Available online at www.sciencedirect.com
ScienceDirect

Theoretical Computer Science 364 (2006) 166 – 195

Theoretical Computer Science

www.elsevier.com/locate/tcs

Differential interaction nets[☆]

T. Ehrhard*, L. Regnier

Institut de Mathématiques de Luminy, Campus de Luminy, Case 907, 13288 Marseille Cedex 9, France

Abstract

We introduce interaction nets for a fragment of the differential lambda-calculus and exhibit in this framework a new symmetry between the *of course* and the *why not* modalities of linear logic, which is completely similar to the symmetry between the *tensor* and *par* connectives of linear logic. We use algebraic intuitions for introducing these nets and their reduction rules, and then we develop two correctness criteria (weak typability and acyclicity) and show that they guarantee strong normalization. Finally, we outline the correspondence between this interaction nets formalism and the resource lambda-calculus.
© 2006 Published by Elsevier B.V.

Keywords: Lambda-calculus; Linear logic; Differential lambda-calculus; Interaction nets; Resource lambda-calculus; Lambda-calculus with multiplicities

1. Introduction

In previous semantic investigations [7,8], the first author observed that it can perfectly make sense to extend linear logic and the lambda-calculus with differential constructions. From these observations, he derived with the second author in [9] a *differential lambda-calculus*, which is a lambda-calculus equipped with a notion of formal differentiation (and new term constructions corresponding to differentials). In the same paper, it is moreover advocated that derivation is a very natural operation from an operational viewpoint: it corresponds to the possibility of feeding a term (of type $A \to B$ for instance) with *exactly one copy* of its argument. In other words, taking the derivative of a term means extracting the *linear part* of the term, where "linear" takes here its logical (or computer science) meaning of "using ones argument exactly once". One of the main ideas developed in [9] is precisely that this logical notion of linearity coincides with the mathematical meaning (remember that the derivative, in mathematics, is the best linear approximation of a function). This idea is further reinforced by a result relating the Taylor formula to the linear head reduction strategy of the lambda-calculus (the reduction implemented by Krivine's machine for instance).

☆ Supported by the ACI project GEOCAL.
* Corresponding author.
E-mail address: ehrhard@iml.univ-mrs.fr (T. Ehrhard).

0304-3975/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.tcs.2006.08.003

1.1. Exponential and differentiation

In the present paper, our goal is to give a linear logic account of this extension of functional languages with differential constructs. More precisely, motivated by the structures of the models mentioned above, we want to express the differential operations developed in the differential lambda-calculus as new operations on the *exponential* "!" of linear logic. Remember that, in linear logic, the structural operations of weakening and contraction obtain a logical status thanks to the introduction of the two exponential modalities "?" and "!", which are De Morgan dual of each other (through linear negation, which is involutive). So exponentials are responsible for the possibility of erasing and copying data, which is of course essential during computations. In linear logic without exponentials, a proof uses linearly its hypotheses.

In Köthe spaces [7] as well as finiteness spaces [8], formulae are interpreted as topological vector spaces, and proofs of linear logic as linear continuous maps between these spaces. Then exponentials appear as "symmetric tensor algebra" constructions [2]. In linear logic, an intuitionist implication $A \Rightarrow B$ is translated as $!A \multimap B$ (where this $\multimap$ symbol corresponds to linear implication). In the models considered here, linear maps from $!X$ to Y can be seen as "analytic functions" (that is, functions definable by a power series) from the vector space X to the vector space Y. For each space X, the space $!X$ is equipped with the following structure, which is standard from the viewpoint of linear logic:

- There is a map $!X \to X$ (dereliction) and a map $!X \to !!X$ (digging) which are natural and turn "!" (which must be a functor) into a comonad. The basic use of dereliction (which also explains its name) is that, by composition on the left, it allows to turn a linear function from X to Y into an analytic function from X to Y (it is a forgetful operation: linear functions *are* particular analytic functions).
- There is a map $!X \to 1$ (where 1 is the unit of the tensor product) and a map $!X \to !X \otimes !X$ which correspond, respectively, to weakening and contraction. These maps turn $!X$ into a $\otimes$-comonoid (co-algebra). Contraction allows to turn a two-parameter analytic function f into a one-parameter g defined by $g(x) = f(x, x)$. Similarly, given a vector y of Y, seen as a linear function from 1 to Y (1 is the field and this map is just the linear function $t \mapsto ty$), by composing on the left with weakening, we define the constant analytic function $X \to Y$, with value y.

In these models, as we have said, the morphisms of the (cartesian closed) Kleisli category of the comonad "!" can naturally be seen as analytic functions, and therefore can be differentiated. Classically, the derivative of a function $f : X \to Y$ is a function $f' : X \to (X \multimap Y)$ such that for each $x \in X$, the linear function $f'(x)$ (the derivative of f at point x) is the "best linear approximation" of the function $X \to Y$ which maps $u \in X$ to $f(x+u) \in Y$ (the general definition is local). Of course, if the function f is "very regular" (for instance, smooth or even analytic) then the derivative f' will have the same degree of regularity. In the analytic case, differentiation turns a linear function $f : !X \to Y$ into a linear function $f' : !X \to (X \multimap Y)$, that is, $f' : (!X \otimes X) \multimap Y$. It turns out that, just like for the structural logical operations above, f' can be obtained from f by composing it (as a linear function from $!X$ to Y) on the left with a particular linear morphism $(!X \otimes X) \to !X$. This morphism itself can be defined in terms of more primitive operation on $!X$ which appear somehow as symmetrical to the above mentioned structural morphisms.

- There is a linear map $X \to !X$ (co-dereliction) which corresponds to a particular case of differentiation: given an analytic function $f : X \to Y$ seen as a linear function $!X \to Y$, by composing f with co-dereliction, we obtain a linear map $X \to Y$ which is the derivative of f at point 0. Unlike the transformation of a linear function into an analytic function using dereliction, the transformation of an analytic function into a linear function using co-dereliction is very destructive, since all the non-linear part of the function is forgotten (think of the case where the analytic function is constant).
- There is a linear map $!X \otimes !X \to !X$ (co-contraction) which has the following effect. Given an analytic function $f : X \to Y$ seen as a linear function $!X \to Y$, by composing f with co-contraction, we obtain a linear map $!X \otimes !X \to Y$, that is an analytic function g with two parameters of type X, which is the function defined by $g(x, y) = f(x + y)$. Similarly, there is a particular vector in $!X$ (categorically, a linear map $1 \to !X$), called co-weakening, and whose effect by left-composition on an analytic f seen as a linear map $!X \to Y$ is to produce the value $f(0)$. These maps turn $!X$ into a $\otimes$-monoid (or algebra).

Of course, there are a number of commutations satisfied by these morphisms and the morphisms associated to the "?" modality. One can see the present formalism of differential nets as a graphical presentation of these commutations.

1.2. Differential nets

The exponential fragment of linear logic contains constructions for the weakening, contraction and dereliction operations of “?”; they can be equivalently presented in sequent calculus or proof nets formalisms [11]. In proof nets, unary, ternary and binary links (respectively) are associated with these three operations. On the other hand, linear logic has only one rule for “!”, an introduction rule called *promotion*. It is certainly the most complicated rule of linear logic: in sequent calculus, its application requires the context to have a certain shape —all formulae being ?-formulae —and in proof nets, it requires the introduction of *boxes*, whose main operational effect is to delimit sub-nets to be erased or duplicated as a whole by weakening or contraction links during cut elimination.

In the present setting, we have more rules associated with “!”: co-weakening, co-contraction and co-dereliction, as we have seen. Moreover, among these rules, co-weakening and co-dereliction appear as new introduction rules, exactly as weakening and dereliction are introduction rules for “?”. We can hope that it will make sense to extend proof nets with three new kinds of links associated to these rules, and even that such a system could exist *without promotion*, since we have other ways of introducing “!”-formulae. We develop here such a promotion-free formalism, using Lafont’s notion of *interaction nets* [14] rather than proof nets. [1]

In interaction nets, logical rules are represented by *cells*. Nets are made of cells connected with each other by *wires* through their *ports*. Each cell has a unique *principal port* and reduction occurs between cells connected by their principal ports: this means that the cuts of proof nets are replaced in interaction nets by wires between principal ports (see [15]).

As we have seen, there is a complete symmetry between ?-cells and !-cells (weakening vs. co-weakening, contraction vs. co-contraction and dereliction vs. co-dereliction). Strikingly, this symmetry extends to the reduction rules we have deduced from our models: if a redex consisting of two exponential cells connected by their principal ports (a “?”-cell, for instance a dereliction, and a “!”-cell, for instance a co-contraction) reduces to a net π, then the dual redex obtained by swapping “?” and “!” reduces to π where “?” and “!” are swapped.

One fundamental point which departs this syntax from standard interaction nets or proof nets is that, just like in [9], we are obliged to consider sums of nets, due to the fact that our reduction is “non-deterministic” (this point is extensively discussed and illustrated in [9]). These sums are created by the dereliction/co-contraction and co-dereliction/contraction reduction rules (the 0-ary versions of these rules, dereliction/co-weakening and co-dereliction/weakening yield zero nets, where 0 is the neutral element of the sum). In spite of this non-determinism, the calculus enjoys the Church–Rosser confluence property, as soon as reduction is properly extended to sums. The same phenomenon was already observed in [9]).

1.3. Correctness

Not all nets obtained by connecting together tensor, par, weakening, contraction, dereliction, co-weakening, co-contraction and co-dereliction cells are correct differential interaction nets. Obviously, it does not make sense to connect two tensor cells through their principal ports for instance. We avoid such ill-formed nets by introducing a typing system allowing recursive types and we prove that this notion of typability is preserved under reduction. But being typable in such a system is not enough for a differential net to be acceptable [2] : just as in multiplicative proof nets, a correctness criterion is needed. In ordinary linear logic, a proof net satisfying the Girard, or Danos–Regnier [6] criterion (for instance, but others have been designed) has the fundamental property of being sequentializable [11], which means that it comes from a proof in the sequent calculus of linear logic. Here we do not propose any sequent calculus corresponding to our interaction nets (though such a calculus exists, and sequentialization certainly holds), but we nevertheless define a correctness criterion, obviously inspired by the Danos–Regnier criterion: contraction is handled like a par and co-contraction like a tensor. What we prove is that this criterion is preserved under reduction, which guarantees that the normal forms of correct nets (typable nets satisfying the criterion) do not contain cycles (deadlocks). We also sketch a proof that all nets satisfying this criterion are strongly normalizable.

[1] This is basically an aesthetic choice: in proof nets, one of our reduction rules, the contraction/co-contraction reduction, would require the introduction of new axiom links in the reduced net, whereas axiom links are kept “implicit” in interaction nets as wires between auxiliary ports of cells.

[2] At least from a logical viewpoint. Such “incorrect” nets might be interesting from a purely computational viewpoint.

