A Local Graph-rewriting System for Deciding Equality in Sum-product Theories (Extended Abstract)

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

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

1 Introduction

The *point-free* style of programming [1] has been defended as a good choice for reasoning about functional programs. However, when one actually tries to construct a decision procedure for the associated equational theory, one faces problems, even when small fragments of the theory are considered.

In this paper we outline how a graph-based decision procedure can be given for the functional calculus with sums and products (but no exponentials – the expressions we use here can not really be seen as a programming language). We show in turn how the system covers *reflexivity* equational laws, *fusion* laws, and *cancelation* laws.

The decision procedure has interest independently of our initial motivation. The term language (and its theory) can be seen as the internal language of a category with binary products and coproducts. 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.

2 The Term Language and Theory

Consider the following language $\mathcal{T}_{\mathrm{PF}}$ for types and terms:

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

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

where A is a set of *base types* and $C^{\mathsf{Type},\mathsf{Type}}$ 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 $A : f : B$ the assertion that term f has domain A and codomain B. The typing rules associated to the language are the following

$$\frac{}{A : c^{A,B} : B}\; c^{A,B} \in C^{A,B} \qquad \frac{}{A : id^A : A} \qquad \frac{A : f : B \qquad B : g : C}{A : g \cdot f : C}$$

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

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

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

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

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

Deciding equality under the theory defined by these equations requires producing a decision procedure. The simplest way to accomplish this is to orient the equations to obtain a confluent, terminating rewriting system (possibly by means of a completion process). Unfortunately, in this case it is not possible to conduct this program. Even considering the multiplicative fragment alone (i.e. ignoring the terms that involve sums), we face problems when constructing a rewriting system from the corresponding laws.

2.1 Difficulties

In the multiplicative sub-system, the orientation *left to right* seems sensible for the equations given above, but creates unsolvable critical pairs induced by the reflection laws. To illustrate this problem consider the following derived law (*surjective pairing*)

$$f = \mathsf{id} \cdot f = \langle \pi_1, \pi_2 \rangle \cdot f = \langle \pi_1 \cdot f, \pi_2 \cdot f \rangle$$

Both extremes of the equality chain are in normal form with respect to the rewrite system obtained, thus it fails to be complete.

A closer look at the reflection law gives us a hint of what the problem is – it drops from the term structural information that is essential for the confluence of the system. An approach to overcoming this problem consists in imposing that all the rewrites preserve the structural information (allowing for the reconstruction of types), together with the proviso that the starting term contains all the structural information to reconstruct its type structure. In practice, we can drop *identities* from the language, except at base types, and the reflexivity law can be dropped from the rewriting system – it becomes a rule for defining identities of structured types. As an example, the identity of type $(A \times B) \times C$ is defined as $\langle\langle \pi_1, \pi_2 \rangle \,.\, \pi_1, \pi_2 \rangle$.

Constant functions should also carry their structural information. To avoid restricting constant functions to base types, we may instead exhibit that information by composing the functions with appropriate identities (defined as above). This means that a normal form of a constant function f with codomain $A \times B$ is the normal form of $\langle \pi_1, \pi_2 \rangle \cdot f$, that is $\langle \pi_1 \cdot f, \pi_2 \cdot f \rangle$. Equations like surjective pairing are then satisfied by construction.

Obviously, restricting our attention to the additive fragment will lead to dual arguments. However, when both products and sums are considered, a simple rewriting approach faces irremediable problems: not only does associativity of composition become a concern (there no longer exists a sensible orientation for it), but products and sums interact in such a symmetrical way that the rewriting system cannot "choose" a certain form to the detriment of its dual. To exhibit an example that illustrates this last observation, consider the following equality derivation (known as the *exchange law*):

$$\begin{aligned}
\langle [f,g],[h,k] \rangle &= \langle [f,g],[h,k] \rangle \cdot [\mathrm{i}_1, \mathrm{i}_2] \\
&= [\langle [f,g],[h,k] \rangle \cdot \mathrm{i}_1, \langle [f,g],[h,k] \rangle \cdot \mathrm{i}_2] \\
&= [\langle [f,g] \cdot \mathrm{i}_1, [h,k] \cdot \mathrm{i}_1 \rangle, \langle [f,g] \cdot \mathrm{i}_2, [h,k] \cdot \mathrm{i}_2 \rangle] \\
&= [\langle f,h \rangle, \langle g,k \rangle]
\end{aligned}$$

In fact, to decide equality of the sum-product theory through a rewriting system, we must work modulo an appropriate equational theory that handles these equalities (see for instance [3]).

In this paper we follow a totally different approach: the graph-rewriting system introduced in the next sections captures associativity of composition for free, and moreover the interaction between the multiplicative and the additive fragments is adequately treated (for instance the two sides of the exchange law have the same normal form). The system includes however the treatment of reflexivity outlined above.

2.2 Local Graph Rewriting

A graphical representation for terms

2.2.1 Fusion Rules

Fusion is accomplished by the interaction with (co-)duplicators. Intuitively, a duplicator interacting with a net should perform a copy of that net (see Figure 1). However, this "duplication" should take in account that we intend it to be performed locally, i.e. the (co-)duplicators interact only with individual agents. 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 our presentation considering the multiplicative fragment of our language. Later, we elaborate on the adjustments required for dealing with the full language.

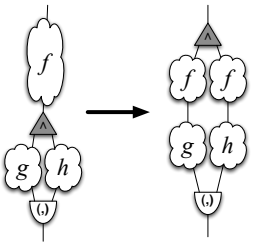


Fig. 1. Fusion as Net Duplication

Multiplicative Fusion

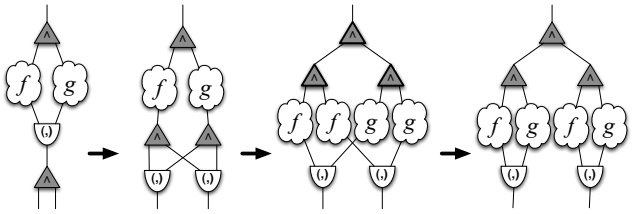
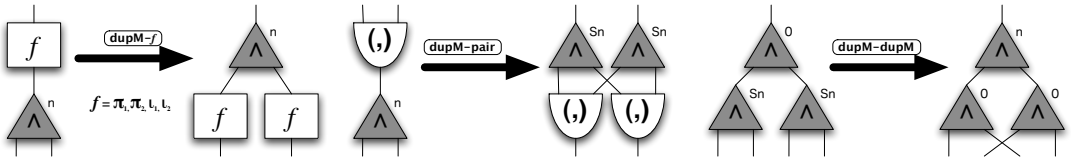


Fig. 2. Duplication of a Structured Net

When a duplicator meets a structured net (e.g. a split of two terms), it must split itself in order to duplicate each component of the net. Moreover, once concluded the duplication of each sub-net, 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 2). To control this reorganization of duplicators, we introduce *indexes*. We are lead to the following rules governing the interaction with duplicators.

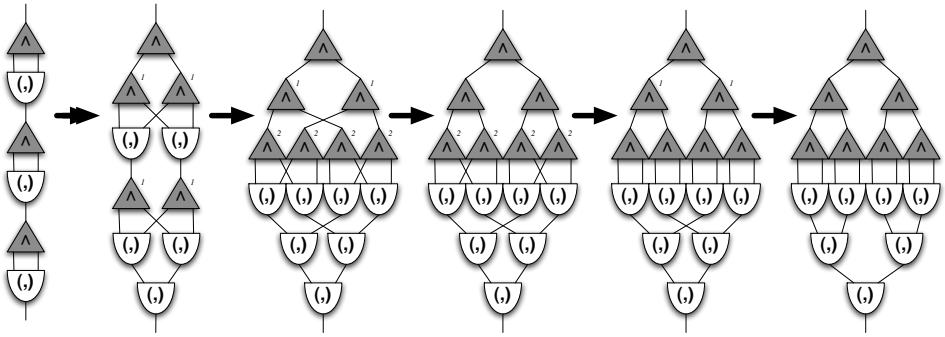


The first one is fairly obvious — the interaction with single input/single output agents simply duplicates them. When a duplicator interacts with a pair constructor, it not only duplicates it, but also splits itself in two in order

to duplicate each subnet. The last rule is the commutation rule between duplicators and is actually the counterpart of the spliting of duplicators referred in the previous rule — it closes the interaction with structured nets since, apart from the duplication of the top agent, it also rejoin the "once splited duplicator".

