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

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

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

Abstract

In this paper we outline how a graph-based decision procedure can be given for the functional calculus with sums and products. 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.

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

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

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:

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

where A 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

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

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

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

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.

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 [4]).

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.

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 $\reflectbox{$\varepsilon$}$.
- a *cancel* symbol with arity and co-arity 1, depicted $\square$; its dual *co-cancel* is depicted $\blacksquare$.

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, $\wedge$, $\vee$, $\blacksquare$ and $\square$ nodes in a net will be labelled indexes (pair of integers for $\wedge$ and $\vee$ nodes, integers for *cancel* and $\square$ nodes). These will be used to control the duplication and mutual annihilation of nodes in the reduction system presented in section 6.

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 1 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 1 hold for every node ($S\ n$ denotes the successor of n). Well-formedness imposes a structural invariant on nets.

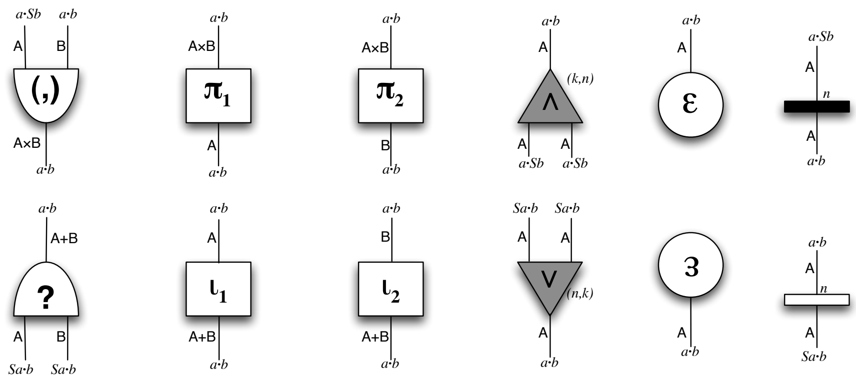


Fig. 1. Typing and Positioning Constraints

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.

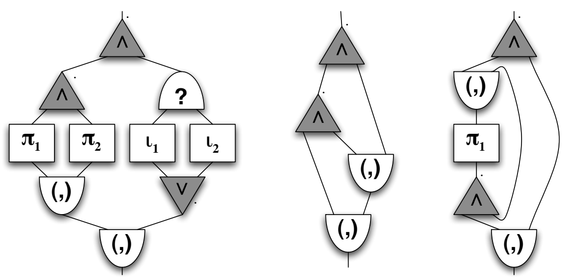


Fig. 2. Examples of Nets

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

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 set to zero.

The translation expands identities as explained before, so that only identities of atomic types are represented as edges. Moreover, we remark that the translation of $\langle , \rangle$ and $[,]$ may also expand identities to allow for the correct treatment of terms to which the *exchange law* may be applied.

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.

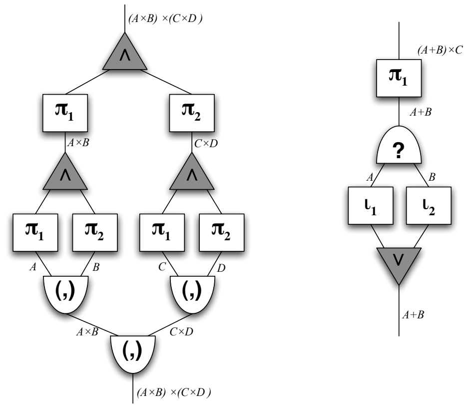
Either

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 4.1 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 Deciding Equality by Local Graph Rewriting

Fusion

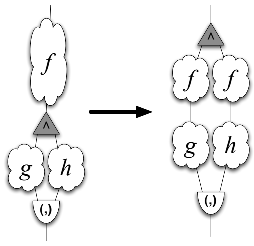


Fig. 3. Fusion as Net Duplication

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 3). However, this "duplication" should take into account that we intend it to 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.

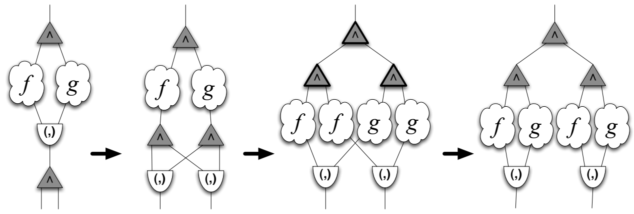
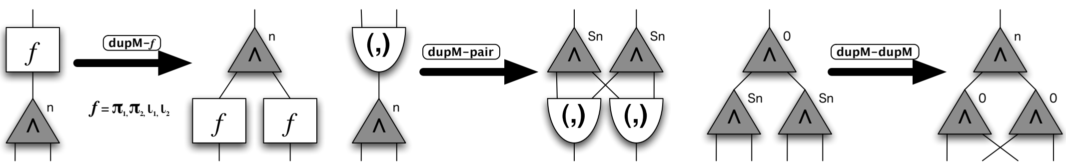