Observe that this criterion is the only feature of the presented formalism which breaks the symmetry between its positive ($\otimes$, !) and negative ($\parr$, ?) connectives.

1.4. Resource lambda-calculus

Then, we illustrate the connection between these nets and lambda-calculus. Of course, since we have decided to exclude promotion from this presentation, it is impossible to translate the (differential) lambda-calculus in differential interaction nets.[3] However, in [10], we observed that, when Taylor—expanding *all the applications* of a lambda-term, only a fragment of the differential lambda-calculus is needed. The target language of this translation can be seen as a *resource lambda-calculus* similar to those previously introduced and studied by various authors [3,4,13]. We translate this resource lambda-calculus in differential interaction nets.

1.5. Promotion

Last, we shortly discuss the issue of adding promotion (and hence ordinary application) to this resource calculus. Inspired by the Taylor formula, which is syntactically valid in lambda-calculus (see [9,10]), we present promotion as an "exponential function" on terms, subject to congruences similar in spirit to the "$!P = P\ |!P$" congruence of process calculi (see [17]) and meaning that a promoted term can be used as many times as needed. Such a presentation of promotion could easily be adapted to the present interaction net setting.

The more traditional presentation of promotion would require the introduction of "exponential boxes" (in the interaction nets formalism, such a box can be represented as a cell labeled by the net contained in the box), and, just as with proof-nets of ordinary linear logic, these boxes could interact with other boxes and cells through their principal and auxiliary ports. This is perfectly possible, the associated rewriting rules are the standard rules for exponential boxes in linear logic (see [11]), plus reduction rules corresponding to the interaction of a co-weakening, co-contraction or co-dereliction cell with an auxiliary port of a box, which can easily be deduced from the semantics [8,7], or from the intuitions presented in the present paper. For instance, the interaction between a co-dereliction and an auxiliary port of a box corresponds to the well known *chain rule* of differential calculus. The corresponding commuting categorical diagrams are made explicit in [1].

The resulting formalism, however, is more complicated than ordinary interaction nets. In particular, because boxes can interact with other boxes or with cells through their auxiliary ports, one loses the nice property of interaction nets which makes confluence proofs so easy: interaction can occur only between principal ports of cells. For that reason, and also because promotion breaks fundamentally the symmetry between "?" and "!" that we want to emphasize here, we have preferred not to give a detailed account of promotion in the present paper.

Our differential nets can be seen as a non-deterministic extension of multiplicative proof nets of linear logic, with exponential operations of contraction and co-contraction considered as multiplexing operators allowing several agents to communicate in a common "broadcast area", and dereliction and co-dereliction transforming agents into communicating agents. This viewpoint on exponentials strongly suggests to address the possible connections between differential nets and process calculi.

2. Syntax and reduction rules

2.1. General interaction nets

We assume to be given a collection of *symbols*. Each of these symbols is given together with a non-negative integer, its *arity*.

An *interaction net* consists of the following data:

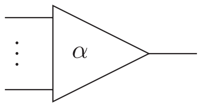
- a finite set P of *free ports*;
- a finite set C of *cells*, each cell c being given with a symbol $\mathsf{l}(c)$ (and the pairs (c, i) where i is a non-negative integer at most equal to the arity of c are called *connected ports*);

[3] This is actually not quite true: lambda-terms can be interpreted as generally infinite sums of differential nets, by Taylor expanding all applications.

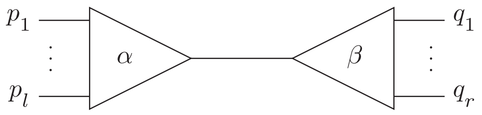
- a set W of *wires* together with a function ∂ mapping each wire to a set of 0 or 2 (free or connected) ports, and such that the collection of these sets be a partition of the set of all ports.

Each cell $c \in C$ has a particular port $(c, 0)$ which is called its *principal port*, the others being called *auxiliary ports*.

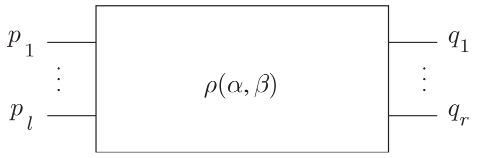
A cell of symbol α of arity n is usually pictured as follows:



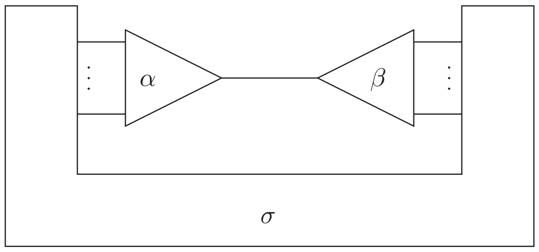
the singled out port on the right being the principal port, and the other $n-1$ ports on the left being the auxiliary ports. Interaction always occurs between two cells connected by their principal ports (such an interaction net is called a *redex*), and consists in replacing this redex by a given interaction net (depending only on the redex) having the same "free ports". To be more precise, a redex is a net



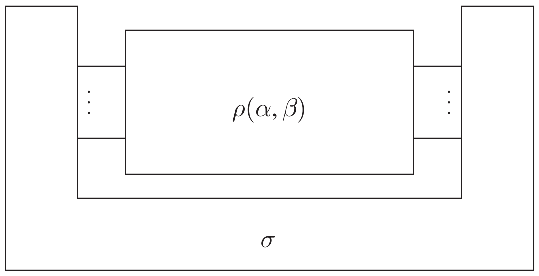
reducing to a net with the same ports $p_1, \ldots, p_l, q_1, \ldots, q_r$, and which only depends on the labeling symbols α and β:



This means that if the redex occurs within an interaction net π as follows:



then this interaction net reduces to



An *oriented wire* is a wire $w \in W$ such that ∂w has two elements, together with an ordering of these elements (that is, an ordered pair (p_1, p_2) with $\partial w = \{p_1, p_2\}$). Such an oriented wire w will be naturally pictured as an arrow, and can obviously be reversed into an oriented wire w^*. Typing[4] a net will consist in associating to each of its oriented

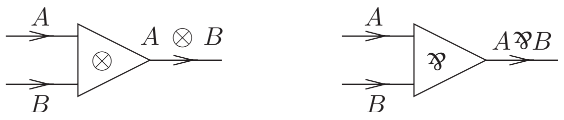
[4] We shall give later a more precise definition of net typing.

wires a formula of linear logic, with the constraint that the formula associated to w^* must be the linear negation of the formula associated to w.

When dealing with interaction nets, we shall not use this completely formal presentation which would lead to very boring developments. We shall stay at a more informal level, but all of our reasonings can be formalized.

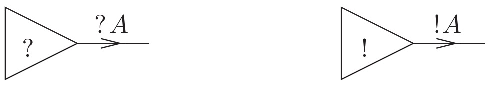
2.2. *Differential interaction nets*

We consider the following symbols. First, we have two *multiplicative* cells of arity 2: the tensor cell and the par cell, depicted as follows, together with their typing rules:

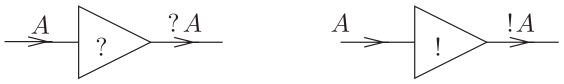


For each of the two exponentials, we have 3 cells of respective arities 0, 1 and 2. We give them together with their typing rule.

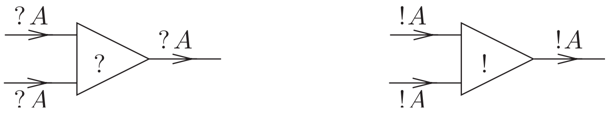
Weakening and co-weakening (or application to 0):



Dereliction and co-dereliction (or derivation at 0):



Contraction and co-contraction (or convolution product[5]):



The interaction nets built with these cells will be called *simple nets*, and general nets will be linear combinations of simple nets. We shall see that reducing simple nets gives rise in general to non-trivial linear combinations. These linearly combined simple nets must have the same free ports, which will be considered as the free ports of the linear combination itself. This allows to connect together general nets and not only simple nets, through their free ports. To turn such a connection of linear combinations of simple nets into a general net, we simply apply multi-linearity. In this way, we are reduced to considering only simple nets and linear combinations thereof.

More precisely, if we have two nets π and ρ, both of which are sums of simple nets ($\pi = \sum_{i=1}^{l} \pi_i$ and $\rho = \sum_{j=1}^{r} \rho_j$) and if we want to connect these two nets through their common free ports $p_1, \ldots, p_n$ (of course, π and ρ can have other free ports), then the result of this operation is the sum of the $l \times r$ simple nets obtained by connecting[6] π_i with ρ_j through the ports $p_1, \ldots, p_n$ for all i, j.

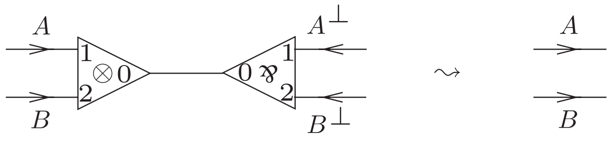
[5] This terminology comes from an analogy with distributions, see the end of [7].

[6] This operation should be defined precisely and the correct definition would be rather long because each of the two connected nets can contain wires w with ∂w consisting of two elements belonging to $\{p_1, \ldots, p_n\}$; such wires can be connected in the resulting net just like electric extensions, and these composed wires must be reduced to standard ones, producing possibly loops in the resulting net (wires w with $\partial w = \emptyset$). But there is of course no conceptual difficulty here.

2.3. Reduction rules

The first reduction rule concerns the multiplicative cells and it is completely standard.

2.3.1. Tensor/par

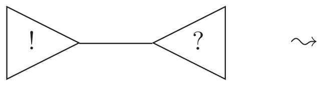


One has to be careful about the fact that the ports of a cell have indexes and reduction respects these indexes. We have written here these indexes (0, 1 and 2) explicitly for both cells. This is also essential for typing: if the port 1 of a $\otimes$ cell is connected to an incoming wire of type A and the port 2 to an incoming wire of type B, then any outcoming wire connected to its principal port 0 will carry type $A \otimes B$ (and not $B \otimes A$!). Similarly for ⅋ cells, of course. A priori, one has also to take this into account for contraction and co-contraction cells. But we shall see in the reduction rules that they are actually commutative (this point will be made more explicit in future work).

Next come the rules concerning the exponential cells. As we shall see, they are completely symmetrical.

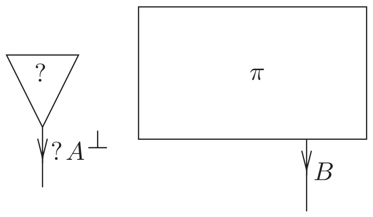
2.3.2. Co-weakening/weakening

This yields nothing (or more precisely, a multiplicative nothing), as follows:

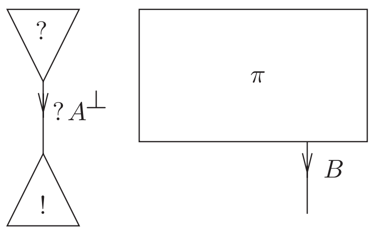


This means that if a simple net contains the weakening/co-weakening redex, it reduces to the same net, but without this redex.