Indexes attached to duplicators are simply integers that record "how deep" they are in the transversal of a structured net [4]. When a duplicator have its index set to zero, we will call it *ground duplicator* and often omit its index from its graphical representation. Notice that the commutation rule actually restricts its top duplicator to be grounded.

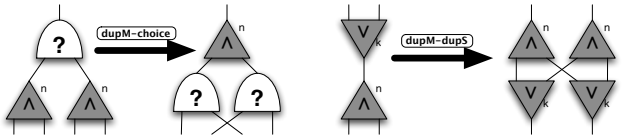
We should also stress the distinction made between "duplication" and "spliting" — a duplicator get splited to perform duplication of the two sub-nets, and eventually rejoin by duplicating the ground duplicator that tops both sub-nets (commutation rule). The following reduction sequence illustrates these rules in action:



Interaction of Sums and Products

What should be the interaction rule when a duplicator meets a co-duplicator? Structurally, it is fairly obvious that they should pass-through each other. The question is then whether the agents get duplicated or splited during this process. In other words, what is the impact *indexes* their indexes.

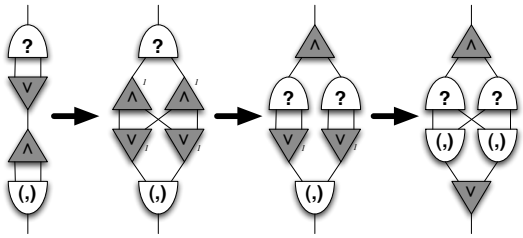
A first solution would be to state that only duplications take place. This corresponds to keep unaltered both indexes and immediately leads to a rule that allows duplicators to freely pass through choice agents (i.e. without index regulation). It corresponds to the following additional interaction rules (and their duals):



[4] In the conference version of this article, we had used a more informative notion of indexes that record the path taken during the net transversal. These indexes allow a finer control on the reduction process, but lead to much more complex arguments concerning the properties of the system.

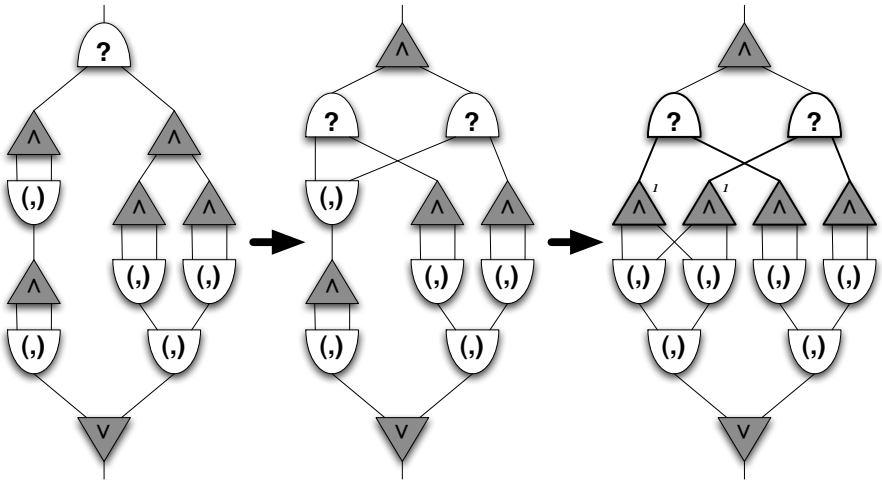
Let us start by enunciating the virtues of this approach:

- Keeping the multiplicative/additive indexes independent mean that we actually manage to keep both subsystems independent (or orthogonal). It certainly contributes to the simplicity and elegance of the system.
- The system automatically exhibits the "full symmetry" reclaimed above for terms like those appearing in both sides of the exchange law (as shown in the following simple reduction):



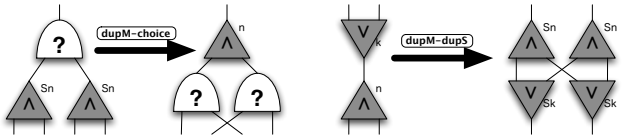
These normal forms exhibit that are not direct translations of splits or eithers. We will call these *full normal forms*, as they capture the fact that these nets can be read-back as different terms.

Unfortunately, the indexes no longer constrains appropriately the commutations that might take place in a reduction sequence. To see this, consider the following reduction sequence:

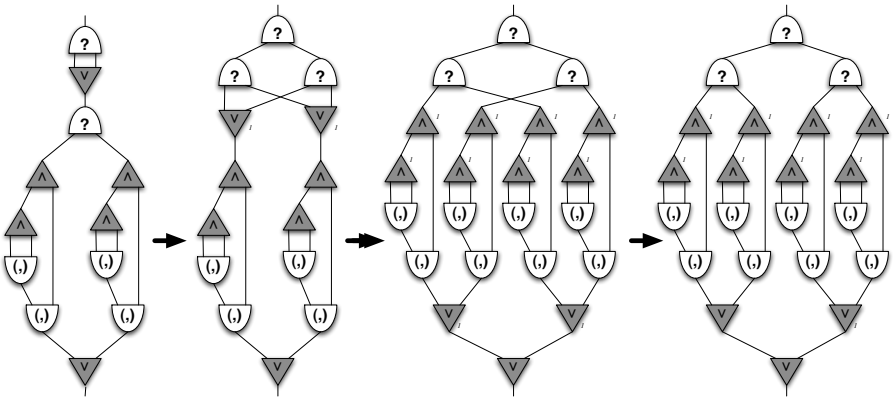


Note that the application of rule `dupM-choice` removes the ground duplicator needed to close the fusion started on the left hand-side sub-net. In fact, this is exactly the pattern of divergence that the restriction in the rule `dupM-dupM` avoids by restricting the top duplicator to be ground. This shows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

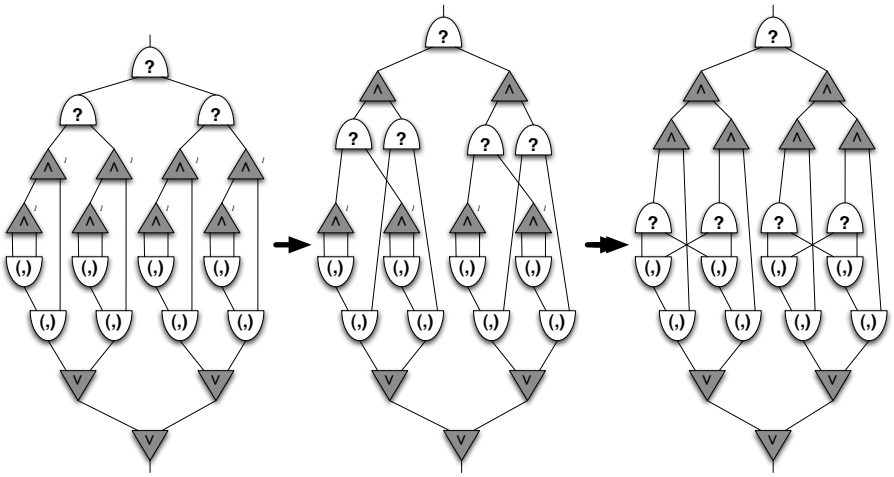
A second approach would be to let both agents get splited (i.e. both agents increment their index). These are the corresponding rules:



Here the problem became a lot more subtle. Let us first exhibit its manifestation a later refer why it is actually a problem. Consider the following reduction sequence:



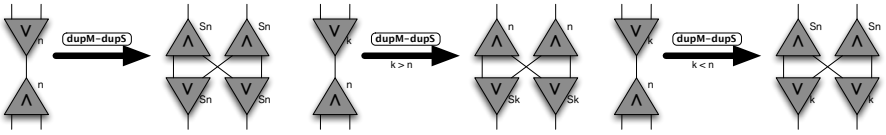
Notice that the transversal of the net by the coduplicator on the top actually lifts the indexes for all the duplicators in the lower sub-nets. This is not in accordance with the information description given above, but can actually appear as an interesting feature that can be exploited by the system. In fact, if we continue the reduction process, we get:



This example suggests that, in order to reach full normal forms, it is sufficient to promote enough fusions (something that might be accomplished by pre or post-composition with identities). The problem is that we do the additional reductions relying strongly on the symmetry of the net (more precisely, on the number and the tree shape of duplicators in both sides of the choice agent). It is true that the nets that come from well-typed point-free terms do exhibit an high degree of symmetry. However, we are not able to assure that all nets, under all reductions strategies, do possess the symmetry required by

this set of rules (a counterexample will be presented below).

A final approach for the interaction of both fragments would be to take a sort of a mixed version of the previous solutions: informally, we take one of the agents to be the "dominator" of the interaction. This agent would split itself (increases the index), and the other is simply duplicated (indexes remain unaltered). The rules became:

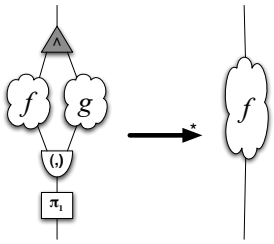


When both agents share the same index, we will treat them with equal status (as in the previous solution). When an agent has an higher index we consider it as the dominator, as it higher index reveals that it is deeper in the transversal of the net – as such, the other agent is part of a subnet.

It can be easily seen that this set of rules only lifts the index of the top duplicator in a net such as the one presented above. As such, it no longer depends on the symmetry of the sub-nets to proceed the reduction. On the down-side, we can also note that simple trick of identity composition to reach full normal forms is no longer enough. We will return to this theme later (Section 2.2.3).

2.2.2 Cancelation Rules

Cancelation is, by its own nature, related with erasure. Consider the structure of a cancelation rule:



The interaction of the bottom agents should trigger the erasure of the net g and the top duplicator.

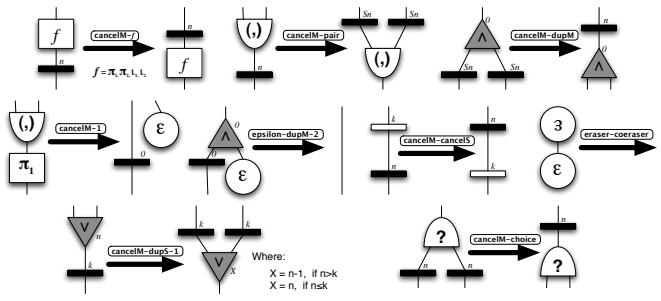


Fig. 3. Cancelation Rules

For the erasure of net g, we will take the standard approach of introducing special erasing agents ε / $\ni$ (denoted *eraser* and *coeraser*) that annihilate any other agent that interact with them. The main difficulty is thus the removal of the top agent, that also acts as delimiter for the erasure process. For that, we introduce another pair of agents (denoted *cancel* and *cocancel*, depicted respectively as small black and white filled boxes respectively) whose function is to transverse the net f and remove the top duplicator as soon as the erasure of the net g take place. Once again, we should take in account indexes that will assure that this process do not interfere with other cancelations/fusions that might take place on the net.

A representative subset of the rules for cancelation is given in figure 7. Here are some example reductions:

Reduções de exemplo (slides do jsp)

As a final example, let us show an example where both fusion and cancelation take place simultaneously.

FIGURA COM REDE ASSIMETRICA... e respectiva redução

The relevance of this example is that it is an instance of an asymmetrical net (and thus constitutes a counterexample for the second approach for the interaction of multiplicative and additive fusion). In a sense, this is a contrived example — it avoids the "natural" symmetry imposed by well-typedness through cancelation. If we reduce the net on the left hand-side of the choice agent, we recover a fully symmetric net.

2.2.3 Full normal forms

We already notice that the adopted solution for the interaction of multiplicative and additive fusion does not lead directly to full normal forms (such as the ones that would equalize both sides of the exchange law).

Let us note that this, in itself, does not turn the system unsound or of no use — we can still have a sound system, that simplifies point-free expressions. But it does mean that this system will fail to be "out of the box" complete (in the sense that, we cannot rely on simple syntactic comparison of normal forms to assert equality of the original terms). The question is thus if it is possible to augment the system in order to guaranty that full normal forms are reached.

The solution for this problem is guided by the observation that our system actually performs a first step towards full normal forms. In fact, we see that one of the duplicators actually crosses the choice agent since it starts with equal index relative to its associated closing coduplicator of the choice. The problem is that the inner duplicators are simply copied and do not see their indexes increased to a level that allow the commutation with the choice agent (in fact, we saw that such an increase would lead to confluence problems). What we would like is some manner to trigger these commutations. We will try to achieve this by composition with an appropriate net.

Consider the effect of composing the normal form obtained in the example

given by a net that "looks like" identity (denoted as *identity nets*).

Figura com a composição da rede com a ID

We saw that the obtained normal form is now in its full form, but several comments are certainly appropriate:

(i) Attached nets contain nonzero indexes — this is certainly non-standard from what we have presented earlier, where indexes appear during reduction. This is actually the "trick" that allow the activation of the intermediate duplicators that do not have enough height in their indexes to pass through the choice agent.

(ii) The normal form contain duplicators with non-empty indexes. In fact, this is hardly a surprise, since we start the reduction process with those duplicators. The situation should not be confused with the apparition of spurious indexes that we made reference above.

(iii) In general, we should attach two identity nets — one on the top and one on the bottom (according to the domain and codomain type of the corresponding net).

(iv) We must avoid the commutation between (co)duplicators inside identity nets. This is why we give them a special status (and color). The commutation rule for duplicators (and dually, for coduplicators) is restricted accordingly.

3 Sum-product Nets

Sum-product Nets will be built from instances of *symbols*; each symbol has an associated number of input ports (or *arity*) and number of output ports (or *co-arity*). We organize these symbols in *dual* pairs where the arity and co-arity are exchanged. These symbols are:

- a *duplicator* symbol with arity 1 and co-arity 2, depicted $\wedge$; its dual is the *co-duplicator*, depicted $\vee$;
- a *makepair* symbol with arity 2 and co-arity 1, depicted $(,)$; its dual is *choice* and 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 $\ni$.
- a *cancel* symbol with arity and co-arity 1, depicted $\blacksquare$; its dual *co-cancel* is depicted $\square$.

A *Net* is a tuple (S, E, I, O) where S is a set of occurrences of symbols, E is a set of *edges*, and I, O are two sets of *input ports* and *output ports* of the net. Input and output ports of the net do not belong to any symbol occurrence. Let S^I, S^O denote respectively the sets of input and output ports

of the symbol occurrences in S. Then each edge in E connects a port in $S^O \cup I$ (the output port of some symbol occurrence or an input of the net) to a port in $S^I \cup O$ (the input port of some symbol occurrence or an output of the net). Every port in $S^I \cup S^O \cup I \cup O$ belongs to exactly one edge. In the rest of the paper we refer to occurrences of symbols as *nodes*.

