semantics or with an LTS semantics, the latter given in terms of labelled transition systems, where the label of each transition is a suitable encoding of the action performed in the associated step.

In the case of reduction semantics, one of the most relevant paradigms is probably Milner's *action calculi* [14], where some basic components (*names*, *abstraction* and *controls*, the latter construct offering a general way of encapsulating agents) suffice to describe the *statics* of most studied calculi (also in the presence of higher-order features) and where the *dynamics* can be described in terms of a suitable pre-order over the states, similarly to what happens in the *chemical abstract machine* proposed by Berry and Boudol [1]. This approach presents some analogies also with Meseguer's *rewriting logic* [12], where an algebra of proofs, usually obtained by lifting the state structure to computations, replaces the pre-order of action calculi. We refer to [8] for a comparison between rewriting logic and action calculi.

In the case of LTS semantics, the bidimensional nature of process calculi — the static description of system states and the *observable* part of their dynamics — is directly represented in a recently developed compositional model of computation, called *tile logic*. It collects many ideas in the field, ranging from the classical SOS rules by Plotkin [17], to the already mentioned rewriting logic, including *structured transition systems* [6] and *context systems* [11]. The basic reference for tiles is [9], but several tile formats have been investigated, that can deal with different aspects of distributed and interactive systems [7, 4]. Paper [8] extends its comparison also to the tile approach.

Tile logic consists of a sequent calculus over a rule-based model (i.e., the *tile model*), whose rules define the behaviour of certain *open* (e.g., partially specified) *configu-*

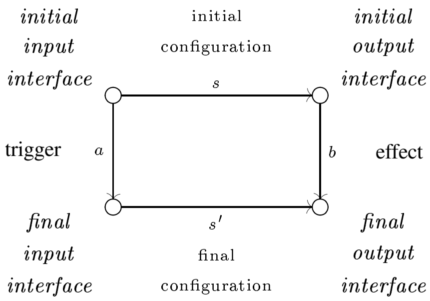

Figure 1. A tile.

rations, which may interact through their *interfaces*. Then, the behaviour of a system as a whole consists of a coordinated evolution of its sub-systems. The name "tile" is due to the graphical representation of such rules, which have the form in Figure 1 but can also be written as sequents $s \xrightarrow[b]{a} s'$, stating that the *initial configuration* s of the system evolves to the *final configuration* s' producing the *effect* b, which can be observed by the rest of the system. However, such a step is allowed only if the subcomponents of s (which is in general an open configuration) evolve to the subcomponents of s', producing the *trigger* a.

Triggers and effects are called *observations*. The vertices of each tile are called *interfaces*. Tiles can be composed horizontally (synchronizing an effect with a trigger), vertically (computational evolutions of a fixed component), and in parallel (concurrent steps) to generate larger steps. Thanks to these composition properties, there exists a precise correspondence between tiles and the cells of suitable *double categories*.

While the interpretation of tiles as double cells is already described in [9], in previous joint work with José Meseguer [3, 5] and in the PhD thesis of one of the authors [2] a precise categorical formulation is given for those tile systems whose configurations and observations rely on the same algebraic structure (e.g., symmetric monoidal or cartesian). The categorical models for such systems have been defined in terms of the novel notions of *symmetric monoidal* and *cartesian double categories*. However, in the previous work, only first-order structures have been considered.

In this paper, following the analogous one-dimensional construction, we thus define the *closed* version of cartesian double categories (CCDC). However we deal with an abstraction mechanism over objects only. Inspired by the well-known connections between cartesian closed categories and λ-calculus, we then investigate the possibility of adopting this new paradigm for the analysis of mobile communicating systems. To this purpose, we develope a syntactical notation for representing the cells of CCDC's in a convenient way, where typed variables and the usual functional application of λ-calculus are suitably included. However, since object abstraction only is allowed (as opposed to configuration abstraction or observation abstraction), we must explicitly introduce in the notation the sequential compositions of double cells to recover the full expressiveness of the categorical framework.

The double λ-notation is built over a set of type constants and three sets of term constants (for configurations, observations and tiles). The type system we propose assigns to a term as its type the "contour" of the tile it represents. Type judgments for terms are defined under a sequence Γ of type assignments to the free variables involved, as it is the case for simply typed λ-calculus. Although this notation can be manipulated and used in the natural way, depending on the applications we are interested in, a more linear but equivalent notation can be employed. We adopt the alternative notation for encoding the early semantics of π-calculus.

This presentation can be considered a still preliminary investigation on the subject, because a lot of open questions remain to be answered. In particular, the equality on terms is defined in the paper by looking at their corresponding cells in the initial categorical model. Since CCDC are axiomatized, we can lift the axiomatization to terms, but reduction to normal form is not considered yet, and we leave this topic for further research.

Structure of the paper. In Section 2, we fix the categorical notation that is used in the rest of the paper. This ranges from cartesian and cartesian closed categories to double categories and cartesian double categories.

In Section 3, we introduce and discuss the definition of cartesian closed double categories, making explicit their axiomatization. Section 4 presents the double λ-notation and its categorical interpretation in terms of CCDC's. In Section 5, the recasting of the early semantics of π-calculus in the double λ-notation concludes the main part of the paper.

To make this presentation almost self-contained, two appendices have also been included. In Appendix A both the syntax and the categorical interpretation of simply typed λ-calculus are briefly discussed. Hopefully, this should help to see the analogy with the bidimensional structures that we consider. In Appendix B, we recall from [5] the axiomatization of (canonical) cartesian double categories.

2. Background

We review some basics on cartesian closed categories that can be useful to see the analogy with their corresponding bidimensional versions in the theory of double categories. Since double categories might be unfamiliar to a large part of the community of computer scientists, we devoted Section 2.2 to their introduction.

2.1. Cartesian closed categories

A *cartesian* category is a category such that for each (possibly empty) tuple of objects $a_1, ..., a_n$, there exists a *product object* that precisely represents such tuple. It is usually denoted by $a_1 \times \cdots \times a_n$, but notice that the product is unique only up to isomorphism, and that it is completely determined by specifying its projections[1] $\Pi_1^{a_1 \times \cdots \times a_n}, ..., \Pi_n^{a_1 \times \cdots \times a_n}$, with

$$\Pi_i^{a_1 \times \cdots \times a_n} : a_1 \times \cdots \times a_n \to a_i,$$

for $i \in \{1, ..., n\}$. For example this means that for any two arrows $f : c \to a$ and $g : c \to b$, then there exists a unique arrow from c to $a \times b$, usually denoted by $\langle f, g\rangle$, such that $f = \langle f, g\rangle; \Pi_1$ and $g = \langle f, g\rangle; \Pi_2$. Given two arrows $f : a \to c$ and $g : b \to d$, then the arrow

$$\langle \Pi_1; f, \Pi_2; g\rangle : a \times b \to c \times d$$

is usually denoted by $f \times g$ (see Remark 1).

If $\mathcal{C}$ has binary products and a terminal object (i.e., the nullary product), then it has all *finite* products. It is often convenient to choose a specific product diagram for each tuple of objects, as explained in Appendix A.2. For instance, this simplifies the classical interpretation of simply typed λ-terms (see Table 8).

By exploiting the universal property of products, cartesianity can be defined in terms of suitable *adjunctions*.

Proposition 1 (Cartesian Category) *Let $\mathcal{C}$ be a category: (1) $\mathcal{C}$ has a terminal object iff the unique functor $!_\mathcal{C} : \mathcal{C} \to \mathbf{1}$ has a right adjoint; (2) $\mathcal{C}$ has binary products iff the diagonal functor $\Delta_\mathcal{C} : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ sending each arrow f of $\mathcal{C}$ to (f, f), has a right adjoint; (3) $\mathcal{C}$ is cartesian iff both $!_\mathcal{C} : \mathcal{C} \to \mathbf{1}$ and $\Delta_\mathcal{C} : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ have right adjoints.*

Remark 1 *The right adjoint to $!_\mathcal{C}$ corresponds to the choice of a final element in $\mathcal{C}$ that is the unique object in the image of $\mathbf{1}$. The right adjoint to $\Delta_\mathcal{C}$ corresponds to a suitable product choice $\_ \times \_$.*

A cartesian category is called *closed* if for each pair of objects a and b, there exists an *exponent* object b^a that can represent the homset $\mathcal{C}[a, b]$ (see Definition 9). As for cartesian categories, also cartesian closed categories admit a characterization that exploits the notion of adjunction.

Definition 1 (CCC as Adjunction) *A cartesian category $\mathcal{C}$ is* cartesian closed *if for each object a the functor $\_ \times a : \mathcal{C} \to \mathcal{C}$ has a right adjoint $\_^a : \mathcal{C} \to \mathcal{C}$.*

[1] We will omit the superscripts when obvious from the context.

$$\begin{tikzcd} a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & b \arrow[d, "u"] \\ c \arrow[r, "g"'] & d \end{tikzcd}$$

Figure 2. Graphical representation of a cell.

The component at b of the counit of the adjunction is denoted by $eval_{a,b} : b^a \times a \to b$. We recall that $eval$ generalizes the set-theoretic evaluation $eval(f, x) = f(x)$. The universal property of the exponent is explained in detail in Appendix A.2 (see Figure 11).

If we denote by $\Lambda_c^{a,b}$ the isomorphism (at c and b) from $\mathcal{C}[c \times a, b]$ to $\mathcal{C}[c, b^a]$ that arises from the adjunction, then the functor $\_^a : \mathcal{C} \to \mathcal{C}$ can be explicitly defined as:

$$(\_^a)(g) \quad = \quad \Lambda_{b^a}^{a,b}(eval_{a,b}; g)$$

for any arrow $g : b \to d$ in $\mathcal{C}$ (thus, $g^a : b^a \to d^a$).

We will extend this characterization of cartesian closedness to obtain its corresponding version in the double category setting.

To shorten the notation we will avoid the superscript $\_^{a,b}$ that specifies the objects a and b to which the map Λ_c is associated, as a and b are often clearly determined from the context. An axiomatic presentation of cartesian closed categories is shown in Appendix A.2.

2.2. Double categories

Definition 2 (Double Category) *A double category $\mathcal{D}$ consists of a collection $a, b, c, ...$ of* objects*, a collection $h, g, f, ...$ of* horizontal arrows*, a collection $v, u, w, ...$ of* vertical arrows *and a collection $A, B, C, ...$ of* cells *(also called* double cells*).*

1. *Objects and horizontal arrows form the* horizontal 1-category *$\mathcal{H}$, with identity id^a for each object a, and composition $\_ * \_$.*

2. *Objects and vertical arrows form also a category, called the* vertical 1-category *$\mathcal{V}$, with identity id_a for each object a, and composition $\_ \cdot \_$ (usually we will refer to both id^a and id_a either with the object name a or with id_a).*

3. *Cells are assigned* horizontal source *and* target *(which are vertical arrows) and* vertical source *and* target *(which are horizontal arrows); furthermore given a cell A, with vertical source h, vertical target g, horizontal source v, and horizontal target u, then the sources and targets of h, g, v, and u must be* compatible*, in the sense that they must compose a square as*

$$\begin{tikzcd} a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & b \arrow[r, "f"] \arrow[d, "u"] \arrow[dr, phantom, "B"] & b_1 \arrow[d, "w"] \\ c \arrow[r, "g"'] & d \arrow[r, "k"'] & d_1 \end{tikzcd} = A * B \qquad \begin{tikzcd} a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & b \arrow[d, "u"] \\ c \arrow[r, "g"] \arrow[d, "z"'] \arrow[dr, phantom, "C"] & d \arrow[d, "s"] \\ c_1 \arrow[r, "h'"'] & d_2 \end{tikzcd} = A \cdot C$$

Figure 3. Horizontal and vertical composition.

illustrated in Figure 2. We write $A : h \xrightarrow[u]{v} g$ for such a cell, and we refer to its sides as north, south, west *and* east*, with the obvious meaning.*

4. *Cells can be composed both horizontally ($\_ * \_$) and vertically ($\_ \cdot \_$) as follows: given $A : h \xrightarrow[u]{v} g$, $B : f \xrightarrow[w]{u} k$, and $C : g \xrightarrow[s]{z} h'$, then $A * B : (h*f) \xrightarrow[w]{v} (g*k)$, and $A \cdot C : h \xrightarrow[u \cdot s]{v \cdot z} h'$ are cells (see Figure 3). Moreover, given a fourth cell $D : k \xrightarrow[t]{s} f'$, horizontal and vertical compositions verify the following* exchange law*:*

$$(A \cdot C) * (B \cdot D) = (A * B) \cdot (C * D)$$

Under these rules, cells form both a horizontal category $\mathcal{D}^*$ and a vertical category $\mathcal{D}^{\cdot}$, with respective identities $1_v : a \xrightarrow[v]{v} c$ and $1^h : h \xrightarrow[b]{a} h$. Given $1^h : h \xrightarrow[b]{a} h$ and $1^g : g \xrightarrow[c]{b} g$, the equation $1^h * 1^g = 1^{h*g}$ must hold (and similarly for vertical composition of horizontal identities). Furthermore, horizontal and vertical identities of identities coincide, i.e., $1_{id_a} = 1^{id^a}$, and are denoted by 1_a, or simply a.

As a matter of notation, sometimes we will use $\_;\_$ to denote the composition on both the horizontal and vertical 1-categories.

Example 1 (Quartet Category) *Given a category $\mathcal{C}$, we denote by $\Box\mathcal{C}$ the* category of quartets *of $\mathcal{C}$: its objects are the objects of $\mathcal{C}$, its horizontal and vertical arrows are the arrows of $\mathcal{C}$, and its cells are the commuting square diagrams of arrows in $\mathcal{C}$*

$$\begin{tikzcd} a \arrow[r, "f"] \arrow[d, "v"'] & b \arrow[d, "u"] \\ c \arrow[r, "g"'] & d \end{tikzcd}$$

where $f, v, u, g \in \mathcal{C}$ and $f;u = v;g$ in $\mathcal{C}$. Notice that $\mathcal{C}$ is both the horizontal and the vertical 1-category of $\Box\mathcal{C}$.

Definition 3 (Double functor) *Given two double categories $\mathcal{D}$ and $\mathcal{E}$, a* double functor *$F : \mathcal{D} \to \mathcal{E}$ is a 4-tuple of functions mapping objects to objects, horizontal and vertical arrows to horizontal and vertical arrows, and cells to cells, preserving identities and compositions of all kinds.*

Double functors can be composed in the obvious way and any double category has associated an identity functor mapping each element into itself. This yields a category $\mathbf{DCat}$ whose objects are double categories and whose arrows are double functors.

Example 2 *The discrete category $\mathbf{1}$ (with just one object) can be viewed as a double category, which is also a final object in $\mathbf{DCat}$.*

Diagonal categories. Sometimes, due to the particular kind of cells involved in a complex composition, it is possible to adopt a more concise and convenient notation. In particular, for any double category, it is always possible to characterize two suitable *diagonal* sub-categories.

