Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 176 (2007) 139–163
www.elsevier.com/locate/entcs

A Local Graph-rewriting System for Deciding Equality in Sum-product Theories

José Bacelar Almeida[1], Jorge Sousa Pinto[2] and Miguel Vilaça[3]

Departamento de Informática / CCTC
Universidade do Minho
4710-057 Braga, Portugal

Abstract

In this paper we give a graph-based decision procedure for a calculus with sum and product types. Although our motivation comes from the Bird-Meertens approach to reasoning algebraically about functional programs, the language used here can be seen as the internal language of a category with binary products and coproducts. As such, the decision procedure presented has independent interest.
A standard approach based on term rewriting would work modulo a set of equations; the present work proposes a simpler approach, based on graph-rewriting. We show in turn how the system covers *reflection* equational laws, *fusion* laws, and *cancellation* laws.

Keywords: Graph-based decision procedure, point-free programming

1 Introduction

The *point-free* style of programming [3] has been defended as a good choice for reasoning about functional programs, as it avoids reference to variables. Instead, programs are constructed by a pure first order language over a basic set of combinators, whose semantics is given by a set of equations. The equality of two programs is established by constructing derivations in the corresponding equational theory. This ought to be compared with the pointwise approach, where programs are terms of an appropriate λ-calculus. The notion of equality induced by β-reduction is not sufficient for establishing that two programs behave equally (i.e. produce the same result when applied to equal arguments), which motivates the inclusion of extensionality into the reduction relation.

[1] Email:jba@di.uminho.pt
[2] Email:jsp@di.uminho.pt
[3] Email:jmvilaca@di.uminho.pt

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2006.10.031

The correspondence between point-free languages and extensional λ-calculi is well-known from Categorical Logic [10]. In fact both may be seen as different axiomatizations for the universal properties of categorical constructs: in the former, axioms are given in terms of morphisms, and in the latter in terms of elements (Plotkin [11] appropriately calls these "external" and "internal" axiomatizations).

In this paper, we consider a fragment containing only binary products and coproducts. Due to the lack of exponentials, this fragment has very little computational power. Nevertheless, the problem of deciding equality already exhibits interesting subtleties, since the interplay between products and coproducts forces certain equalities that must traditionally be handled by working modulo an equivalence relation [12,6].

We propose a decision procedure for equality, using a local graph-rewriting system. Our approach is based on the point-free presentation of the theory – the graphs are essentially data-flow representations for terms, with composition corresponding to an edge linking together two graphs. It is shown that this perspective effectively overcomes the need to consider an external equivalence.

Related Work.

The exact subject of this work has already been addressed in [12], where the authors describe a deductive system for categories with finite products and coproducts, which corresponds precisely to the theories under consideration here. A procedure for checking equality of morphisms is established in two steps: (i) a standard rewriting system is studied and proved to be strongly normalizing and confluent modulo an appropriate equational theory E, and (ii) an algorithm is given for deciding equality under E.

Also related is the work of Ghani [6] on an extensional lambda-calculus with sums (sums had been considered earlier [5,4] with restricted versions of the extensionality axiom). Once again, decidability is established in two steps: a strongly normalizing reduction relation and a procedure for checking a 'commuting' conversion of (quasi-)normal forms. More recently, Altenkirch and colleagues [1] have used a *normalization by evaluation* approach for solving the same problem. Although the latter approach does not rely on rewriting, it shares with the work presented here the fact that normal forms do not correspond to terms, since they are computed in a distinctive semantic domain.

Our work also has similarities with work on optimal implementations of the λ-calculus [2,8], and in particular to the use of *safe operators* that allow for the application of non-interaction rules (i.e. rules involving ports that are not principal) in an indexed, local graph-rewriting system. We remark however that the reasons for introducing indices are totally unrelated in both works.

Structure of the Paper.

The paper is organized as follows: Section 2 presents the sum-product theory. Section 3 defines sum-product nets, and Section 4 gives a translation of terms into these nets; Section 5 introduces informally the principles of our graph-rewriting

system. The main results of the paper appear in Section 6, where the system is formally defined and its properties are studied. We conclude in Section 7.

2 The Term Language and Theory

Consider the following language T_{PF} for types and terms:

$$\begin{aligned}\mathsf{Type} &::= A \mid \mathsf{Type} \times \mathsf{Type} \mid \mathsf{Type} + \mathsf{Type}\\ \mathsf{Term} &::= C^{\mathsf{A};\mathsf{A}} \mid \mathsf{id}^{\mathsf{Type}} \mid \mathsf{Term} \circ \mathsf{Term} \mid \langle \mathsf{Term}, \mathsf{Term} \rangle \mid \pi_1{}^{\mathsf{Type};\mathsf{Type}} \mid\\ &\quad \pi_2{}^{\mathsf{Type};\mathsf{Type}} \mid [\mathsf{Term}, \mathsf{Term}] \mid \mathrm{i}_1{}^{\mathsf{Type};\mathsf{Type}} \mid \mathrm{i}_2{}^{\mathsf{Type};\mathsf{Type}}\end{aligned}$$

where A ranges over a set of *base types* and $C^{\mathsf{A};\mathsf{A}}$ is a set of *constant functions* (we assume that the sets in this indexed family are pairwise disjunct – thus a constant symbol uniquely determines its indexing types).

To each term we associate a *domain* and a *codomain* type – we denote $f : A \mid B$ the assertion that term f has domain A and codomain B. The typing rules associated to the language are the following

$$\frac{}{c^{\mathsf{A};\mathsf{B}} : \mathsf{A} \mid \mathsf{B}}\ c^{\mathsf{A};\mathsf{B}} \in C^{\mathsf{A};\mathsf{B}} \qquad \frac{}{\mathrm{id}^{\mathsf{A}} : \mathsf{A} \mid \mathsf{A}} \qquad \frac{\mathsf{f} : \mathsf{A} \mid \mathsf{B} \qquad \mathsf{g} : \mathsf{B} \mid \mathsf{C}}{\mathsf{g} \circ \mathsf{f} : \mathsf{A} \mid \mathsf{C}}$$

$$\frac{\mathsf{f} : \mathsf{A} \mid \mathsf{C} \qquad \mathsf{g} : \mathsf{B} \mid \mathsf{C}}{[\mathsf{f};\mathsf{g}] : (\mathsf{A} + \mathsf{B}) \mid \mathsf{C}} \qquad \frac{}{\pi_1{}^{\mathsf{A};\mathsf{B}} : (\mathsf{A} \times \mathsf{B}) \mid \mathsf{A}} \qquad \frac{}{\pi_2{}^{\mathsf{A};\mathsf{B}} : (\mathsf{A} \times \mathsf{B}) \mid \mathsf{B}}$$

$$\frac{\mathsf{f} : \mathsf{A} \mid \mathsf{B} \qquad \mathsf{g} : \mathsf{A} \mid \mathsf{C}}{\langle \mathsf{f};\mathsf{g} \rangle : \mathsf{A} \mid (\mathsf{B} \times \mathsf{C})} \qquad \frac{}{\mathrm{i}_1{}^{\mathsf{A};\mathsf{B}} : \mathsf{A} \mid (\mathsf{A} + \mathsf{B})} \qquad \frac{}{\mathrm{i}_2{}^{\mathsf{A};\mathsf{B}} : \mathsf{B} \mid (\mathsf{A} + \mathsf{B})}$$

In the following, when referring to a term we assume its well-typedness. We will omit the type superscripts, which can be inferred from the context.

The type constructors $\times$ and $+$ are characterized through their universal properties, which may in turn be captured by the following set of equations:

Composition	$\mathsf{id} \circ f = f \circ \mathsf{id} = f$	$(f \circ g) \circ h = f \circ (g \circ h)$
Reflection laws	$\langle \pi_1, \pi_2 \rangle = \mathsf{id}$	$[\mathrm{i}_1, \mathrm{i}_2] = \mathsf{id}$
Fusion laws	$\langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle$	$f \circ [g, h] = [f \circ g, f \circ h]$
Cancellation laws	$\pi_1 \circ \langle f, g \rangle = f$	$[f, g] \circ \mathrm{i}_1 = f$
	$\pi_2 \circ \langle f, g \rangle = g$	$[f, g] \circ \mathrm{i}_2 = g$

This presentation of the equational theory is standard in the field of Algebraic Programming [3]. In the λ-calculus perspective, cancellation laws are β-equalities, and the reflection and fusion laws together are equivalent to η-equalities. For products for instance, η-equality would be written as $f = \langle \pi_1 \circ f, \pi_2 \circ f \rangle$, also known as *surjective pairing*.

Deciding equality under the theory defined by these equations requires producing a decision procedure. The standard way to accomplish this would be to orient the equations to obtain a confluent, terminating rewriting system (by means of a

completion process). This method fails for this system, for reasons that will be hinted at when we explain our proposed solutions in Section 5. It will become clear that the factorization of the η rules is useful since it allows to treat independently issues raised by reflection and by fusion, which are of different natures.

3 Sum-product Nets

Sum-product Nets will be built from an alphabet of *symbols*; each symbol has an associated *arity* and *co-arity*. We introduce these symbols in *dual* pairs where the arity and co-arity are exchanged:

a *makepair* symbol with arity 2 and co-arity 1, depicted $(,)$; its dual *choice* is depicted ?;

two *pair projection* symbols with arity 1 and co-arity 1, depicted π_1 and π_2; their duals are the *choice injections* depicted i_1 and i_2;

an *eraser* symbol with arity 1 and co-arity 0, depicted ε; the dual *co-eraser* is depicted $\ni$.

for every natural number n, a *cancel* symbol with arity and co-arity 1, depicted n; its dual *co-cancel* is depicted n.

for every pair of natural numbers (n_1, n_2), a *duplicator* symbol with arity 1 and co-arity 2, depicted $\wedge^{(\mathsf{n}_1;\mathsf{n}_2)}$; its dual is the *co-duplicator*, depicted $\_^{(\mathsf{n}_1;\mathsf{n}_2)}$;

A *node* is a triple (s, I, O) where s is a symbol with arity a_{s} and co-arity c_{s}, I is a set of *input ports*, and O is a set of *output ports* such that the number of elements of I (resp. O) is a_{s} (resp. c_{s}).