In what follows, $\wedge$, $\vee$, $\square$ and $\blacksquare$ nodes in a net will be labelled with non-negative integer indexes (we usually omit them when they are zero). These will be used to control the duplication and mutual annihilation of nodes in the reduction system presented in section 4.

A net is *well-typed* is 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 4 hold for every node (type variables are depicted as capital letters).

A *position* is a pair of non-negative integers (a, b), depicted as $a \cdot b$. 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 4 hold for every node ($S\ n$ denotes the successor of n). Well-formedness imposes a structural invariant on nets.

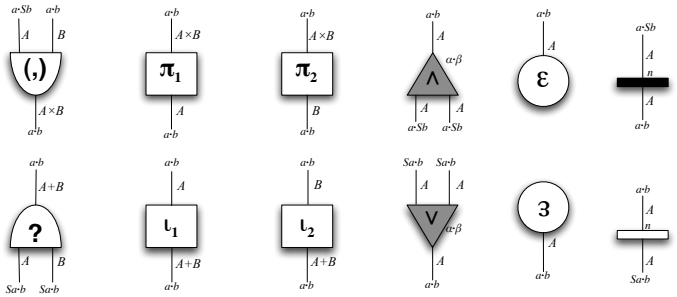


Fig. 4. Typing and Positioning Constraints

Definition 3.1 A *sum-product net* is an acyclic, well-typed and well-formed net with a single input and output, both labelled with empty positions. A sum-product net without $\square$ and $\blacksquare$ nodes and with all indexes set to zero is called an *elementary net*.

Figure 5 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 Sum-product Net Rewriting

A local graph-rewriting system will now be given for sum-product nets. 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, such that the sets of input and output ports are the same in both nets (in

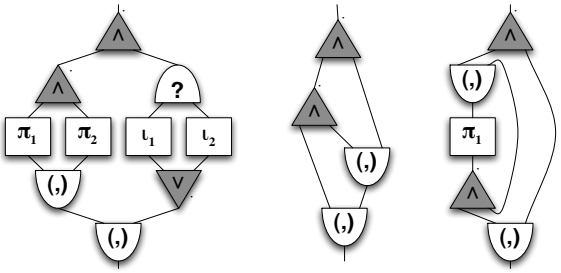


Fig. 5. Examples of Nets

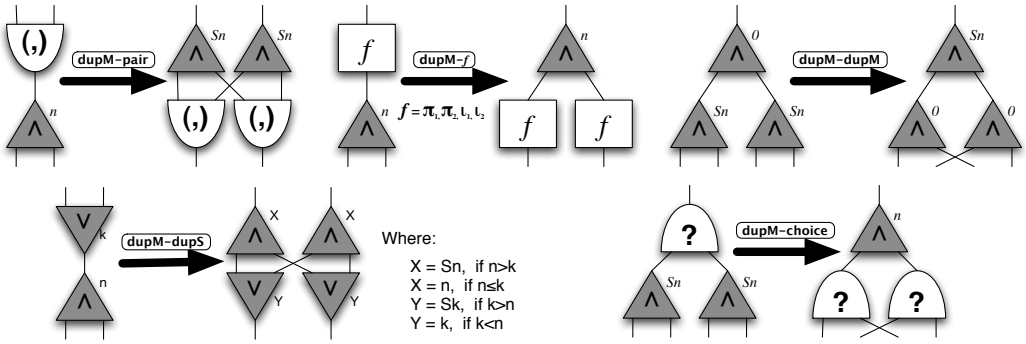


Fig. 6. Fusion Rules

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 [2]; 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 is sufficient to guarantee strong local confluence, which is not a property of our system.

The rules are introduced in two sets: a first set allows to decide the theory minus the cancelation laws; a second set of rules addresses these laws.

Fusion Rules.

Fusion rules are given in Figure 6 (we omit those rules that can be obtained by duality). These rules promote the upward movement of duplicators (dually,

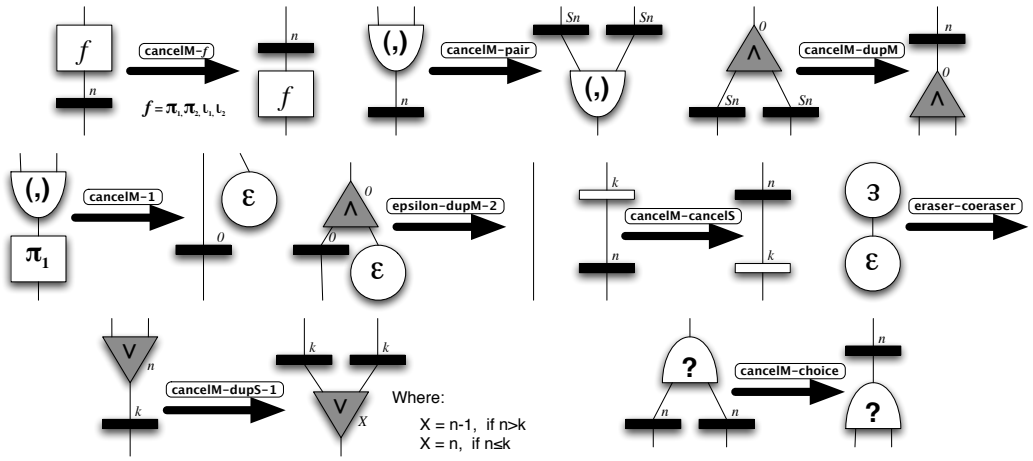


Fig. 7. Cancelation Rules

downward for co-duplicators). During XXXXXXXXX

Some remarks are in order:

(i) The comparison of indexes in the `dupM-dupS` rule guaranties that the index of a duplicator (or co-duplicator) is incremented only if it could reach the corresponding pair-constructor (or choice) agent.

Cancelation Rules.

The set of rules for cancelation is given in Figure 7. We omit the rules that can be obtained by duality and the garbage-collection rules for ε and $\ni$. The full set of rules is given on Appendix A. In these rules, cancelation is trigger by the interaction of the pair-constructor and a projection agent (rule `cancelM-1`. After that, two new agents will perform the work: the ε agent will discard the portion of the net that correspond to the canceled sub-term; the $\square$ agent will transverse the preserved sub-term and synchronize with the ε agent on the top duplicator. In a sense, $\square$ agents behave like duplicators. However, we should note the following differences: we should emphasize the following differences:

- They do not duplicate the transversed net (in fact, they could not, since they are single output agents);
- During the transversal, they perform a correction on the indexes of co-duplicators. This is because co-duplicators might have crossed the top duplicator (and thus, splited themselves), and thus requiring the corresponding index to be decreased to preserve the well-formedness of the net.
- Like duplicators, an index is attached to $\square$ agents. However, there is not commutation rule for $\square$ agents — they simply vanish when the top duplicator is reached.
- The indexes attached to $\square$ agents are not affected by the additive constructs — this simplification is possible due to the absence of a commutation rule

for □ agents.

Two example reductions are shown below. The fundamental role taken by indexes in controlling commutations is evident (rules `dupM-dupS` and `dupM-dupM`).

This reduction system thus induces the following definition of equivalence of sum-product nets. Let $\equiv$ denote structural equality of nets.

Definition 4.1 Two sum-product nets G_1, G_2 are *equivalent*, written $G_1 = G_2$, if there exist G_1', G_2' such that $G_1 \longrightarrow^* G_1'$ and $G_2 \longrightarrow^* G_2'$, and $G_1' \equiv G_2'$.

4.1 Termination

It is straightforward to see that the system is *strongly normalizing* (in general $\wedge$ and □ nodes go up; $\vee$ and ■ nodes go down; commutations between three $\wedge$ or $\vee$ nodes impose unique configurations).