In fact, those cells having identities both as south and east sides are the arrows of a *diagonal* category $\mathcal{D}^{\triangleleft}$ whose composition $\_ \triangleleft \_$ is uniquely defined as depicted below:

$$\begin{tikzcd} a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & b \arrow[d, "b"] \\ b \arrow[r, "b"'] & b \end{tikzcd} \triangleleft \begin{tikzcd} b \arrow[r, "g"] \arrow[d, "u"'] \arrow[dr, phantom, "B"] & c \arrow[d, "c"] \\ c \arrow[r, "c"'] & c \end{tikzcd} = \begin{tikzcd} a \arrow[r] \arrow[d] \arrow[dr, phantom, "A"] & b \arrow[r] \arrow[d] \arrow[dr, phantom, "1^g"] & c \arrow[d] \\ b \arrow[r] \arrow[d] \arrow[dr, phantom, "1_u"] & b \arrow[r] \arrow[d] \arrow[dr, phantom, "B"] & c \arrow[d] \\ c \arrow[r] & c \arrow[r] & c \end{tikzcd}$$

i.e., $A \triangleleft B = (A * 1^g) \cdot (1_u * B)$.

Remark 2 *The composition schema associated to $\triangleleft$ is well-defined also when the horizontal and vertical targets of B are not identities. Abusing the notation, we will write $A \triangleleft B$ also for such cases.*

In a similar way, we can define a diagonal composition $\_ \triangleright \_$ for those cells having identities both as north and west sides (yielding the double category $\mathcal{D}^{\triangleright}$):

$$\begin{tikzcd} a \arrow[r, "a"] \arrow[d, "a"'] \arrow[dr, phantom, "A"] & a \arrow[d, "v"] \\ a \arrow[r, "h"'] & b \end{tikzcd} \triangleright \begin{tikzcd} b \arrow[r, "b"] \arrow[d, "b"'] \arrow[dr, phantom, "B"] & b \arrow[d, "u"] \\ b \arrow[r, "g"'] & c \end{tikzcd} = \begin{tikzcd} a \arrow[r] \arrow[d] \arrow[dr, phantom, "A"] & a \arrow[r] \arrow[d] \arrow[dr, phantom, "1_v"] & a \arrow[d] \\ a \arrow[r] \arrow[d] \arrow[dr, phantom, "1^h"] & b \arrow[r] \arrow[d] \arrow[dr, phantom, "B"] & b \arrow[d] \\ b \arrow[r] & b \arrow[r] & c \end{tikzcd}$$

i.e., $A \triangleright B = (A * 1_v) \cdot (1^h * B)$.

Cartesian double categories. To fully exploit the bidimensional structure of double categories, in joint work with José Meseguer [5] we proposed a notion of double products that requires the products to exist according to the four possible compositions that we have seen: horizontal ($\_ * \_$), vertical

(_·_), and diagonals (_◁_ and _▷_). The fact that also the diagonal sub-categories have a cartesian structure asserts some sort of consistency between the horizontal and the vertical cartesianity. For example this means that one has several ways for *pairing* two cells, but they are all strictly correlated, so that it is possible to step from one to another. This is discussed in detail in Appendix B.2. In this section, cartesian double categories are instead defined in terms of suitable adjunctions between double functors. Such adjunctions act in all the dimensions under consideration.

Definition 4 (Double Adjoint) *Given two double categories $\mathcal{D}$ and $\mathcal{E}$ and a double functor $F : \mathcal{D} \to \mathcal{E}$, we say that F has a* double right adjoint *(or equivalently, that F is a* double left adjoint*) if there exists a double functor $G : \mathcal{E} \to \mathcal{D}$ that is a right adjoint of F in $\mathcal{D}^*$, $\mathcal{D}^\cdot$, $\mathcal{D}^\triangleleft$, and $\mathcal{D}^\triangleright$. A double adjoint is* strict *if the object components of the counits (respectively units) for $\mathcal{D}^*$ and $\mathcal{D}^\cdot$ are the non-trivial horizontal and vertical arrows in the corresponding counits (respectively units) for $\mathcal{D}^\triangleleft$ and $\mathcal{D}^\triangleright$.*

The fact the G is the right adjoint of F along the four possible compositions is clearly different from requiring the existence of four right adjoints, one for each composition. Indeed, this expresses the consistency of the adjunction along the possible compositions.

In the following definition, let $\Delta : \mathcal{D} \to \mathcal{D} \times \mathcal{D}$ be the double functor that makes a copy of the argument, i.e., such that for each $A \in \mathcal{D}$, $\Delta(A) = (A, A)$.

Definition 5 (Cartesian Double Category) *A double category $\mathcal{D}$ is* cartesian *iff both the double functor $!_\mathcal{D} : \mathcal{D} \to \mathbf{1}$ and $\Delta_\mathcal{D} : \mathcal{D} \to \mathcal{D} \times \mathcal{D}$ have strict double right adjoints.*

As said above, the fact that the right adjoint is the same functor in all the dimensions guarantees the consistency of products and terminal objects in the two dimensions.

For example, this means that given two cells $A : h \overset{v}{\underset{b}{\longrightarrow}} b$ and $B : g \overset{v}{\underset{b}{\longrightarrow}} b$ with the same horizontal source $v : a \to b$ and with identities as targets, then they can be paired not only horizontally, but also diagonally (w.r.t., composition _ ◁ _), just exploiting the common source a.

3. Cartesian closed double categories

The definition of the analogous of cartesian closedness in the theory of double categories is problematic, and the approach we propose can appear a bit controversial. The problem is that the objects of the horizontal category are arrows of the vertical 1-category, and vice versa: In principle one might require that for each cell $A : h \overset{v}{\underset{u}{\longrightarrow}} g$ both the cells A^f and A^w would exists, for any horizontal arrow f and vertical arrow w. Such request implies the existence of higher order arrows h^f, v^w that has no counterpart in cartesian closed categories, and this contradict our intuition of having two (coherent) cartesian closed categories as underlying horizontal and vertical structures.

The solution we adopt is to allow the abstraction only w.r.t. objects. Indeed they are the only elements that the vertical and the horizontal 1-categories have in common, and moreover we think that this feature suffices for modelling a wide class of applications. In particular, given any cell A and any object a, it makes sense to define the exponentiation A^a, and such construction acts as a double functor.

$$\begin{tikzcd} b^a \times a \arrow[r, "eval_{a,b}"] \arrow[d, "v^a \times a"'] \arrow[dr, phantom, "Heval_{a,v}"] & b \arrow[d, "v"] \\ b'^a \times a \arrow[r, "eval_{a,b'}"'] & b' \end{tikzcd} \qquad \begin{tikzcd} b^a \times a \arrow[r, "h^a \times a"] \arrow[d, "eval'_{a,b}"'] \arrow[dr, phantom, "Veval_{a,h}"] & b'^a \times a \arrow[d, "eval'_{a,b'}"] \\ b \arrow[r, "h"'] & b' \end{tikzcd}$$

$$\begin{tikzcd} b^a \times a \arrow[r, "eval_{a,b}"] \arrow[d, "eval'_{a,b}"'] \arrow[dr, phantom, "Deval_{a,b}"] & b \arrow[d, "b"] \\ b \arrow[r, "b"'] & b \end{tikzcd} \qquad \begin{tikzcd} b^a \times a \arrow[r, "b^a \times a"] \arrow[d, "b^a \times a"'] \arrow[dr, phantom, "Deval'_{a,b}"] & b^a \times a \arrow[d, "eval'_{a,b}"] \\ b^a \times a \arrow[r, "eval_{a,b}"'] & b \end{tikzcd}$$

Figure 4. The double evaluation maps.

Definition 6 (CCDC) *A double category $\mathcal{D}$ is* cartesian closed *(CCDC) iff it is cartesian and for each object a the double functor $\_ \times a$ has a strict double right adjoint.*

The extensive definition of CCDC implies that the horizontal (respectively vertical) 1-category of $\mathcal{D}$ is CCC with exponent b^a, evaluation map *eval* and natural isomorphism Λ (respectively, CCC with exponent b^a, evaluation map $eval'$ and natural isomorphism Λ') and also the existence of the following cells (see also Figure 4):

- for any vertical arrow $v : b \to b'$ and for any object a, there exists the cell

$$Heval_{a,v} : eval_{a,b} \overset{v^a \times a}{\underset{v}{\longrightarrow}} eval_{a,b'};$$

- for any horizontal arrow $h : b \to b'$ and for any object a, there exists the cell

$$Veval_{a,h} : h^a \times a \overset{eval'_{a,b}}{\underset{eval'_{a,b'}}{\longrightarrow}} h;$$

- for any objects a and b, there exists the cell

$$Deval_{a,b} : eval_{a,b} \overset{eval'_{a,b}}{\underset{b}{\longrightarrow}} b;$$

- for any objects a and b, there exists the cell

$$Deval'_{a,b} : b^a \times a \xrightarrow[eval'_{a,b}]{b^a\times a} eval_{a,b}.$$

Moreover, we have the following isomorphisms between the cells of $\mathcal{D}$:

- for any object a and vertical arrows u and v, there is a bijection[2] $\Lambda^{\mathtt{H}}_u : \{A : h \xrightarrow[v]{u\times a} g\} \to \{B : \Lambda h \xrightarrow[v^a]{u} \Lambda g\}$, where $\Lambda^{\mathtt{H}}_u(A)$ is the *horizontal abstraction* of A, with

$$\begin{array}{rcll} (\Lambda^{\mathtt{H}}_u(A)\times a) * Heval_{a,v} &=& A & (1)\\ \Lambda^{\mathtt{H}}_u((B\times a) * Heval_{a,v}) &=& B; & (2)\end{array}$$

- for any object a and horizontal arrows g and h, there is a bijection $\Lambda^{\mathtt{V}}_g : \{A : g\times a \xrightarrow[u]{v} h\} \to \{B : g \xrightarrow[\Lambda' u]{\Lambda' v} h^a\}$, where $\Lambda^{\mathtt{V}}_g(A)$ is the *vertical abstraction* of A, with

$$\begin{array}{rcll} (\Lambda^{\mathtt{V}}_g(A)\times a) \cdot Veval_{a,h} &=& A & (3)\\ \Lambda^{\mathtt{V}}_g((B\times a) \cdot Veval_{a,h}) &=& B; & (4)\end{array}$$

- for any objects a, b and c, there is a bijection $\Lambda^{\triangleleft}_c$ from the set of cells $\{A : h \xrightarrow[b]{v} b \mid h, v : c\times a \to b\}$ to the set of cells $\{B : h' \xrightarrow[b^a]{v'} b^a \mid h', v' : c \to b^a\}$, with $\Lambda^{\triangleleft}(A)_c : \Lambda h \xrightarrow[b^a]{\Lambda' v} b^a$ such that

$$\begin{array}{rcll} (\Lambda^{\triangleleft}_c(A)\times a) \triangleleft Deval_{a,b} &=& A & (5)\\ \Lambda^{\triangleleft}_c((B\times a) \triangleleft Deval_{a,b}) &=& B; & (6)\end{array}$$

- for any objects a, b and c, there is a bijection $\Lambda^{\triangleright}_c$ from the set $\{A : c\times a \xrightarrow[v]{c\times a} h \mid h, v : c\times a \to b\}$ to the set $\{B : c \xrightarrow[v']{c} h' \mid h', v' : c \to b^a\}$, with $\Lambda^{\triangleleft}(A)_c : c \xrightarrow[\Lambda' v]{c} \Lambda h$ such that

$$\begin{array}{rcll} (\Lambda^{\triangleright}_c(A)\times a) \triangleright Deval'_{a,b} &=& A & (7)\\ \Lambda^{\triangleright}_c((B\times a) \triangleright Deval'_{a,b}) &=& B. & (8)\end{array}$$

Given a cell $A : h \xrightarrow[u]{v} g$, from the definition of double adjunction it turns out that the possible horizontal and vertical definitions of A^a coincide (see Figure 5).

$$\begin{array}{rcl} A^a &=& \Lambda^{\mathtt{H}}_{v^a}(Heval_{a,v} * A)\\ &=& \Lambda^{\mathtt{V}}_{h^a}(Veval_{a,h}\cdot A)\end{array}$$

[2]We just recall that every arrow $h' : c \to b^a$ is in the image through Λ of an arrow $h : c\times a \to b$, and therefore $\Lambda^{\mathtt{H}}_u$ is indeed a bijection between $\{A : h \xrightarrow[v]{u\times a} g\}$ and $\{B : h' \xrightarrow[v^a]{u} g'\}$.

$$\begin{tikzcd} b \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & d \arrow[d, "u"] \\ b' \arrow[r, "g"'] & d' \end{tikzcd} \qquad \begin{tikzcd} b^a \arrow[r, "h^a"] \arrow[d, "v^a"'] \arrow[dr, phantom, "A^a"] & d^a \arrow[d, "u^a"] \\ b'^a \arrow[r, "g^a"'] & d'^a \end{tikzcd}$$

Figure 5. A cell A and its exponentiation A^a.

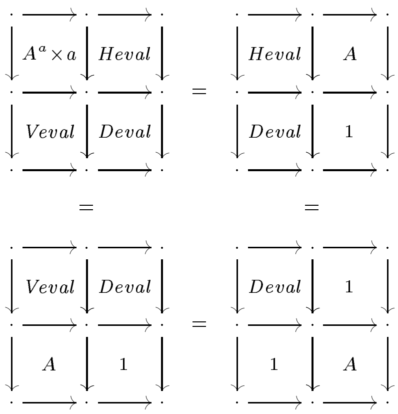

Figure 6. Commuting compositions of cells.

Notice that if $A : h \xrightarrow[d]{v} d$ or $A : b \xrightarrow[v]{b} h$, then also the two following *diagonal* definitions are feasible, and they coincide with the previous one:

$$\begin{array}{lcl} \text{if } A : h \xrightarrow[d]{v} d \text{ then } A^a &=& \Lambda^{\triangleleft}_{b^a}(Deval_{a,b} \triangleleft A)\\ \text{if } A : b \xrightarrow[v]{b} h \text{ then } A^a &=& \Lambda^{\triangleright}_{b^a}(Deval'_{a,b} \triangleright A)\end{array}$$

The cell $Deval_{a,b}$ and $Deval'_{a,b}$ can also be used to characterize *Heval* and *Veval* (viewed as functors from the vertical/horizontal 1-category to the vertical/horizontal category of cells) in terms of double transformations (see Appendix B.1). Indeed, the following naturality axioms are satisfied (for all a, b, h and v):

$$\begin{array}{rcl} Heval_{a,v}\cdot Deval_{a,b'} &=& Deval_{a,b}\cdot 1_v\\ Veval_{a,h} * Deval_{a,b'} &=& Deval_{a,b} * 1^h\\ 1_{v^a\times a}\cdot Deval'_{a,b'} &=& Deval'_{a,b}\cdot Heval_{a,v}\\ 1^{h^a\times a} * Deval_{a,b'} &=& Deval_{a,b} * Veval_{a,h}\\ Deval'_{a,b} * Deval_{a,b} &=& 1^{eval_{a,b}}\\ Deval'_{a,b}\cdot Deval_{a,b} &=& 1_{eval'_{a,b}}\end{array}$$

For instance, all the pastings of cells in Figure 6 coincide.

Theorem 1 *The complete axiomatization of* CCDC*'s is given by axioms (1)–(8) together with axioms* **(A)–(E)** *in Appendix B.2.*

4. Double λ-notation

We aim at developing a general type formalism for process calculi, which has a natural interpretation in terms of cartesian closed double categories. Our formalism takes into account both the syntactic structure of the calculi and its observable dynamics. Since it acts in two dimensions we call it *double λ-notation*.