We justify this rule as follows. Imagine that we have a (simple) net π with conclusion B, and that we build a constant function $f : A \to B$ of value π as the following net:



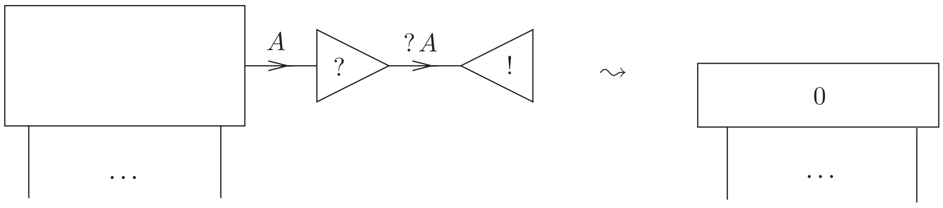
Applying this function to the value 0 amounts to building the net, whose only pending wire is of type B:



Obviously, the value of this expression is π, which justifies the reduction rule.

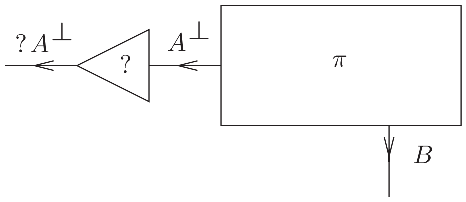
2.3.3. *Co-weakening/dereliction and co-dereliction/weakening*

These rules cancel completely simple nets. Let us start with the dereliction/co-weakening which has a ludics [12] flavor (if we identify co-weakening with *daemon*):

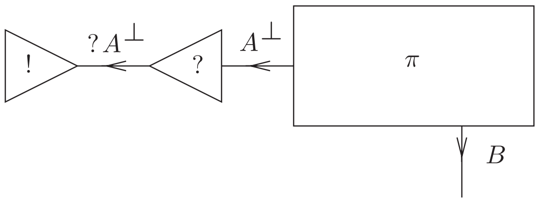


where the box on the right-hand side denotes the 0 net, with the same free ports as the left-hand net.

As an intuitive justification, consider the following situation, where π can be considered as a linear function from A to B:

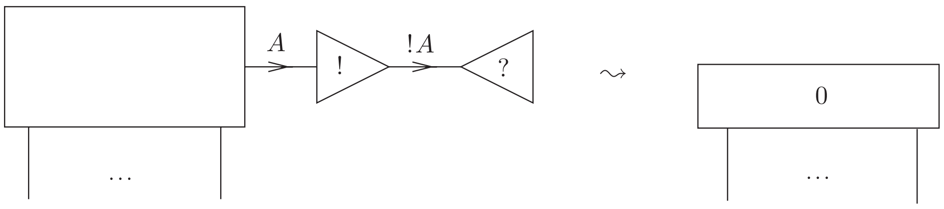


The complete net above, with conclusions $?A^\perp$ and B has therefore to be considered as an "analytic" function from A to B which happens to be linear: this is precisely the purpose of the dereliction cell. Plugging a co-weakening as follows:

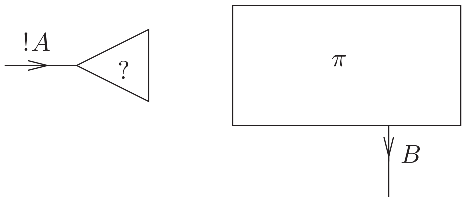


amounts to applying this linear function to 0, and we obtain the value 0 of type B, as prescribed by our reduction rule. In this reduction, the whole simple net π disappears (whether connected or not) and our initial configuration is replaced by a 0-net of type B.

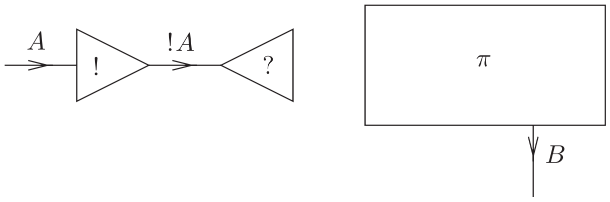
The co-dereliction/weakening interaction is completely symmetrical.



Again, let us give an intuitive justification. Using a weakening cell, we can build a constant "analytic" function f from A to B, which always takes the value corresponding to π:



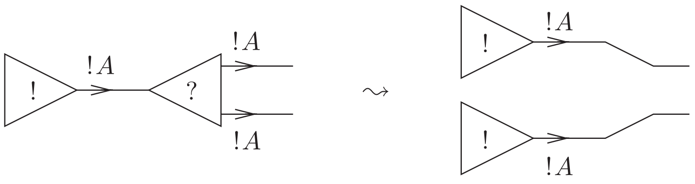
The following net represents then the derivative of *f* at point 0:



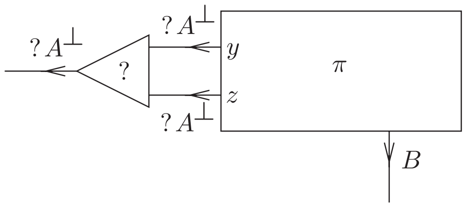
which indeed must be a linear function from *A* to *B*. This function must be null since *f* is constant. This is exactly what our reduction rules prescribe.

2.3.4. *Co-weakening/contraction and weakening/co-contraction*

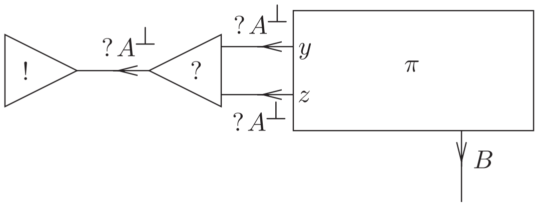
Both rules are duplications. The first rule is as follows:



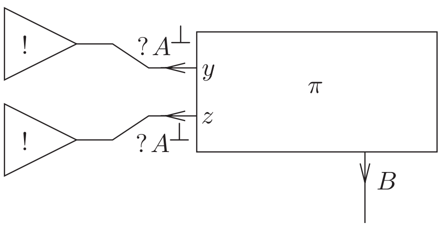
As a justification, consider the following net, where π represents an analytic function $f(y, z)$ with two parameters (named here y and z for notational convenience, and that we indicate in the picture below) of type *A* and value of type *B*



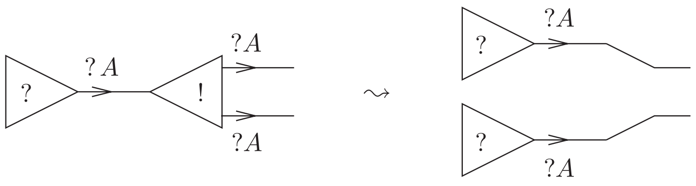
This net represents the analytic function *g* from *A* to *B* such that $g(x) = f(x, x)$. Building the net



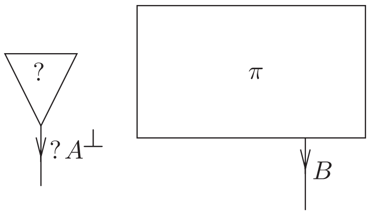
amounts to apply the function *g* to 0, which clearly produces the same result as the following net (which corresponds to $f(0, 0)$):



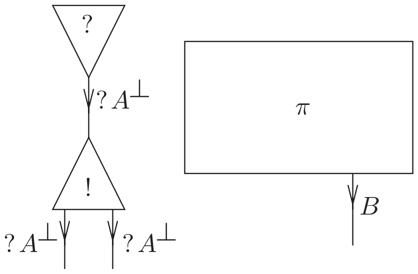
The second reduction rule is symmetrical to that one.



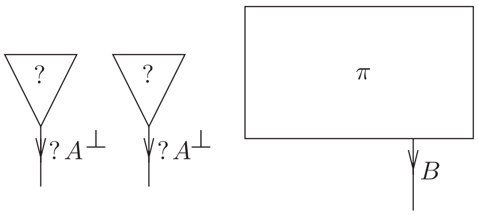
Consider indeed the following net, which represents a constant function *f* (with value *b* of type *B*, represented by π), and with one parameter of type *A*:



Then we build the following two parameter function *g* by plugging a co-contraction cell

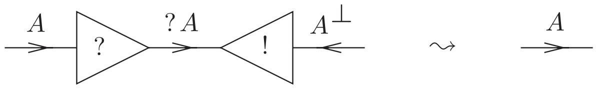


This function is defined by $g(y, z) = f(y + z) = b$ and is just a two parameter constant function, in accordance with our reduction rule which produces the following net:

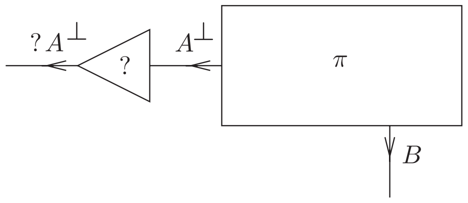


2.3.5. Co-dereliction/dereliction

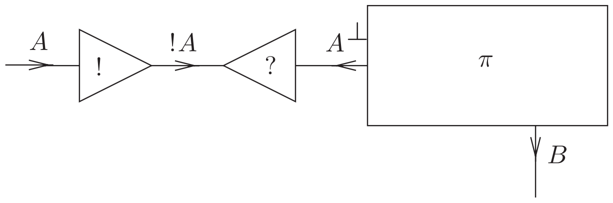
This is a simplification without much surprises.



Consider again the following net:



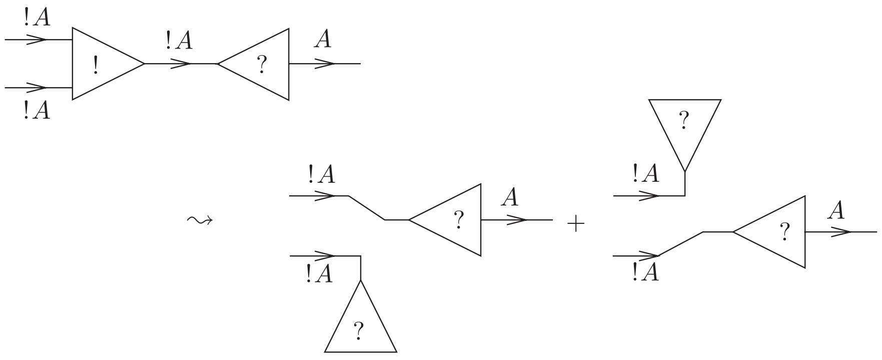
which represents an analytic function *f* from *A* to *B* which happens to be the linear function *h* represented by the net π. Then the following net:



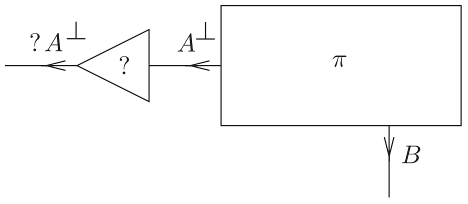
represents the linear function $f'(0)$, the derivative of *f* at point 0; it is well known that this function must be *h*, in accordance with our reduction rule.