A *net* is a tuple $(S, E, \hat{I}, \hat{O})$ where

S is a set of *nodes*; let $I_{\mathsf{S}} = \bigcup_{(\mathsf{s};\mathsf{I};\mathsf{O})2\mathsf{S}} I$ and $O_{\mathsf{S}} = \bigcup_{(\mathsf{s};\mathsf{I};\mathsf{O})2\mathsf{S}} O$;

$\hat{I}, \hat{O}$ are respectively the sets of *input ports* and *output ports* of the net;

$E : O_{\mathsf{S}} [\hat{I} ! I_{\mathsf{S}} [\hat{O}$ is a bijection. Pairs (o, i) 2 E are called *edges* of the net.

A net is *well-typed* if there exists a labelling of the input and output ports of each of its nodes with a type, such that every edge connects equally labelled ports, and the constraints shown in Figure 1 hold for every node (capital letters denote type variables).

A *position* is a pair of non-negative integers (a, b), depicted as $a\ b$. A net is *well-formed* if there exists a labelling of the input and output ports of each of its nodes with a position, such that every edge connects equally labelled ports, and the constraints also shown in Figure 1 hold for every node ($S\ n$ denotes the successor of n). Well-formedness imposes a structural invariant on nets.

De nition 3.1 A *sum-product net* is an acyclic, well-typed and well-formed net with a single input and output.

Indices in sum-product nets will be used to control the duplication and mutual annihilation of nodes in the reduction system presented in section 6. Non-indexed

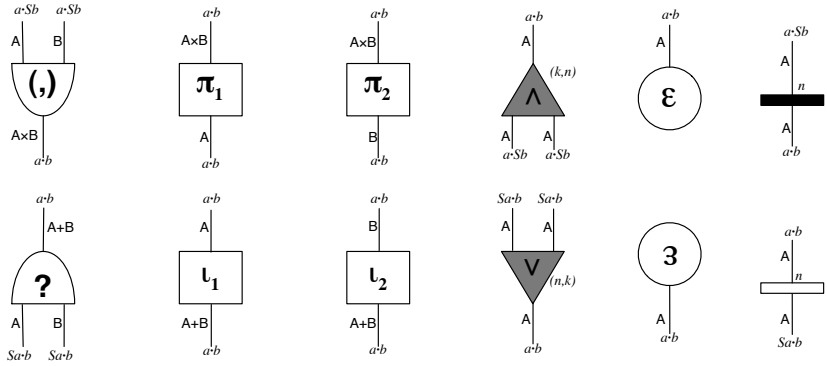


Fig. 1. Typing and Positioning Constraints

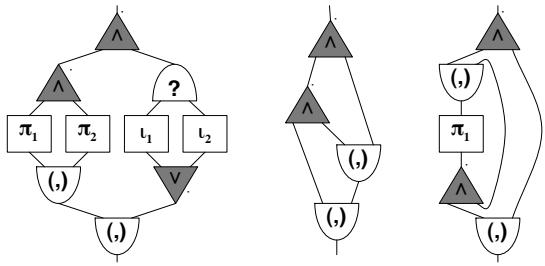


Fig. 2. Examples of Nets

or zero-indexed nodes – 0 for (co-)cancel, or $(0,0)$ for (co-)duplicators – will be called *ground*, and their indices will often be omitted. Figure 2 contains examples of nets that are not sum-product nets: the first net is not well-typed; the second is not well-formed; the third net has a cycle.

4 Term nets

In this section we give a type-directed translation $\mathsf{T}(\,)$ from terms of T_{PF} into sum-product nets.

The translation will already solve a confluence problem that is inherent to reflection laws: taking the multiplicative sub-system, the orientation *left to right* seems sensible, but the reflection law creates unsolvable critical pairs. To illustrate this, consider the following derivation of surjective pairing

$$f = \mathsf{id} \quad f = \mathsf{h}\pi_1, \pi_2\mathsf{i} \quad f = \mathsf{h}\pi_1 \;\; f, \pi_2 \;\; f\mathsf{i}$$

Both extremes of the equality chain are in normal form with respect to the rewrite system obtained, thus it fails to be complete. The problem is that the rule resulting from orienting the reflection law drops structural information from the term that is essential for the confluence of the system – $\mathsf{h}\pi_1, \pi_2\mathsf{i}$ is certainly more informative than id, since the type of the latter is more general.

The graph-rewriting system defined in the next sections always preserves the

structural information, which must be explicitly present in the initial representations. Reflection is entirely dropped from the rewriting system, and becomes a rule for defining identities of structured types. As an example, the identity of type $(A \quad B) \quad C$ will be represented as $\mathsf{hh}\pi_1, \pi_2\mathsf{i} \,.\, \pi_1, \pi_2\mathsf{i}$.

The translation given below expands identities in accordance with this discussion, so that only identities of atomic types are represented as edges. Equations like surjective pairing are then satisfied by construction.

In the following, when a net is used to construct a bigger net, we assume that the input and output in the initial net are removed, and a new pair of input/output ports and corresponding edges are introduced in the new net. All the introduced nodes are ground.

Identity

$\mathsf{T}(\mathsf{id}^{\mathsf{A!\ A}})$, where A is a base type, is defined as the sum-product net consisting of a single edge connecting the input to the output;

$\mathsf{T}(\mathsf{id}^{\mathsf{A\ \ B!A\ \ B}})$ is the sum-product net I obtained by introducing 4 new nodes, ^, π_1, π_2, and $(,)$, and new edges connecting the first (resp. second) output of ^ to the input of π_1 (resp. π_2), the output of π_1 (resp. π_2) to the input of I_{A} (resp. I_{B}), and the output of I_{A} (resp. I_{B}) to the first (resp. second) input of $(,)$, where $I_{\mathsf{A}} = \mathsf{T}(\mathsf{id}^{\mathsf{A!A}\ \ })$ and $I_{\mathsf{B}} = \mathsf{T}(\mathsf{id}^{\mathsf{B!B}\ \ })$; and finally setting the input of I to be the input of ^ and the output of I to be the output of $(,)$.

$\mathsf{T}(\mathsf{id}^{\mathsf{A+B!A\ \ +B}})$ is the sum-product net I obtained by introducing 4 new nodes, ?, i_1, i_2, and _, and new edges connecting the first (resp. second) output of ? to the input of I_{A} (resp. I_{B}), the output of I_{A} (resp. I_{B}) to the input of i_1 (resp. i_2), and the output of i_1 (resp. i_2) to the first (resp. second) input of _, where $I_{\mathsf{A}} = \mathsf{T}(\mathsf{id}^{\mathsf{A!A}\ \ })$ and $I_{\mathsf{B}} = \mathsf{T}(\mathsf{id}^{\mathsf{B!\ B}})$; and finally setting the input of I to be the input of ? and the output of I to be the output of _.

Composition

$\mathsf{T}(u \,.\, t^{\mathsf{A!C}\ \ })$ is the sum-product net V obtained by connecting an edge from the output of T to the input of U, where $T = \mathsf{T}(t^{\mathsf{A!\ B}})$ and $U = \mathsf{T}(u^{\mathsf{B!C}\ \ })$. Naturally, the input of T becomes the input of V, and the output of U becomes the output of V.

Constant Function

$\mathsf{T}(\pi_1{}^{\mathsf{A\ \ B!\ A}})$ is the net P_1 obtained by introducing a new node π_1 and a new edge connecting its output to the input of I_{A}, where $I_{\mathsf{A}} = \mathsf{T}(\mathsf{id}^{\mathsf{A!\ A}})$, and setting the input of P_1 to be the input of π_1 and the output of P_1 to be the output of I_{A}.

$\mathsf{T}(\pi_2{}^{\mathsf{A\ \ B!\ B}})$ is the net P_2 obtained by introducing a new node π_2 and a new edge connecting its output to the input of I_{B}, where $I_{\mathsf{B}} = \mathsf{T}(\mathsf{id}^{\mathsf{B!\ B}})$, and setting the input of P_2 to be the input of π_2 and the output of P_2 to be the output of I_{B}.

$\mathsf{T}(\mathsf{i}_1{}^{\mathsf{A!\ A+B}})$ is the net I_1 obtained by introducing a new node i_1 and a new edge connecting the output of I_{A} to the input of i_1, where $I_{\mathsf{A}} = \mathsf{T}(\mathsf{id}^{\mathsf{A!\ A}})$, and setting the input of I_1 to be the input of I_{A} and the output of I_1 to be the

output of i_1.

$\mathsf{T}(i_2{}^{B!A\ \ +B})$ is the net I_2 obtained by introducing a new node i_2 and a new edge connecting the output of I_B to the input of i_2, where $I_B = \mathsf{T}(id^{B!\ \ B})$, and setting the input of I_2 to be the input of I_B and the output of I_2 to be the output of i_2.

Split

Let G be the sum-product net obtained by introducing two new ^ and $(,)$ nodes, and 4 new edges connecting the outputs of ^ to the inputs of T and U, and the outputs of T and U to the inputs of $(,)$, where $T = \mathsf{T}(t^{E!A\ \ })$ and $U = \mathsf{T}(u^{E!\ \ B})$; the input of ^ becomes the input of G and the output of $(,)$ becomes the output of G. Then:

$\mathsf{T}(\mathsf{h}t, u\mathsf{i}^{E!\ \ A\ \ \ B})$, with $E = C + D$, is the sum-product net G^0 obtained by constructing the net $I = \mathsf{T}(id^{C+D!\ \ C+D})$, and an edge connecting its output to the input of G, setting the input of G^0 to be the input of I, and the output of G^0 to be the output of G.