The idea is to extend the tile formalism (see e.g., [9, 2]) with more expressive configurations and observations. In particular, we assume that they are both described by simply typed λ-terms over two distinct signatures (over the same sorts). Therefore, this format allows us to introduce abstractions both on configurations and on observations. For instance, this is very important when designing the rules for the input context in mobile calculi, where channel names are both syntactic elements and also entities that can be communicated, thus appearing in the behavioural dimension.

As discussed in Appendix A.1, the usual way of presenting syntactic judgments for simply typed λ-calculus is defined by the format $\Gamma \vdash M : \sigma$, where Γ is an environment and σ is the type of the term M (defined over a certain signature $\langle B, C \rangle$). Our type system must take into account the fact that terms are double cells and their types are essentially their borders, i.e., the typed terms that define the north, south, east and west sides.

4.1. Double typing

A *double signature* for the double λ-notation is a 4-tuple

$$\Sigma = \langle \mathbb{B}, \mathbb{H}, \mathbb{V}, \mathbb{C} \rangle$$

where $\mathbb{B}$ is the set of *type constants*, $\mathbb{H}$ is the set of *horizontal term constants* $h : \sigma, g : \tau, \ldots$ typed over $\mathbb{B}$ (see Appendix A.1 for the definition of σ, τ, ...), $\mathbb{V}$ is the set of *vertical term constants* $v : \sigma, u : \tau, \ldots$ typed over $\mathbb{B}$, and $\mathbb{C}$ is the set of *tile term constants* $\alpha \square t, \beta \square t', \ldots$, with t, t', ... *closed contour types* (see below).

The pair $\langle \mathbb{B}, \mathbb{H} \rangle$ defines the *horizontal λ-notation* $\Sigma_{\mathtt{H}}$, and the pair $\langle \mathbb{B}, \mathbb{V} \rangle$ defines the *vertical λ-notation* $\Sigma_{\mathtt{V}}$. Contour types essentially describe the border of tile terms, and thus their definition involves a "mixture" of horizontal and vertical terms.

The idea is that just by looking at the contour type of a tile term, it must be possible to understand the ways in which it can be composed with other tile terms, i.e., its feasible interactions with the environment. This can be done by considering the terms that form the border of the tile. A generic contour type has the form:

$$H : \tau \xrightarrow[U:\tau \to \sigma]{V:\rho} G : \rho \to \sigma$$

where H represents the north side, U represents the east side, V represents the west side, and G represents the south side. Here H and G are horizontal terms, whilst U and V are vertical terms. A contour type is well-typed under the type assignment Γ if H, G, V and U are well-typed when employing Γ (with types as written in the formula). A contour type t is closed if it can be typed by an empty environment, i.e., if H, G, V and U are closed terms.

Starting from the constants in the signature, more complex terms can be constructed using the following syntax (notice the difference between the symbol $\star$ for the unit and the symbol $*$ for horizontal composition):

$$\begin{array}{lll} M & ::= & \alpha \mid h \mid v \mid x \mid \star \mid \\ & & M * M \mid M \cdot M \mid M \triangleleft M \mid M \triangleright M \mid \\ & & \langle M, M \rangle^{\mathtt{H}} \mid \mathit{Proj}_i^{\mathtt{H}} M \mid \langle M, M \rangle^{\mathtt{V}} \mid \mathit{Proj}_i^{\mathtt{V}} M \mid \\ & & \langle M, M \rangle^{\triangleleft} \mid \mathit{Proj}_i^{\triangleleft} M \mid \langle M, M \rangle^{\triangleright} \mid \mathit{Proj}_i^{\triangleright} M \mid \\ & & \lambda^{\mathtt{H}} x : \sigma . M \mid M @^{\mathtt{H}} M \mid \lambda^{\mathtt{V}} x : \sigma . M \mid M @^{\mathtt{V}} M \mid \\ & & \lambda^{\triangleleft} x : \sigma . M \mid M @^{\triangleleft} M \mid \lambda^{\triangleright} x : \sigma . M \mid M @^{\triangleright} M \end{array}$$

where $\alpha \in \mathbb{C}$, $h \in \mathbb{H}$, $v \in \mathbb{V}$, x is a generic variable and pairings, projections, abstractions and applications (the latter denoted by the symbol @) are possible along all the compositional dimensions of tiles (horizontal, vertical and diagonal, the latter in two distinct ways). Moreover, horizontal, vertical and diagonal compositions ($M * M$, $M \cdot M$, $M \triangleleft M$ and $M \triangleright M$) are also given. We will see that since abstraction is restricted to objects, these compositions cannot be expressed by functional application.

We are now ready to present the inference rules of the type system for the double λ-notation. In the next section, we will give the translation of well-typed tile terms into the corresponding cells of the categorical models, which are cartesian closed double categories. In particular, the equivalences between tile terms are exactly those that can be derived in the initial model. We leave to future work the study of a more direct equational system for tile terms and the definition of possible normal form reduction strategies.

A generic type sentence has the form

$$\Gamma \vdash M \square\ H : \tau \xrightarrow[U:\tau \to \sigma]{V:\rho} G : \rho \to \sigma\ .$$

The type system is defined by the axioms and inference rules listed below. They are numbered to enhance the correspondence with the categorical interpretation rules given in Section 4.2. Due to space limitations, for each kind of

$$
\begin{array}{rcl}
h^{\mathtt{H}} & = & h \\
\star^{\mathtt{H}} & = & \star \\
x^{\mathtt{H}} & = & x \\
(\langle H, G\rangle)^{\mathtt{H}} & = & \langle H^{\mathtt{H}}, G^{\mathtt{H}}\rangle^{\mathtt{H}} \\
(Proj_i H)^{\mathtt{H}} & = & Proj_i^{\mathtt{H}} H^{\mathtt{H}} \\
(\lambda x : \sigma.H)^{\mathtt{H}} & = & \lambda^{\mathtt{H}} x : \sigma.H^{\mathtt{H}} \\
(HG)^{\mathtt{H}} & = & H^{\mathtt{H}} @^{\mathtt{H}} G^{\mathtt{H}}
\end{array}
$$

Table 1. Definition of the meta-notation $H^{\mathtt{H}}$.

construct (identities, pairing, projection, abstraction, application, and composition) the corresponding rule is shown only in the horizontal "dimension." The others can be obtained analogously.

To keep the notation shorter we use the following abbreviations: for all type σ let $id_\sigma = \lambda y : \sigma.y$ (with type $\sigma \to \sigma$) and for all type ϱ and vertical term $\Gamma \vdash U : \tau \to \sigma$ then let $U^\varrho = \lambda y : (\varrho \to \tau).\lambda x : \varrho.U(yx)$ (with type $(\varrho \to \tau) \to (\varrho \to \sigma)$), where $y, x \notin \mathrm{fv}(U)$ (with $\mathrm{fv}(U)$ denoting the free variables of U), and similarly for the horizontal term $\Gamma \vdash G : \rho \to \sigma$.

1. *tile term constants:*
$\dfrac{\alpha \square t \in \mathbb{C}}{\emptyset \vdash \alpha \square t}$;

2. *horizontal identities:*
$$\frac{\Gamma \vdash H : \tau}{\Gamma \vdash H^{\mathtt{H}} \square\ H : \tau \xrightarrow[id_\tau : \tau \to \tau]{\star : \epsilon} \lambda z : \epsilon.H : \epsilon \to \tau};$$
where $z \notin \mathrm{fv}(H)$ and the superscript $\_^{\mathtt{H}}$ is just a meta-notation to represent the tile term associated to the horizontal term H and can be easily defined by induction on the structure of H as in Table 1;

3. *horizontal pairing:*
$$\frac{\begin{array}{c}\Gamma \vdash M_1 \square\ H_1 : \tau_1 \xrightarrow[U_1 : \tau_1 \to \sigma_1]{V : \rho} G_1 : \rho \to \sigma_1\ , \\ \Gamma \vdash M_2 \square\ H_2 : \tau_2 \xrightarrow[U_2 : \tau_2 \to \sigma_2]{V : \rho} G_2 : \rho \to \sigma_2\end{array}}{\Gamma \vdash \langle M_1, M_2\rangle^{\mathtt{H}} \square\ \langle H_1, H_2\rangle : \tau_1 \times \tau_2 \xrightarrow[U' : \tau']{V : \rho} G' : \rho'}$$
where $U' = \lambda\langle y_1, y_2\rangle : \tau_1 \times \tau_2.\langle U_1 y_1, U_2 y_2\rangle$
with type $\tau' = (\tau_1 \times \tau_2) \to (\sigma_1 \times \sigma_2)$
and $G' = \lambda z : \rho.\langle G_1 z, G_2 z\rangle$
with type $\rho' = \rho \to (\sigma_1 \times \sigma_2)$
for $y_1, y_2 \notin \mathrm{fv}(U_1, U_2)$ and $z \notin \mathrm{fv}(G_1, G_2)$ (for $\langle \_, \_\rangle^{\mathtt{V}}$, $\langle \_, \_\rangle^{\triangleleft}$ and $\langle \_, \_\rangle^{\triangleright}$, in the first case the requirement is that $H_1 = H_2$ but the V's can be different, in the second case just Γ must be the same, and in the last case both $H_1 = H_2$ and $V_1 = V_2$ must hold);

4. *horizontal projection:*
$$\frac{\Gamma \vdash M \square\ H : \tau_1 \times \tau_2 \xrightarrow[U : (\tau_1 \times \tau_2) \to (\sigma_1 \times \sigma_2)]{V : \rho} G : \rho \to \sigma_1 \times \sigma_2}{\Gamma \vdash Proj_1^{\mathtt{H}} M \square\ Proj_1 H : \tau_1 \xrightarrow[\lambda y_1 : \tau_1.U_1 : \tau_1 \to \sigma_1]{V : \rho} G' : \rho \to \sigma_1}$$
where $U = \lambda\langle y_1, y_2\rangle : \tau_1 \times \tau_2.\langle U_1, U_2\rangle$
and $G' = \lambda z : \rho.Proj_1(Gz)$
provided that $z \notin \mathrm{fv}(G)$, $y_1 \notin \mathrm{fv}(U_2)$ and $y_2 \notin \mathrm{fv}(U_1)$, and similarly for $Proj_2^{\mathtt{H}}$;

5. *horizontal abstraction:*
$$\frac{\Gamma, x : \varrho \vdash M \square\ H : \tau \xrightarrow[U : \tau \to \sigma]{V : \rho} G : \rho \to \sigma}{\Gamma \vdash \lambda^{\mathtt{H}} x : \varrho.M \square\ \lambda x : \varrho.H : \varrho \to \tau \xrightarrow[U^\varrho]{V : \rho} G' : \sigma'}$$
where U^ϱ has type $(\varrho \to \tau) \to (\varrho \to \sigma)$
and $G' = \lambda z : \rho.\lambda x : \varrho.Gz$
has type $\sigma' = \rho \to (\varrho \to \sigma)$,
provided that $x \notin \mathrm{fv}(V, U)$ and $z \notin \mathrm{fv}(G)$ (for vertical abstraction $x \notin \mathrm{fv}(H, G)$ must hold, whereas $\lambda^{\triangleleft} x : \varrho.M$ requires $x \notin \mathrm{fv}(U, G)$ and $\lambda^{\triangleright} x : \varrho.M$ requires $x \notin \mathrm{fv}(H, V)$);

6. *horizontal application:*
$$\frac{\begin{array}{c}\Gamma \vdash M_1 \square\ H_1 : \tau_1 \xrightarrow[id_{\tau_1} : \tau_1 \to \tau_1]{V_1 : \rho_1} G_1 : \rho_1 \to \tau_1\ , \\ \Gamma \vdash M_2 \square\ H_2 : \tau_1 \to \tau_2 \xrightarrow[U_2^{\tau_1}]{V_2 : \rho_2} G_2 : \rho_2 \to (\tau_1 \to \sigma_2)\end{array}}{\Gamma \vdash M_2 @^{\mathtt{H}} M_1 \square\ H_2 H_1 : \tau_2 \xrightarrow[U_2 : \tau_2 \to \sigma_2]{\langle V_2, V_1\rangle : \rho_2 \times \rho_1} G : (\rho_2 \times \rho_1) \to \sigma_2}$$
where $U_2^{\tau_1}$ has type $(\tau_1 \to \tau_2) \to (\tau_1 \to \sigma_2)$
and $G = \lambda\langle z_2, z_1\rangle : \rho.G_2 z_2(G_1 z_1)$,
with $z_1, z_2 \notin \mathrm{fv}(G_1, G_2)$;

7. *horizontal composition:*
$$\frac{\begin{array}{c}\Gamma \vdash M_1 \square\ H_1 : \tau_1 \xrightarrow[V_2 : \tau_1 \to \rho_2]{V_1 : \rho_1} G_1 : \rho_1 \to \rho_2\ , \\ \Gamma, x : \tau_1 \vdash M_2 @^{\triangleleft} x \square\ H_2 x : \tau_2 \xrightarrow[U_2 : \tau_2 \to \sigma_2]{V_2 x : \rho_2} G_2 : \rho_2 \to \sigma_2\end{array}}{\Gamma \vdash M_1 * M_2 \square\ H : \tau_2 \xrightarrow[U_2 : \tau_2 \to \sigma_2]{V_1 : \rho_1} G : \rho_1 \to \sigma_2}$$
where $H = H_2 H_1$ and $G = \lambda z : \rho_1.G_2(G_1 z)$, with $z \notin \mathrm{fv}(G_1, G_2)$ and $x \notin \mathrm{fv}(M_2, H_2, U_2, G_2)$.

The diagonal application in the second premise of rule 7 serves only as a mean to shorten the notation. In particular, the matching between the east side of M_1 (i.e., V_2) and the west side of M_2 can be expressed in a natural way, whereas the abstraction w.r.t. x would result in a more complicated condition. This is also in the style of the translation we need for the π-calculus example.

The following rules are in some sense auxiliary, since they deal with rearrangements of interfaces but do not substantially change the meaning of judgments in the premises. The symbol $\simeq$ signals that also the converse rule must be considered.

i $\dfrac{\Gamma \vdash M \,\square\, t}{\Gamma, x : \sigma \vdash M \,\square\, t}$;

ii $\dfrac{\Gamma_1, x : \tau, y : \rho, \Gamma_2 \vdash M \,\square\, t}{\Gamma_1, y : \rho, x : \tau, \Gamma_2 \vdash M \,\square\, t}$;

iii $\dfrac{\Gamma, x : \varrho \vdash M \,\square\;\; H : \tau \xrightarrow[U : \tau \to \sigma]{V : \rho} G : \rho \to \sigma}{\Gamma, x : \varrho \vdash M \,\square\; \langle x, H\rangle : \varrho \times \tau \xrightarrow[U' : (\varrho \times \tau) \to \sigma]{V : \rho} G : \rho \to \sigma} \simeq$

where $U' = \lambda\langle x, y\rangle : \varrho \times \tau . Uy$, with $y \notin \mathrm{fv}(U)$ and $x \in \mathrm{fv}(U)$ (but notice that $x \notin \mathrm{fv}(U')$). There is also the analogous rule for avoiding that $x \in \mathrm{fv}(G)$, but it is very similar and thus omitted.