2.3.6. Co-contraction/dereliction and contraction/co-dereliction

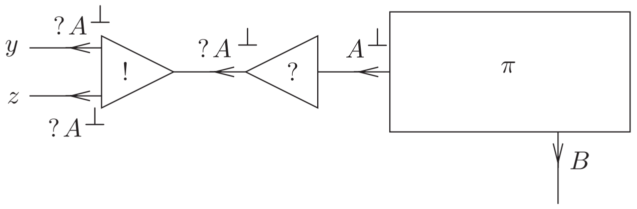
Both situations lead to sums as follows. The first rule is



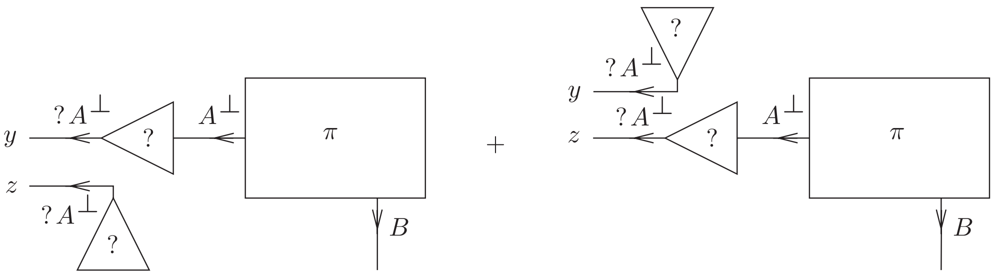
Consider once more the net



which represents an analytic function *f* from *A* to *B* which happens to be the linear function *h* represented by the net π. Then the following net:

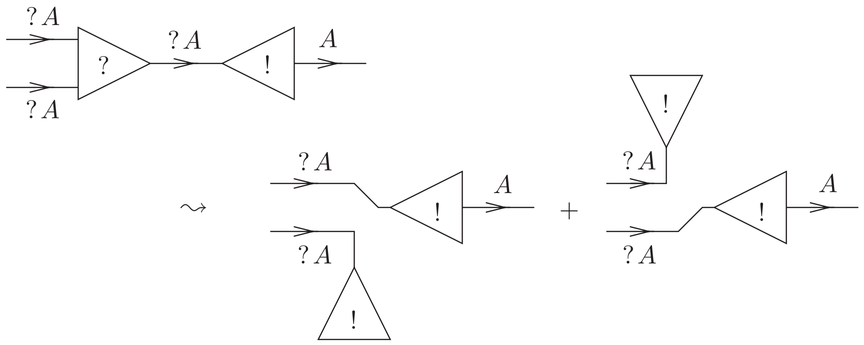


represents the two parameters analytic function g given by $g(y, z) = f(y + z)$. By linearity of f, we have $g(y, z) = f(y) + f(z)$. But this latter expression is represented by the following sum of simple nets:

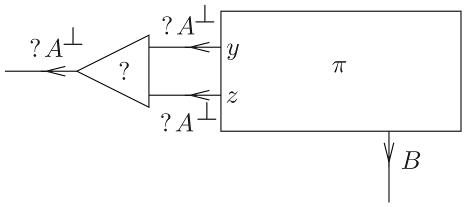


In the first summand, the variable z is not used, whence the weakening cell, and symmetrically for the second summand. This is exactly the equation implemented by the reduction rule under consideration (taking into account our conventions on distributivity).

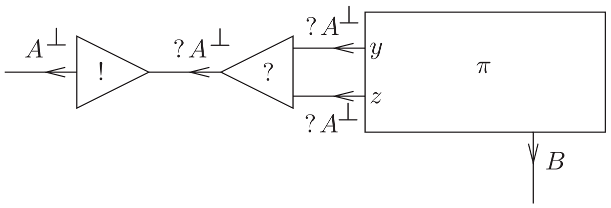
Strikingly enough, the contraction/co-dereliction configuration leads to a completely symmetrical pattern, but for rather different mathematical reasons.



Consider indeed the following net, where π represents an analytic function $f(y, z)$ with two parameters (named here y and z for notational convenience, and that we indicate in the picture below) of type A and value of type B:

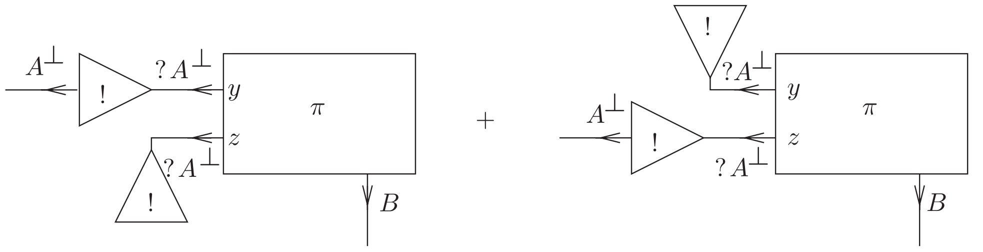


This net represents the analytic function g from A to B such that $g(x) = f(x, x)$. Then the following net represents the linear function $g'(0)$, from A to B



Given a vector u of type A, we know (by the usual laws of calculus) that $g'(0) \cdot u = f'_y(0, 0) \cdot u + f'_z(0, 0) \cdot u$ ($f'_y(0, 0)$ and $f'_z(0, 0)$ are the two partial derivatives, computed at point $(0, 0)$, which are linear functions from A to B).

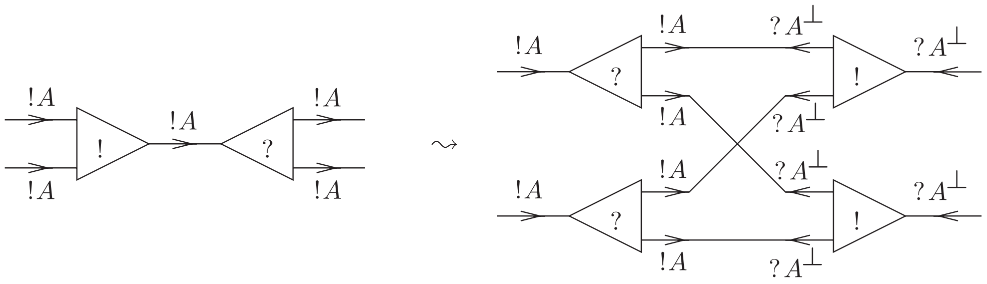
This latter expression corresponds to the following sum of nets:



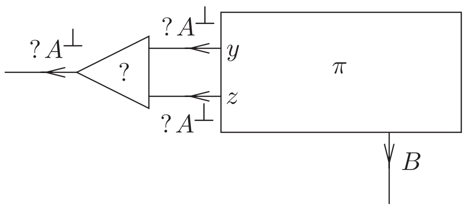
which is the net obtained by applying the co-dereliction/contraction reduction above, as well as our convention on distributivity.

2.3.7. *Co-contraction/contraction*

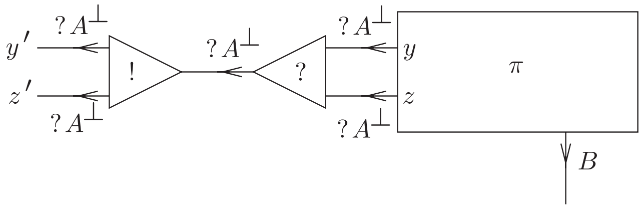
This is just the standard rule of bi-algebras.



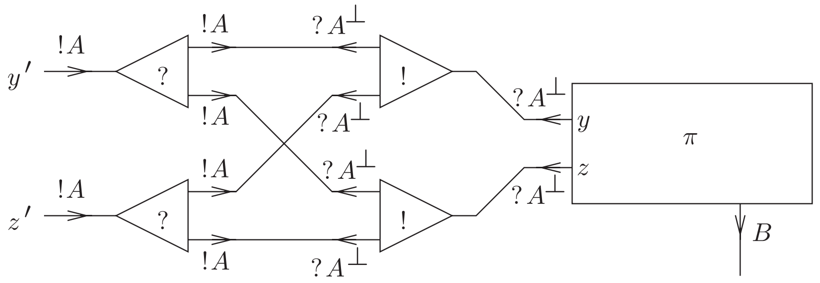
We have typed the wires of the right-hand net in various different ways for illustrating our conventions. Consider again the following net, where π represents an analytic function $f(y, z)$ with two parameters of type A and value of type B:



This net represents the analytic function g from A to B such that $g(x) = f(x, x)$. Plugging a co-contraction, we get the net



which represents the two parameters function h given by $h(y', z') = f(y'+z', y'+z')$. Now it is clear that the following net represents exactly the same function. Here, the variables y' and z' are duplicated before being added, whereas in the net above, duplication was performed after addition; the result is obviously the same, in accordance with our reduction rule.

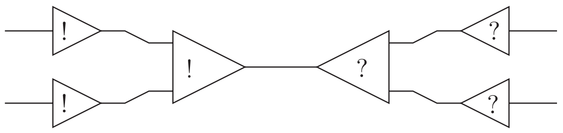


2.4. *What we have defined so far*

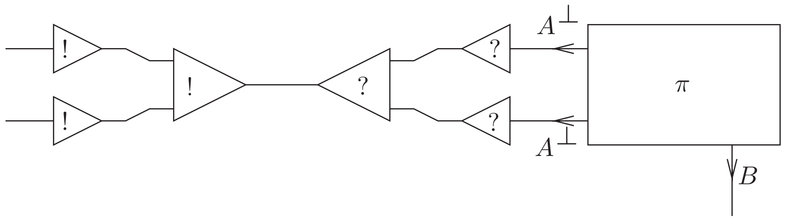
To be precise, we have defined a relation $\rightsquigarrow$, which is a relation from simple nets to nets (actually, sums of 0, 1 or 2 simple nets only). This relation will be called *simple one step reduction* and will be generalized later, when we shall be concerned with the confluence issue.