$\mathsf{T}(\mathsf{h}t, u\mathsf{i}^{E!\ \ A\ \ \ B})$, where E is not of the form $C + D$, is just G.

Either

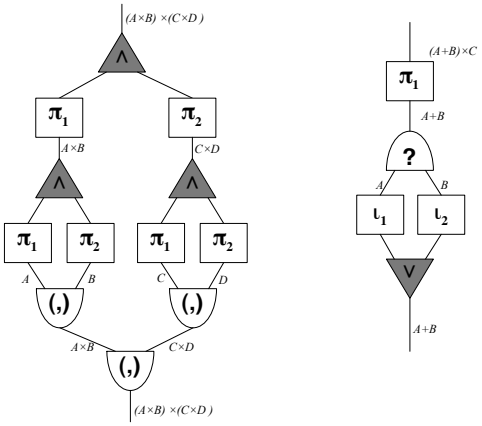
Let G be the sum-product net obtained by introducing two new ? and _ nodes, and 4 new edges connecting the outputs of ? to the inputs of T and U, and the outputs of T and U to the inputs of _, where $T = \mathsf{T}(t^{A!\ \ E})$ and $U = \mathsf{T}(u^{B!E\ \ })$; then the input of ? becomes the input of G and the output of _ becomes the output of G. We have:

$\mathsf{T}([t, u]^{A+B!E\ \ })$, where $E = C\ \ D$, is the sum-product net G^0 obtained by constructing the net $I = \mathsf{T}(id^{C\ \ D!C\ \ \ D})$, and an edge connecting the output of G to the input of I; the input of G^0 is the input of G, and the output of G^0 is the output of I.

$\mathsf{T}([t, u]^{A+B!E\ \ })$, where E is not of the form $C\ \ D$, is just G.

Definition 4.1 The class of sum-product nets constructed by the translation $\mathsf{T}(\ )$ are designated *term nets.*

The term nets $\mathsf{T}(id^{(A\ \ B)\ (C\ \ \ D)!\ (A\ \ B)\ (C\ \ \ D)})$ and $\mathsf{T}(\pi_1{}^{(A+B)\ \ C!A\ \ +B})$ are shown below as examples.



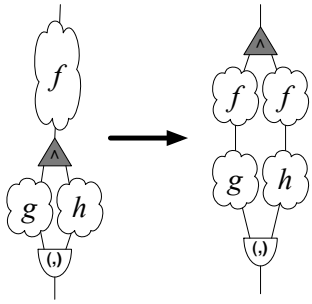


Fig. 3. Fusion as Net Duplication

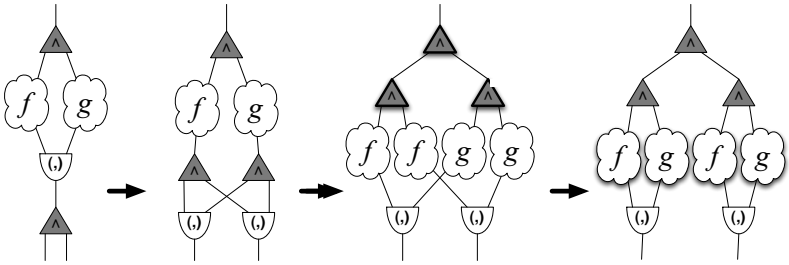


Fig. 4. Duplication of a Structured Net

It is straightforward to see that the net $\mathsf{T}(t^{\,A!B}\,)$ has input of type A and output of type B. Two differently-typed, syntactically equal terms may be translated as different term nets. The principal type of the term represented by a net can always be uniquely determined.

5 Deciding Equality by Local Graph Rewriting

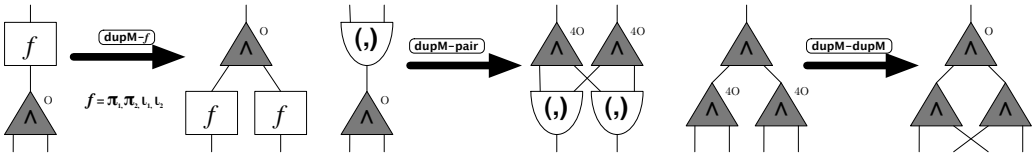
In this section we introduce informally the graph-rewriting system and the difficulties involved in its definition. The system will then be formally defined in the next section, and its properties studied. Here we consider in turn the treatment of fusion and cancellation.

Fusion

Fusion is accomplished by (co-)duplicator nodes. Intuitively, a duplicator interacting with a net should perform a copy of that net (see Figure 3). However, this "duplication" should be performed locally, i.e. the (co-)duplicators interact only with individual nodes. Moreover, both kinds of fusion (additive and multiplicative) can occur simultaneously and thus some care must be taken in order to avoid interferences in the process.

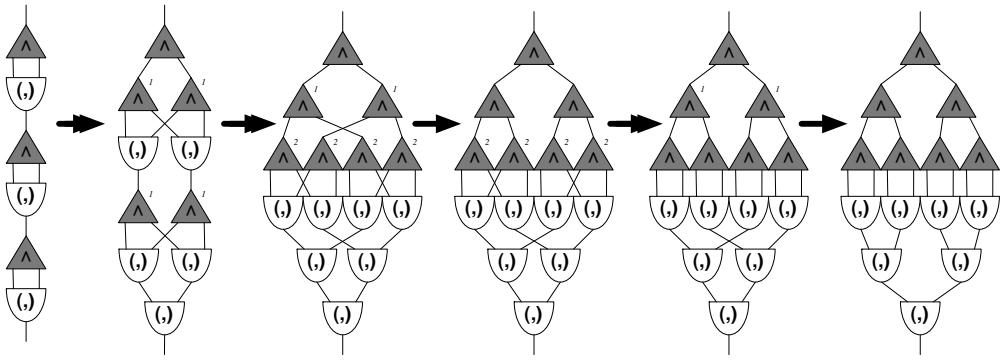
For the sake of clarity, we start by considering only the multiplicative fragment of our language. When a duplicator meets a structured net (necessarily a split of two terms), it must itself split in order to duplicate each component of the net. Once the duplication of each sub-net is concluded, it is still necessary to reorganize the duplicators on the top of the net to get the correct outcome for the duplication

of the structured net (see Figure 4). The need to control this reorganization of duplicators justifies the presence of indices in the duplicator symbol – let us assume indices are integers (they will be refined later to be pairs of integers). We are led to the following rules governing the interaction with duplicators.



The first rule is fairly obvious – the interaction with single-input / single-output nodes simply duplicates them. When a duplicator interacts with a pair constructor node, not only does it duplicate the node, but it also splits itself in two in order to duplicate each subnet. The last rule is the commutation rule between duplicators that allows for the two split copies of the original duplicator to be rejoined, while at the same time duplicating the top duplicator. Note the difference between splitting and duplication of duplicators.

Indices record "how deep" the corresponding duplicator is in the traversal of a structured net. Notice that the commutation rule restricts the top duplicator to be ground. We show an example of the duplication process:



We now turn to the interaction between sums and products. The above treatment of fusion corresponds to a left-to-right orientation of the fusion law. One problem that is raised at the term level when both sum and product types are present has to do with associativity of composition: there no longer exists a preferred orientation for it, suggested by the left-to-right orientation of fusion. At the graph level this problem is avoided since graphs capture associativity of composition for free, and no rewriting is required.

A second problem has to do with the fact that products and sums interact in such a symmetrical way that it is not possible to choose a certain form to the detriment of its dual. As an example to illustrate this last observation, consider the following equality derivations for the same term:

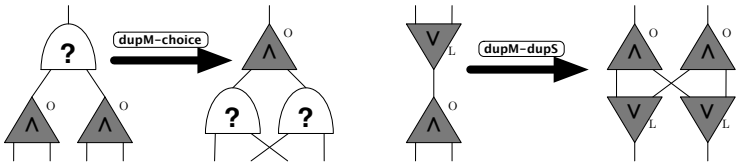
$$\text{hid,idi}\quad [\text{id},\text{id}] = \text{hid}\ [\text{id},\text{id}],\text{id}\ \ [\text{id},\text{id}]\text{i} = \text{h}[\text{id},\text{id}],[\text{id},\text{id}]\text{i}$$

$$\text{hid,idi}\quad [\text{id},\text{id}] = [\text{hid,idi}\quad \text{id},\text{hid,idi}\quad \text{id}] = [\text{hid},\text{idi},\ \text{hid,idi}]$$

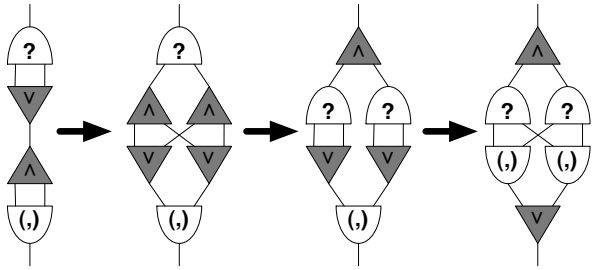
In fact this is an instance of a more general equality, known as the *exchange law*: $\mathsf{h}[f,g],[h,k]\mathsf{i} = [\mathsf{h}f,h\mathsf{i},\mathsf{h}g,k\mathsf{i}]$. Rather than working modulo an appropriate equational theory that equates these terms (as in [12]), our graph-rewriting system will assign them to a unique normal form. The essence of this relies on letting duplicators and co-duplicators perform their traversals without interfering with each other.