The third rule can be applied only once to a given M (because after its application the premise $x \in \mathrm{fv}(U)$ is no longer valid). Moreover, if $x \in \mathrm{fv}(G) \cap \mathrm{fv}(U)$, the application of rule iii and then of the analogous one yields the same result as the application of the two rules in the reverse order.

In practice, the several restrictions on names for the application of the type rules listed above do not limit the expressiveness of the formalism. In fact, if a rule cannot be applied, we can always find equivalent contour types for which the application of the rule is possible. This is achieved by applying rule iii, which makes explicit the propagation of global variables via H and V to U and G.

Also notice that different type judgments are possible with the same Γ and M; indeed they all represent "isomorphic" situations. This should not be surprising since also in the one-dimensional case a term M can be typed with different assignments Γ, and in a contour type the north and west sides have in part the rôle of type assignments for the east and south sides.

4.2. Categorical semantics

As already said, the double λ-notation has been designed having in mind the category where its models should live, i.e., the category of CCDC and structure preserving double functors. Therefore, the initial model associated to a signature is the cartesian closed double category where

(a) the objects are the elements of the higher-order monoid freely generated by the type constants in $\mathbb{B}$;

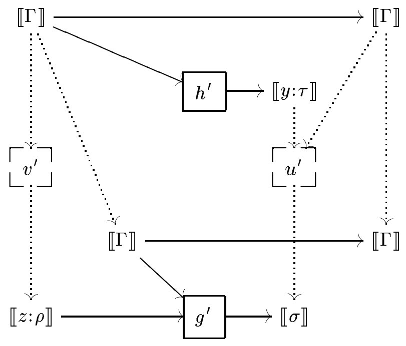

Figure 7. The contour of the cell $[\![\Gamma \vdash M \,\square\, t]\!]$.

(b) the horizontal (respectively, vertical) arrows are those of the cartesian closed category freely generated by the signature[3] $\Sigma_{\mathtt{H}}$ (respectively, $\Sigma_{\mathtt{V}}$);

(c) the tile term constants α are mapped to the corresponding cells $[\![\alpha]\!]$;

(d) and the cells are freely generated via the available operations.

Let $\Gamma = x_1 : \sigma_1, ..., x_n : \sigma_n$ and let $\Gamma \vdash M \,\square\, t$ be a generic type judgment with

$$t = \; H : \tau \xrightarrow[U : \tau \to \sigma]{V : \rho} G : \rho \to \sigma$$

Then, the categorical semantics $[\![\Gamma \vdash M \,\square\, t]\!]$ yields a cell with border $h \xrightarrow[u]{v} g$, where

$$\begin{array}{rcl} h &=& [\![\Gamma \vdash \langle x_1, ..., x_n, H\rangle \,:\, \sigma_1 \times \cdots \times \sigma_n \times \tau]\!] \\ v &=& [\![\Gamma \vdash \langle x_1, ..., x_n, V\rangle \,:\, \sigma_1 \times \cdots \times \sigma_n \times \rho]\!] \\ u &=& [\![\Gamma, y : \tau \vdash \langle x_1, ..., x_n, Uy\rangle \,:\, \sigma_1 \times \cdots \times \sigma_n \times \sigma]\!] \\ g &=& [\![\Gamma, z : \rho \vdash \langle x_1, ..., x_n, Gz\rangle \,:\, \sigma_1 \times \cdots \times \sigma_n \times \sigma]\!] \end{array}$$

As the reader can notice, we have chosen to propagate all the variables in Γ around the contour of the tile. In this way, the names in Γ can be used also in G and U without being explicitly propagated via H and V. This is very convenient for representing calculi based on names, because one can type the terms within a *global* environment.

This interpretation is graphically illustrated in Figure 7, where an informal *wire and box* notation is used to represent the contour of the cell, with

$$\begin{array}{llll} h' = & [\![\Gamma \vdash H \,:\, \tau]\!], & g' = & [\![\Gamma, z : \rho \vdash Gz \,:\, \sigma]\!], \\ v' = & [\![\Gamma \vdash V \,:\, \rho]\!], & u' = & [\![\Gamma, y : \tau \vdash Uy \,:\, \sigma]\!]. \end{array}$$

[3] In particular, this means that for each horizontal term constant $h : \sigma \in \mathbb{H}$ there is a basic arrow $[\![h]\!] : e \to [\![\sigma]\!]$ in the category.

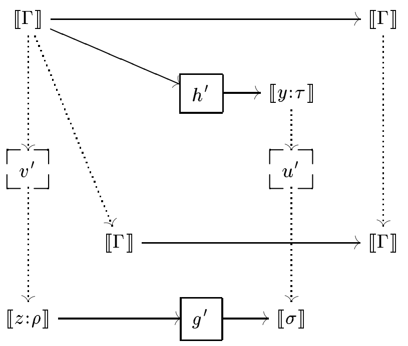

Figure 8. A cell with $\mathrm{fv}(G, U) = \emptyset$.

Type constants, horizontal term constants and vertical term constants are interpreted in the usual way (see Appendix A.3). Similarly, each tile term constant $\alpha \square t$ is interpreted as a tile of the appropriate type (i.e., as in Figure 7 with $\llbracket \Gamma \rrbracket = \llbracket \emptyset \rrbracket = e$, which is the unit object).

The use of Γ as a global environment facilitates the use of the double λ-notation, but complicates at some extent the presentation of the categorical semantics. For example, if one swaps the position of two variables in Γ, this changes also the contour of the tile, because Γ is propagated everywhere. As another example, the cell for the diagonal pairing $\langle \_, \_ \rangle^{\triangleleft}$ can be defined in terms of the cells associated to the components, but then some additional composition with double dischargers, double duplicators and double symmetries has to be done to get rid of the two copies of the variables in Γ that appear in the images of the two components. Therefore we make use of the last rule of the type system (and its specular version) by considering only judgments with explicit variable propagation, i.e., such that $\mathrm{fv}(G, U) = \emptyset$ (see Figure 8, where $u' = \llbracket y : \tau \vdash Uy \ : \ \sigma \rrbracket$ and $g' = \llbracket z : \rho \vdash Gz \ : \ \sigma \rrbracket$, i.e., U and G can be typed without Γ). For such cases it is always possible to discharge the global variables propagated along the contour, and thus we can assume that

$$\llbracket \Gamma \vdash M \square t \rrbracket \quad = \quad \boldsymbol{\pi}_{\llbracket \Gamma \rrbracket} \triangleleft (1_{\llbracket \Gamma \rrbracket} \otimes \{\![\Gamma \vdash M \square t]\!\})$$

or, vice versa, that

$$\{\![\Gamma \vdash M \square t]\!\} \quad = \quad (\llbracket \Gamma \vdash M \square t \rrbracket * (\boldsymbol{!}_{\llbracket \Gamma \rrbracket} \otimes 1_{u'})) \cdot ((\boldsymbol{\dagger}_{\llbracket \Gamma \rrbracket} * \boldsymbol{\phi}_{\llbracket \Gamma \rrbracket}) \otimes 1_{g'})$$

where $\boldsymbol{\pi}$, $\boldsymbol{!}$, $\boldsymbol{\dagger}$ and $\boldsymbol{\phi}$ are the auxiliary cells of cartesian double categories as defined in Appendix B.2 (they are boldfaced to avoid confusion with the greek letters ranging over types). This correspondence is very convenient, because the simpler mapping $\{\![\Gamma \vdash M \square t]\!\}$ illustrated in Figure 9 (and therefore $\llbracket \_ \rrbracket$ via the transformation above) can be inductively defined as follows. Notice that for mapping terms of the one-dimensional horizontal (vertical) λ-notation, parentheses $\llbracket \_ \rrbracket$ are used with the meaning of Table 8. To keep the presentation shorter, we avoid to detail the contour types t, t', t_1, t_2, ... that are those of the associated typing rules, possibly transformed via the application of rule iii when necessary to obtain $\mathrm{fv}(G, U) = \emptyset$.

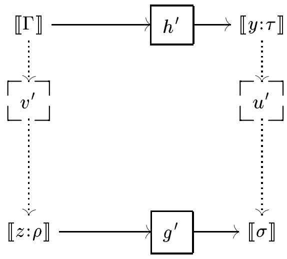

Figure 9. The contour of the cell $\{\![\Gamma \vdash M \square t]\!\}$.

1. $\{\![\emptyset \vdash \alpha \square t]\!\} = \llbracket \emptyset \vdash \alpha \square t \rrbracket = \llbracket \alpha \rrbracket$;
2. $\{\![\Gamma \vdash H^{\mathtt{H}} \square t]\!\} = 1^{\llbracket \Gamma \vdash H \,:\, \tau \rrbracket}$;
3. $\{\![\Gamma \vdash \langle M_1, M_2 \rangle^{\mathtt{H}} \square t]\!\} =$ $\boldsymbol{\nabla}_{\llbracket \Gamma \vdash V \,:\, \rho \rrbracket} * (\{\![\Gamma \vdash M_1 \square t_1]\!\} \otimes \{\![\Gamma \vdash M_2 \square t_2]\!\})$;
4. $\{\![\Gamma \vdash Proj_1^{\mathtt{H}} M \square t]\!\} =$ $\{\![\Gamma \vdash M \square t_1]\!\} * (1^{\llbracket y_1 : \tau_1 \vdash U_1 \,:\, \sigma_1 \rrbracket} \otimes \boldsymbol{!}_{\llbracket y_2 : \tau_2 \vdash U_2 \,:\, \sigma_2 \rrbracket})$;
5. $\{\![\Gamma \vdash \lambda^{\mathtt{H}} x : \varrho . M \square t]\!\} =$ $\Lambda^{\mathtt{H}}_{\llbracket \Gamma \vdash V \,:\, \rho \rrbracket} \{\![\Gamma, x : \varrho \vdash M \square t']\!\}$;
6. $\{\![\Gamma \vdash M_2 @^{\mathtt{H}} M_1 \square t]\!\} =$ $(\boldsymbol{\pi}_{\llbracket \Gamma \rrbracket} \triangleleft (\{\![\Gamma \vdash M_2 \square t_2]\!\} \otimes \{\![\Gamma \vdash M_1 \square t_1]\!\}))*$ $Heval_{\llbracket \tau_1 \rrbracket, \llbracket y_2 : \tau_2 \vdash U_2 y_2 \,:\, \sigma_2 \rrbracket}$
7. $\{\![\Gamma \vdash M_1 * M_2 \square t]\!\} =$ $(\boldsymbol{\pi}_{\llbracket \Gamma \rrbracket} \triangleleft (\boldsymbol{\dagger}_{\llbracket \Gamma \rrbracket} \otimes \{\![\Gamma \vdash M_1 \square t_1]\!\}))*$ $((\{\![\Gamma \vdash M_2 \square t_2]\!\} \otimes 1_{\llbracket \tau_1 \rrbracket}) \cdot$ $Veval_{\llbracket \tau_1 \rrbracket, \llbracket y_2 : \tau_2 \vdash U_2 y_2 \,:\, \sigma_2 \rrbracket})*$ $(Heval_{\llbracket \tau_1 \rrbracket, \llbracket z_2 : \rho_2 \vdash G_2 z_2 \,:\, \sigma_2 \rrbracket} \cdot Deval_{\llbracket \tau_1 \rrbracket, \llbracket \sigma_2 \rrbracket})$

i $\{\![\Gamma, x : \sigma \vdash M \square t]\!\} = \{\![\Gamma \vdash M \square t]\!\} \otimes \boldsymbol{\phi}_{\llbracket \sigma \rrbracket}$;

ii $\{\![\Gamma_1, x_2 : \varrho_2, x_1 : \varrho_1, \Gamma_2 \vdash M \square t]\!\} =$ $(1_{\llbracket \Gamma_1 \rrbracket} \otimes \boldsymbol{\sigma}_{\llbracket \varrho_2 \rrbracket, \llbracket \varrho_1 \rrbracket} \otimes 1_{\llbracket \Gamma_2 \rrbracket}) \triangleleft$ $\{\![\Gamma_1, x_1 : \varrho_1, x_2 : \varrho_2, \Gamma_2 \vdash M \square t]\!\}$;

iii $\llbracket \Gamma, x : \varrho \vdash M \ : \ t \rrbracket =$ $(\boldsymbol{\nabla}_{\llbracket \Gamma, x : \varrho \rrbracket} * 1_{\llbracket \Gamma, x : \varrho \rrbracket \otimes \llbracket \Gamma, x : \varrho \vdash H \,:\, \tau \rrbracket} *$ $(1_{\llbracket \Gamma \rrbracket} \otimes \boldsymbol{\tau}_{\llbracket \varrho \rrbracket} \otimes 1_{\llbracket \tau \rrbracket})) \cdot \llbracket \Gamma, x : \varrho \vdash M \ : \ t' \rrbracket$;

Notice that the categorical semantics is defined by induction on the proof of each type judgment and therefore its argument should be more precisely a derivation and not just a type judgment. However, it is possible to see that the translation is independent from the chosen proof.

We consider as equivalent those terms that are mapped to the same cell. In this way, it is possible to show that the usual properties of λ-calculus, as e.g., the β-axiom, the η-axiom, the congruence rules ξ and ν, and the axioms for unit, product and projections still hold in each dimension (horizontal, vertical and diagonal).

Moreover, all the cells of the initial model are in the codomain of the mapping, i.e., each cell can be suitably expressed in the double λ-notation.

5. Typing the π-calculus

5.1. π-calculus

The π-calculus [15] is probably the most developed example of *mobile* process calculi. It is an extension of CCS [13] in which channel names can be passed in the communications. Therefore new channels can be dynamically created and the interconnection structure of processes can be dynamically reconfigured. These features allow for the encoding of higher-order communications [18] that can be used for modelling code migration.

Syntax. Although many versions of π-calculus have appeared in the literature, we consider the core of the *monadic* calculus under the *early* semantics only, which suffices to illustrate the expressiveness of our framework. The detailed treatment of other versions of the π-calculus is left for the full paper, as well as the modelling of other semantics studied in the literature (e.g., the *late* or the *open* semantics).

Let $\mathcal{N}$ be the set of *names*, ranged over by $x, y, \ldots$, then the π-calculus *agents*, ranged over by $P, Q, \ldots$, are defined by the syntax:

$$P ::= nil \mid (\nu y)P \mid \tau.P \mid x(y).P \mid \bar{x}y.P \mid P|P$$

where τ is the *silent* action. The *free names* of P are defined in the obvious way and denoted by $\mathrm{fn}(P)$. The free occurrences of y in P are *bound* in $x(y).P$ and $(\nu y)P$. As usual, π-agents are taken up to alpha-conversion of bound names and the notation $P[y/x]$ denotes the replacement of all the free occurrences of x in P with y (possibly after the renaming of bound names to avoid capture).