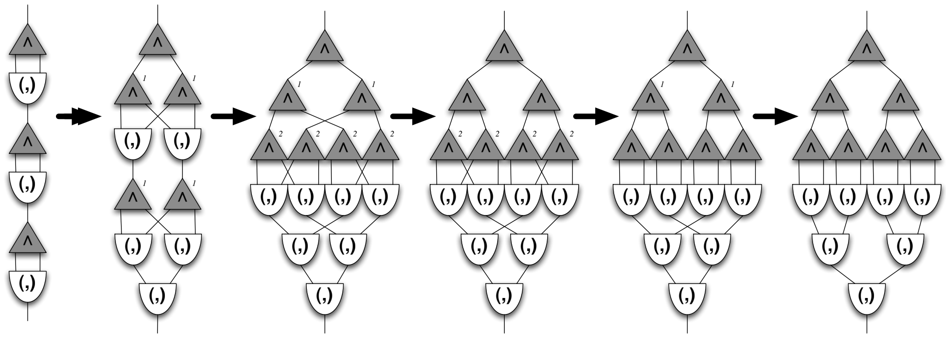
Fig. 4. Duplication of a Structured Net

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. 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 4). The need to control this reorganization of duplicators justifies the presence of *indexes* in the nodes. We are led to the following rules governing the interaction with duplicators.



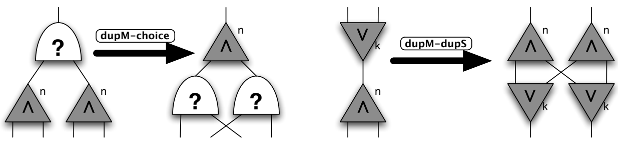
The first rule is fairly obvious — the interaction with single input/single output nodes simply duplicates them. When a duplicator interacts with a pair constructor node, not only does it duplicate the node, but it also splits itself in two in order to duplicate each subnet. The last rule is the commutation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

For now, we let the indexes attached to duplicators to be integers (later, we will elaborate these to be pair of integers). They record "how deep" the corresponding duplicator is in the transversal of a structured net. When a duplicator has its index set to zero, we call it a *ground duplicator* and often omit its index. Notice that the commutation rule actually restricts the top duplicator to be ground. We show an example of the duplication process:

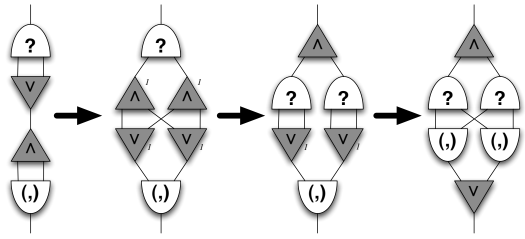


We now turn to the interaction between sums and products. Structurally, it is fairly obvious that when a duplicator and a co-duplicator meet, they should pass through each other. The question is whether the nodes get duplicated or split during this process (i.e. what the impact on their indexes is).

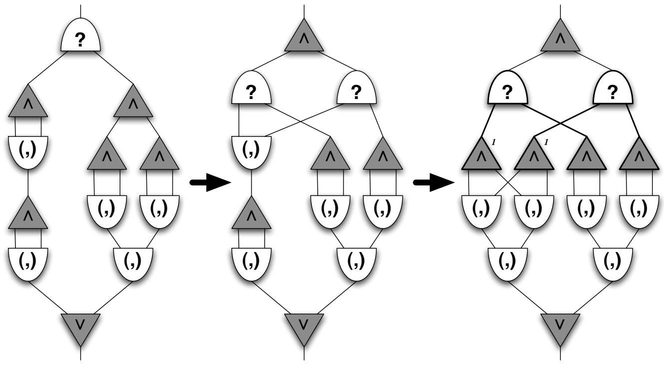
A first solution would be to state that only duplications take place. This corresponds to keeping both indexes unaltered, and leads to a rule that allows duplicators to freely pass through choice nodes (i.e. without index regulation). The following additional interaction rules (and their duals) are required:



We show an example of this. Note that, as expected from the discussion in Section 2, the above normal form is not a direct translation of any term (split or either).

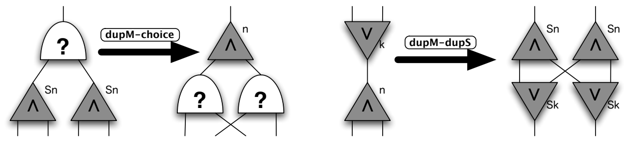


Unfortunately, with these rules the indexes no longer constrain appropriately the commutations that might take place in a reduction sequence. To see this, consider the following reduction sequence:

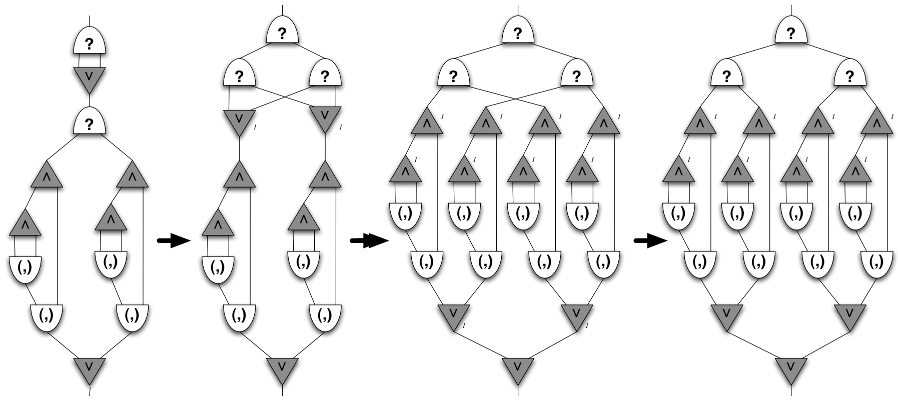


Note that the application of rule `dupM-choice` removes the ground duplicator needed to close the fusion started on the left hand-side sub-net. In fact, this is exactly the pattern of divergence that rule `dupM-dupM` avoids by restricting the top duplicator to be ground. This shows that indexes should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