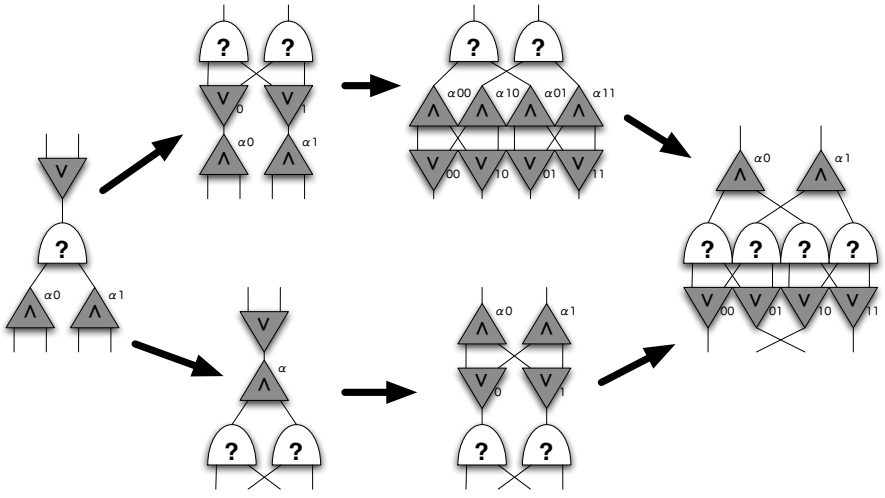
Do we really need (want) to prove this?

4.2 Confluence

The rewriting system exhibit a considerable number of critical pairs. Most of these do have a purely local resolution. For some, however, we must rely on the well-formedness assumption of sum-product nets. A nice example is given by the critical pair formed by the rules `dupM-dupM` and `dupM-dupS` — it is necessary to consider two cases, each of which fall in one of the categories referred above.

Consider the case where the co-duplicator has a non-zero index. The following two reduction sequences establish the confluence of this critical pair.

FIGURA COM PAR CRÍTICO...



On the other hand, when the index of the co-duplicator is zero, we need to invoke the well-formedness of the net:

Figura com par crítico e TWIN EQUIV.

The equivalence XXXXXXXXX

Twin equivalence!!!

5 Term nets

As mentioned in the introduction, reflection laws are handled by the translation to sum-product nets. More precisely, identities are restricted to atomic types XXXXXX

$$\mathcal{T} <- > \mathcal{E}$$

5.1 Term Translation

We now give a type-directed translation $\mathbf{T}(\cdot)$ from terms of $\mathcal{T}_{\mathrm{PF}}$ into sum-product nets. When a smaller net is used to construct some other net, we assume that the input and output in the initial net are removed. We also assume that a new pair of input/output ports and corresponding edges are introduced in the new net. The indexes of new nodes are initially empty.

Identity

- $\mathbf{T}(\mathsf{id}^{A\to A})$, where A is a base type, is defined as the sum-product net consisting of a single edge connecting the input to the output;
- $\mathbf{T}(\mathsf{id}^{A\times B\to A\times B})$ is the sum-product net I obtained by introducing 4 new nodes, $\wedge$, π_1, π_2, and $(,)$, and new edges connecting the first (resp. second) output of $\wedge$ 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 $I_A = \mathbf{T}(\mathsf{id}^{A\to A})$ and $I_B = \mathbf{T}(\mathsf{id}^{B\to B})$; and finally setting the input of I to be the input of $\wedge$ and the output of I to be the output of $(,)$.
- $\mathbf{T}(\mathsf{id}^{A+B\to A+B})$ is the sum-product net I obtained by introducing 4 new nodes, ?, i_1, i_2, and $\vee$, 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 $\vee$, where $I_A = \mathbf{T}(\mathsf{id}^{A\to A})$ and $I_B = \mathbf{T}(\mathsf{id}^{B\to B})$; and finally setting the input of I to be the input of ? and the output of I to be the output of $\vee$.

Composition

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

Constant Function

- $\mathbf{T}({\pi_1}^{A\times B\to A})$ 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 $I_A = \mathbf{T}(\mathsf{id}^{A\to A})$, 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.

- $\mathbf{T}({\pi_2}^{A \times B \to B})$ 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 $I_B = \mathbf{T}(\mathsf{id}^{B \to B})$, 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.
- $\mathbf{T}({\mathrm{i}_1}^{A \to A + B})$ 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 $I_A = \mathbf{T}(\mathsf{id}^{A \to A})$, 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.
- $\mathbf{T}({\mathrm{i}_1}^{B \to A + B})$ 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 $I_B = \mathbf{T}(\mathsf{id}^{B \to B})$, 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 $\wedge$ and $(,)$ nodes, and 4 new edges connecting the outputs of $\wedge$ to the inputs of T and U, and the outputs of T and U to the inputs of $(,)$, where $T = \mathbf{T}(t^{E \to A})$ and $U = \mathbf{T}(u^{E \to B})$; the input of $\wedge$ becomes the input of G and the output of $(,)$ becomes the output of G. Then:

- $\mathbf{T}(\langle t, u \rangle^{E \to A \times B})$, with $E = C + D$, is the sum-product net G' obtained by constructing the net $I = \mathbf{T}(\mathsf{id}^{C + D \to C + D})$, and an edge connecting its output to the input of G, setting the input of G' to be the input of I, and the output of G' to be the output of G.
- $\mathbf{T}(\langle t, u \rangle^{E \to A \times B})$, where E is not of the form $C + D$, is just G.

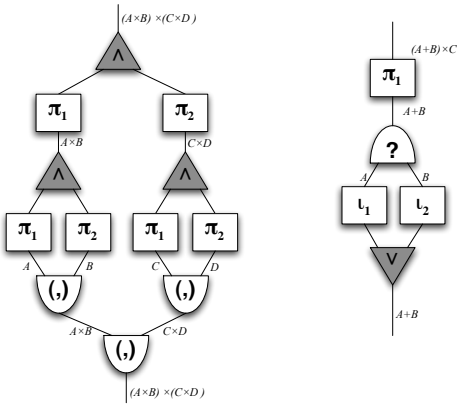
Either

Let G be the sum-product net obtained by introducing two new ? and $\vee$ 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 $\vee$, where $T = \mathbf{T}(t^{A \to E})$ and $U = \mathbf{T}(u^{B \to E})$; then the input of ? becomes the input of G and the output of $\vee$ becomes the output of G. We have: cancela

- $\mathbf{T}([t, u]^{A + B \to E})$, where $E = C \times D$, is the sum-product net G' obtained by constructing the net $I = \mathbf{T}(\mathsf{id}^{C \times D \to C \times D})$, and an edge connecting the output of G to the input of I; the input of G' is the input of G, and the output of G' is the output of I.
- $\mathbf{T}([t, u]^{A + B \to E})$, where E is not of the form $C \times D$, is just G.

Definition 5.1 The class of sum-product nets constructed by the translation $\mathbf{T}(\cdot)$ are designated *term nets.*

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



It is straightforward to see that $\mathbf{T}(t^{A\to B})$ is indeed a term net with input of type A and output of type B. A distinctive feature of the translation is that two differently-typed, syntactically equal terms may be translated as different term nets. The translation introduces in the nets sufficient structural information to allow for the typing information to be discarded. The principal type of the term represented by a net can always be uniquely determined.

5.2 Soundness

The reduction system thus induces the following definition of equivalence of sum-product nets. Let $\equiv$ denote structural equality of nets.

Definition 5.2 Two term nets G_1, G_2 are *equivalent*, written $G_1 = G_2$, if there exist G_1', G_2' such that $G_1 \longrightarrow^* G_1'$ and $G_2 \longrightarrow^* G_2'$, and $G_1' \equiv G_2'$.