Early operational semantics. The *actions* that agents can perform are defined by the syntax:

$$\mu ::= \tau \mid xy \mid \bar{x}y \mid \bar{x}(y)$$

denoting respectively *synchronization*, *input*, *free output* and *bound output*. The name x is the *subject* and y is the *object* (i.e., the transmitted value) of the actions. The name y is bound in $\bar{x}(y)$, but it is free in both xy and $\bar{x}y$, whereas x is always free. As usual, we denote by $\mathrm{n}(\mu)$, $\mathrm{bn}(\mu)$ and $\mathrm{fn}(\mu)$ respectively the *names*, the *bound names* and the *free names* of μ, with $\mathrm{n}(\mu) = \mathrm{bn}(\mu) \cup \mathrm{fn}(\mu)$.

$$[\textsc{tau}]\quad \tau.P \xrightarrow{\tau} P,$$

$$[\textsc{in}]\quad x(y).P \xrightarrow{xz} P[z/y]$$

$$[\textsc{out}]\quad \bar{x}y.P \xrightarrow{\bar{x}y} P$$

$$[\textsc{par}]\quad \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \text{ if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$[\textsc{comm}]\quad \frac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{xy} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$[\textsc{res}]\quad \frac{P \xrightarrow{\mu} P'}{(\nu y)P \xrightarrow{\mu} (\nu y)P'} \text{ if } y \notin \mathrm{n}(\mu)$$

$$[\textsc{open}]\quad \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(y)} P'} \text{ if } y \neq x$$

$$[\textsc{close}]\quad \frac{P \xrightarrow{\bar{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P|Q \xrightarrow{\tau} (\nu z)(P'|Q')} \text{ if } z \notin \mathrm{fn}(Q)$$

Table 2. Early operational semantics.

The transition system representing the *early operational semantics* of the calculus under consideration is defined by the axioms and inference rules of Table 2 (the rules symmetric to [PAR], [COMM], and [CLOSE], are omitted).

Driven by the operational intuition, we briefly comment on the syntax of π-calculus. The output prefix $\bar{x}y.P$ states that the value y is transmitted along the channel x. In the input prefix $x(y).P$ the name x still represents the channel where the communication takes place, but y is instead a formal variable, that can be used in P and that must be instantiated by the actual value when either the input transition is performed (see rule [IN]) or input and output transition are synchronized (see rules [COMM] and [CLOSE]). Notice that since we are considering the early semantics, although the input prefix has the form $x(y).\_$, the associated transition is labelled with xz. Indeed y is a formal parameter, and z is the actual value. Also *name extrusion* is allowed (see rule [OPEN]) and therefore a corresponding label $\bar{x}(z)$ for the bound output must be used, so that previously restricted names can be communicated to the environment and received by other agents (rule [CLOSE]).

Since in the input transition it is not possible to know in advance which name will be received, it follows that even

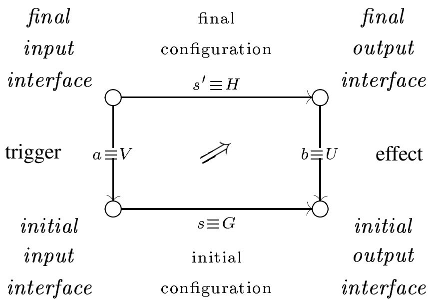

Figure 10. A "reversed" tile.

simple agents can perform an infinite number of transitions (i.e., the transition system is not finitely branching).

In the next section we show that the double λ-notation discussed in Section 4.1 can provide a framework where several aspects of name handling are managed.

5.2. The type system

To represent the π-calculus in the double λ-notation we find it more convenient to reverse the direction of vertical arrows. Henceforth, according to this choice, tiles must be interpreted as in Figure 10. Indeed this allows us to use the discharger ! (see Appendix A.2) for dealing with name creation (since the computation grows upwards instead than downwards).

The idea is then to define a signature for the πcalculus such that each proof of the transition $P \xrightarrow{\mu} Q$ is represented as a tile term $\Gamma \vdash M \,\square\, [\![Q]\!] \xrightarrow[{[\![\mu]\!]}]{id_e} [\![P]\!]$ (for suitable encondings of agents and actions).

Interfaces. The type constants on which the type system for the π-calculus is defined in the double λ-notation are $name$ and $proc$ (associated to names and processes respectively). To shorten the notation we will often write n and p in place of $name$ and $proc$ respectively.

Configurations. The horizontal term constants yielding the syntactic structure of π-agents are the following

$$\begin{array}{rcl} \mathrm{nil} & : & proc, \\ \mathrm{in} & : & name \to (name \to proc) \to proc, \\ \mathrm{out} & : & \langle name, name \rangle \to proc \to proc, \\ \mathrm{tau} & : & proc \to proc, \\ \mathrm{par} & : & \langle proc, proc \rangle \to proc, \\ \nu & : & (name \to proc) \to proc, \\ c & : & name. \end{array}$$

Here c is a name constant which turns out to be useful in the tile constant for the [CLOSE] rule. The translation of π-agents into horizontal terms (all of type $proc$) is inductively defined in Table 3.

$$\begin{array}{rcl} [\![nil]\!] & = & \mathrm{nil} \\ [\![x(y).P]\!] & = & \mathrm{in}\ x\ \lambda y : n.[\![P]\!] \\ [\![\bar{x}y.P]\!] & = & \mathrm{out}\ \langle x, y \rangle\ [\![P]\!] \\ [\![\tau.P]\!] & = & \mathrm{tau}\ [\![P]\!] \\ [\![P_1 | P_2]\!] & = & \mathrm{par}\ \langle [\![P_1]\!], [\![P_2]\!] \rangle \\ [\![(\nu x)P]\!] & = & \nu\ \lambda x : n.[\![P]\!] \end{array}$$

Table 3. Interpretation of π-agents.

Observations. The vertical term constants yielding the observations of π-agents are the following (with the obvious correspondence with the labels μ of the transition system):

$$\begin{array}{rcl} \mathbf{In} & : & \langle name, name \rangle \to proc \to proc, \\ \mathbf{Out} & : & \langle name, name \rangle \to proc \to proc, \\ \mathbf{bOut} & : & \langle name, name \rangle \to proc \to proc, \\ \mathbf{Tau} & : & proc \to proc. \end{array}$$

Tiles. The tile term constants are listed in Table 4 (first block) using a slightly different, more linear, notation than the one described in Section 4.1, that has been designed to be more natural for interactive calculi. Hence, a generic contour type

$$H : \tau \xrightarrow[\lambda y : \tau . U' : \tau \to \sigma]{V : \rho} \lambda z : \rho . G' : \rho \to \sigma$$

becomes written as $G'[V/z] \Rightarrow U'[H/y] : \sigma$ where the occurrences of H and V after substitution are written inside square brackets to separate terms that live in orthogonal dimensions. This is consistent with the choice of reversing the direction of vertical arrows as explained at the beginning of this section. See also Figure 10 for the interpretation of the symbol $\Rightarrow$.

Since τ and ρ can be the product of smaller types, the pairs of square brackets where each argument of the tuple is inserted are labelled by the position in the tuple of the argument itself. For example, $f([t]_1, [t]_2, [t]_1)$ represents a context with two arguments that is applied to two instances of the same term $t : \tau$, and where the first instance is used twice. This is clearly different from $f([t]_1, [t]_2, [t]_2)$, where the second instance is used twice. They are both different from $f([t]_1, [t]_1, [t]_1)$. In fact, in the first two cases the interface is typed $\tau \times \tau$, whilst in the third case is typed τ. Furthermore, in the first case the third argument of f is an instance of the first element of the interface pair, while in the second case it is an instance of the second element. Therefore the notation $f([t], [t], [t])$ might be ambiguous (because

$$
\begin{array}{rclcl}
q : p & \vdash & \mathtt{Tau}@^{\lhd}q & \square & \mathrm{tau}\,[q] \;\Rightarrow\; \mathbf{Tau}\,[q] \mathbin{\boldsymbol{:}} p \\
x,z:n,\; q:n\to p & \vdash & \mathtt{Input}@^{\lhd}\langle x,z\rangle @^{\lhd}q & \square & \mathrm{in}\,x\,[q] \;\Rightarrow\; \mathbf{In}\,\langle x,z\rangle\,[qz] \mathbin{\boldsymbol{:}} p \\
x,y:n,\; q:p & \vdash & \mathtt{Output}@^{\lhd}\langle x,y\rangle @^{\lhd}q & \square & \mathrm{out}\,\langle x,y\rangle\,[q] \;\Rightarrow\; \mathbf{Out}\langle x,y\rangle\,[q] \mathbin{\boldsymbol{:}} p \\
x,y:n,\; q_1,q_2:p & \vdash & \mathtt{Par}@^{\lhd}\langle x,y\rangle @^{\lhd}\langle q_1,q_2\rangle & \square & \mathrm{par}\,\langle[\mathbf{Out}\,\langle x,y\rangle\,q_1]_1,[q_2]_2\rangle \;\Rightarrow\; \mathbf{Out}\,\langle x,y\rangle\,[\mathrm{par}\langle q_1,q_2\rangle] \mathbin{\boldsymbol{:}} p \\
x,y:n,\; q_1,q_2:p & \vdash & \mathtt{Comm}@^{\lhd}\langle x,y\rangle @^{\lhd}\langle q_1,q_2\rangle & \square & \mathrm{par}\,\langle[\mathbf{Out}\,\langle x,y\rangle\,q_1]_1,[\mathbf{In}\,\langle x,y\rangle\,q_2]_2\rangle \;\Rightarrow\; \mathbf{Tau}\,[\mathrm{par}\langle q_1,q_2\rangle] \mathbin{\boldsymbol{:}} p \\
x,z:n,\; q:n\to p & \vdash & \mathtt{Res}@^{\lhd}\langle x,z\rangle @^{\lhd}q & \square & \nu\,[\lambda y:n.\mathbf{Out}\,\langle x,z\rangle\,qy] \;\Rightarrow\; \mathbf{Out}\,\langle x,z\rangle\,[\nu q] \mathbin{\boldsymbol{:}} p \\
x,z:n,\; r:n\to(n\times p) & \vdash & \mathtt{Open}@^{\lhd}\langle x,z\rangle @^{\lhd}r & \square & \nu\,[\lambda y:n.\mathbf{Out}\,\langle x,\mathit{Proj}_1(ry)\rangle\,\mathit{Proj}_2(ry)] \;\Rightarrow\; \mathbf{bOut}\,\langle x,\mathit{Proj}_1[rz]\rangle\,\mathit{Proj}_2[rz] \mathbin{\boldsymbol{:}} p \\
x,z:n,\; q_1,q_2:p & \vdash & \mathtt{Cpar}@^{\lhd}\langle x,z\rangle @^{\lhd}\langle q_1,q_2\rangle & \square & \mathrm{par}\,\langle[\mathbf{bOut}\,\langle x,z\rangle\,q_1]_1,[\mathbf{In}\,\langle x,z\rangle\,q_2]_2\rangle \;\Rightarrow\; \mathbf{Tau}\,[\mathrm{par}\langle q_1,q_2\rangle] \mathbin{\boldsymbol{:}} p \\
q:n\to p & \vdash & \mathtt{Close}@^{\lhd}q & \square & [\lambda z:n.\mathbf{Tau}(qz)]c \;\Rightarrow\; \mathbf{Tau}\,[\nu q] \mathbin{\boldsymbol{:}} p
\end{array}
$$

$$
\begin{array}{rl}
[\![\textsc{tau}]\!] = & \Gamma\vdash [\![P]\!]^{\mathtt{H}} * \mathtt{Tau}\,\square\,[\![\tau.P]\!] \Rightarrow\; \mathbf{Tau}[[\![P]\!]] \mathbin{\boldsymbol{:}} p \\
[\![\textsc{in}]\!] = & \Gamma\vdash (\lambda y:n.[\![P]\!])^{\mathtt{H}} * \mathtt{Input}@^{\lhd}\langle x,z\rangle\,\square\,[\![x(y).P]\!] \Rightarrow\; \mathbf{In}\langle x,z\rangle[[\![P[z/y]]\!]] \mathbin{\boldsymbol{:}} p \\
[\![\textsc{out}]\!] = & \Gamma\vdash [\![P]\!]^{\mathtt{H}} * \mathtt{Output}@^{\lhd}\langle x,y\rangle\,\square\,[\![\bar{x}y.P]\!] \Rightarrow\; \mathbf{Out}\langle x,y\rangle[[\![P]\!]] \mathbin{\boldsymbol{:}} p \\
\text{if } \Gamma\vdash M\,\square\,[\![P]\!] \Rightarrow \mathbf{Out}\langle x,y\rangle[[\![P']\!]] \mathbin{\boldsymbol{:}} p \text{ then} \quad [\![\textsc{par}]\!]\,M = & \Gamma\vdash \langle M,[\![Q]\!]^{\mathtt{H}}\rangle^{\mathtt{H}} * \mathtt{Par}@^{\lhd}\langle x,y\rangle\,\square\,[\![P|Q]\!] \;\Rightarrow\; \mathbf{Out}\,\langle x,y\rangle\,[[\![P'|Q]\!]] \mathbin{\boldsymbol{:}} p \\
\multicolumn{2}{l}{\text{if } \Gamma\vdash M_1\,\square\,[\![P]\!] \Rightarrow \mathbf{Out}\langle x,y\rangle[[\![P']\!]] \mathbin{\boldsymbol{:}} p \;\text{ and }\; \Gamma\vdash M_2\,\square\,[\![Q]\!] \Rightarrow \mathbf{In}\langle x,y\rangle[[\![Q']\!]] \mathbin{\boldsymbol{:}} p \;\text{ then}} \\
[\![\textsc{comm}]\!]\,M_1\,M_2 = & \Gamma\vdash \langle M_1,M_2\rangle^{\mathtt{H}} * \mathtt{Comm}@^{\lhd}\langle x,y\rangle\,\square\,[\![P|Q]\!] \;\Rightarrow\; \mathbf{Tau}[[\![P'|Q']\!]] \mathbin{\boldsymbol{:}} p \\
\text{if } \Gamma,y:n\vdash M\,\square\,[\![P]\!] \Rightarrow \mathbf{Out}\langle x,z\rangle[[\![P']\!]] \mathbin{\boldsymbol{:}} p \text{ then} \quad [\![\textsc{res}]\!]\,M = & \Gamma\vdash (\lambda^{\mathtt{H}}y:n.M) * \mathtt{Res}@^{\lhd}\langle x,z\rangle\,\square\,[\![(\nu y)P]\!] \;\Rightarrow\; \mathbf{Out}\,\langle x,z\rangle[[\![(\nu y)P']\!]] \mathbin{\boldsymbol{:}} p \\
\text{if } \Gamma,y:n\vdash M\,\square\,[\![P]\!] \Rightarrow \mathbf{Out}\langle x,y\rangle[[\![P']\!]] \mathbin{\boldsymbol{:}} p \text{ then} [\![\textsc{open}]\!]\,M = & \Gamma,z:n\vdash (\lambda^{\mathtt{H}}y:n.M) * \mathtt{Open}@^{\lhd}\langle x,z\rangle\,\square\,[\![(\nu y)P]\!] \;\Rightarrow\; \mathbf{bOut}\langle x,z\rangle[[\![P'[z/y]]\!]] \mathbin{\boldsymbol{:}} p \\
\multicolumn{2}{l}{\text{if } \Gamma,z:n\vdash M_1\,\square\,[\![P]\!] \Rightarrow \mathbf{bOut}\langle x,z\rangle[[\![P']\!]] \mathbin{\boldsymbol{:}} p \;\text{ and }\; \Gamma,z:n\vdash M_2\,\square\,[\![Q]\!] \Rightarrow \mathbf{In}\langle x,z\rangle[[\![Q']\!]] \mathbin{\boldsymbol{:}} p \;\text{ and }\; \Gamma\vdash[\![Q]\!] : p \;\text{ then}} \\
[\![\textsc{close}]\!]\,M_1\,M_2 = & \Gamma\vdash (\lambda^{\mathtt{H}}z\!:\!n.\langle M_1,M_2\rangle^{\mathtt{H}} * \mathtt{CPar}@^{\lhd}\langle x,z\rangle) * \mathtt{Close}\,\square\,[\![P|Q]\!] \Rightarrow \mathbf{Tau}[[\![(\nu z)P'|Q']\!]] \mathbin{\boldsymbol{:}} p
\end{array}
$$

Table 4. Tile term constants for the π-calculus.

the interface is not recoverable) and is allowed only if there is a unique argument as in $g([t])$, avoiding the subscript $_{\text{-}1}$. Discarded arguments of the interface (i.e., not used in the context) are made explicit by extending the tuple with the term $![t]_i$ (if t is the i-th argument of the interface and it is discarded). We remark that this notation abstracts from the formal variables y and z. Moreover, variables in the global environment Γ can be referred to without specifying their position in the interface, thus avoiding the square brackets.