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



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



Notice that the traversal 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 informal 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:

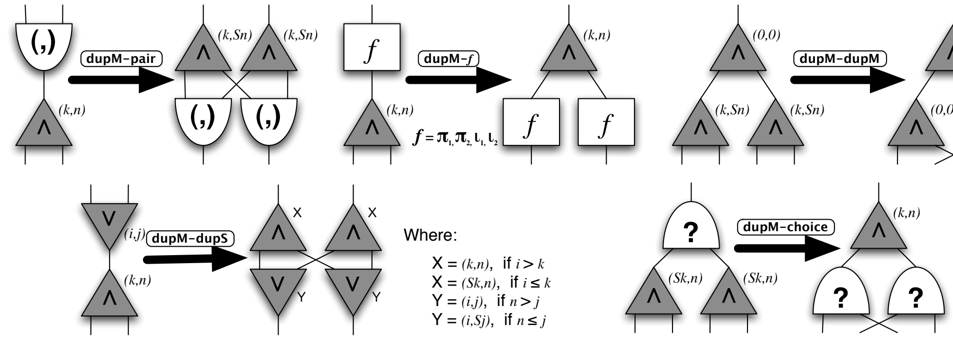
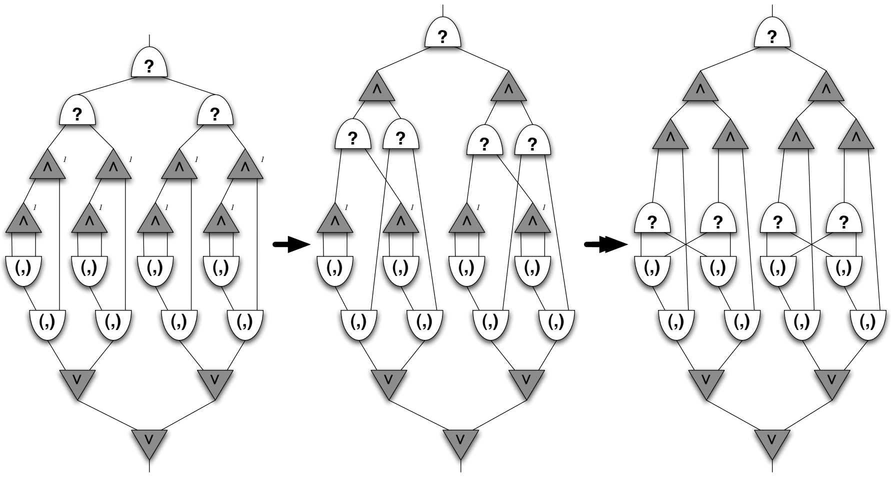


Fig. 5. Fusion Rules

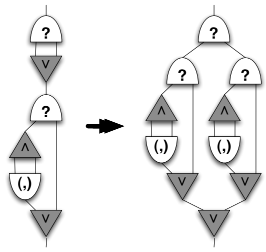


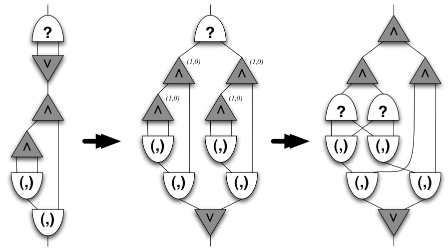
This example suggests that, in order to reach full normal forms, it is sufficient to promote enough fusions (something that might be accomplished by pre or post-composition with identities). The problem is that we do the additional reductions relying strongly on the symmetry of the net (more precisely, on the number and the tree shape of duplicators in both sides of the choice node). Even though term nets do exhibit a high degree of symmetry, we are not able to assure that all nets, under all reductions strategies, do possess the symmetry required by this set of rules

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 splits itself (its index is increased), and the other is simply duplicated (its index remains unaltered).

The problem become how to define the domination relation. In fact, the most obvious solutions fail to define a confluent system. Our solution is given by the following characterization (for the sake of clarity, we present it in terms of the (co-)duplicator initial positions): "a $\vee$ dominates a $\wedge$ when there is no ? node between them". In order to express this with indexes, we need to distinguish an additive and a multiplicative component. Figure 5 sum up our discussion (we omit those rules that can be obtained by duality).

The following two examples show the distinct treatment given for duplicators when traversed by the top co-duplicator.

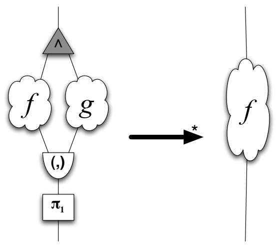




Note that we only allow $\wedge$s to commute with the ? node that is associated with the co-duplicator that is performing the fusion (even when this ? only become available after commutation with other $\wedge$s). Commutations with ? nodes duplicated during the fusion process are inhibited.