2.5. *Example of reduction*

Consider the following simple net (the reader can check that it is typable):

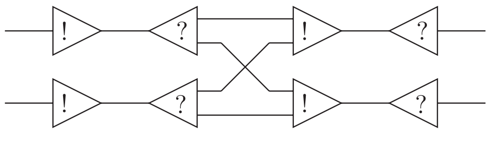


We suggest the reader to try the following game: on the right side of this configuration, connect a simple net π representing a *bilinear* function $f(y, z)$, taking two parameters of type A, and yielding values of type B, as follows:

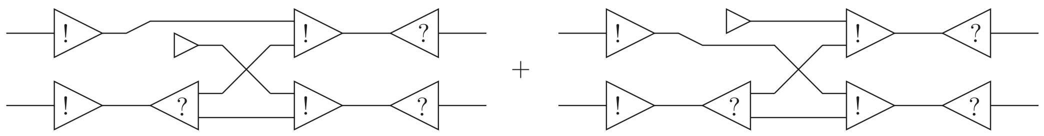


Using the above explanation about the cells as operating on analytic functions, try to figure out what should be the function corresponding to this configuration, and compare your computation with the following reduction.

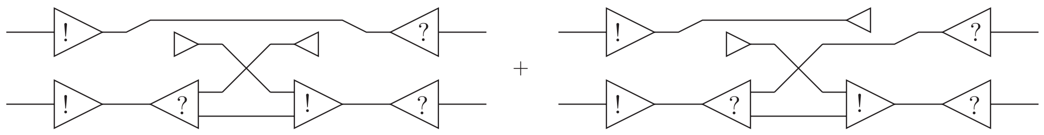
There is only one redex, and the reduction yields



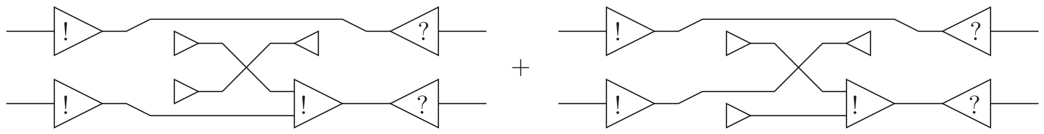
a simple net with 4 redexes. Each of these redexes leads to a sum of two simple nets. So we obtain a sum of 16 simple nets, 14 of which will disappear reducing to 0 as we shall see. After reducing the upper-left redex, we get a sum of two similar simple nets π_1 and π_2:



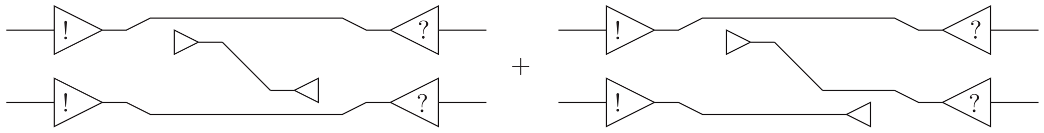
We consider only the first of these nets. If now we reduce the upper-right redex, we get



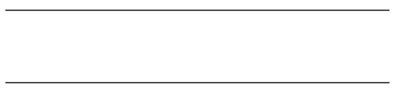
In the right-hand term of this sum, we have a co-dereliction/weakening redex which annihilates completely this simple net, and the sum reduces to its first member. Reducing the lower-left redex in this net, we get



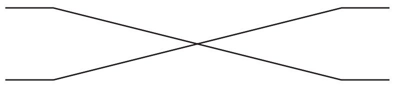
The right-hand member of this sum contains a co-dereliction/weakening redex and therefore reduces to 0. In the left-hand member, we can fire the weakening/co-weakening redex, which vanishes, and the co-contraction/dereliction redex, leading to



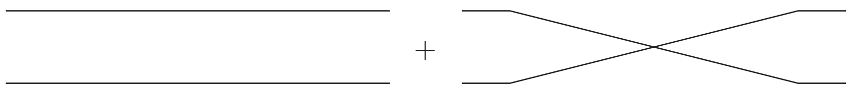
The right-hand term of this sum reduces again to 0 (it contains a dereliction/co-weakening and a co-dereliction/weakening redex), whereas the first net, after having fired the weakening/co-weakening and the two dereliction/co-dereliction redexes, leads simply to



Performing the same computation on π_2, we obtain



Finally, the normal form of our simple net is the sum of these two wirings:



3. Weak typing

The nets defined in the previous section can contain *pathological patterns* consisting of two cells connected by their principal ports and for which we have given no reduction rules—for instance two tensor cells connected by their principal ports—or can reduce to nets containing such pathological patterns. We therefore introduce a very weak typing system which will have the following two key features:

- a weakly typable net contains no pathological pattern (weak correctness), and
- if π is weakly typable and π reduces to π', then π' is also weakly typable (subject reduction).

Let us assume to be given a denumerable set $\mathcal{A}$ of type variables. Types are defined by the following syntax.

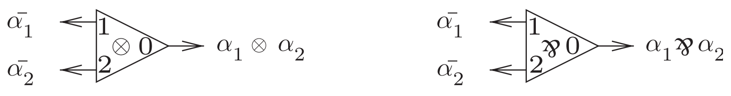
- If α is a type variable, then α and $\bar{\alpha}$ are (atomic) types. The atomic type $\bar{\alpha}$ corresponds to the linear negation of α and is called a "negated type variable".
- If A and B are types, then $A \otimes B$ and $A \parr B$ are types.
- If A is a type then $!A$ and $?A$ are types.

If A is a type, then $A^\perp$ is defined by induction using the De Morgan laws of linear logic (with of course $\alpha^\perp = \bar{\alpha}$ and $(\bar{\alpha})^\perp = \alpha$). An important operation is substitution of a formula for a type variable within another formula; this is defined as usual, and requires (for the negated occurrences of the substituted variable) computing linear negations "on the fly". For instance, $(\alpha \otimes \bar{\alpha})[!\beta/\alpha] = !\beta \otimes ?\bar{\beta}$.

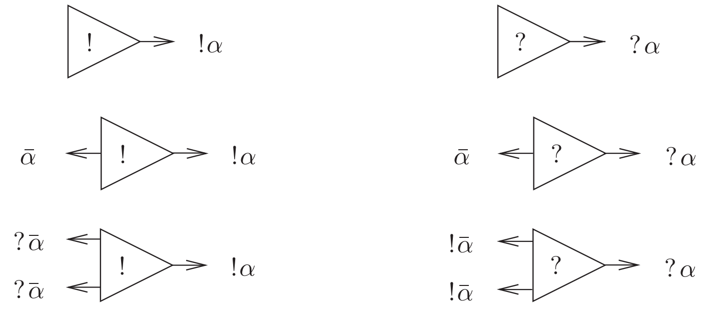
3.1. Typing a net

Let π be a simple net. To the cells c of π, we associate pairwise disjoint sets of type variables S_c and we assign types to all ports of all cells (considered conventionally, for each cell, as oriented toward the outside) as follows. We also assign to each free port of π type variables, these variables being distinct from each other and distinct from the variables assigned to connected ports. This assignment of types to ports will sometimes be called the *primitive type assignment* of π.

If c is a tensor or a par cell, then S_c must have two elements α_1 and α_2, and the types associated to the ports are



If c is an exponential cell, then S_c must have one element α, and the associated types are



Warning: for a technical reason related to the dereliction/co-contraction and co-dereliction/contraction reduction rules, in the case where c is a weakening or co-weakening cell, and only in these cases, we allow also the unique element of S_c to be a negated type variable.

In this way, each port of π receives a type. We define now a set $\mathcal{E}_\pi$ of equations between types: it is the set of all the equations $A \simeq B^\perp$ where A and B are types associated to ports related by a wire in π. Up to renaming of type variables, this set of equations is uniquely defined and depends only on π.

A set $\mathcal{E}$ of equations between types is called a *simultaneous recursion* (following [5]) if $\mathcal{E} = \{\alpha_1 \simeq A_1, \dots, \alpha_n \simeq A_n\}$ (we use the symbol "$\simeq$" rather than "$=$" for denoting these formal equations) where the α_i's are *pairwise distinct* type variables and the A_i are types which are not of the shape $\bar{\beta}$ (negated atom).[7]

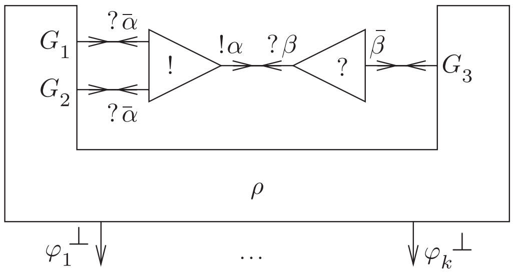
Let π be a simple net, and let $p_1, \dots, p_k$ a repetition-free list of its free ports. We say that the simple net π is *weakly typable* of type $A_1, \dots, A_k$ if there is a simultaneous recursion $\mathcal{E}$ such that $\mathcal{E} \vdash_{\mathrm{eq}} \mathcal{E}_\pi$ where $\vdash_{\mathrm{eq}}$ stands for standard equational reasoning (with the multiplicative and exponential connectives of linear logic as function symbols, and substitution defined as explained above) and $\mathcal{E} \vdash_{\mathrm{eq}} \varphi_i = A_i{}^\perp$ for $i = 1, \dots, k$ (where φ_i are the type variables associated to the free ports p_i by the primitive type assignment of π). In this situation, we write

$$\mathcal{E} \vdash \pi \mid p_1 : A_1, \dots, p_k : A_k$$

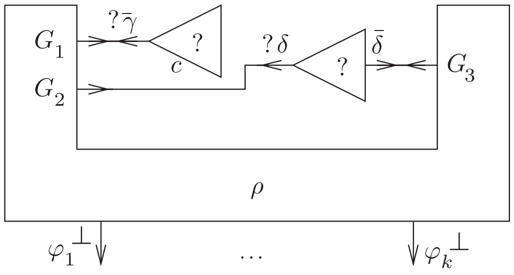
3.2. Subject reduction

The first property of this notion of typing is subject reduction. If π is a simple net, if $\mathcal{E} \vdash \pi \mid p_1 : A_1, \dots, p_k : A_k$ and if π reduces to a sum of simple nets $\sum_{j=1}^n \pi_n$, then there is a simultaneous recursion $\mathcal{E}'$ such that $\mathcal{E}' \vdash \pi_j \mid p_1 : A_1, \dots, p_k : A_k$ for each j.