Returning to the $\mathsf{hid,idi}\quad[\mathsf{id},\mathsf{id}]$ term, let us focus on the interaction of the co-duplicator and duplicator. Structurally, it is fairly obvious that when a duplicator and a co-duplicator meet, they should pass through each other. The question is whether the nodes should be *duplicated* or *split* during this process (the impact on their indices is different).

A first solution would be to state that only duplications take place. This corresponds to keeping both indices unaltered, and leads to a rule that allows duplicators to freely pass through choice nodes (without index regulation). The following additional rules (and their duals) would be required:

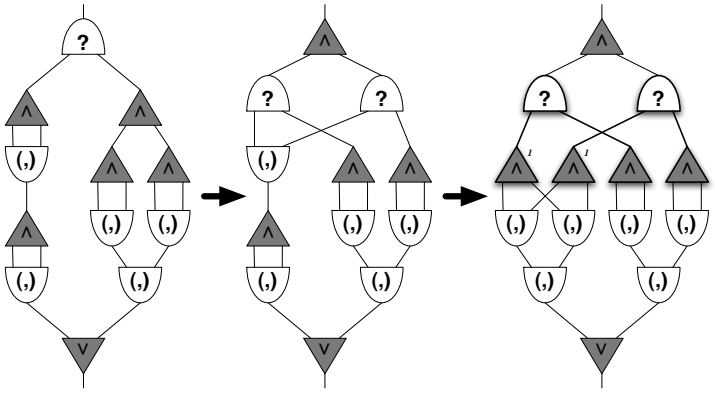


Our example term $\mathsf{hid,idi}\quad[\mathsf{id},\mathsf{id}]$ reduces to a unique normal form, which *does not correspond to the translation of any term*. We will call these *full normal forms*, as they capture the fact that these nets can be read-back as different terms.



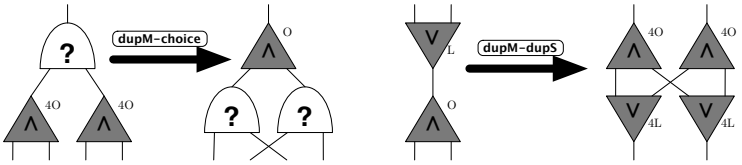
It can be readily verified that the translation of $\mathsf{h}[\mathsf{id,id}],[\mathsf{id},\mathsf{id}]\mathsf{i}$ (resp. $[\mathsf{hid},\mathsf{idi},\ \mathsf{hid,idi}]$) also has the same normal form. Reduction is triggered by an extra identity net introduced by the translation function at the top (resp. bottom) of the net.

Unfortunately, with the above rules the indices no longer constrain appropriately the commutations that might take place. To see this, consider the following reduction sequence:

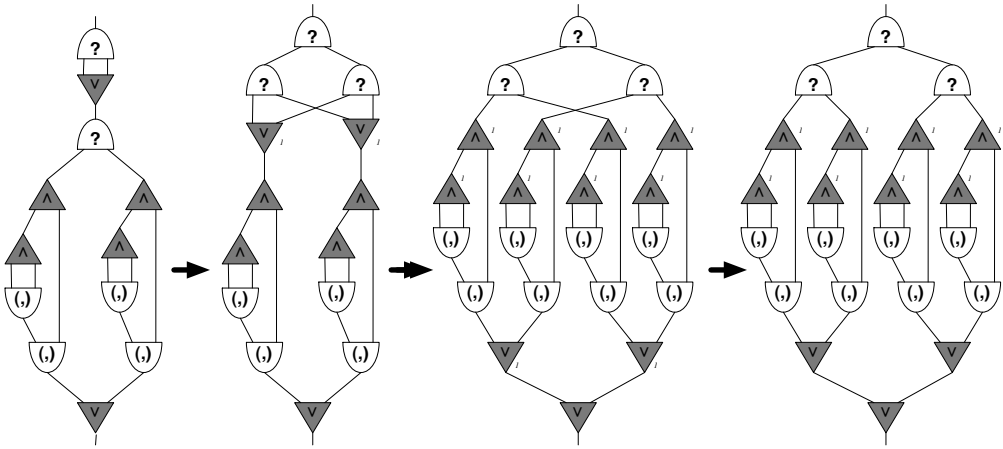


Note that the application of rule dupM-choice removes the ground duplicator needed to close the fusion started on the left-hand side sub-net. In fact, this is exactly the pattern of divergence that rule dupM-dupM avoids by restricting the top duplicator to be ground. This shows that indices should also restrict the application of rule dupM-choice.

A second approach would be to let both nodes be split (i.e. both nodes have their indices incremented). These are the corresponding rules:



Here the problem becomes subtler. Consider the following reduction sequence:



Notice that the traversal of the net by the co-duplicator on the top lifts the indices of all the duplicators in the lower sub-nets. This feature is not in accordance with the informal description given above, but it can be exploited fruitfully. In fact, if we continue the reduction process, we get:

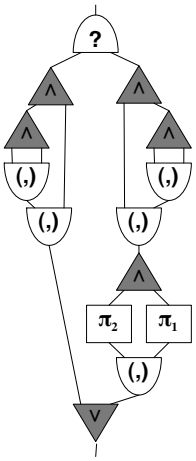
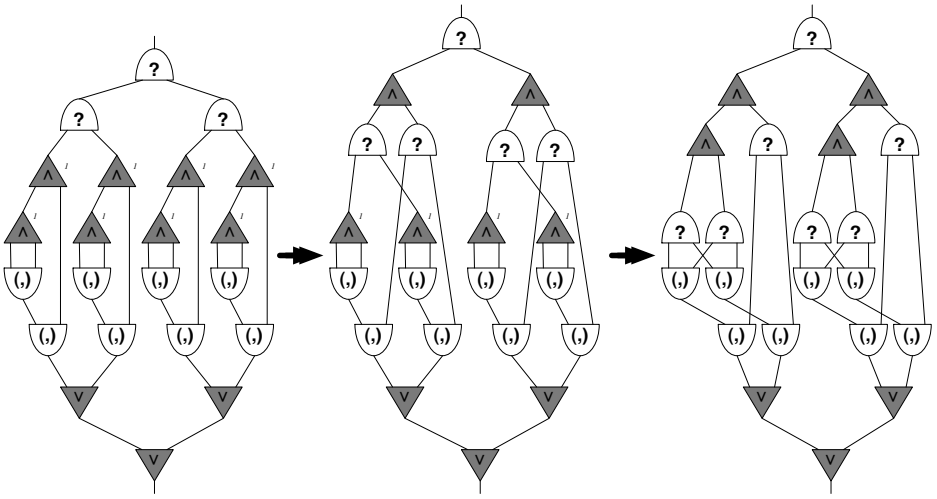


Fig. 5. Example of an asymmetrical net



This example suggests that in order to reach full normal forms, it is sufficient to promote enough fusions (something that may be accomplished by pre- or post-composing with identities). The problem is that the additional reductions rely strongly on the *symmetry* of the net. More precisely, they rely on the number of duplicators and on the shape of the duplicator tree, on both sides of the choice node. However, even though term nets do exhibit a high degree of symmetry, it cannot be assured that all nets, under all reduction strategies, possess the required symmetry. Figure 5 shows precisely an example of an asymmetrical net. The reader is invited to apply fusion to it with $[id, id]$ to check the problems of this approach.

A final approach is to have a mixed version of the previous solutions: informally, we take one of the nodes to be the 'dominator' of the interaction. This node is split (its index is increased), and the other is simply duplicated (its index remains unaltered). The problem here is how to define the domination relation, since the most obvious solutions fails to define a confluent system.

Our solution is given by the following characterization: a _ node dominates a ^

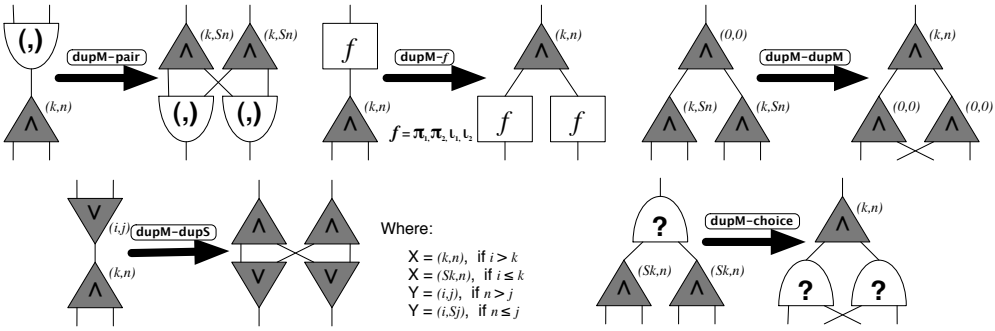


Fig. 6. Fusion Rules

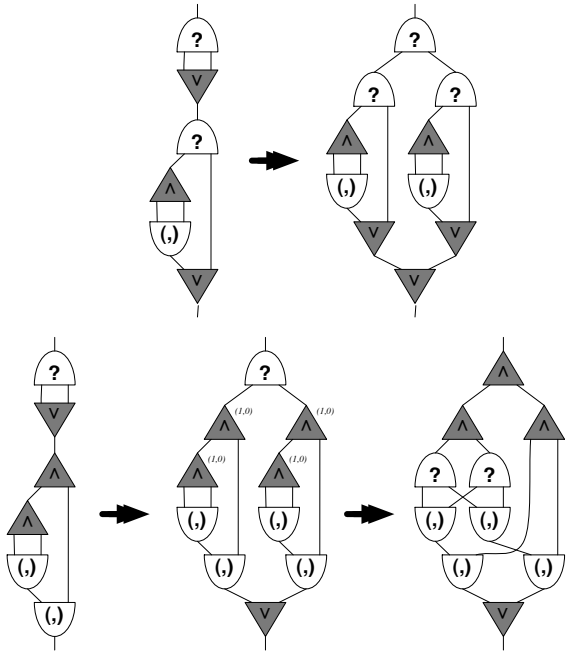


Fig. 7. Examples Fusions with Rules in Fig. 6

when "there is no ? node between them". Being able to express this in the system is in fact the key reason for having *pairs of integers* as indices, since an additive and a multiplicative component need to be distinguished. Figure 6 implements this solution (a dual set of rules is used for co-duplicators), and Figure 7 illustrates its application.