Cancelation

Cancelation is by nature related with erasing. Consider the structure of a cancelation rule:



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 special erasing nodes ε and ɜ that annihilate any other node that interacts with them. The main difficulty is the removal of the top node, that also acts as delimiter for the erasing process. For that, we use $\blacksquare$ and $\square$ nodes, whose function is to traverse the net f and remove the top duplicator as soon as the erasure of the net g takes place. Once again, it must be ensured that this process does not interfere with other cancelations and fusions that may be taking place.

In a sense, $\blacksquare$ nodes behave like duplicators as they move upwards in the net. Like duplicators, an index is attached to $\blacksquare$ nodes. However, these are simplified indexes as they are not affected by the additive constructs – this simplification is possible due to the simpler commutation rule for $\blacksquare$ nodes.

During the traversal, $\blacksquare$ nodes perform a correction on the indexes of co-

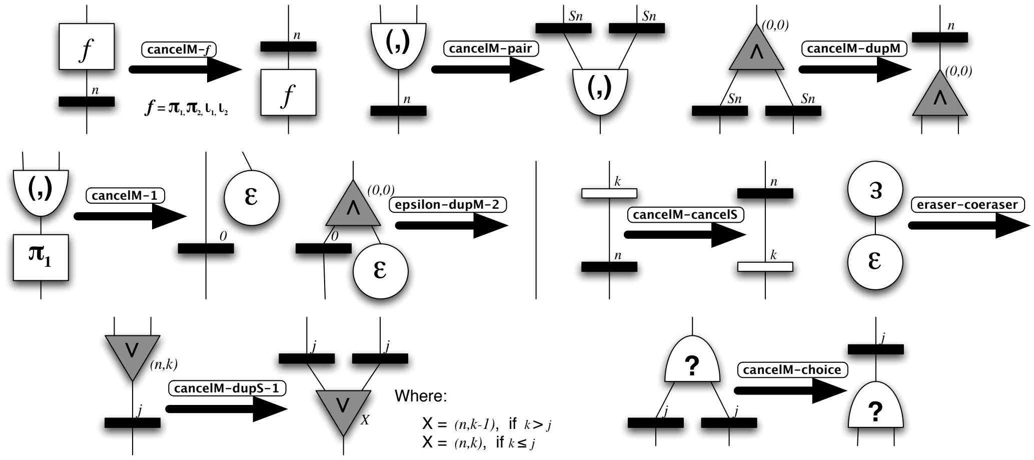


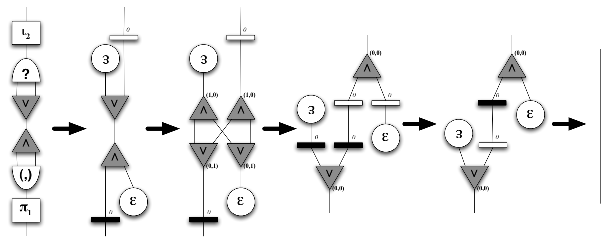
Fig. 6. Cancelation Rules

duplicators. This is because co-duplicators might have crossed the top duplicator and updated their indexes.

A representative subset of the rules for cancelation is given in figure 6. We omit the rules that can be obtained by duality and the garbage-collection rules for ε and ϵ. The full set of rules is given in Appendix A.

In these rules, cancelation is triggered by the interaction of the pair-constructor and a projection node (rule `cancelM-1`). After that, the ε node will discard the portion of the net that corresponds to the canceled sub-term; the ■ node will traverse the preserved sub-term and synchronize with the ε node on the top duplicator.

The reduction shown below illustrate the interaction between the additive and multiplicative fragment and with cancelations.



To conclude this informal presentation, we present in Figure 7 an example of an asymmetrical net. It constitutes a counterexample for the second approach outlined above to the interaction of multiplicative and additive fusion. The reader is invited to perform its fusion with $[id, id]$ to verify the problems of such an approach.

6 Sum-product Net Rewriting

A local graph-rewriting system will now be given for sum-product nets, based on the ideas discussed infromally in the previous section. We first need to establish an appropriate notion of graph-rewriting rule: both the left-hand

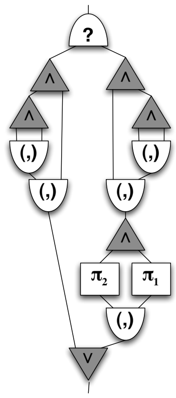


Fig. 7. Example of an asymmetrical net

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 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 [3]; 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.

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

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

Duality allow a considerable economy of effort during the study of the rewriting system: in practice, one need to check (roughly) half of node types and rules. In what follows, we will always implicitly invoke it.

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

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

Confluence

It is not difficult to realize that the rewriting system presented above is not confluent for general sum-product nets. However, we are interested in a particular class of nets, namely admissible nets, and for these we will be able to exploit their regularity properties to show that the reduction paths actually converge.