This is proved by simply examining each reduction. We just consider the two most complicated cases. Assume that π is of the shape



Then we know that $\mathcal{E} \vdash_{\mathrm{eq}} G_1 \simeq {!\alpha}$, $\mathcal{E} \vdash_{\mathrm{eq}} G_2 \simeq {!\alpha}$, $\mathcal{E} \vdash_{\mathrm{eq}} {!\alpha} \simeq {!\bar{\beta}}$, $\mathcal{E} \vdash_{\mathrm{eq}} G_3 \simeq \beta$ and $\mathcal{E} \vdash_{\mathrm{eq}} \varphi_j \simeq A_j{}^\perp$ for $j = 1, \dots, k$. According to the co-contraction/dereliction reduction rule, this net reduces to a sum of two simple nets $\pi_1 + \pi_2$; we consider only π_1, the other one being similar:



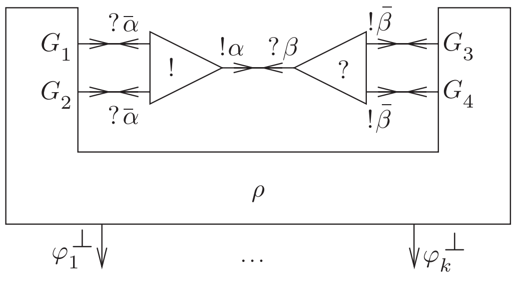
We have kept the same primitive assignment for the subnet ρ, and we have introduced fresh type variables γ and δ for the cells obtained by reducing the redex under consideration (*warning*: for the new weakening cell c, we have taken $S_c = \{\bar{\gamma}\}$). Of course, this primitive assignment induces the same types as above for the free ports of ρ (namely, $\varphi_1,\dots,\varphi_k$, G_1, G_2 and G_3). Let $\mathcal{E}' = \mathcal{E} \cup \{\gamma \simeq \alpha, \delta \simeq \beta\}$. Then $\mathcal{E}'$ is a simultaneous recursion. By hypothesis, and since $\mathcal{E}' \supseteq \mathcal{E}$, we know that $\mathcal{E}' \vdash_{\mathrm{eq}} \mathcal{E}_\rho$ and $\mathcal{E}' \vdash_{\mathrm{eq}} \varphi_j \simeq A_j{}^\perp$ for $j = 1, \dots, k$. Moreover, $\mathcal{E}' \vdash_{\mathrm{eq}} G_1 \simeq {!\alpha}$, hence $\mathcal{E}' \vdash_{\mathrm{eq}} G_1 \simeq {!\gamma}$ since $\mathcal{E}' \vdash_{\mathrm{eq}} \gamma \simeq \alpha$. Similarly $\mathcal{E}' \vdash_{\mathrm{eq}} G_3 \simeq \delta$. Last, by hypothesis we have $\mathcal{E}' \vdash_{\mathrm{eq}} G_2 \simeq {!\alpha}$, and also $\mathcal{E}' \vdash_{\mathrm{eq}} {!\alpha} \simeq {!\bar{\beta}}$.

[7] The purpose of this restriction is to avoid the dramatic cyclic definition $\alpha \simeq \bar{\alpha}$; we have no trouble with equations between positive atoms, we shall see that they are necessary.

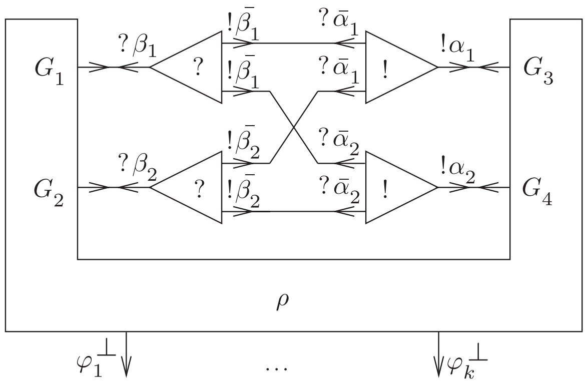
But $\mathcal{E}' \vdash_{\mathrm{eq}} \delta \simeq \beta$, hence $\mathcal{E}' \vdash_{\mathrm{eq}} !\alpha \simeq !\bar{\delta}$. Therefore $\mathcal{E}' \vdash_{\mathrm{eq}} G_2 \simeq !\bar{\delta}$, and finally $\mathcal{E}' \vdash \pi_1 \mid p_1 : A_1, \ldots, p_k : A_k$. One checks easily that, with a similar extension of $\mathcal{E}'$ as a simultaneous recursion $\mathcal{E}''$, one has $\mathcal{E}'' \vdash \pi_2 \mid p_1 : A_1, \ldots, p_k : A_k$ (and of course $\mathcal{E}'' \vdash \pi_1 \mid p_1 : A_1, \ldots, p_k : A_k$ remains valid).

The case of a co-dereliction/contraction reduction is completely similar.

Last, let us consider the co-contraction/contraction reduction. So we start with a simple net of the shape



We know as above that $\mathcal{E} \vdash_{\mathrm{eq}} G_1 \simeq !\alpha$, $\mathcal{E} \vdash_{\mathrm{eq}} G_2 \simeq !\alpha$, $\mathcal{E} \vdash_{\mathrm{eq}} !\alpha \simeq !\bar{\beta}$, $\mathcal{E} \vdash_{\mathrm{eq}} G_3 \simeq ?\beta$, $\mathcal{E} \vdash_{\mathrm{eq}} G_4 \simeq ?\beta$ and $\mathcal{E} \vdash_{\mathrm{eq}} \varphi_j \simeq A_j{}^{\perp}$ for $j = 1, \ldots, k$. Firing this redex, we obtain the following simple net π':



Again, we assume that the type variables α_i and β_i are fresh. Let $\mathcal{E}' = \mathcal{E} \cup \{\alpha_i \simeq \alpha, \beta_i \simeq \beta \mid i = 1, 2\}$ which is a simultaneous recursion. Then we have $\mathcal{E}' \vdash_{\mathrm{eq}} G_1 \simeq !\alpha$ and $\mathcal{E}' \vdash_{\mathrm{eq}} !\alpha \simeq !\bar{\beta}$ by hypothesis, hence $\mathcal{E}' \vdash_{\mathrm{eq}} G_1 \simeq !\bar{\beta}_1$; the equations concerning G_2, G_3 and G_4 are dealt with similarly. The "internal" equations such as $\mathcal{E}' \vdash_{\mathrm{eq}} !\bar{\beta}_1 = !\alpha_2$ result from $\mathcal{E} \vdash_{\mathrm{eq}} !\alpha \simeq !\bar{\beta}$ and from the four equations we have added to $\mathcal{E}$.

The other reduction rules are similarly simple to deal with.

Observe that this general notion of typing subsumes the simple typing outlined at the beginning of the paper: a net π with ports $p_1, \ldots, p_n$ is simply typable with type A_i for port p_i iff $\mathcal{E} \vdash \pi \mid p_1 : A_1, \ldots, p_n : A_n$ with a recursion $\mathcal{E} = \{\alpha_j \simeq B_j \mid i = 1, \ldots, m\}$ such that, for each j and each $l \leqslant j$, the variable α_j does not appear in the type B_l (such a recursion will be called a *loop-free* recursion).

3.3. Weak correctness of weakly typed nets

Let us say that a net is *weakly correct* if it contains no pathological pattern. It is very easy to prove that a weakly typed net is weakly correct. It suffices to consider the set $\mathcal{T}$ of infinite types built with the connectives $\otimes$, $⅋$, ! and ? (the infinite trees whose nodes are either binary, and then labeled by $\otimes$ or by $⅋$, or are unary, and then labeled by ! or by ?).

A valuation $\mathcal{I} : \mathcal{A} \to \mathcal{T}$ can be extended to all finite types in the obvious way. Given a set $\mathcal{E}$ of equations, let us write $\mathcal{I} \vDash \mathcal{E}$ if $\mathcal{I}(A) = \mathcal{I}(B)$ for all equation $A \simeq B$ belonging to $\mathcal{E}$. Equational reasoning on types is sound with respect to this semantics: if $\mathcal{E} \vdash_{\mathrm{eq}} \mathcal{E}'$ and $\mathcal{I} \vDash \mathcal{E}$, then $\mathcal{I} \vDash \mathcal{E}'$. Now we conclude observing first that, if $\mathcal{E}$ is a simultaneous recursion, there exists a valuation $\mathcal{I} : \mathcal{A} \to \mathcal{T}$ such that $\mathcal{I} \vDash \mathcal{E}$ (this is fundamentally due to our restriction that simultaneous recursions cannot contain equations of the shape $\alpha = \bar{\beta}$ where α and β are type variables, and to the fact that, in a simultaneous recursion, a type variable cannot occur twice as left member of an equation), and second that if for some valuation $\mathcal{I}$, one has $\mathcal{I} \models \mathcal{E}_\pi$, then the simple net π is weakly correct.

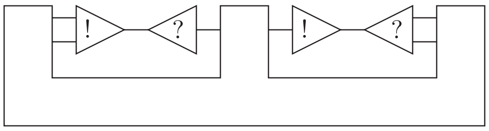
4. Confluence

One of the nice features of Lafont's interaction nets is that they enjoy the Church–Rosser property in the strongest sense: the diamond property holds for the one-step reduction. This will also be the case here, but we have to be careful when extending the one-step reduction $\rightsquigarrow$ to arbitrary nets (that is, to sums of simple nets).

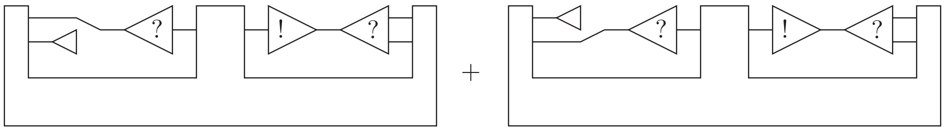
An adequate definition of this concept is the following. Let π be a net which is a sum $\pi = \sum_{i=1}^{n} \pi_i$ of simple nets, and let π' be another net. We say that π *reduces in one step* to π' if $\pi' = \sum_{i=1}^{n} \pi'_n$ where for each i, $\pi'_i = \pi_i$ or $\pi_i \rightsquigarrow \pi'_i$, and $\pi_i \rightsquigarrow \pi'_i$ for at least one i. In that case, we write $\pi \rightsquigarrow \pi'$. Observe that when π is simple, then the original notion of simple one-step reduction coincides with this new notion, so it makes sense to keep the same notation for both. Observe also that if π_1 and π_2 are nets and if $\pi_i \rightsquigarrow \pi'_i$ for $i = 1, 2$, then $\pi_1 + \pi_2 \rightsquigarrow \pi'_1 + \pi'_2$. This may seem weird for a notion of one-step reduction, but one must keep in mind that the sum represents a kind of non-deterministic superimposition of simple nets.

Now the following statement is easy to check. Let π, π_1 and π_2 be nets. If $\pi \rightsquigarrow \pi_i$ for $i = 1, 2$, then $\pi_1 = \pi_2$ or there exists a net π' such that $\pi_i \rightsquigarrow \pi'$ for $i = 1, 2$. From our definition of one-step reductions for general nets, we can restrict our attention to the case where π is simple. For producing π_1 and π_2, we choose two redexes in π, that we can of course assume to be distinct. If none of these redexes is a co-dereliction/contraction or dereliction/co-contraction redex, then we conclude as in standard interaction nets, using the fact that reduction is purely local (there is of course the case where one of the redexes is a co-dereliction/weakening or co-weakening/dereliction, but in that case we can converge to the 0 net).