Note that we only allow ^ nodes to commute with the ? node associated with the co-duplicator that is performing the fusion (even if this ? becomes available only after commutation with other ^s). Commutations involving ? nodes duplicated during the fusion process are inhibited.

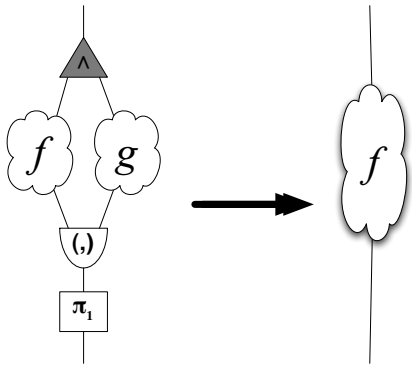


Fig. 8. Cancellation as Erasing

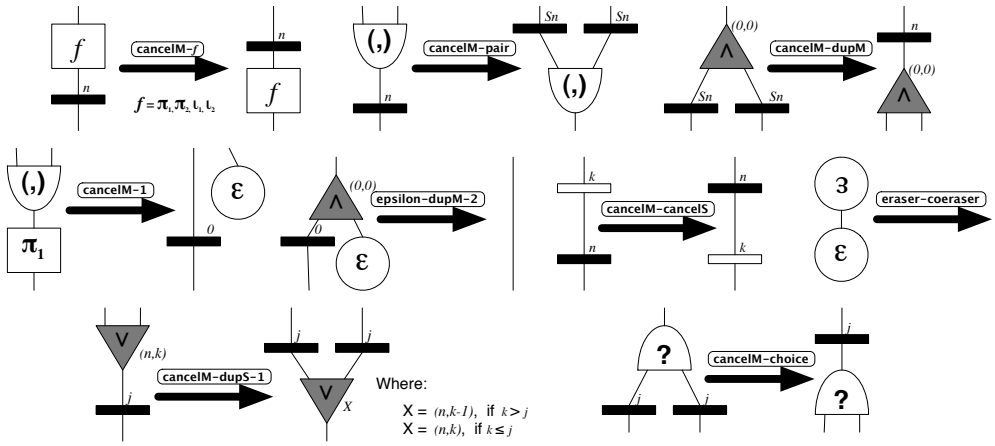


Fig. 9. Cancellation Rules

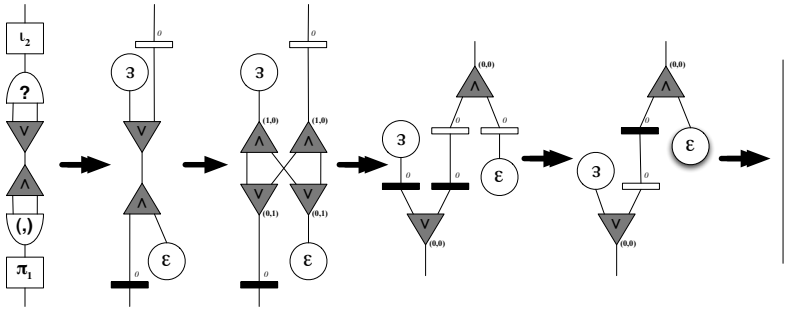
Cancellation

Cancellation is by nature related with erasing. Consider the structure of a cancellation rule given in Figure 8. The interaction of the bottom nodes should trigger the removal of the net g and the top duplicator. We follow the standard approach of introducing a special erasing node ε (and its dual $\ni$) that annihilates any other node. The main difficulty is the removal of the top node, which acts as delimiter for the erasing process. For that, we use a (and dual) node, whose function is to traverse the net f and remove the top duplicator as soon as g has been erased. Once again, it must be ensured that this process does not interfere with other cancellations and fusions.

With respect to indices, nodes behave like duplicators as they move upwards in the net. Indices are single integers, as they are not affected by the additive constructs. During the traversal, nodes perform a correction on the indices of co-duplicators. This is because co-duplicators may have crossed the top duplicator and had their indices updated.

A representative subset of the rules for cancellation is shown in Figure 9. In

brief, cancellation is triggered by the interaction between a pair-constructor and a projection node (rule cancelM-1), after which the ε node will discard the portion of the net that corresponds to the cancelled subterm. The node will then traverse the preserved subterm and synchronize with the ε node on the top duplicator. The example reduction shown below illustrates the use of cancellation laws.



6 Sum-product Net Rewriting

A local graph-rewriting system will now be given for sum-product nets, based on the ideas discussed informally in the previous section. We first need to establish an appropriate notion of graph-rewriting rule: both the left-hand side (LHS) and the right-hand side (RHS) of the rule are finite nets, with the same sets of input and output ports (in other words the rule preserves the interface of the net). Moreover both the LHS and RHS nets are well-typed and well-formed, the rule preserves type and position labellings of the inputs and outputs, and does not introduce cycles.

The application of a rule in a typed net replaces any subnet matching its LHS by its RHS; the conditions above guarantee that there will be no edges left dangling. The system introduced below enjoys additionally the following:

There are no two rules in the system with the same LHS, or such that the LHS of a rule is a subnet of the LHS of the other;

The RHS of each rule does not contain as a subnet the LHS of another rule;

The set of rules is *dual-complete*: the dual of each rule is also in the system.

This has some of the defining properties of an interaction net system [9]; further requirements of such a system are that each node should have a distinguished principal port, and the LHS of every rule should consist of two nodes with an edge connecting both principal ports. This requirement would be sufficient to guarantee strong local confluence, which is not a property of our system.

De nition 6.1 The *Sum-Product Rewriting System* is defined by the rules given in Appendix B. An *admissible net* is any reduct of a term net.

It is straightforward to see that the system is *strongly normalizing* (in general ^ and nodes go up; _ and nodes go down; commutations between three ^ or _ nodes are index-regulated in a way that prevents non-termination). In what follows, we will implicitly invoke duality, which allows for a considerable economy

of effort during the study of the rewriting system.

When we restrict attention to admissible nets, the system has a controlled behaviour that will be explored later when proving the main results of the paper. Let us now state one of these results for future reference (see appendix A for proof).

Lemma 6.2 *In admissible nets, indices can be reset by reduction.*

Confluence

It is easy to see that the rewriting system presented above is not in general confluent for sum-product nets. However it is confluent for admissible nets.

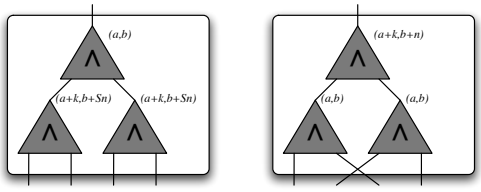
As usual, confluence of the system will be established by studying all the critical pairs induced by the rules. Most are resolved in a purely local fashion. For some pairs however, this will not be enough since the convergence of the reduction paths relies on global properties of admissible nets. In order to regain the locality in the analysis of critical-pairs, we will now introduce a suitable notion of equivalence.

De nition 6.3 Let N be a single input, n-output net and A a two-input, single-output node. We define $\mathsf{JOIN}(A,\mathsf{N})$ as the two-input, n-output net composed of two copies of N and n copies of A (say A_i) where the inputs of the net are the inputs of each copy of N, the ith output of the first (resp. second) copy of N is connected to the first (resp. second) input of A_i, and the ith-output of the net is the output of A_i.

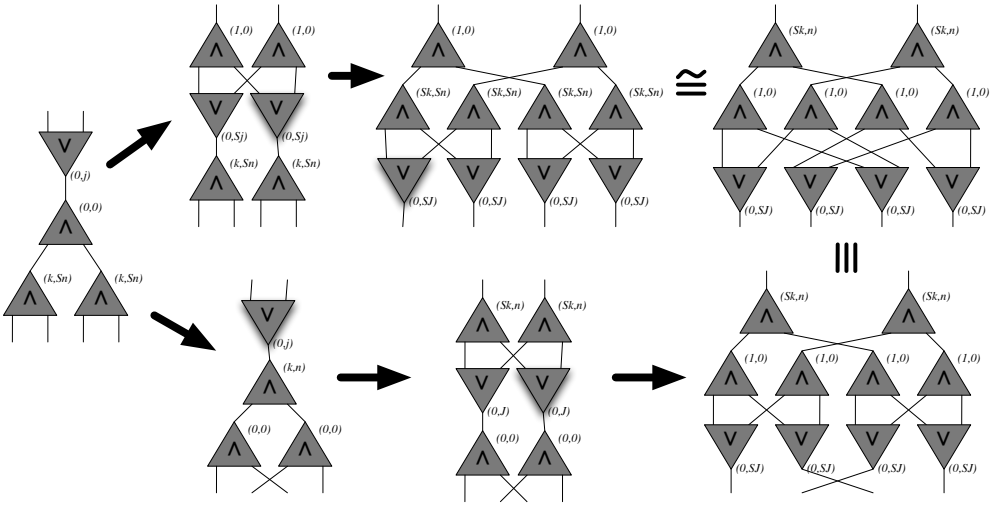
De nition 6.4 Let N_1 and N_2 be two single-input, n-output nets. They are said to be *twin equivalent with respect to a node* A, written $\mathsf{N}_1 =^\mathsf{A} \mathsf{N}_2$, when for every net N containing an occurrence of $\mathsf{JOIN}(A,\mathsf{N}_1)$, N has a common reduct with the net resulting from substituting $\mathsf{JOIN}(A,\mathsf{N}_1)$ by $\mathsf{JOIN}(A,\mathsf{N}_2)$ in N.