As usual, the confluence of the system will be established by resolution of the critical pairs induced by the rules. Most of these pairs are resolved in a purely local fashion, applying the rules of the system. For some, however, that will be not enough since the convergence of both reduction paths do rely on the global properties of admissible nets. In order to regain the local flavor in the analysis of critical-pairs, we will introduce a notion of equivalence that will capture this dependency.

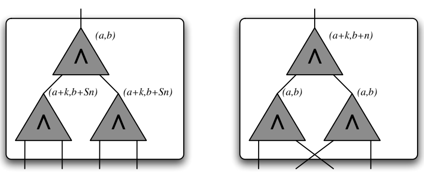
Definition 6.3 Let $\mathcal{N}$ be single input, n-output net and A a two-input, single-output node. We define $\mathsf{JOIN}(A, \mathcal{N})$ as a net composed of two copies of $\mathcal{N}$ and n copies of A (say A_i) where the inputs of the net are the inputs of each copy of $\mathcal{N}$, the ith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

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

Let us illustrate a concrete example of a twin equivalence.

Lemma 6.5 *The following nets are twin-equivalents with respect to* $\vee(i, j)$.

Proof. Let us denote by $N_1^{(a,b)}$ and $N_2^{(a,b)}$ the following nets:



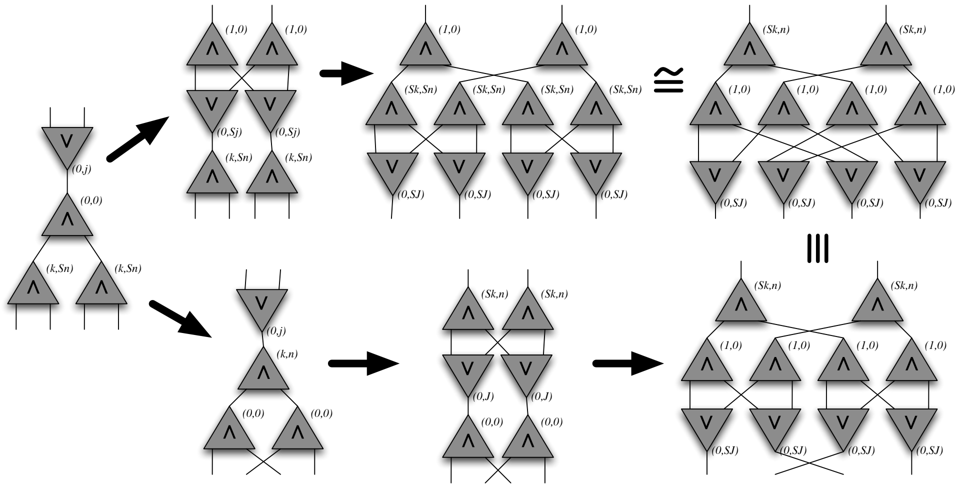
Note that $N_1^{(0,0)}$ reduces to $N_2^{(0,0)}$. The idea is that we can treat each of these parametrized nets as (indexed) nodes — for that, we compute the derived rules of interaction between these nets and each kind of node defined in sum-product nets. Doing that we realize that, for every node type, they are exactly the same for $N_1^{(a,b)}$ and $N_2^{(a,b)}$. On the other side, we know by Lemma 6.2 that the top duplicator in $N_1^{(a,b)}$ do have a reduction sequence that reset its index. Consider that reduction sequence performed by $N_1^{(a,b)}$ as a block and finally applying rule `dupM-dupM`. The above discussion tell us that the impact on $\mathcal{N}$ is precisely the same if performed by $N_2^{(a,b)}$. □

This example actually shows what will be the primary purpose of twin-equivalences in the proof of confluence: they allow to overcome the restrictions on the application of the commutation rules. Curiously, these restriction were imposed precisely to achieve confluence, as they avoid critical pairs between commutations.

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

Proof. We consider the critical pairs induced by the rules given in appendix A. Let us illustrate with the most interesting one: the critical pair formed by rules `dupM-dupM` and `dupM-dupS`.

We need to consider several cases depending on the indexes of the agents. Let us first assume that the additive index of the $\vee$ is zero. We are lead to the following pair of reduction sequences:



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

The symbol $\equiv$ denotes structural equality in nets and $\cong$ is an application of the twin-equivalence presented in Lemma 6.5.

When $\vee$ has a strictly positive additive index, the top-duplicator does not alter its index, and the invocation of the twin-equivalence can be replaced by the execution of rule `dupM-dupM`. □

7 Soundness and Completeness

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

Definition 7.1 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 7.2 (Soundness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms. Then*

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

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

Proposition 7.3 (Completeness) *Let t, u be $\mathcal{T}_{\mathrm{PF}}$ terms. Then*

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

Proofs of both results can be found in Appendix B.

8 Conclusions and Future Work

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.
- 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 $\langle a, b\rangle . [c, d]$, which may happen in two directions. In our system, such a fusion results in a (unique) graph which is no longer a term net.