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

Proposition 5.3 (Soundness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms. Then*

$$t = u \implies \mathbf{T}(t) = \mathbf{T}(u)$$

This can be proved by induction on the definition of equality in $\mathcal{T}_{\mathrm{PF}}$. Together, the translation $\mathbf{T}(\cdot)$ and the graph-rewriting system solve three problems:

- The translation directly captures the reflexivity laws, because it expands identities according to their types. For instance, the second example in Apendix C represents the term net $\mathbf{T}({\pi_1}^{(A+B)\times C\to A+B})$.
- To ensure that the fusion laws are effectively captured, commutations between configurations involving 3 nodes must be allowed, as exemplified in appendix D. These commutations (rules dupM-dupM, dupM-choice, dupS-dupS and pair-dupS) are regulated by a bit string indexing scheme.
- Finally, this indexing scheme must be capable of handling fusions in terms such as $\langle a, b\rangle .[c, d]$, which may happen in two directions. In our system, such a fusion results in a (unique) graph which is no longer a term net.

5.3 Completeness

The reader is invited to verify that reduction of a term net does not necessarily produce another term net. As such, in the following completeness result, the common reduct of $\mathbf{T}(t)$ and $\mathbf{T}(u)$ may no longer be a term net.

Proposition 5.4 (Completeness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms. Then*

$$\mathbf{T}(t) = \mathbf{T}(u) \implies t = u$$

This can be proved using a path semantics for graphs, that will be introduced in the long version of this paper.

6 Further Work

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.

We also intend to use this graph-rewriting system in the context of a visual language for functional programming.

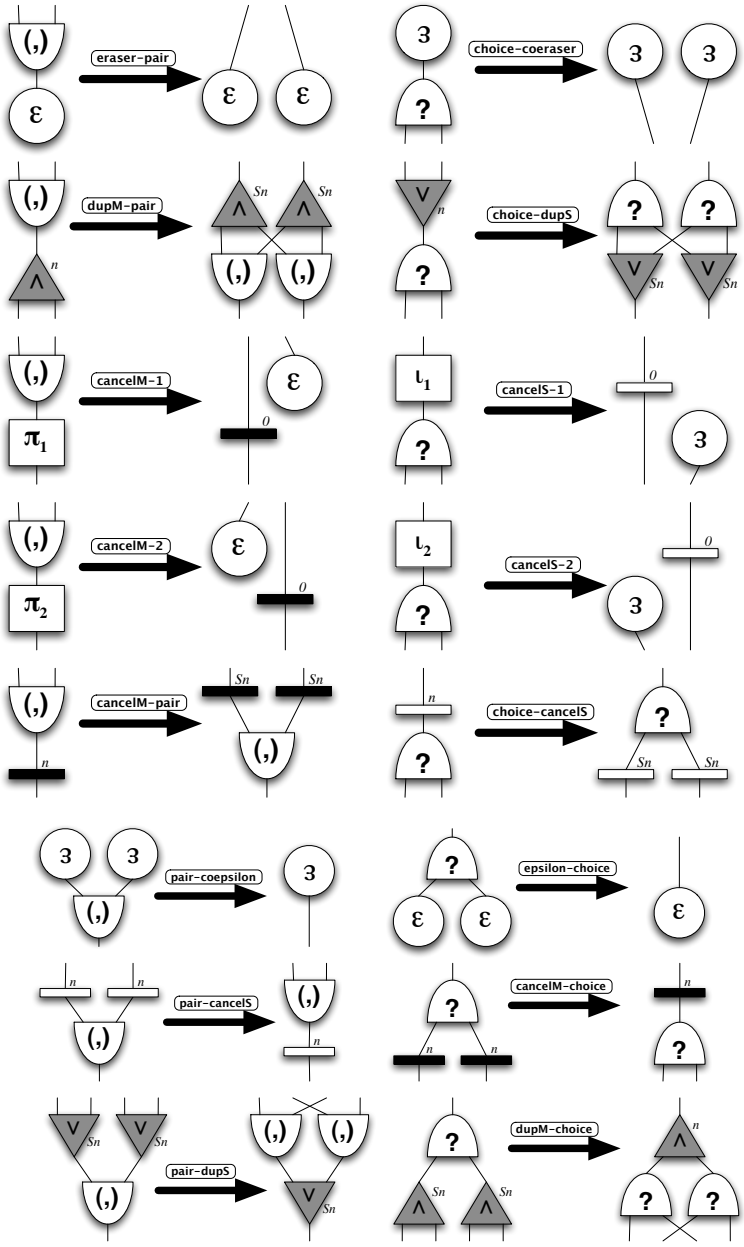
References

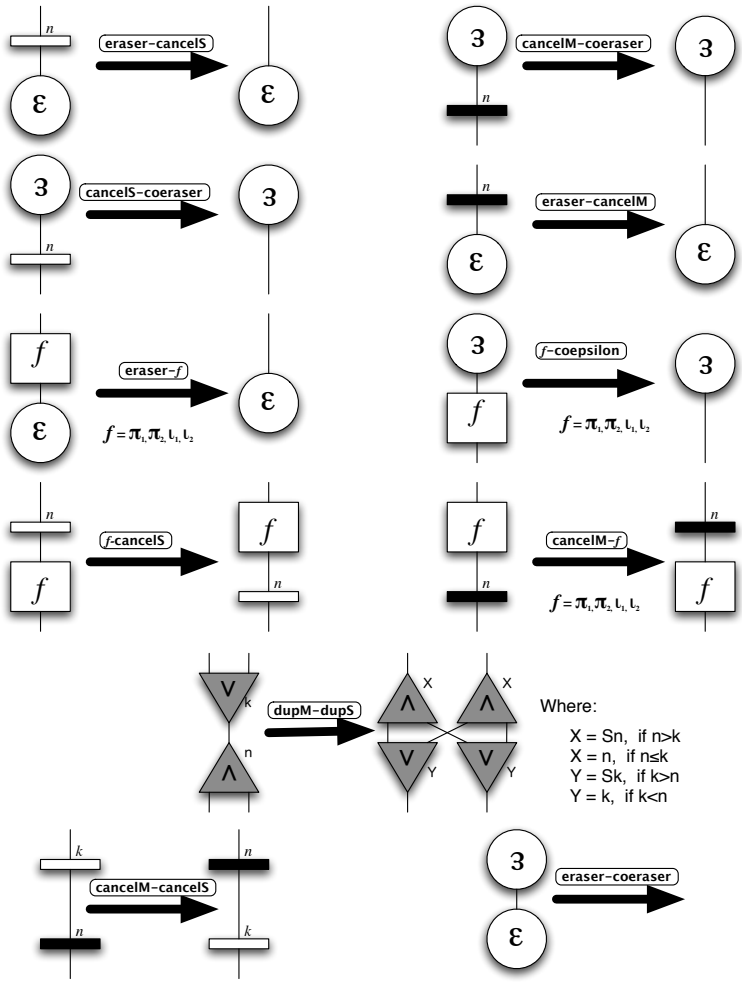
[1] Bird, R., de Moor, O.: *Algebra of Programming*, Prentice Hall, 1997.