Lemma 6.5 *The following nets are twin-equivalents with respect to a* _ *node with some index* (i,j).

Proof. Let us denote by $\mathsf{N}_1^{(\mathsf{a};\mathsf{b})}$ and $\mathsf{N}_2^{(\mathsf{a};\mathsf{b})}$ the following nets:



Each of these parameterized nets can be treated as an (indexed) node, by computing the derived rules of interaction between the net and (a node for) each symbol.



where $J = \begin{cases} j & \text{, if } n > j \\ Sj & \text{, otherwise:} \end{cases}$

Fig. 10. Example Critical Pair

For every such symbol, the derived rule is exactly the same for $N_1^{(a;b)}$ and $N_2^{(a;b)}$. Now consider a net N containing an occurrence of $\mathsf{JOIN}(A, N_1^{(a;b)})$. By Lemma 6.2, the top duplicator in $N_1^{(a;b)}$ has a reduction sequence that resets its index to $(0,0)$. Finally, observe that this reduction sequence can be mimicked in the net $N_2^{(a;b)}$ yielding $N_2^{(0;0)}$, and $N_1^{(0;0)}$ reduces by rule dupM-dupM to $N_2^{(0;0)}$. The impact on N is precisely the same. 2

This proof illustrates the primary purpose of twin-equivalence in the proof of confluence: it allows to overcome the restrictions on the application of the commutation rules, which incidentally are introduced in the system precisely to achieve confluence, as they avoid critical pairs between commutations.

Proposition 6.6 *The rewriting system is confluent for admissible nets.*

Proof. Considering all the critical pairs generated by the rules given in appendix B.

Let us illustrate this with the critical pair formed by rules dupM-dupM and dupM-dupS, which is an interesting case. We first assume that the first (additive) component of the _ index is 0. We are led to the pair of reduction sequences shown in Figure 10, where denotes structural equality of nets and = denotes the fact that the two nets have a common reduct, as granted by Lemma 6.5. When _ has a positive additive index, the index of the top duplicator is not modified, and the invocation of twin-equivalence can be replaced by the application of rule dupM-dupM. 2

Soundness and Completeness

The reduction system given in Appendix B induces the following definition of equivalence of term nets. Let denote structural equality of nets.

Definition 6.7 Two term nets G_1, G_2 are *equivalent*, written $G_1 = G_2$, if there exist sum-product nets G_1^0, G_2^0 such that G_1 ! G_1^0 and G_2 ! G_2^0, and G_1^0 G_2^0.

We may now establish the main results relating the equational theory and the graphical system.

Proposition 6.8 (Soundness) *Let t, u be T_{PF} terms. Then*

$$t = u \;\Rightarrow\; \mathsf{T}(t) = \mathsf{T}(u)$$

Since reduction of a term net does not necessarily produce another term net, in the following completeness result the common reduct of $\mathsf{T}(t)$ and $\mathsf{T}(u)$ may be a sum-product net that is not a term net.

Proposition 6.9 (Completeness) *Let t, u be T_{PF} terms. Then*

$$\mathsf{T}(t) = \mathsf{T}(u) \;\Rightarrow\; t = u$$

Proofs of both results can be found in Appendix A.

7 Conclusions and Future Work

Together, the translation $\mathsf{T}(\ )$ and the graph-rewriting system solve three problems:

The translation directly captures the reflection laws, because it expands identities according to their types.

To ensure that the fusion laws are effectively captured, commutations between configurations involving 3 nodes must be allowed (rules dupM-dupM, dupM-choice, dupS-dupS and pair-dupS), regulated by an indexing scheme.

Finally, this indexing scheme must be capable of handling fusions in terms such as $\mathsf{h}a, b\mathsf{i}.[c, d]$, which may happen in two directions. In our system, such a fusion results in a (unique) net, which is no longer a term net.

It is possible to draw a parallel between our translation phase and the use of expansions in extensional lambda-calculi. This is sharply seen when we compare it with the simulation account of expansions found in [4], where a similar preprocessing phase performs all the required expansions. Our translation is considerably simpler, since we do not need to simulate expansions on the translated terms.

An adequate treatment of the exponential fragment of the calculus is the next obvious step. This introduces new problems, related to the work on encodings of the λ-calculus into interaction nets. The initial and terminal objects and their associated morphisms can easily be incorporated in our system.

References

[1] Thorsten Altenkirch and Peter Dybjer and Martin Hofmann and Phil Scott, Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303-310. IEEE Press, 2001.

[2] Asperti, A., Giovannetti, C., and Naletto, A. (1996). The Bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810.

[3] Bird, R., de Moor, O.: *Algebra of Programming*, Prentice Hall, 1997.

[4] Roberto di Cosmo and Delia Kesner. Simulating expansions without expansions. Math. Structures in Computer Science, 4(3):1-48, 1994

[5] D. Dougherty, Some -calculi with categorical sums and products. In *Rewriting Techniques and Applications*, volume 690 of Lecture Notes in Computer Science, pages 137–151. Springer Verlag, 1993.

[6] Neil Ghani, Beta-Eta Equality for Coproducts, In *Proceedings of TLCA'95*, pages 171-185, vol.902 LNCS. Springer-Verlag, 1995.

[7] J.-Y. Girard: Towards a Geometry of Interaction, In *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*, pages 69–108. 1989.

[8] Gonthier, G., Abadi, M., and Lvy, J.-J. (1992a). The geometry of optimal lambda reduction. In *Proceedings of ACM Symposium Principles of Programming Languages*, pages 15–26.

[9] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[10] Joachim Lambek and Phil J. Scott. *Introduction to higher order categorical logic*, volume 7 of Cambridge studies in advanced mathematics. Cambridge University Press, 1986.

[11] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh. Available from http://www.dcs.ed.ac.uk/home/gdp/publications/, 1981.

[12] J. R. B. Cockett and R. A. G. Seely Finite sum - product logic. In *Theory and Applications of Categories*, Vol. 8, 2001, No. 5, pp 63-99.

A Proofs

Proof of Lemma 6.2

Rules dupM-dupMand dupM-choice are called *commutation rules* as they promote a swap of nodes. A net is said to admit "half a commutation" if it contains a subnet matching the top and one of the bottom nodes of the LHS of a commutation rule.

Lemma A.1 *Let G be an admissible net that admits half a commutation. Then there exists a net G^0 such that G ! G^0 and the partial match in G^0 is extended to the full match of the LHS of the commutation rule.*

Proof. In an admissible net, for every ? or ^ node with index $(0,0)$, the agent destined to commute with it (if any) can be identified by a recursive procedure that follows the definition of the net, starting from one output of the agent. The absence of commutation rules with non-ground duplicators guarantees that this node does not change until commutation actually occurs. Moreover, the fact that rules dupM-pair and dupM-dupSinject duplicators in both inputs of the interacting agents, guarantees that the result is the same when computed on any of the output ports of the given node. 2

With this we can now prove Lemma 6.2:

Proof. Indices are adjusted during reduction in accordance with well-formedness. On the other hand, rules for indexed nodes are such that they do not constrain interaction when the indices are non-zero. This means that an indexed node can only survive in a normal form if: (i) it reaches the top of the net; or (ii) it gets stuck in a commutation that is not possible. The well-formedness criterion guarantees that (i) is not possible for admissible nets, and Lemma A.1 shows that (ii) could never occur in a normal form. 2

Soundness

Lemma A.2 *Let t be a T_{PF} term, and $G^{\wedge}(t)$ the net obtained by connecting a $\wedge$ node with index (a,m) where $m>0$, to the output of the term net $\mathsf{T}(t)$. Then $G^{\wedge}(t)$! $G^{\wedge}(t)$, where $G^{\wedge}(t)$ consists of a $\wedge$ node with index (a,m), whose outputs are connected to the inputs of two nets G_{l}, G_{r} such that $G_{\mathsf{l}}=G_{\mathsf{r}}=\mathsf{T}(t)$.*

Proof. By induction on the structure of t. Note that the lemma is not valid for $m=0$, in the particular case that t is of the form (or a composition of terms ending with) $[f,g]$, in which case the duplicator node does not dominate the _. 2

Lemma A.3 *Let t be a T_{PF} term, and $G^{\shortparallel}(t)$ the net obtained by connecting an ε node to the output of the term net $\mathsf{T}(t)$. Then $G^{\shortparallel}(t)$! $G^{\shortparallel}$, where $G^{\shortparallel}$ consists of a single ε node.*

Proof. By induction on the structure of t, using the rules where ε appears in the left-hand side. 2

Lemma A.4 *Let t be a T_{PF} term, G (t) the net obtained by connecting the input of a node indexed with n to the output of the term net $\mathsf{T}(t)$, and G (t) obtained by connecting the output of a node (again indexed with n) to the input of $\mathsf{T}(t)$. Then G (t) ! G (t).*

Proof. By induction on the structure of t, using rules where appears in the left-hand side. 2

Lemma A.5 *Let N be a sum-product net with a single input and output, and let $\mathsf{N}-$ denote the net obtained by incrementing the second component of the indices of top co-duplicators in N, i.e. (a,m) becomes $(a,m+1)$ for every co-duplicator that is not located under another co-duplicator.*

Let also N^{0} be the net obtained by plugging a net $\mathsf{T}(\mathsf{id})$ (where the identity has the appropriate type) on top of N. Then $\mathsf{N}-=\mathsf{N}^{0}$.