Finally notice that in Table 4 we type the tile term constants under non-empty environments. This is just a shorthand that avoids unnecessary redundancy in the notation. For example, the constant for the silent action prefix should be given as:

$$
\begin{array}{ll}
\emptyset\vdash \mathtt{Tau}\,\square\,\lambda q:p.\mathrm{tau}\;([\lambda q':p.q']q) & \Rightarrow \\
\qquad \lambda q:p.\mathbf{Tau}\;([\lambda q':p.q']q) & \mathbin{\boldsymbol{:}} p\to p
\end{array}
$$

This expression is hard to read (and the other constants are still more complicated), but fortunately, the diagonal application allows simplifying the presentation as in Table 4.

We briefly comment on tile term constants. The first and the third constants, `Tau` and `Out`, are very simple, since they generate effects (**Tau** and **Out** $\langle x,y\rangle$ respectively) that are specular to the operators that prefix the configurations. The constant `Input` is more interesting, since in the initial configuration the prefix is abstracted w.r.t. the name to be received (indeed q has type $\mathit{name}\to\mathit{proc}$), whilst the effect is instantiated with a name z taken from the environment, according to the early semantics of π-calculus.

The tiles `Par` and `Res` just propagate output observations through par and ν. Analogous tiles are needed for propagating input, bound output and silent observations.

The constants `Comm` and `Cpar` are very similar: Both synchronize two complementary observations, but `Cpar` deals with bound output and, together with `Close`, models the reception of extruded names.

The more interesting tile is probably the constant `Open` that models name extrusion. To better understand how it works, a simple example is fully illustrated at the end of this section.

As for the operational semantics rules, we have omitted the tile term constants symmetric to `Par`, `Comm` and `Cpar`.

Transitions as double terms. We are now ready to define the correspondence between the (proofs of the) transitions and the well-typed tile terms over the signature for the π-calculus. The precise mapping is given in the second block of Figure 4. Due to space limitations, it is represented as a schema parametrized w.r.t. the terms associated to the premises of each inference rule in Table 2. For example, since the application of rule [COMM] requires two proofs $\mathcal{P}_1$ and $\mathcal{P}_2$ for the transitions $P \xrightarrow{\bar{x}y} P'$ and $Q \xrightarrow{xy} Q'$, then the associated schema $[\![\textsc{comm}]\!]$ has been made parametric w.r.t. the terms M_1 and M_2 associated to such transition proofs.

The execution of an action prefix is obtained by horizontally composing a suitable identity for the idle agent P (abstracted w.r.t. the name y in the case of input) with the tile constant associated to the prefix kind (`Tau` for $\tau.\_$, `Input` for input prefix and `Output` for output prefix).

Similarly, rules [PAR], [COMM] and [RES] require just the horizontal composition of the "premises" with the corresponding constant `Par`, `Comm` and `Res` respectively. In the table, we provided the mapping for [PAR] and [RES] only in the case of an **Out** action. The mapping for an **In** action is analogous. For propagating bound outputs, we must ensure in addition that $z \notin \mathrm{fv}(Q)$, as in [PAR]. This can be done by requiring that $\Gamma \vdash [\![Q]\!] : p$ and

$$\Gamma, z : n \vdash M \,\Box\, [\![P]\!] \;\Rightarrow\; \mathbf{bOut}\;\langle x,z\rangle\;[[\![P']\!]] \;\mathbf{:}\; p.$$

As an example of the extrusion mechanism, we derive now the tile term corresponding to the π-calculus transition $(\nu y)\bar{x}y.nil \xrightarrow{\bar{x}(y)} nil$. The proof of this transition is given by instantiating (with nil) the [OUT] axiom, obtaining $\bar{x}y.nil \xrightarrow{\bar{x}(y)} nil$, and by applying the [OPEN] rule. In the tile approach, we start from the `Output` tile term constant, which we rewrite here in the ordinary form to emphasize the contour type.

$$x, y : n, q : p \vdash \mathtt{Output@}^{\triangleleft}\langle x,y\rangle\mathtt{@}^{\triangleleft}q\Box \qquad (9)$$
$$q \xrightarrow[\lambda q'.\mathbf{Out}\langle x,y\rangle q']{q} \lambda q'.\mathrm{out}\langle x,y\rangle q'$$

The translation $[\![\mathrm{OUT}]\!]$ of the instantiated axiom can be found in the second block of Table 4.

$$x, y : n \vdash \mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)\Box \qquad (10)$$
$$\mathrm{nil} \xrightarrow[\lambda q.\mathbf{Out}\langle x,y\rangle q]{\star} \lambda i : \epsilon.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

Notice that this tile term can be obtained by horizontal composition (see rule 7 of both Section 4.1 and 4.2) of the tile term $x, y : n \vdash \mathrm{nil}\Box\;\mathrm{nil} \xrightarrow[\lambda q.q]{\star} \lambda i : \epsilon.\mathrm{nil}$ and of the tile term (9). The term constant `Open` can be written as

$$x, z : n, r : n \to (n \times p) \vdash \mathtt{Open@}^{\triangleleft}\langle x,z\rangle\mathtt{@}^{\triangleleft}r\Box \qquad (11)$$
$$rz \xrightarrow[\lambda\langle w,q\rangle.\mathbf{bOut}\langle x,w\rangle q]{\lambda y.\mathbf{Out}\langle x, Proj_1(ry)\rangle Proj_2(ry)} \nu$$

To understand why a variable r with an unusual type as $n \to (n \times p)$ is needed, we should try to apply horizontal abstraction (see rule 5 of both Section 4.1 and 4.2) to the tile term (10). This is necessary, since the restriction operator ν requires an argument of type $n \to p$. A straightforward application of horizontal abstraction would be possible if (as it is the case for [RES]) the object of the action would not include the variable we want to abstract from. In fact, the applicability condition of horizontal abstraction requires $y \notin \mathrm{fv}(V,U)$. We have no problem with V (it is $\star$), but to eliminate y from $U = \lambda q.\mathbf{Out}\langle x,y\rangle q$ we must apply the auxiliary rule iii, thus obtaining

$$x, y : n \vdash \mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)\Box$$
$$\langle y, \mathrm{nil}\rangle \xrightarrow[\lambda\langle y',q\rangle.\mathbf{Out}\langle x,y'\rangle q]{\star} \lambda i : \epsilon.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

We can now abstract w.r.t. the name y

$$x : n \vdash \lambda^{\mathtt{H}}y.\mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)\Box \quad (12)$$
$$\lambda y.\langle y, \mathrm{nil}\rangle \xrightarrow[\lambda r:n\to(n\times p).\lambda y'.U]{\star} \lambda i : \epsilon.\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

with $U = \mathbf{Out}\langle x, Proj_1(ry')\rangle Proj_2(ry')$. As suggested in the second block of Table 4, the final tile term for our example is obtained by horizontally composing tile terms (12) and (11). Notice however that we first have to create a fresh name. This can be accomplished by the auxiliary rule i.

$$x, z : n \vdash \lambda^{\mathtt{H}}y.\mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)\Box$$
$$\lambda y.\langle y, \mathrm{nil}\rangle \xrightarrow[\lambda r:n\to(n\times p).\lambda y'.U]{\star} \lambda i : \epsilon.\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

The horizontal composition yields

$$x, z : n \vdash (\lambda^{\mathtt{H}}y.\mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)) * \mathtt{Open@}^{\triangleleft}\langle x,z\rangle\Box$$
$$(\lambda y.\langle y, \mathrm{nil}\rangle)z \xrightarrow[\lambda\langle y',q\rangle.\mathbf{bOut}\langle x,y'\rangle q]{\star} \lambda i : \epsilon.\nu\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

which, by β-conversion and by applying the auxiliary rule iii backwards, becomes

$$x, z : n \vdash (\lambda^{\mathtt{H}}y.\mathrm{nil} * (\mathtt{Output@}^{\triangleleft}\langle x,y\rangle)) * \mathtt{Open@}^{\triangleleft}\langle x,z\rangle\Box$$
$$\mathrm{nil} \xrightarrow[\lambda q.\mathbf{bOut}\langle x,z\rangle q]{\star} \lambda i : \epsilon.\nu\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil}$$

whose contour type, written in the linear notation is

$$\begin{aligned}(\lambda i : \epsilon.\nu\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil})[\star] &\;\Rightarrow\; (\lambda q.\mathbf{bOut}\langle x,z\rangle q)[\mathrm{nil}]\\ \equiv\; \nu\lambda y.\mathrm{out}\langle x,y\rangle\mathrm{nil} &\;\Rightarrow\; \mathbf{bOut}\langle x,z\rangle[\mathrm{nil}]\\ \equiv\; [\![(\nu y)\bar{x}y.nil]\!] &\;\Rightarrow\; \mathbf{bOut}\langle x,z\rangle[[\![nil]\!]]\end{aligned}$$

This concludes the example.

6. Conclusion

Pursuing the analogy with simply typed λ-calculus and CCC's, we have introduced the new notion of *cartesian closed double categories* (CCDC) and a corresponding λ-notation. We hope that this approach provides a natural framework for the study of communicating systems with name passing and creation.

It is worth remarking that only object abstraction is allowed according to our definition of CCDC, because otherwise our approach based on a double adjunction would not work. Nevertheless, since objects naturally model name tuples, and mobile calculi also abstract from names, we are confident that our framework is expressive enough to serve as a basis for most of the proposed calculi. The case study of (synchronous) π-calculus illustrates some basic mechanisms of the new framework.

Acknowledgements. We want to thank Mariangiola Dezani, Gianluigi Ferrari, Vladimiro Sassone and Paolo Baldan for helpful comments.

This research has been supported by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; by Esprit Working Groups *CONFER2* and *COORDINA*; and by MURST project *Tecniche Formali per Sistemi Software*.

References

[1] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Comput. Sci.*, 96:217–248, 1992.

[2] R. Bruni. *Tile logic for synchronized rewriting of concurrent systems*. PhD thesis, University of Pisa - Department of Computer Science, 1999.

[3] R. Bruni, J. Meseguer, and U. Montanari. Process and term tile logic. Technical Report SRI-CSL-98-06, SRI International, 1998. Also Technical Report TR-98-09, Department of Computer Science, University of Pisa.

[4] R. Bruni, J. Meseguer, and U. Montanari. Executable tile specifications for process calculi. In J.-P. Finance, editor, *Fundamental Approaches to Software Engineering*, volume 1577 of *LNCS*, pages 60–76. Springer Verlag, 1999.

[5] R. Bruni, J. Meseguer, and U. Montanari. Symmetric monoidal and cartesian double categories as a semantic framework for tile logic. *Mathematical Structures in Computer Science*, 1999. To appear.

[6] A. Corradini, G. Ferrari, and U. Montanari. Transition systems with algebraic structure as models of computations. In I. Guesserian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 185–222. Springer Verlag, 1990.

[7] G. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Information and Computation*, 1999. To appear.

[8] F. Gadducci and U. Montanari. Comparing logics for rewriting: Rewriting logic, action calculi and tile logic, 1999. Submitted for publication. Available at the URL http://www.di.unipi.it/~ugo/ABSTRACT.html.

[9] F. Gadducci and U. Montanari. The tile model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 1999. To appear.

[10] Y. Lafont. Equational reasoning with 2–dimensional diagrams. In H. Comon and J.-P. Jouannaud, editors, *Term Rewriting*, volume 909 of *LNCS*, pages 170–195. Springer Verlag, 1995.

[11] K. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In M. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *LNCS*, pages 526–539. Springer Verlag, 1990.

[12] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Comput. Sci.*, 96:73–155, 1992.

[13] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[14] R. Milner. Calculi for interaction. *Acta Inf.*, 33:707–737, 1996.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. Part I and II. *Information and Computation*, 100:1–77, 1992.

[16] J. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[17] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.

[18] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Laboratory for Foundations of Computer Science, Computer Science Department - Edinburgh University, 1993.

A. Cartesian closed categories and the λ-calculus

A.1. λ-calculus

To fix the notation, we recall the classical simply typed λ-calculus. In particular, we consider typed λ-terms in $\lambda^{\rightarrow,\times,\epsilon}$, i.e., with *product types* and *unit*. We refer to [16] for an extensive presentation of the subject.

A λ-*signature* Σ is a pair $\langle B, C\rangle$, where B is the set of *type constants* (i.e., the basic types allowed), and C is the set of (typed) *term constants*. The *types* are built over the type constants $b \in B$ using the following grammar:

$$\sigma ::= b \mid \epsilon \mid \sigma\times\sigma \mid \sigma\rightarrow\sigma$$

Since term constants must be typed, the set C is usually represented as a collection $\{c_1 : \sigma_1, ..., c_n : \sigma_n, ...\}$, where σ_i is the type of the term constant c_i, for $i = 1, ..., n, ...$.

A *term* (also λ-*term*) M over Σ with variables $x \in X$ can then be constructed accordingly to the syntax

$$\begin{array}{rcl} M & ::= & c \mid \star \mid x \mid \lambda x:\sigma.M \mid MM \mid \\ & & \langle M, M\rangle \mid \mathit{Proj}_1^{\sigma,\sigma} M \mid \mathit{Proj}_2^{\sigma,\sigma} M \end{array}$$

However, we are only interested in well-typed terms. Since terms can contain variables, the *type judgments* must be parametrized with respect to suitable *assignments* of types to the (free) variables. A generic assignment Γ has the form

$$\frac{c : \sigma \in C}{\emptyset \vdash c : \sigma} \qquad \frac{}{\emptyset \vdash \star : \epsilon} \qquad \frac{}{x : \sigma \vdash x : \sigma}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma, x : \tau \vdash M : \sigma} \qquad \frac{\Gamma_1, x : \tau, y : \rho, \Gamma_2 \vdash M : \sigma}{\Gamma_1, y : \rho, x : \tau, \Gamma_2 \vdash M : \sigma}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}$$

$$\frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathit{Proj}_1^{\sigma,\tau} M : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathit{Proj}_2^{\sigma,\tau} M : \tau}$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x : \tau . M : \tau \to \sigma} \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}$$

Table 5. Well-typed terms.