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

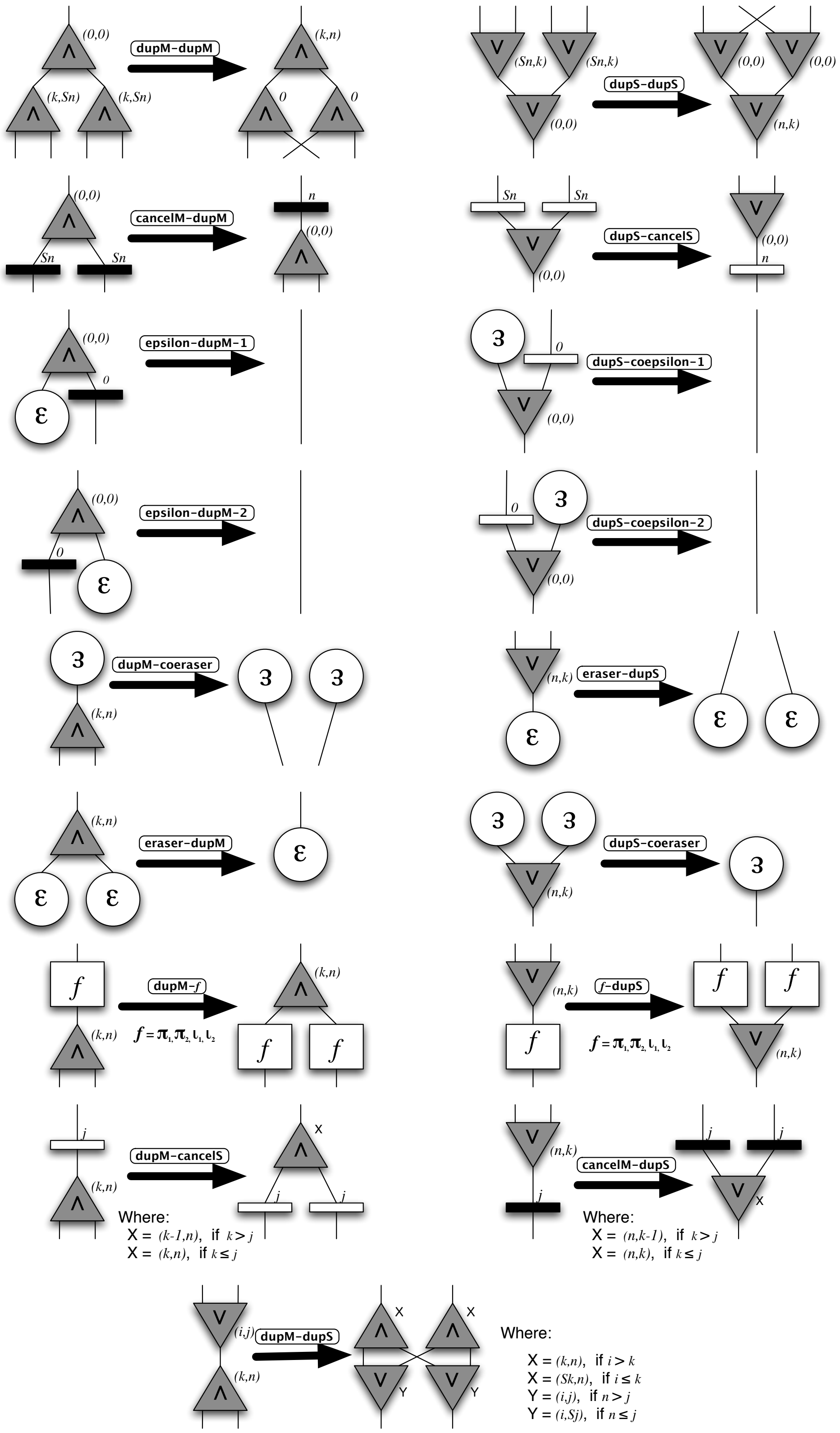
[1] Bird, R., de Moor, O.: *Algebra of Programming*, Prentice Hall, 1997.