Proof. Straightforward induction on the type of the identity. 2

Dual lemmas to the above can be proved. We will refer to these as Lemmas A.2', A.3', A.4',and A.5'.

Proposition A.6 (Soundness) *Let t, u be T_{PF} terms with $t=u$. Then $\mathsf{T}(t)=\mathsf{T}(u)$.*

Proof. By cases of the definition of equality of terms.

$\mathsf{T}(\mathsf{id}\ \ f)\ \ \mathsf{T}(f\ \ \mathsf{id})\ \ \mathsf{T}(f)$
(straightforward)

$\mathsf{T}((f\ \ g)\ \ h)\ \ \mathsf{T}(f\ \ (g\ \ h))$
(straightforward)

$\mathsf{T}(\langle\pi_1,\pi_2\rangle)\ \ \mathsf{T}(\mathsf{id})$
Guaranteed by construction. Observe that the term $\langle\pi_1,\pi_2\rangle$ necessarily has type $A\ \ B\ !\ \ A\ \ B$ for some types A, B, and

$$\mathsf{T}(\langle\pi_1,\pi_2\rangle^{A\ \ B!\ A\ \ B})\ \ \mathsf{T}(\mathsf{id}^{A\ \ B!\ A\ \ B})$$

$\mathsf{T}([\mathsf{i}_1,\mathsf{i}_2])\ \ \mathsf{T}(\mathsf{id})$
Dual to the previous case.

$\mathsf{T}(\langle f,g\rangle\ \ h) = \mathsf{T}(\langle f\ \ h,g\ \ h\rangle)$
By induction on the structure of h

(i) $h = h_1\ \ h_2$ – we use $\mathsf{T}((f\ \ g)\ \ h)\ \ \mathsf{T}(f\ \ (g\ \ h))$ and then the inductive hypothesis applies.

(ii) h is a constant – straightforward.

(iii) $h = \langle a,b\rangle$ – straightforward using Lemma A.2, which can be used since the duplicator has index (0,1) after duplicating the $(,)$ node.

(iv) $h = [a,b]$ – this is the hard case since Lemma A.2 does not apply.

In the net $\mathsf{T}(\langle f,g\rangle\ \ [a,b])$ after one step of rule dupM-dupSthe split duplicators have indices (1,0); they will duplicate the nets $\mathsf{T}(a)$ and $\mathsf{T}(b)$ incrementing the (second component of the) indices of the top co-duplicators. Finally, the split duplicator will commute with the top ? node.

In the net $\mathsf{T}(\langle f\ \ [a,b],g\ \ [a,b]\rangle)$ on the other hand, the translation introduces the encoding of an identity of sum type on top. Proceeding with reduction one obtains a net that is similar to the previously obtained, except that the nets $\mathsf{T}(a)$ and $\mathsf{T}(b)$ appear intact, with identities of sum type on top of each such net. Lemma A.5 then yields $\mathsf{T}(\langle f,g\rangle\ \ [a,b]) = \mathsf{T}(\langle f\ \ [a,b],g\ \ [a,b]\rangle)$.

$\mathsf{T}(h\ \ [f,g]) = \mathsf{T}([h\ \ f,h\ \ g])$
Dual to the previous case using Lemmas A.2' and A.5'.

$\mathsf{T}(\pi_1\ \ \langle f,g\rangle) = \mathsf{T}(f)$ and symmetrically $\mathsf{T}(\pi_2\ \ \langle f,g\rangle) = \mathsf{T}(g)$
If the domain of $\langle f,g\rangle$ is not a sum type and the codomain of π_1 is a ground type, then $\mathsf{T}(\pi_1\ \ \langle f,g\rangle\ \ !\ \ \mathsf{T}(f)$ by rule cancelM-1, LemmasA.3 and A.4, and finally rule epsilon-dupM-2.

Otherwise,

(i) If the domain of $\langle f,g\rangle$ is of the form $C+D$, the translation introduces an additional net on top, corresponding to the encoding of an identity of type $C+D$. We have $\mathsf{T}(\pi_1\ \ \langle f,g\rangle\ \ !\ \ \mathsf{T}(f\ \ \mathsf{id}) = \mathsf{T}(f)$.

(ii) If the codomain of π_1 is not a ground type, the translation introduces an additional net at the bottom, corresponding to the encoding of the identity at that type. We have $\mathsf{T}(\pi_1\ \ \langle f,g\rangle\ \ !\ \ \mathsf{T}(\mathsf{id}\ \ f) = \mathsf{T}(f)$.

$\mathsf{T}([f,g]\ \ \mathsf{i}_1) = \mathsf{T}(f)$ and symmetrically $\mathsf{T}([f,g]\ \ \mathsf{i}_1) = \mathsf{T}(g)$

Dual to the previous case using Lemmas A.3' and A.4'.

2

Completeness

The proof of completeness uses a path-based interpretation of terms, in the style of the Geometry of Interaction [7].

De nition A.7 (Read-back) We define a labelling of sum-product nets from top to bottom as follows, where the labels are T_{PF} terms extended with the constants L, R, and ε.

If the input of a ^ node is labelled α then both its outputs are labelled α.

If the inputs of a $(,)$ node are labelled $\beta\ \ \alpha$ and $\beta^0\ \ \alpha$ (where α is the longest common suffix) then its output is labelled $\mathsf{h}\beta, \beta^0\mathsf{i}\ \ \alpha$. Note that if the labels are equal then we take $\beta = \beta^0 = \mathsf{id}$.

If the input of a ? node is labelled γ then its outputs are labelled $L\ \ \gamma$ and $R\ \ \gamma$ respectively.

For _ nodes there are two cases:

If its inputs are labelled $\beta\ \ \alpha\ \ L\ \ \gamma$ and $\beta\ \ \alpha^0\ \ R\ \ \gamma$ (where β is the longest common prefix and γ is necessarily a common suffix) then its output is labelled $\beta\ \ [\alpha, \alpha^0]\ \ \gamma$. Note that if β extends until L and R then $\alpha = \alpha^0 = \mathsf{id}$.

If its inputs are labelled α and $\beta\ \ \varepsilon$ (in any order) then its output is labelled α.

If the input of a pair projection node π_1 (resp. π_2) is labelled α then its output is labelled $\pi_1\ \ \alpha$ (resp. $\pi_2\ \ \alpha$).

If the input of a choice injection node i_1 (resp. i_2) is labelled α then its output is labelled $\mathrm{i}_1\ \ \alpha$ (resp. $\mathrm{i}_2\ \ \alpha$).

The output of a ε node is labelled ε.

If the input of a node is labelled α then its output is also labelled α.

If the input of a node is labelled α then its output is also labelled α.

Given $x\ \ \mathsf{T}_{\mathrm{PF}}$ and a sum-product net N with a single input and a single output, we define its *read-back* $\mathsf{R}_{\mathsf{x}}(\mathsf{N})$ as the label of its output, given uniquely from the above rules after labelling the input of N with x. We will write simply $\mathsf{R}(\mathsf{N})$ for $\mathsf{R}_{\mathsf{id}}(\mathsf{N})$.

For sum-product nets in general the read-back can be generalized as taking a vector of n inputs and producing a vector of m outputs (both indexed from left to right), $\mathsf{R}_{\mathsf{x}}(\mathsf{N}) = y_1, \ldots, y_{\mathsf{m}}$ where $\mathsf{x} = x_1, \ldots, x_{\mathsf{n}}$.

Finally, we extend the equational theory of terms with the following equations relating the new constants introduced in the labels (ranged over by α):

$$L\ \ \mathrm{i}_1 = \mathsf{id} \qquad L\ \ \mathrm{i}_2 = \varepsilon$$
$$R\ \ \mathrm{i}_1 = \varepsilon \qquad R\ \ \mathrm{i}_2 = \mathsf{id}$$
$$\alpha\ \ \varepsilon = \varepsilon$$

Lemma A.8 *Let* $\mathsf{N}_1, \mathsf{N}_2$ *be sum-product nets; if* $\mathsf{N}_1\ !\ \mathsf{N}_{\ 2}$ *then* $\mathsf{R}_\mathsf{x}(\mathsf{N}_1) = \mathsf{R}_\mathsf{x}(\mathsf{N}_2)$.

Proof. All the net reduction rules preserve the read-back. We give two examples: in rule choice-dupS the inputs must have labels respectively of the form $\beta\ \ \alpha_1\ \ L\ \ \gamma$ and $\beta\ \ \alpha_2\ \ R\ \ \gamma$, or else α and $\beta\ \ \varepsilon$; in the first case, on both sides of the rule the outputs will be labelled $L\ \ \beta\ \ [\alpha_1, \alpha_2]\ \ \gamma$ and $R\ \ \beta\ \ [\alpha_1, \alpha_2]\ \ \gamma$; in the second case the outputs are labelled $L\ \ \alpha$ and $R\ \ \alpha$ on both sides.

In rule cancel-S1, for input α we have output $L\ \ \mathrm{i}_1\ \ \alpha$ on the left-hand side and α on the right-hand side, which are equal under the augmented equational theory. 2

We remark that the read-back of an admissible net is necessarily a term of T_{PF}, and in particular:

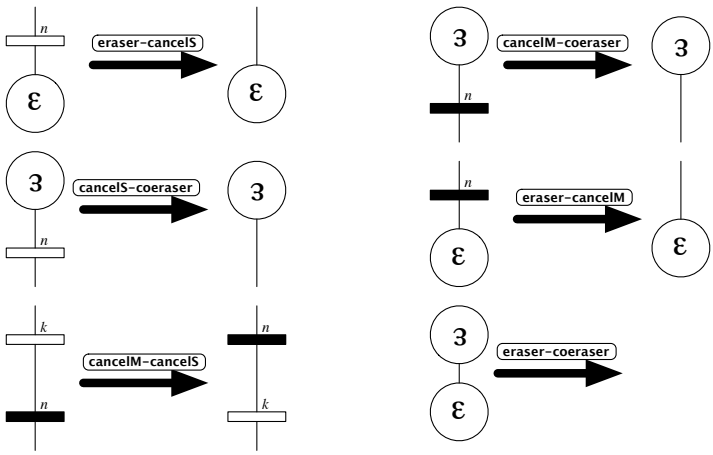
Lemma A.9 *For any* t 2 T_{PF}, $\mathsf{R}\,(\mathsf{T}(t)) = t$.