$\{x_1 : \sigma_1, ..., x_k : \sigma_k\}$. Since for simplicity we will consider cartesian closed categories with chosen products as suitable models, we assume that each assignment is a list, instead of a set, and that each variable can appear at most once in the list. We use the notation $\Gamma, x : \sigma$ to denote the assignment obtained by appending $x : \sigma$ at the end of Γ. Similarly, we will write also Γ_1, Γ_2 to state that the assignment is the concatenation of Γ_1 followed by Γ_2.

A type judgment has the form

$$\Gamma \vdash M : \sigma,$$

which means that the term M is well-typed with type σ under the assumption of Γ. The correct type judgments for terms over the signature $\langle B, C \rangle$ are those generated by applying the rules in Table 5.

We omit the axiomatization that identifies equivalent terms and the corresponding rules for reduction to normal form. Notice however that such identifications are exactly those that hold in the initial model category illustrated in Table 8. The idea is that such category has strings of types as objects (with monoidal composition $\otimes$ and unit e), and each type judgment $x_1 : \sigma_1, ..., x_n : \sigma_n \vdash M : \sigma$ is mapped into a suitable arrow from $\sigma_1 \otimes \cdots \otimes \sigma_n$ to σ.

A.2. Enriched monoidality

The category $\mathcal{C}$ has *chosen products* if a specific product diagram is given for each finite list of objects. This is often useful to reason about complex product expressions from a more intuitive perspective. For example, in $\mathbf{Set}$ one can imagine that the product of two sets A and B is just their cartesian product $A \times B$ with the obvious projections also if, in general, any other isomorphic set is as good as $A \times B$.

A classical result shows that "chosen" cartesianity can be expressed in terms of monoidal categories enriched with

$$\begin{aligned}
\gamma_{a\otimes b,c} &= (id_a \otimes \gamma_{b,c}); (\gamma_{a,c} \otimes id_b) \\
\gamma_{a,b}; \gamma_{b,a} &= id_{a\otimes b} \\
\nabla_{a\otimes b} &= (\nabla_a \otimes \nabla_b); (id_a \otimes \gamma_{a,b} \otimes id_b) \\
!_{a\otimes b} &= !_a \otimes !_b \\
\nabla_a; (1_a \otimes \nabla_a) &= \nabla_a; (\nabla_a \otimes 1_a) \\
\nabla_a; \gamma_{a,a} &= \nabla_a \\
\nabla_a; (1_a \otimes !_a) &= id_a \\
\nabla_e &= id_e = !_e
\end{aligned}$$

Table 6. Coherence axioms.

suitable natural transformations. In particular, we will consider the case in which the monoidal category is *strict* (i.e., the tensor product is associative and with unit element). Obviously, one could also adopt a more general view (e.g., by requiring tensor product to be associative only up to iso) but this introduces unnecessary and complicated details. Moreover, each non-strict monoidal category has a monoidal-equivalent strict monoidal category [10], which is surely more convenient for the analysis of computational models.

Definition 7 (SMC) *A strict monoidal category (*SMC*) is a category $\mathcal{C}$ together with a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ and a constant functor $e : \mathcal{C} \to \mathcal{C}$ such that $\otimes$ is associative and has the object $e \in \mathcal{C}$ as unit.*

Definition 8 (Chosen Cartesianity) *A* cartesian category with chosen products *is a tuple $(\mathcal{C}, \otimes, e, \gamma, \nabla, !)$, where*

- *$(\mathcal{C}, \otimes, e)$ is a strict monoidal category;*
- *γ is a natural transformation from $\otimes$ to $X; \otimes$, where $X : \mathcal{C} \times \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ is the functor that swaps its arguments, i.e., such that $X(f, g) = (g, f)$;*
- *∇ is a natural transformation from the identity on $\mathcal{C}$ to $\Delta; \otimes$, where $\Delta : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ is the diagonal functor that duplicates the argument, i.e., such that $\Delta(f) = (f, f)$;*
- *$!$ is a natural transformation from the identity to the constant functor e;*
- *finally, the natural transformations γ, ∇ and $!$ must satisfy the* coherence axioms *in Table 6.*

For example, given two objects a and b and two arrows $f : c \to a$ and $g : c \to b$, we have that $a \times b = a \otimes b$, $\Pi_1^{a\times b} = a \otimes !_b$, and $\langle f, g \rangle = \nabla_c; (f \otimes g)$.

Similarly, the exponent has an axiomatic definition, which is more suitable for readers not acquainted with category theory. This should hopefully emphasize the universal property of the exponent.

$$
\begin{tikzcd}
c \arrow[d, "h"'] \\
b^a
\end{tikzcd}
\qquad
\begin{tikzcd}
c \times a \arrow[r, "f"] \arrow[d, "h \times a"'] & b \\
b^a \times a \arrow[ur, "eval_{a,b}"'] &
\end{tikzcd}
$$

Figure 11. The universal property of exponent.

$$
\begin{tikzcd}
c \\
c' \arrow[u, "g"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{C}[c, b^a] \arrow[r, "\Lambda_c^{-1}"] \arrow[d, "g;\_"'] & \mathcal{C}[c \times a, b] \arrow[d, "g \times a;\_"] \\
\mathcal{C}[c', b^a] \arrow[r, "\Lambda_{c'}^{-1}"'] & \mathcal{C}[c' \times a, b]
\end{tikzcd}
$$

Figure 12. Naturality of Λ^{-1}.

Definition 9 (Exponent) *Let $\mathcal{C}$ be a cartesian category. Given two objects a and b of $\mathcal{C}$, the* exponent *of a and b is an object b^a together with a morphism $eval_{a,b} : b^a \times a \to b$ (called* evaluation map*) and for every object c an operation $\Lambda_c : \mathcal{C}[c \times a, b] \to \mathcal{C}[c, b^a]$ such that for all morphisms $f : c \times a \to b$, $h : c \to b^a$, the following equations hold:*

$$(\Lambda_c(f) \times a); eval_{a,b} = f \tag{13}$$

$$\Lambda_c((h \times a); eval_{a,b}) = h \tag{14}$$

Observe that Λ_c defines a bijection. Indeed, by (13) and (14), Λ_c^{-1} is the operation which takes every $h : c \to b^a$ to $\Lambda_c^{-1}(h) = (h \times a); eval_{a,b}$.

Notice also that if $\Lambda_c : \mathcal{C}[c \times a, b] \to \mathcal{C}[c, b^a]$ is a bijection and (13) holds, then (14) is necessarily true. [4] The following is thus an equivalent definition of exponent.

Proposition 2 (Exponent Revisited) *Let $\mathcal{C}$ be a cartesian category. Given two objects a and b of $\mathcal{C}$, the exponent of a and b is an object b^a together with a morphism $eval_{a,b} : b^a \times a \to b$, where for all morphisms $f : c \times a \to b$, there exists a unique arrow $h : c \to b^a$ such that the diagram in Figure 11 commutes.*

Then, Definition 1 can be reformulated by saying that a cartesian category $\mathcal{C}$ is cartesian closed if for every pair of objects a and b, there is an exponent.

The collection $\Lambda = \{\Lambda_c\}_{c \in \mathcal{C}}$ of isomorphisms defines a natural isomorphism from the (contravariant) functor

$$\mathcal{C}[\_ \times a, b] : \mathcal{C} \to \mathbf{Set}$$

to the (contravariant) functor $\mathcal{C}[\_, b^a] : \mathcal{C} \to \mathbf{Set}$. Indeed, for any $g : c' \to c$ and $f \in \mathcal{C}[c \times a, b]$ we have that

$$
\begin{aligned}
\Lambda_{c'}(g \times a; f) &= \Lambda_{c'}(g \times a; \Lambda_c(f) \times a; eval_{a,b}) \\
&= \Lambda_{c'}((g; \Lambda_c(f)) \times a; eval_{a,b}) \\
&= g; \Lambda_c(f)
\end{aligned}
$$

[4] To verify this simply take $f \in \mathcal{C}[c \times a, b]$ such that $h = \Lambda_c(f)$ and substitute h by $\Lambda_c(f)$ in (14).

$$
\frac{}{e \in M_B} \qquad \frac{b \in B}{b \in M_B} \qquad \frac{a, b \in M_B}{a \otimes b \in M_B} \qquad \frac{a, b \in M_B}{b^a \in M_B}
$$

Table 7. Free higher-order monoid.

Moreover, by defining $\Lambda^{-1} = (\_ \times a); eval_{a,b}$, then $\Lambda; \Lambda^{-1} = id$ by (13), and $\Lambda^{-1}; \Lambda = id$ by (14). Therefore Λ is a natural isomorphism. Also the converse is true, i.e., if for all objects a and b, there is an object b^a and a natural isomorphism $\Lambda : \mathcal{C}[\_ \times a, b] \to \mathcal{C}[\_, b^a]$ (the naturality of Λ^{-1} is expressed by the commutative diagram in Figure 12), then, we can define $eval_{a,b} = \Lambda_{b^a}^{-1}(id_{b^a})$ and prove that b^a is an exponent with $eval_{a,b}$ as evaluation map. Hence we have the following characterization of CCC's.

Proposition 3 (CCC Revisited II) *A cartesian category $\mathcal{C}$ is cartesian closed iff for all objects a and b there is an object b^a and a natural isomorphism $\Lambda : \mathcal{C}[\_ \times a, b] \to \mathcal{C}[\_, b^a]$.*

Notice that, by the properties of evaluation maps, it follows that for any $g : b \to b'$ in $\mathcal{C}$:

$$
\begin{aligned}
(g^a \times a); eval_{a,b'} &= (\Lambda_{b^a}(eval_{a,b}; g) \times a); eval_{a,b'} \\
&= eval_{a,b}; g.
\end{aligned}
$$

A.3. Categorical semantics for simply typed λ-calculus

Now that the subject of cartesian closed category has been extensively discussed, we are ready to present the categorical semantics for λ-terms over a given signature $\langle B, C \rangle$. The idea is to consider the cartesian closed category with chosen products freely generated from the term constants in C, and then to interpret the well-typed terms by structural induction on the associated type-proofs: The objects of the category are the elements of the higher-order monoid freely generated from the type constants in B (i.e., the monoid M_B whose elements are generated by the inference rules in Table 7), and each term constant $c : \sigma$ is mapped into a basic arrow $[\![c]\!] : e \to [\![\sigma]\!]$ (where $[\![\sigma]\!]$ is defined as below).

$[\![\emptyset \vdash c : \sigma]\!] = [\![c]\!]$
$[\![\emptyset \vdash \star : \epsilon]\!] = id_e$
$[\![x : \sigma \vdash x : \sigma]\!] = id_{[\![\sigma]\!]}$
$[\![\Gamma, x : \tau \vdash M : \sigma]\!] = [\![\Gamma \vdash M : \sigma]\!] \otimes !_{[\![\tau]\!]}$
$[\![\Gamma_1, y : \rho, x : \tau, \Gamma_2 \vdash M : \sigma]\!] =$
$(id_{[\![\Gamma_1]\!]} \otimes \gamma_{[\![\rho]\!],[\![\tau]\!]} \otimes id_{[\![\Gamma_2]\!]});$
$[\![\Gamma_1, x : \tau, y : \rho, \Gamma_2 \vdash M : \sigma]\!]$
$[\![\Gamma \vdash \langle M, N\rangle : \sigma \times \tau]\!] = \langle [\![\Gamma \vdash M : \sigma]\!], [\![\Gamma \vdash N : \tau]\!]\rangle$
$[\![\Gamma \vdash Proj_1^{\sigma,\tau} M : \sigma]\!] = [\![\Gamma \vdash M : \sigma \times \tau]\!]; (id_{[\![\sigma]\!]} \otimes !_{[\![\tau]\!]})$
$[\![\Gamma \vdash Proj_2^{\sigma,\tau} M : \tau]\!] = [\![\Gamma \vdash M : \sigma \times \tau]\!]; (!_{[\![\sigma]\!]} \otimes id_{[\![\tau]\!]})$
$[\![\Gamma \vdash \lambda x : \tau.M : \tau \to \sigma]\!] = \Lambda_{[\![\Gamma]\!]}[\![\Gamma, x : \tau \vdash M : \sigma]\!]$
$[\![\Gamma \vdash MN : \sigma]\!] =$
$\langle [\![\Gamma \vdash M : \tau \to \sigma]\!], [\![\Gamma \vdash N : \tau]\!]\rangle; eval_{[\![\tau]\!],[\![\sigma]\!]}$

Table 8. The categorical semantics of terms.

We first map types into objects as follows:

$$\begin{array}{rcl} [\![\epsilon]\!] & = & e \\ [\![b]\!] & = & b \\ [\![\sigma \times \tau]\!] & = & [\![\sigma]\!] \otimes [\![\tau]\!] \\ [\![\tau \to \sigma]\!] & = & [\![\sigma]\!]^{[\![\tau]\!]} \end{array}$$

Then, we generalize such translation to assignments:

$$\begin{array}{rcl} [\![\emptyset]\!] & = & e \\ [\![\Gamma, x : \sigma]\!] & = & [\![\Gamma]\!] \otimes [\![\sigma]\!] \end{array}$$

Finally, the type judgments for terms are mapped into arrows of the free CCC as defined in Table 8. It is possible to see that different proofs for the same type judgment yield the same result.

B. Axiomatization of cartesian double categories

In the one-dimensional case, we have seen that cartesianity can be obtained via suitable natural transformations. An analogous result holds for double cartesianity, but since double categories have two compositions, the naturality of a transformation between double functors is more involved. For example, following the internal construction approach (double categories are **Cat**-objects in **Cat**), an internal natural transformation is an arrow in **Cat** which verifies the naturality conditions w.r.t. one composition and which is functorial w.r.t. the other composition.

B.1. Natural double transformations

We recall from [5] the following definition of *natural double transformation* (originally called *generalized natural transformation*), referring to that paper and to [2] for an extensive explanation of the subject.

As a matter of notation, we call natural *comp*-transformation a transformation which satisfies the naturality requirement w.r.t. the composition operator *comp* and which is functorial w.r.t. the other composition operator. A natural double transformation is the key to express relationships between natural $*$-transformations and natural $\cdot$-transformations.