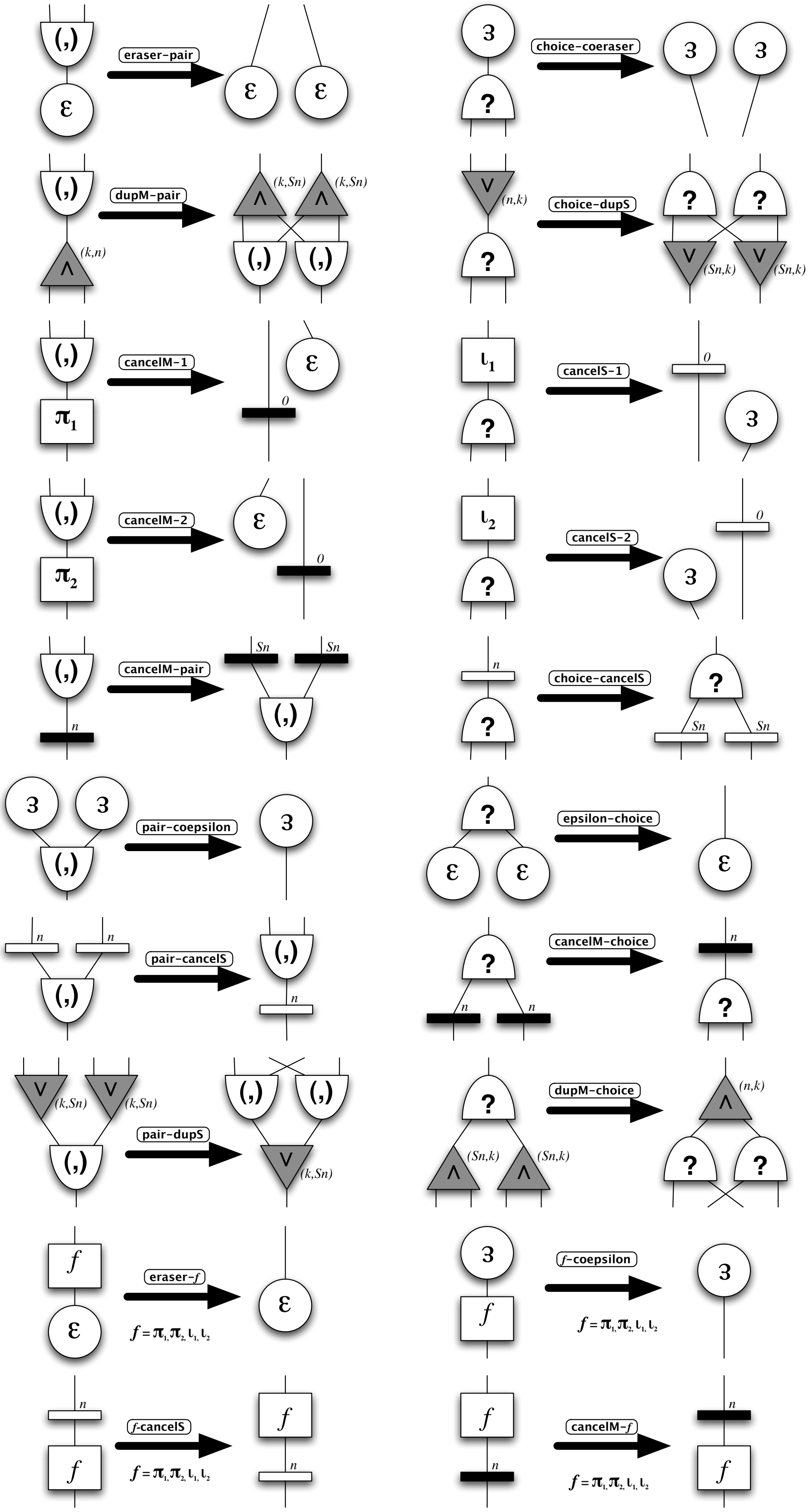
[2] J.-Y. Girard: Towards a Geometry of Interaction, In *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*, pages 69–108. 1989.

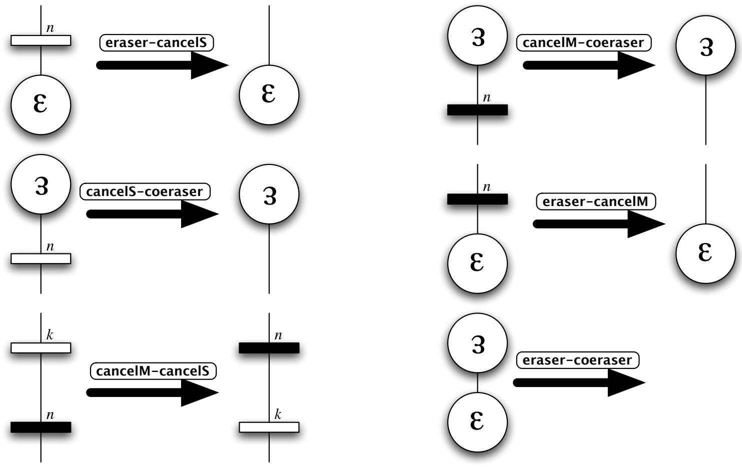
[3] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[4] 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 Rewrite Rules




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



B Proofs

Admissible Nets

Rules `dupM-dupM` and `dupM-choice` are called *commutation rules* as they promote a swap of nodes. For convenience, let us say that a net admits "half a commutation" if that net contains a subnet matching the top and one of the bottom nodes of the LHS of a commutation rule (a partial match).

Lemma B.1 *Let $\mathcal{N}$ be an admissible net that admits half a commutation. Then, there exists a net $\mathcal{N}'$ such that $\mathcal{N} \Rightarrow^{\star} \mathcal{N}'$ and $\mathcal{N}'$ admits to extend the partial match to the full match of the commutation rule LHS.*

Proof. In an admissible net, and for every ? and $\wedge$ node with index $(0, 0)$, we are able to identify what is the agent "scheduled" for commutation with it (if any). This can be computed 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 guaranty that this node does not change until commutation actually occurs. Moreover, the fact that rules `dupM-pair` and `dupM-dupS` inject duplicators in both inputs of the interacting agents guaranty that, in admissible nets, the result is the same when computed on any of the output ports of the given node. □

Corollary B.2 (Lemma 6.2) *In admissible nets, indexes can be reset by reduction.*

Proof. Indexes are adjusted during reduction in accordance with the well-formedness criteria. On the other hand, rules for indexed nodes are such that do not constrain interaction when the indexes are non-zero. This mean that an indexed node can only survive in a normal form if: (i) it reaches the top of the net; (ii) it get stuck in a commutation that is not possible. The well-formedness criteria guaranties that (i) is not possible for admissible nets. Lemma B.1 shows that (ii) could never occur in a normal form. □