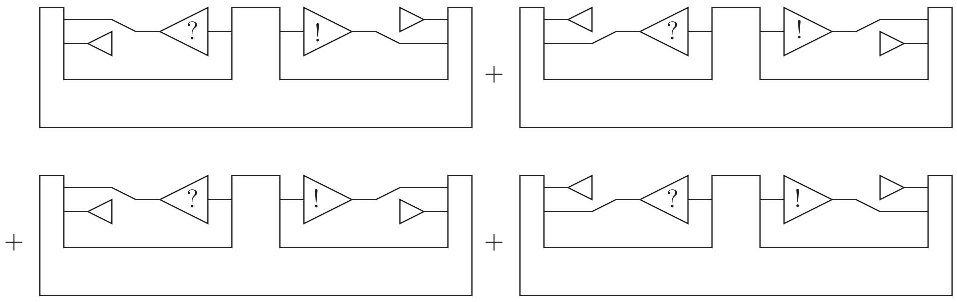
Assume for instance that one of the redexes is a co-dereliction/contraction redex and that the other one is a co-contraction/dereliction redex, so that our net π has the following shape:



Reducing the left redex, we obtain in one step the following sum:



Now reducing the right redex in both terms of this sum, we obtain *in one step* again the following sum, which we also obtain from the original net by reducing first the right redex, and then the left redexes (up to associativity and commutativity of addition, of course):



The other cases are similar.

5. Acyclicity criterion

We extend the Danos–Regnier criterion for multiplicative proof nets to the present setting. We shall not prove a sequentialization theorem, simply because we do not present here any sequent calculus corresponding to these nets. The criterion is nevertheless essential because it will prevent the appearance of cyclic structures during reduction, and thereby ensure strong normalization.

Let π be a simple net. A *switching* is a map S from the set of par and contraction cells of π to the set $\{1, 2\}$ (the names of the auxiliary ports of the cell). Given such a switching S, we define a non-oriented graph $\mathcal{G}(\pi, S)$. The nodes of this graph are the ports of π. Let p and q be two ports of π, there is an edge between p and q in $\mathcal{G}(\pi, S)$ if

- there is a wire between p and q in π, or
- p and q are the two ports of a dereliction or co-dereliction cell, or
- p and q are the principal port and one of the auxiliary ports of a tensor or of a co-contraction cell, or
- p and q are the principal port and the port i ($i = 1$ or $i = 2$) of a par or of a contraction cell c such that $S(c) = i$.

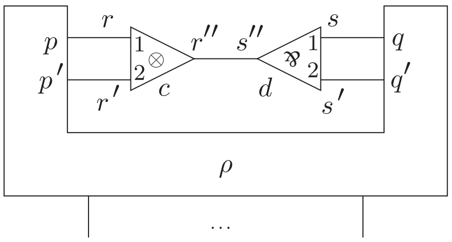
We say that π is *acyclic* if $\mathcal{G}(\pi, S)$ is an acyclic graph (a forest) for each switching S of π.

We cannot require connexity as one usually does in multiplicative linear logic because this property is not preserved by the co-contraction/dereliction reduction (the redex itself is a simple net without par or contraction cells and which gives rise to a connected graph, whereas each summand of the result of the reduction gives rise to is a disconnected graph; this problem might be solved by introducing an edge between the weakening cell and the dereliction cell, but we do not care so much about connexity anyway here).

5.1. Preservation under reduction

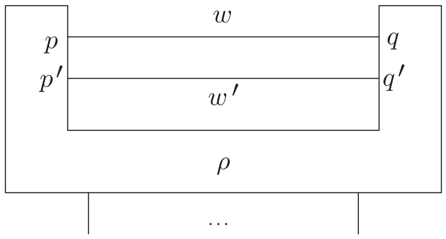
The main property is again that this acyclicity property is preserved under reduction: if an acyclic simple net π reduces to a sum $\sum_{i=1}^{n} \pi_i$ of simple nets, then each of the π_i's is acyclic. The proof is a simple checking of all the possible reductions, as we did for subject reduction. Let us concentrate on the tensor/par and on the contraction/co-contraction reductions, the other ones obviously reducing the connectivity of the net.

Consider first the case where π has the following structure.[8] (we have given names to the ports involved in the reduction):



Due to the acyclicity of π, we must have $p \neq q$, $p \neq q'$, $p' \neq q$ and $p' \neq q'$. To be more precise, what we have pictured here is not a simple net, but two simple nets, namely ρ and the considered redex consisting of c and d, these two simple nets being connected by identifying the free ports p and r, p' and r', etc. As far as acyclicity is concerned, this is harmless.

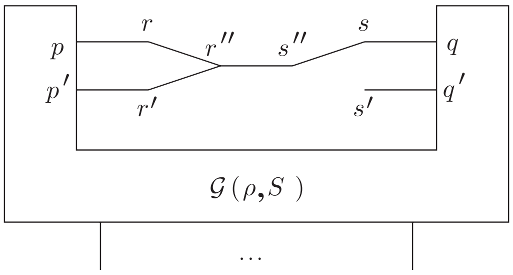
The reduced net π' is



[8] This case is well known by the way, since it corresponds to the standard correctness criterion of MLL proof nets. We give the proof here for self-containedness, and because it corresponds to a sub-case of the contraction/co-contraction situation.

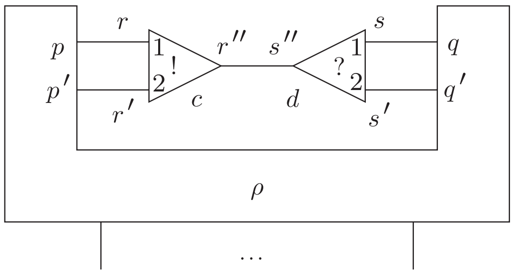
Let S be a switching of π', that is, a switching of the subnet ρ. Assume towards a contradiction that the graph $\mathcal{G}(\pi', S)$ has a cycle $C = (p_1, \dots, p_n)$ (so p_i are pairwise distinct ports of π', such that p_i and p_{i+1} as well as p_n and p_1 are connected in $\mathcal{G}(\pi', S)$, and moreover $n \geqslant 3$). Then C must contain both wires w and w'. Indeed, if it contains none, then C is a cycle in $\mathcal{G}(\rho, S)$ which is impossible since $\mathcal{G}(\rho, S)$ is acyclic because ρ is a subnet of π which is acyclic. If C passes only through w, then extend S to the switching S_1 of π which maps the par cell d to 1; it is easy to transform C into a cycle in the graph $\mathcal{G}(\pi, S_1)$, contradiction.

So we can assume without loss of generality that $p_1 = p$ and $p_2 = q$, and that there exists $i < n$ such that
(1) $i > 2$, $p_i = p'$ and $p_{i+1} = q'$, or
(2) $i > 2$, $p_i = q'$ and $p_{i+1} = p'$.
In the first case, consider the switching S_1 of π obtained by extending S with $d \mapsto 1$. The corresponding graph is

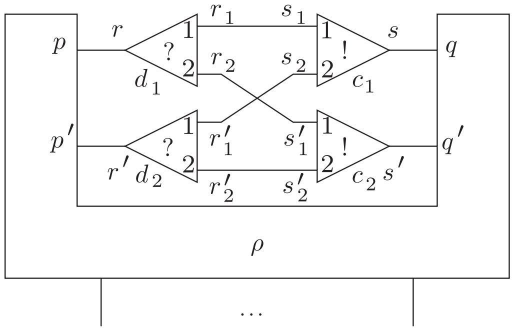


In this graph, $(q, p_3, \dots, p_{i-1}, p', r', r'', s'', s)$ is a cycle which contradicts the acyclicity of π. In the second case, both switching S_1 and S_2 (where $d \mapsto 2$) provide the cycle $(p, r, r'', r', p', p_{i+2}, \dots, p_n)$.

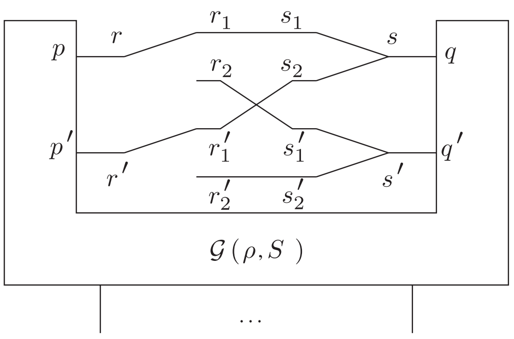
Consider now the co-contraction/contraction situation, in a simple net π:



By reducing this redex, we obtain the following simple net π':

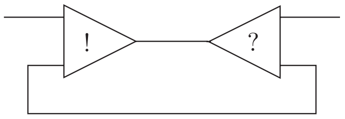


and we must show that $\mathcal{G}(\pi', S')$ is acyclic, whatever be the switching S' of π' that we consider. Such a switching is obtained by extending a switching S of ρ by giving values 1 or 2 to the two contraction cells d_1 and d_2. Up to symmetries, there are only two such extensions: S'_1 which maps d_1 to 1 and d_2 to 2, and S'_2 which maps d_1 and d_2 to 1. The first switching produces a graph $\mathcal{G}(\pi', S'_1)$ whose acyclicity is obtained exactly as in the tensor/par situation above. The second switching produces the following graph $\mathcal{G}(\pi', S'_2)$:



whose acyclicity results from the acyclicity of $\mathcal{G}(\pi, S_1)$ where S_1 is the switching of π which is equal to S on ρ and maps d to 1.

We shall say that a net is *correct* if it is weakly typable and acyclic. As we have seen, if π is correct and reduces to π', then π' is also correct. A simple net, even if it is simply typable, can have infinite reductions, as shown by the following example:



The trouble with this net is of course that it contains a cycle (put the contraction switch on the lower position). The main property of correct nets is that they enjoy strong normalization.

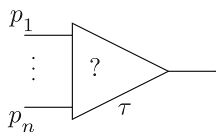
5.2. Co-contraction and contraction trees

We call *contraction trees* the simple nets generated by the following inductive definition (we define in the same induction the principal port and the auxiliary ports of a contraction tree).

- A weakening cell is a contraction tree, whose principal port is the principal port of the cell and which has no auxiliary ports.
- If τ_1 and τ_2 are contraction trees with disjoint sets of ports, and with principal ports p_1 and p_2, respectively, then the net obtained by plugging p_1 and p_2 to the auxiliary ports of a contraction cell c is a contraction tree, whose principal port is the principal port of c, and whose auxiliary ports are those of τ_1 and those of τ_2.

In a completely similar way, one defines the notion of co-contraction tree.