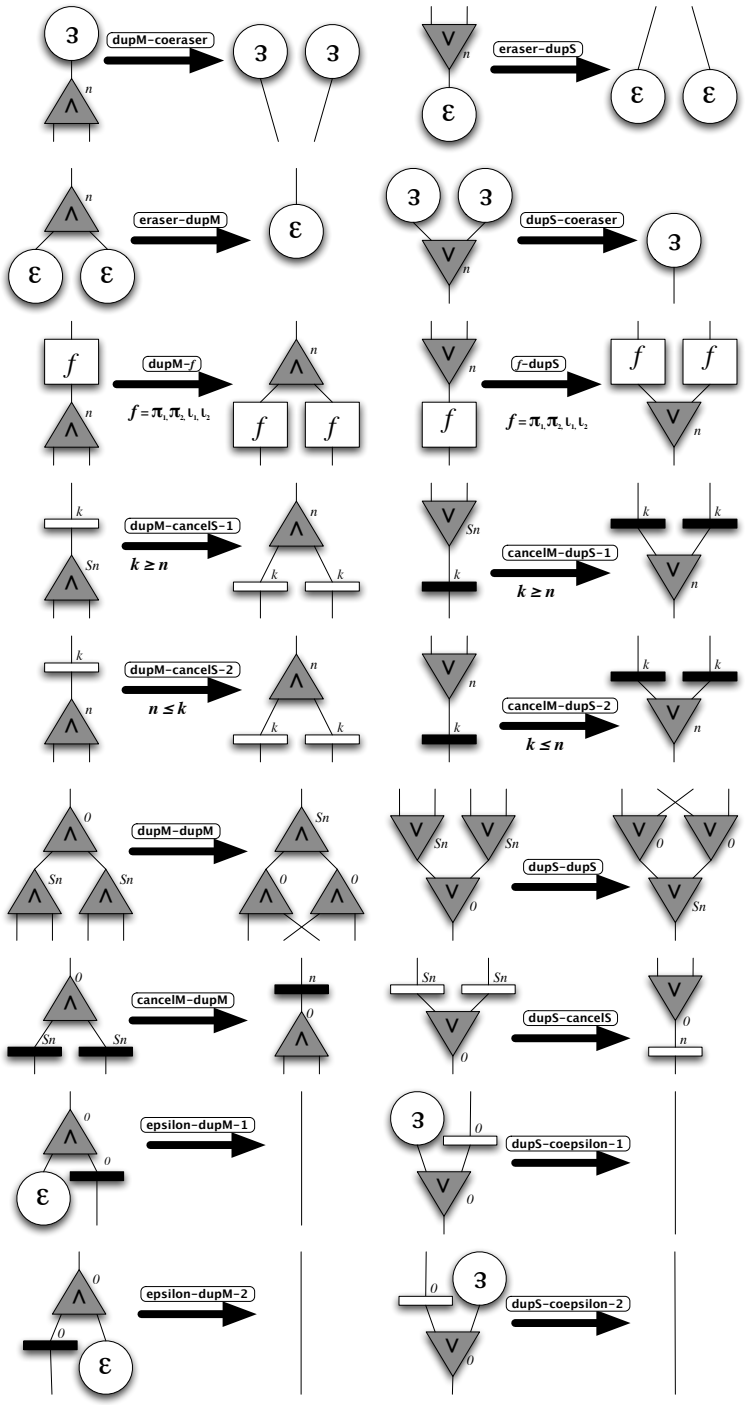
[2] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[3] 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 Full set of rewriting rules




eraser-cancelS
cancelS-coeraser
eraser-f
f = π1, π2, ι1, ι2
f-cancelS
cancelM-coeraser
eraser-cancelM
f-coepsilon
f = π1, π2, ι1, ι2
cancelM-f
f = π1, π2, ι1, ι2
dupM-dupS
Where:
X = Sn, if n>k
X = n, if n≤k
Y = Sk, if k>n
Y = k, if k<n
cancelM-cancelS
eraser-coeraser


dupM-coeraser
eraser-dupS
eraser-dupM
dupS-coeraser
dupM-f
f-dupS
dupM-cancelS-1
cancelM-dupS-1
dupM-cancelS-2
cancelM-dupS-2
dupM-dupM
dupS-dupS
cancelM-dupM
dupS-cancelS
epsilon-dupM-1
dupS-coepsilon-1
epsilon-dupM-2
dupS-coepsilon-2

B Critical Pairs

B.1 Among Common and Commutation

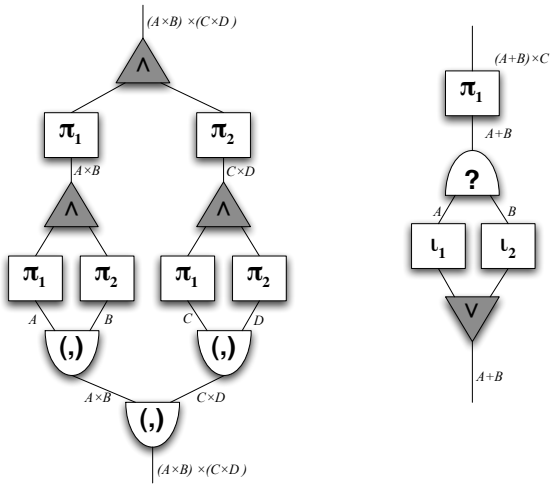
B.2 Among ISA and CR

B.3 Among ISB and CR

B.4 Among two rules of the commutation group

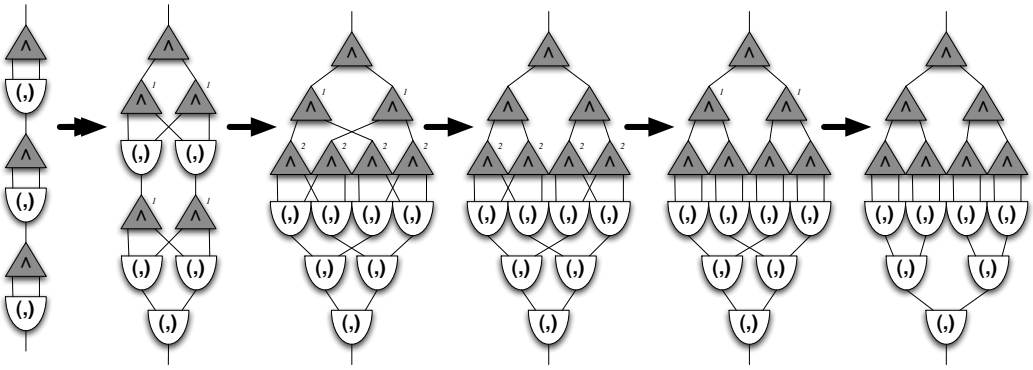
C Example Term nets

The following depict $\mathbf{T}(\mathsf{id}^{(A \times B) \times (C \times D) \to (A \times B) \times (C \times D)})$ and $\mathbf{T}(\pi_1{}^{(A+B) \times C \to A+B})$.

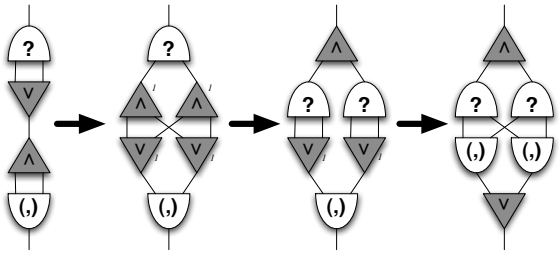


D Example Reductions

$\mathbf{T}(\langle \mathsf{id}, \mathsf{id} \rangle . \langle \mathsf{id}, \mathsf{id} \rangle . \langle \mathsf{id}, \mathsf{id} \rangle)$.



$\mathbf{T}(\langle \mathsf{id}, \mathsf{id} \rangle . [\mathsf{id}, \mathsf{id}])$.