Soundness

Lemma B.3 *Let t be a $\mathcal{T}_{\mathrm{PF}}$ term, and $G_\wedge(t)$ the net obtained by connecting a $\wedge$ node indexed with a, m where $m > 0$, 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 a, m, 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 $m = 0$, 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 $\vee$. $\square$

Lemma B.4 *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. $\square$

Lemma B.5 *Let t be a $\mathcal{T}_{\mathrm{PF}}$ term, $G_\blacksquare(t)$ the net obtained by connecting the input of a $\blacksquare$ node indexed with n to the output of the term net $\mathbf{T}(t)$, and $G^\blacksquare(t)$ obtained connecting the output of a $\blacksquare$ node (again indexed with n) to the input of $\mathbf{T}(t)$. Then $G_\blacksquare(t) \longrightarrow^* G^\blacksquare(t)$.*

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

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

Lemma B.6 *Let N be a sum-product net with a single input and output, and let $N^\vee$ denote the net obtained by incrementing the second component of the indexes of top coduplicators in N (i.e. a, m becomes $a, m+1$ for every coduplicator that is not located under another coduplicator).*

Let also N' be the net obtained by plugging a net $\mathbf{T}(\mathsf{id})$ (where the identitity has the appropriate type) on top of N. Then $N = N'$.

Proof. Straight forward induction on the type of the identity. $\square$

Proposition B.7 (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.

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

$$\mathbf{T}(\langle \pi_1, \pi_2 \rangle^{A \times B \to A \times B}) \equiv \mathbf{T}(\mathsf{id}^{A \times B \to A \times B})$$

- $\mathbf{T}([\mathrm{i}_1, \mathrm{i}_2]) \equiv \mathbf{T}(\mathsf{id})$
  Dual to the previous case.
- $\mathbf{T}(\langle f, g \rangle \cdot h) = \mathbf{T}(\langle f \cdot h, g \cdot h \rangle)$
  We reason by inductively on the structure of h
  (i) $h = h_1 \cdot h_2$ – we use $\mathbf{T}((f \cdot g) \cdot h) \equiv \mathbf{T}(f \cdot (g \cdot h))$ and then the inductive hypothesis applies.
  (ii) h is a constant – straightforward.
  (iii) $h = \langle a, b \rangle$ – straightforward using Lemma B.3, 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 B.3 does not apply.
  In the net $\mathbf{T}(\langle f, g \rangle \cdot [a, b])$ after one step of rule $\mathsf{dupM\text{-}dupS}$ the split duplicators have indexes 1,0; they will duplicate the nets $\mathbf{T}(a)$ and $\mathbf{T}(b)$ incrementing the (second component of the) indexes of the top co-duplicators. Finally, the split duplicator will commute with the top ? node.
  In the net $\mathbf{T}(\langle f \cdot [a, b], g \cdot [a, b] \rangle)$ 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 $\mathbf{T}(a)$ and $\mathbf{T}(a)$ appear intact, with identities of sum type on top of each such net. Lemma B.6 then yields $\mathbf{T}(\langle f, g \rangle \cdot [a, b]) = \mathbf{T}(\langle f \cdot [a, b], g \cdot [a, b] \rangle)$.
- $\mathbf{T}(h \cdot [f, g]) = \mathbf{T}([h \cdot f, h \cdot g])$
  Dual to the previous case using Lemmas B.3' and B.6'.
- $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) = \mathbf{T}(f)$ and symmetrically $\mathbf{T}(\pi_2 \cdot \langle f, g \rangle) = \mathbf{T}(g)$
  If the domain of $\langle f, g \rangle$ is not a sum type and the codomain of π_1 is a ground type, then $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) \longrightarrow^* \mathbf{T}(f)$ by rule $\mathsf{cancelM\text{-}1}$, Lemmas B.4 and B.5, and finally rule $\mathsf{epsilon\text{-}dupM\text{-}2}$.
  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 af an identity of type $C + D$. We have $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) \longrightarrow^* \mathbf{T}(f \cdot \mathsf{id}) = \mathbf{T}(f)$.
  (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 $\mathbf{T}(\pi_1 \cdot \langle f, g \rangle) \longrightarrow^* \mathbf{T}(\mathsf{id} \cdot f) = \mathbf{T}(f)$.
- $\mathbf{T}([f, g] \cdot \mathrm{i}_1) = \mathbf{T}(f)$ and symmetrically $\mathbf{T}([f, g] \cdot \mathrm{i}_1) = \mathbf{T}(g)$
  Dual to the previous case using Lemmas B.4' and B.5'.

□

Completeness

The proof of completeness uses a path-based interpretation of terms, in the style of the Geometry of Interaction [2].

Definition B.8 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 ε.

- 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 $\beta \cdot \varepsilon$ (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 ε node is labelled ε.
- If the input of a $\blacksquare$ node is labelled α then its output is also labelled α.
- If the input of a $\square$ 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):

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

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

$$l \cdot \varepsilon = \varepsilon$$

Lemma B.9 *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 $\beta \cdot \varepsilon$; 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. □

Lemma B.10 *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. □

Proposition B.11 (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 B.9 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 B.10 and because structurally equal nets have the same read-back, we have $t = u$. □