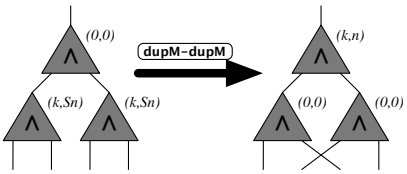
Proof. The stronger result $\mathsf{R}_\mathsf{x}(\mathsf{T}(t)) = t\ \ x$ can be proved by induction on the structure of t. 2

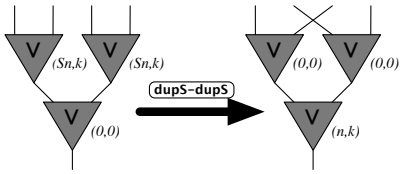
Proposition A.10 (Completeness) *Let* t, u 2 T_{PF} *be such that* $\mathsf{T}(t) = \mathsf{T}(u)$. *Then* $t = u$.

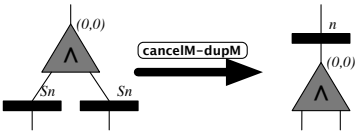
Proof. For $\mathsf{T}(t) = \mathsf{T}(u)$ to hold there must exist sum-product nets $G_\mathsf{t}, G_\mathsf{u}$ such that $\mathsf{T}(t)\ !\ \ G_\mathsf{t}, \mathsf{T}(u)\ !\ \ G_\mathsf{u}$, and $G_\mathsf{t}\ \ G_\mathsf{u}$. By lemma A.8 we have that $\mathsf{R}\,(\mathsf{T}(t)) = \mathsf{R}\,(G_\mathsf{t})$ and $\mathsf{R}\,(\mathsf{T}(u)) = \mathsf{R}\,(G_\mathsf{u})$. Now by lemma A.9 and because structurally equal nets have the same read-back, we have $t = u$. 2

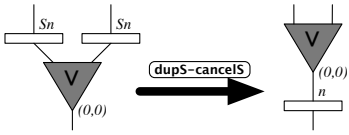
B Full Set of Rewrite Rules

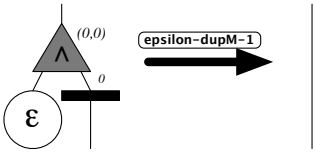


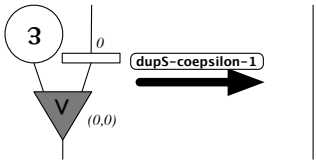


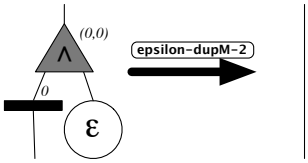


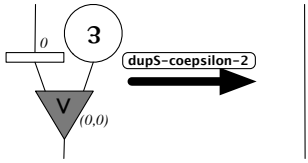


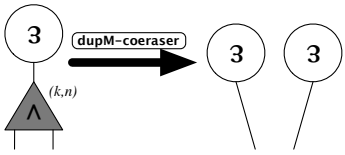


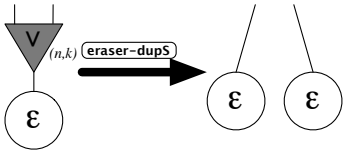


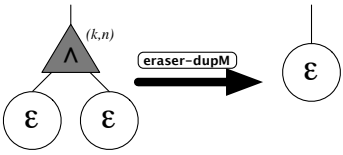


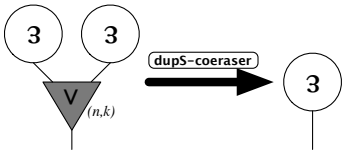


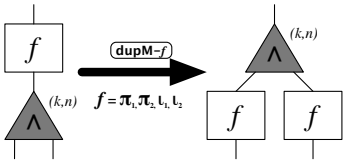


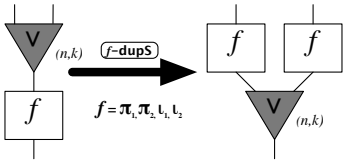


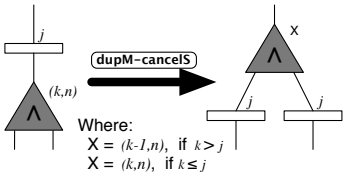


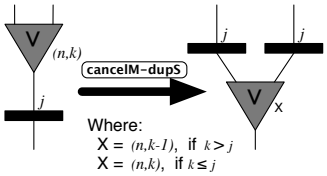


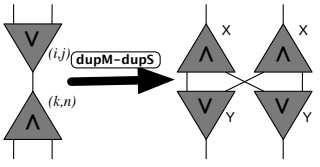






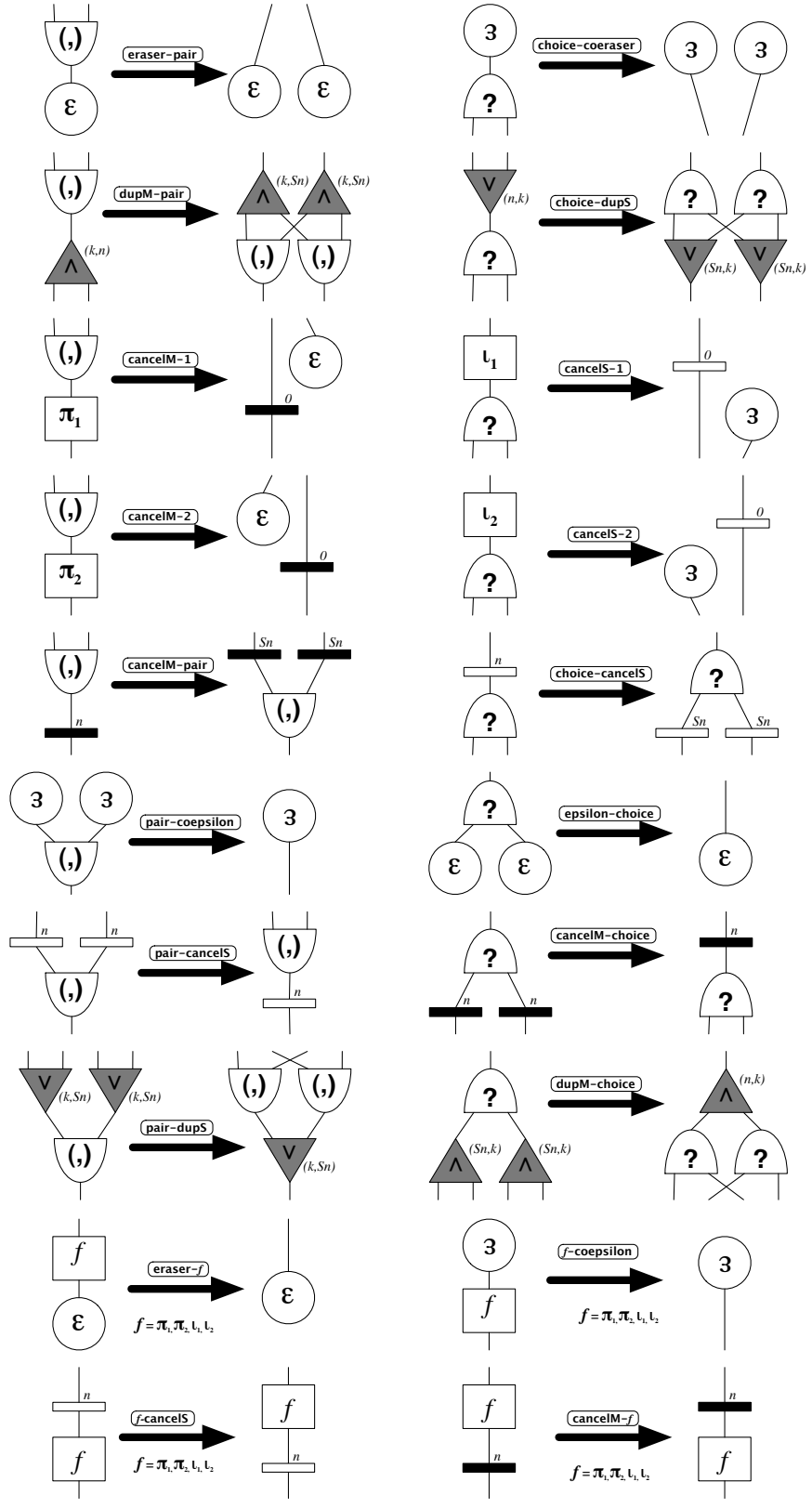






Where:

X = (k,n), if $i > k$
X = (Sk,n), if $i \leq k$
Y = (i,j), if $n > j$
Y = (i,Sj), if $n \leq j$


eraser-pair
choice-coeraser
dupM-pair
choice-dupS
cancelM-1
cancelS-1
cancelM-2
cancelS-2
cancelM-pair
choice-cancelS
pair-coepsilon
epsilon-choice
pair-cancelS
cancelM-choice
pair-dupS
dupM-choice
eraser-f
f-coepsilon
f-cancelS
cancelM-f
f = π1, π2, ι1, ι2