E Proofs

Soundness

Lemma E.1 *Let t be a $\mathcal{T}_{\mathrm{PF}}$ term, and $G_\wedge(t)$ the net obtained by connecting a $\wedge$ node indexed with $n > 0$ to the output of the term net $\mathbf{T}(t)$. Then $G_\wedge(t) \longrightarrow^* G^\wedge(t)$, where $G_\wedge(t)$ consists of a $\wedge$ node indexed with n, whose outputs are connected to the inputs of two nets G_l, G_r such that $G_l = G_r = \mathbf{T}(t)$.*

Proof. By induction on the structure of t. Note that the lemma is not valid for $n = 0$, in the particular case that t is of the form (or a composition of terms ending with) $[f, g]$, in which case a $\vee$ node with index 1 will appear, which is not introduced by the translation in G_l and G_r. □

Lemma E.2 *Let t be a $\mathcal{T}_{\mathrm{PF}}$ term, and $G_\varepsilon(t)$ the net obtained by connecting an ε node to the output of the term net $\mathbf{T}(t)$. Then $G_\varepsilon(t) \longrightarrow^* G_\varepsilon$, where G_ε 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. □

Lemma E.3 *Let t be a $\mathcal{T}_{\mathrm{PF}}$ term, $G_\square(t)$ the net obtained by connecting the input of a $\square$ node indexed with n to the output of the term net $\mathbf{T}(t)$, and $G^\square(t)$ obtained connecting the output of a $\square$ node (again indexed with n) to the input of $\mathbf{T}(t)$. Then $G_\square(t) \longrightarrow^* G^\square(t)$.*

Proof. By induction on the structure of t, using rules where $\square$ appears in the left-hand side. □

Proposition E.4 (Soundness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms with $t = u$. Then $\mathbf{T}(t) = \mathbf{T}(u)$.*

Proof. By cases of the definition of equality of terms. We show here the multiplicative fragment; the remaining cases are dual (and in particular dual lemmas to E.1, E.2 and E.3 apply).

- $\mathbf{T}(\mathsf{id} \cdot f) \equiv \mathbf{T}(f \cdot \mathsf{id}) \equiv \mathbf{T}(f)$
  (straightforward)
- $\mathbf{T}((f \cdot g) \cdot h) \equiv \mathbf{T}(f \cdot (g \cdot h))$
  (straightforward)

- $\mathbf{T}(\langle \pi_1, \pi_2 \rangle) \equiv \mathbf{T}(\mathsf{id})$
  Guaranteed by construction. Observe that the term $\langle \pi_1, \pi_2 \rangle$ necessarily has type $A \times B \to A \times B$ for some types A, B, and
  $$\mathbf{T}(\langle \pi_1, \pi_2 \rangle^{A \times B \to A \times B}) \equiv \mathbf{T}(\mathsf{id}^{A \times B \to A \times B})$$
- $\mathbf{T}(\langle f, g \rangle \cdot h) = \mathbf{T}(\langle f \cdot h, g \cdot h \rangle)$

(i) $h = [a, b]$
  In the net $\mathbf{T}(\langle f, g \rangle \cdot [a, b])$ the rule dupM-dupS can be applied with $k = n = 0$. Lemma E.1 can then be used since the duplicators now have index 1; rule duoM-choice follows. The net thus obtained ... **CONTINUE**

(ii) The remaining cases are straightforward using Lemma E.1 (**mention inductive argument for composition???**).

- $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) = \mathbf{T}(f)$ and symmetrically $\mathbf{T}(\pi_2 \cdot \langle f, g \rangle) = \mathbf{T}(g)$
  There are two cases to consider:

(i) The domain of $\langle f, g \rangle$ is not a sum type. Then $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) \longrightarrow^* \mathbf{T}(f)$ by rule cancelM-1, Lemmas E.2 and E.3, and finally rule epsilon-dupM-2.

(ii) The domain of $\langle f, g \rangle$ is of the form $C + D$. This case is similar to the previous but note that the translation introduces an additional net on top, corresponding to the encoding af an identity of type $C + D$. We have $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) \longrightarrow^* \mathbf{T}(f \cdot \mathsf{id}) = \mathbf{T}(f)$.

□

Completeness

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

Definition E.5 We define a labelling of sum-product nets from top to bottom as follows, where the labels are $\mathcal{T}_{\mathrm{PF}}$ terms extended with the constants L, R, and $\lightning$.

- If the input of a $\wedge$ node is labelled α then both its outputs are labelled α.
- For $\vee$ nodes there are two cases:
  - If its inputs are labelled $\beta \cdot \alpha \cdot L \cdot \gamma$ and $\beta \cdot \alpha' \cdot R \cdot \gamma$ (where β is the longest common prefix and γ is necessarily a common suffix) then its output is labelled $\beta \cdot [\alpha, \alpha'] \cdot \gamma$. Note that if β extends until L and R then $\alpha = \alpha' = \mathsf{id}$.
  - If its inputs are labelled α and $\beta \cdot \lightning$ (in any order) then its output is labelled α.
- If the inputs of a $(,)$ node are labelled $\alpha \cdot \beta$ and $\alpha' \cdot \beta$ (where β is the longest common suffix) then its output is labelled $\langle \alpha, \alpha' \rangle \cdot \beta$. Note that if the labels are equal then $\alpha = \alpha' = \mathsf{id}$.
- If the input of a pair projection node π_1 (resp. π_2) is labelled α then its output is labelled $\pi_1 \cdot \alpha$ (resp. $\pi_2 \cdot \alpha$).
- If the input of a choice injection node i_1 (resp. i_2) is labelled α then its

output is labelled $\mathrm{i}_1 \cdot \alpha$ (resp. $\mathrm{i}_2 \cdot \alpha$).

- The output of a $\mathfrak{z}$ node is labelled $\mathfrak{z}$.
- If the input of a $\square$ node is labelled α then its output is also labelled α.
- If the input of a $\blacksquare$ node is labelled α then its output is also labelled α.

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

For 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), $\mathbf{R}_{\boldsymbol{x}}(G) = l_1, \ldots, l_m$ where $\boldsymbol{x} = x_1, \ldots, x_n$.

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

$$L \cdot \mathrm{i}_1 = \mathsf{id} \qquad L \cdot \mathrm{i}_2 = \mathfrak{z}$$

$$R \cdot \mathrm{i}_1 = \mathfrak{z} \qquad R \cdot \mathrm{i}_2 = \mathsf{id}$$

$$l \cdot \mathfrak{z} = \mathfrak{z}$$

Lemma E.6 *Let G_1, G_2 be sum-product nets; if $G_1 \longrightarrow G_2$ then $\mathbf{R}_{\boldsymbol{x}}(G_1) = \mathbf{R}_{\boldsymbol{x}}(G_2)$.*

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 $\beta \cdot \alpha_1 \cdot L \cdot \gamma$ and $\beta \cdot \alpha_2 \cdot R \cdot \gamma$, or else α and $\beta \cdot \mathfrak{z}$; in the first case, in both sides of the rule the outputs will be labelled $L \cdot \beta \cdot [\alpha_1, \alpha_2] \cdot \gamma$ and $R \cdot \beta \cdot [\alpha_1, \alpha_2] \cdot \gamma$; in the second case the outputs are labelled $L \cdot \alpha$ and $R \cdot \alpha$ in both sides.

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

Lemma E.7 *For any $t \in \mathcal{T}_{\mathrm{PF}}$, $\mathbf{R}(\mathbf{T}(t)) = t$.*

Proof. The stronger result $\mathbf{R}_x(\mathbf{T}(t)) = t \cdot x$ can be proved by induction on the structure of t. $\square$

Proposition E.8 (Completeness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms such that $\mathbf{T}(t) = \mathbf{T}(u)$. Then $t = u$.*

Proof. For $\mathbf{T}(t) = \mathbf{T}(u)$ to hold there must exist sum-product nets G_t, G_u such that $\mathbf{T}(t) \longrightarrow^* G_t$, $\mathbf{T}(u) \longrightarrow^* G_u$, and $G_t \equiv G_u$. By lemma E.6 we have that $\mathbf{R}(\mathbf{T}(t)) = \mathbf{R}(G_t)$ and $\mathbf{R}(\mathbf{T}(u)) = \mathbf{R}(G_u)$. Now by lemma E.7 and because structurally equal nets have the same read-back, we have $t = u$. $\square$