Definition 10 (Double Transformation) *Let $\mathcal{D}$ and $\mathcal{E}$ be double categories. Given a 4-tuple $(F_{00}, F_{10}, F_{01}, F_{11})$ of double functors from $\mathcal{D}$ to $\mathcal{E}$, a* natural double transformation *is a 5-tuple $(\alpha_{0\_}, \alpha_{1\_}, \beta_{\_0}, \beta_{\_1}, \eta)$, which is pictured as the cell*

$$\begin{tikzcd} F_{00} \arrow[r, "\alpha_{0\_}"] \arrow[d, "\beta_{\_0}"'] \arrow[dr, phantom, "\eta"] & F_{01} \arrow[d, "\beta_{\_1}"] \\ F_{10} \arrow[r, "\alpha_{1\_}"'] & F_{11} \end{tikzcd}$$

where:

- *for $i = 0, 1$, the symbol $\alpha_{i\_}$ denotes a natural $*$-transformation from F_{i0} to F_{i1}, i.e., $\alpha_{i\_}$ is also a functor from the category $\mathcal{V}$ of vertical arrows of $\mathcal{D}$ (i.e., the objects of $\mathcal{D}^*$) to the category $\mathcal{E}^{\cdot}$;*
- *for $i = 0, 1$, the symbol $\beta_{\_i}$ denotes a natural $\cdot$-transformation from F_{0i} to F_{1i}, i.e., $\beta_{\_i}$ defines a functor from the category $\mathcal{H}$ of horizontal arrows of $\mathcal{D}$ (i.e., the objects of $\mathcal{D}^{\cdot}$) to the category $\mathcal{E}^*$; and*
- *the symbol η defines both a natural transformation from $\alpha_{0\_}$ to $\alpha_{1\_}$ (seen as functors from $\mathcal{V}$ to $\mathcal{E}^{\cdot}$) and also from $\beta_{\_0}$ to $\beta_{\_1}$ (seen as functors from $\mathcal{H}$ to $\mathcal{E}^*$).*

To shorten the notation, we will denote the natural double transformation just by η, using a picture to represent its other components. The basic idea is that, for each object $a \in \mathcal{O}$, the shape of the cell η_a is

$$\begin{tikzcd} F_{00}a \arrow[r, "\alpha_{0\_,a}"] \arrow[d, "\beta_{\_0,a}"'] \arrow[dr, phantom, "\eta_a"] & F_{01}a \arrow[d, "\beta_{\_1,a}"] \\ F_{10}a \arrow[r, "\alpha_{1\_,a}"'] & F_{11}a \end{tikzcd}$$

and coordinates the components of the horizontal and vertical transformations that compose the border of η_a. As an example, it follows directly from the definition that, given a cell $A : h \xrightarrow[u]{v} g$, all the pastings of cells pictured in Figure 13 yield identical results.

The commuting squares expressing the naturality of a transformation for one dimensional categories, are faithfully extended in the case of double categories to commuting hypercubes such as the one in Figure 14 (to make the

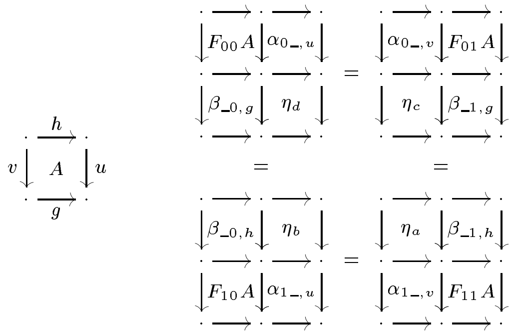

Figure 13. Some properties of η.

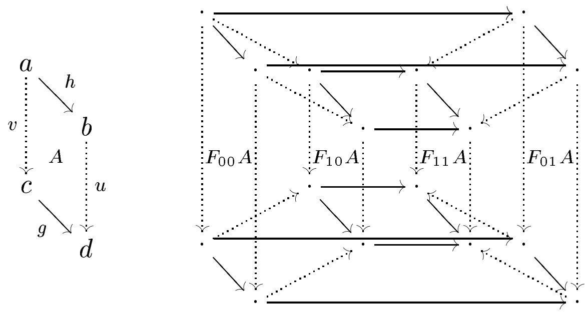

Figure 14. The commuting hypercube.

interpretation easier, vertical arrows are drawn using dotted lines).

The hypercube contains 16 vertices, 24 faces, and 8 cubes. Each vertex is the image of one of the four corner objects of cell A through one of the four functors under consideration. There are eight *empty* faces whose border involves either only vertical or only horizontal arrows. All the other 16 faces are cells of the double category $\mathcal{E}$. Four cells are the image of A w.r.t. the four different functors (see Figure 14). Four cells are the components at h and g of the natural $*$-transformations $\alpha_{0\_}$ and $\alpha_{1\_}$. Four cells are the components at v and u of the natural $\cdot$-transformations $\beta_{\_0}$ and $\beta_{\_1}$. The remaining four cells are the components at the objects of the natural double transformation η. Each cube has two empty faces. The other four faces commute, in the sense that they express a naturality equation. It follows that the hypercube yields eight equations for each cell A. However, the naturality equations for η are both replicated for the two components of each transformation, therefore, there are six distinct equations. The functoriality axioms are given by composing the hypercubes, either one below the other, or one in front of the other.

B.2. Canonical double cartesianity

By Definition 5 we know that a double category $\mathcal{D}$ has a double terminal object and double binary products if all the categories $\mathcal{D}^*$, $\mathcal{D}^{\cdot}$, $\mathcal{D}^{\triangleleft}$, and $\mathcal{D}^{\triangleright}$ have terminal object and binary products. Then, $\mathcal{D}$ is called *cartesian* if it has all binary double products and double terminal objects. However, we characterized also a much tighter notion of product, similar to the choice of a "canonical product." In fact, the more liberal definition does not establish any correspondence between the same notions on different dimensions, but simply states their existence. Thus, we adopt the convention that not only are the products *chosen* in all the four dimensions, but that they are also *consistently chosen*.

In the one-dimensional case, the notion of category with chosen products has an equivalent formulation in terms of *symmetric monoidal category* enriched by two natural transformations called *duplicator* and *discharger*. We adapt from [5] the presentation of the analogous construction within the theory of double categories, where double cartesianity is defined in terms of symmetric strict monoidal double categories with double duplicators and double dischargers.

A *monoidal double category* is an internal category in $\mathbf{MonCat}$ (the category of monoidal categories and monoidal functors), or equivalently, an internal monoidal category in $\mathbf{Cat}$. For the strict case, we have also the following detailed definition.

Definition 11 (SMDC) *A* strict monoidal double category *(SMDC) is a triple $(\mathcal{D}, \otimes, e)$, where $\mathcal{D}$ is the underlying double category, $\otimes : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ is a double functor called the* tensor product*, and e is an object of $\mathcal{D}$ called the* unit object*, the tensor product is associative on objects, arrows and cells, and e is the unit for $\_ \otimes \_$.*

Let $X : \mathcal{D} \times \mathcal{D} \to \mathcal{D} \times \mathcal{D}$ be the double functor which swaps the arguments, i.e., such that for each $A, B \in \mathcal{D}$, $X(A, B) = (B, A)$. In the one-dimensional case, a *symmetry* is a natural isomorphism between the tensor product and the swapped tensor product $X ; \otimes$, which verifies some additional coherence axioms (i.e., the first two axioms in Table 6). To define double symmetries we first introduce the following definition of the inverse of a cell.

Definition 12 (Double Inverse) *Let $A : h \xrightarrow[u]{v} g$ be a cell in a double category $\mathcal{D}$. We say that cell A has a $*$-inverse if and only if there exists a cell A^* such that $A * A^* = 1_v$, and $A^* * A = 1_u$ (i.e., A^* is the inverse of A w.r.t. the* horizontal *composition $*$, and this implies the existence of inverses of the horizontal arrows on the border of A). Similarly, the $\cdot$-inverse $A^{\cdot}$, if it exists, satisfies the equations $A \cdot A^{\cdot} = 1^h$, $A^{\cdot} \cdot A = 1^g$ (this implies the existence of inverses of the vertical arrows on the border of A). Then, A has a* double

$$
\begin{tikzcd}
a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "A"] & b \arrow[d, "u"] \\
c \arrow[r, "g"'] & d
\end{tikzcd}
\qquad
\begin{tikzcd}
b \arrow[r, "h^{-1}"] \arrow[d, "u"'] \arrow[dr, phantom, "A^*"] & a \arrow[d, "v"] \\
d \arrow[r, "g^{-1}"'] & c
\end{tikzcd}
$$

$$
\begin{tikzcd}
c \arrow[r, "g"] \arrow[d, "v^{-1}"'] \arrow[dr, phantom, "A^{\cdot}"] & d \arrow[d, "u^{-1}"] \\
a \arrow[r, "h"'] & b
\end{tikzcd}
\qquad
\begin{tikzcd}
d \arrow[r, "g^{-1}"] \arrow[d, "u^{-1}"'] \arrow[dr, phantom, "A^{-1}"] & c \arrow[d, "v^{-1}"] \\
b \arrow[r, "h^{-1}"'] & a
\end{tikzcd}
$$

Figure 15. Double inverse for the cell A.

$$
\begin{tikzcd}
\otimes \arrow[r, "\gamma"] \arrow[d, "\rho"'] \arrow[dr, phantom, "\sigma"] & X;\otimes \arrow[d, "1"] \\
X;\otimes \arrow[r, "1"'] & X;\otimes
\end{tikzcd}
$$

Figure 16. Double symmetry.

inverse *if and only if A has both a $*$-inverse and a $\cdot$-inverse, and there exists a cell A^{-1} such that:*

- $A^{-1} \cdot A^* = 1^{g^{-1}}$,
- $A^* \cdot A^{-1} = 1^{h^{-1}}$ *(i.e., A^{-1} is the $\cdot$-inverse of A^*),*
- $A^{-1} * A^{\cdot} = 1_{u^{-1}}$, *and*
- $A^{\cdot} * A^{-1} = 1_{v^{-1}}$ *(i.e., A^{-1} is the $*$-inverse of $A^{\cdot}$).*

For example, it follows that $(A \cdot A^{\cdot}) * (A^* \cdot A^{-1}) = (A * A^*) \cdot (A^{\cdot} * A^{-1}) = 1_a$, and that $(A^{-1})^{-1} = A$.

A *double symmetry* (see Figure 16) is a natural double transformation, with a double inverse, and it verifies similar axioms to the first two in Table 6, but w.r.t. the four compositions that we have illustrated (horizontal, vertical and the two diagonals).

A *duplicator* is a natural transformation between the identity and the tensor product of two copies of the argument and verifies some additional coherence axioms involving symmetries and dischargers (see Table 6). A *discharger* is a natural transformation between the identity and the constant functor mapping each element into the unit of the tensor product. Thus, *double duplicators* and *double dischargers* can be defined as natural double transformations verifying similar coherence axioms. Let $\Delta : \mathcal{D} \to \mathcal{D} \times \mathcal{D}$ be the diagonal double functor which makes a copy of the argument, i.e., such that for each $A \in \mathcal{D}$, $\Delta(A) = (A, A)$. Then double duplicators and double dischargers are pictured in Figure 17.

$$
\begin{tikzcd}
1_{\mathcal{D}} \arrow[r, "\boldsymbol{\nabla}"] \arrow[d, "\boldsymbol{\delta}"'] \arrow[dr, phantom, "\boldsymbol{\pi}"] & \Delta;\otimes \arrow[d, "1"] \\
\Delta;\otimes \arrow[r, "1"'] & \Delta;\otimes
\end{tikzcd}
\quad
\begin{tikzcd}
1_{\mathcal{D}} \arrow[r, "1"] \arrow[d, "1"'] \arrow[dr, phantom, "\boldsymbol{\tau}"] & 1_{\mathcal{D}} \arrow[d, "\boldsymbol{\delta}"] \\
1_{\mathcal{D}} \arrow[r, "\boldsymbol{\nabla}"'] & \Delta;\otimes
\end{tikzcd}
\quad
\begin{tikzcd}
1_{\mathcal{D}} \arrow[r, "\boldsymbol{!}"] \arrow[d, "\dagger"'] \arrow[dr, phantom, "\boldsymbol{\phi}"] & e \arrow[d, "1"] \\
e \arrow[r, "1"'] & e
\end{tikzcd}
\quad
\begin{tikzcd}
1_{\mathcal{D}} \arrow[r, "1"] \arrow[d, "1"'] \arrow[dr, phantom, "\boldsymbol{\psi}"] & 1_{\mathcal{D}} \arrow[d, "\dagger"] \\
1_{\mathcal{D}} \arrow[r, "\boldsymbol{!}"'] & e
\end{tikzcd}
$$

Figure 17. Double duplicators (π and τ) and double dischargers (ϕ and ψ).

$$
\begin{array}{rclcrcl}
\boldsymbol{\tau}_a * \boldsymbol{\pi}_a & = & \boldsymbol{\nabla}_a & \qquad & \boldsymbol{\psi}_a * \boldsymbol{\phi}_a & = & \boldsymbol{!}_a \\
\boldsymbol{\tau}_a \cdot \boldsymbol{\pi}_a & = & \boldsymbol{\delta}_a & & \boldsymbol{\psi}_a \cdot \boldsymbol{\phi}_a & = & \dagger_a
\end{array}
$$

Table 9. Double coherence axioms.

The coherence axioms that double symmetries, double duplicators and double dischargers must satisfy are essentially the same of the one-dimensional case (Table 6), but repeated for each one of the four composition, thus

(A) γ, $\boldsymbol{\nabla}$ and $\boldsymbol{!}$ satisfy the coherence axioms w.r.t. $\_ * \_$;

(B) $\boldsymbol{\rho}$, $\boldsymbol{\delta}$, and $\dagger$ satisfy the coherence axioms w.r.t. $\_ \cdot \_$;

(C) $\boldsymbol{\sigma}$, $\boldsymbol{\pi}$, and $\boldsymbol{\phi}$ satisfy the coherence axioms w.r.t. $\_ \triangleleft \_$;

(D) $\boldsymbol{\sigma}^{-1}$, $\boldsymbol{\tau}$, and $\boldsymbol{\psi}$ satisfy the coherence axioms w.r.t. $\_ \triangleright \_$;

(E) moreover, the double coherence axioms illustrated in Table 9 are satisfied.

Then, we can easily define four possible ways of pairing and projecting. For example, given two cells $A : h \xrightarrow[u]{v} g$ and $B : h \xrightarrow[z]{w} f$, then their vertical pairing is $\langle A, B\rangle^{\mathtt{V}} = \boldsymbol{\delta}_h \cdot (A \otimes B)$, and the first vertical projection associated to $f \times g = f \otimes g$ is defined as $\Pi_1^{\mathtt{V}} = 1^f \otimes \dagger_g$. Analogously, given two cells $A : h \xrightarrow[u]{v} g$ and $C : l \xrightarrow[w]{v} f$, then their horizontal pairing is $\langle A, C\rangle^{\mathtt{H}} = \boldsymbol{\nabla}_v * (A \otimes C)$, and the second horizontal projection associated to $u \times w = u \otimes w$ is defined as $\Pi_2^{\mathtt{H}} = \boldsymbol{!}_u \otimes 1_w$. Diagonal pairings $\langle \_, \_\rangle^{\triangleleft}$ and $\langle \_, \_\rangle^{\triangleright}$ are defined in the obvious way. Notice that, by composing auxiliary cells, we have an interesting way of pairing two cells, which is not contained in the previous list: The idea is to get any two cells $A : h \xrightarrow[u]{v} g$ and $B : l \xrightarrow[z]{w} f$ such that h and l have the same source a, and compose them as below

$$(\boldsymbol{\pi}_a * 1^{h \otimes l}) \cdot (1_{v \otimes w} * (A \otimes B))$$

this yields a sort of diagonal pairing that works with any cells, and not only with those in $\mathcal{D}^{\triangleleft}$.