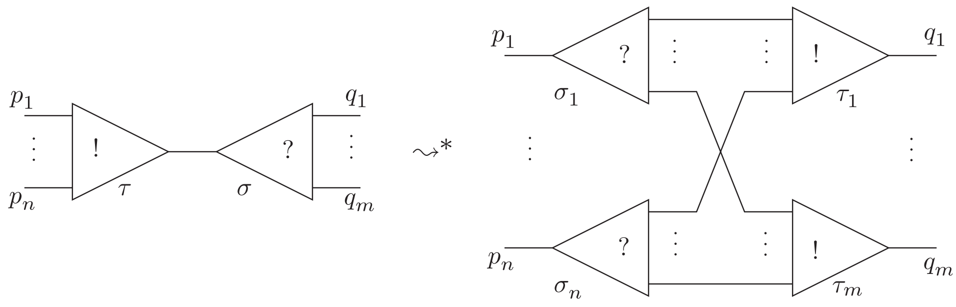
A contraction tree τ with auxiliary ports $p_1, \dots, p_n$ will typically be pictured as follows:



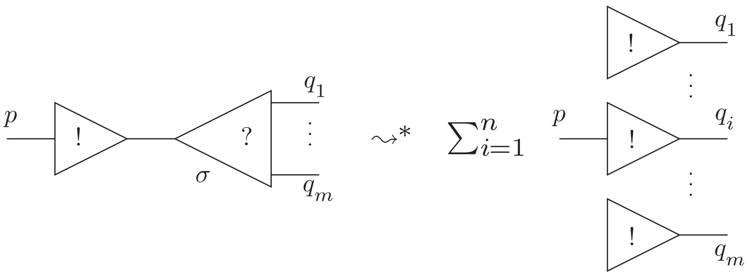
We adopt similar conventions for co-contraction trees.

5.3. Generalized reduction

Observe that the reduction rules involving a contraction cell or a co-contraction cell generalize in the obvious way (but now with several steps of reductions of course). For instance



where $\sigma_1, \ldots, \sigma_n$ are n disjoint copies of σ, and similarly for the τ_j's. Similarly, we have the following reduction, which generalizes both co-dereliction/contraction ($m = 2$) and co-dereliction/weakening ($m = 0$) reductions.



Last, there is a dual reduction which generalizes both dereliction/co-contraction and dereliction/co-weakening (just swap the "!" and the "?" in the picture above).

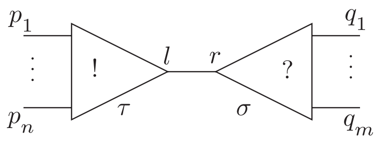
We define a new relation $\leadsto_{\mathrm{g}}$ from simple nets to nets as the reduction which takes these as elementary steps (plus the tensor/par reduction, still considered as an elementary step). We extend this relation to general nets as we did for $\leadsto$. This new reduction relation is clearly included in $\leadsto^*$, the transitive closure of $\leadsto$.

5.4. Weak normalization (sketch)

We outline a proof that $\leadsto$ is a (weakly) normalizing reduction; strong normalization follows, by the diamond property of $\leadsto$, established in Section 4.

If π is a simple net, its *pure size* $|\pi|_{\mathrm{P}}$ is the number of tensor, par, dereliction and co-dereliction cells in π. Its *structural size* $|\pi|_{\mathrm{S}}$ is the number of contraction and co-contraction cells (we do not count weakening and co-weakening cells as one might have expected here).

In the present context, let us call *structural redex* a co-weakening/weakening, co-weakening/contraction, weakening/co-contraction or co-contraction/contraction redex. By examining the auxiliary ports of the two cells involved in such a redex R (whose principal ports will be denoted l and r) within a simple net π, we observe that such a redex is uniquely embedded into a subnet of π which has the following structure



where
- τ is a co-contraction tree and σ is a contraction tree;
- no port p_i is the principal port of a co-contraction tree of π and
- no port q_j is the principal port of a contraction tree of π.

This structure will be called the generalized structural redex associated to R and $j = 1, \dots, m$. Observe moreover that if π is correct, by acyclicity, one has $p_i \neq q_j$ for each $i = 1, \dots, n$. Therefore, when reducing this generalized structural redex, no new structural redex is created: this is the key observation in our proof sketch.

Let $\mathsf{S}(\pi)$ be the number of structural redexes in π.

If π is a correct simple net, we define its measure $\mathsf{M}(\pi)$, a triple of non-negative integers, by setting $\mathsf{M}(\pi) = (|\pi|_{\mathsf{P}}, \mathsf{S}(\pi), |\pi|_{\mathsf{S}})$ if π is not normal, and $\mathsf{M}(\pi) = (0, 0, 0)$ if π is normal. We extend this notion of measure to arbitrary correct nets $\pi = \sum_{i=1}^{n} \pi_i$ (where the π_i's are simple and correct) by setting $\mathsf{M}(\pi) = \max_{i=1}^{n} \mathsf{M}(\pi_i)$, triples of integers being ordered under the lexicographic order.

We define now a particular reduction (a non-deterministic reduction strategy) first as a relation from correct simple nets to correct nets as follows. Let π be a correct simple net and let π' be a net, we say that $\pi \rightsquigarrow_{\mathsf{W}} \pi'$ if
(1) π contains a dereliction/co-weakening or a co-dereliction/weakening redex and $\pi' = 0$, and else if
(2) π contains a tensor/par or a dereliction/co-dereliction redex and π' is obtained from π by reducing this redex, and else if
(3) π contains a structural redex and π' is obtained from π by reducing the associated generalized structural redex (in the sense of the generalized reduction defined above), and else if
(4) π contains a dereliction/co-contraction or a co-dereliction/contraction redex and π' is obtained from π by reducing this redex.

Observe then that $\pi \rightsquigarrow^* \pi'$ (and hence π' is correct) and that $\mathsf{M}(\pi') < \mathsf{M}(\pi)$ (for the lexicographic order on triples of integers). Observe also that if π is a correct simple net which is not normal, then $\pi \rightsquigarrow_{\mathsf{W}} \pi'$ for some net π' since one of the four cases enumerated above must occur. We extend this reduction relation to arbitrary correct nets as follows: if π is a sum $\sum_{i=1}^{n} \pi_i$ of simple correct nets, we say that $\pi \rightsquigarrow_{\mathsf{W}} \pi'$ if $\pi' = \sum_{i=1}^{n} \pi'_i$ where, for each i, $\pi'_i = \pi_i$ if π_i is normal, and $\pi_i \rightsquigarrow_{\mathsf{W}} \pi'_i$ otherwise. Again, if π is an arbitrary correct net and $\pi \rightsquigarrow_{\mathsf{W}} \pi'$ for this extended notion of reduction, then $\pi \rightsquigarrow^* \pi'$ (and hence π' is correct) and $\mathsf{M}(\pi') < \mathsf{M}(\pi)$. This proves that our reduction $\rightsquigarrow$ is weakly normalizing, as announced.

6. Interpreting the resource lambda-calculus in differential nets

The resource lambda-calculus (or lambda-calculus with multiplicities) has been invented by Boudol [3] and further studied also by Curien and Lavatelli [4]. The work of Kfoury [13] develops the same line of ideas consisting in introducing, within an ordinary lambda-calculus, the idea of *linear application*, a new kind of application where the argument is provided exactly once to the function. The motivations of these authors were quite different, but their basic idea of linear application can be interpreted in our setting as differential application.

Let us consider a version of the resource lambda-calculus where application is really linear (not affine). A more comprehensive study of this calculus is available in [10]

The syntax we define has two sorts (simple[9] terms and simple poly-terms) which are defined by mutual induction as follows. As usual, we are given a denumerable supply of variables.
- If x is a variable, then x is a simple resource term.
- If x is a variable and s is a simple resource term, then $\lambda x\, s$ is a simple resource term.
- If s is a simple resource term and S is a simple poly-term, then $\langle s \rangle S$ is a simple resource term.
- If $s_1, \dots, s_n$ are simple resource terms, then $[s_1, \dots, s_n]$ (the multi-set consisting of these terms, repetitions being taken into account) is a simple poly-term.

In the present setting, we find convenient to use the following more algebraic notations for multi-sets representing simple poly-terms: 1 represents the empty simple poly-term, if t is a simple resource term, t also denotes the simple

[9] As before, "simple" here means not being a linear combination.

poly-term $[t]$, and if S and T are simple poly-terms, then ST is the concatenation of the multi-sets S and T (that is, the multi-set union of S and T: the number of occurrences of a simple term in ST is its number of occurrences in S plus its number of occurrences in T). So, the poly-term $[s_1, \ldots, s_n]$ will simply be written as the product $s_1 \cdots s_n$. Similarly, we write T^2 for TT, the multi-set concatenation of T with itself, etc.

From now on we drop the adjective "resource" everywhere and say "term" instead of "resource term". Let Δ be the set of all simple terms and $\Delta^!$ be the set of all simple poly-terms

Let R be a ring.[10] If U is a set, we denote by $R\langle U\rangle$ the free R-module built on U, that is the set of all finite formal linear combinations of elements of U with coefficients in R. The elements of $R\langle\Delta\rangle$ are called terms, and the elements of $R\langle\Delta^!\rangle$ are called poly-terms. All syntactic constructions on simple terms and poly-terms are extended to arbitrary terms and poly-terms by linearity (when they are unary constructions) or multi-linearity (when they are multi-ary constructions: this is the case of application and of poly-term construction). For instance, if $s = \sum_{i=1}^{n} a_i s_i$ is a term which is a linear combination of the pairwise distinct simple terms s_i, then

$$\lambda x\, s = \sum_{i=1}^{n} a_i \lambda x\, s_i$$

and if $S = \sum_{i=1}^{n} a_i S_i$ is a poly-term which is a linear combination of the pairwise distinct simple poly-terms S_i, and $T = \sum_{j=1}^{m} b_j T_j$ is a poly-term which is a linear combination of the pairwise distinct simple poly-terms T_j, then the following equation will hold:

$$ST = \sum_{i=1,j=1}^{n,m} a_i b_j S_i T_j.$$

To give a concrete example of this quite standard convention, the expression $\lambda x\, \langle 2y\rangle(3x+z)^2$ stands for the term

$$18\lambda x\, \langle y\rangle x^2 + 12\lambda x\, \langle y\rangle xz + 2\lambda x\, \langle y\rangle z^2.$$

6.1. Substitution and partial derivation

Substitution of a term t for a variable x in a simple term or poly-term σ (we use this kind of Greek letter for denoting a term or poly-term, when we do not want to make a distinction between these two sorts) is defined as usual by induction on σ (*warning*: t is not assumed to be simple, so the result is not simple in general, and one uses the generalized syntactic constructions defined above by multi-linearity). This operation is then extended to arbitrary σ's by linearity; it is linear in σ but not in t. A particularly important case is when $t = 0$: if σ contains at least one free occurrence of x, then $\sigma[0/x] = 0$, and otherwise $\sigma[0/x] = \sigma$.

More important is the operation of *partial derivation* of terms and poly-terms, *in the direction* of a given term. Given a variable x and a term u, by induction on the simple term or poly-term σ, we define $\frac{\partial\sigma}{\partial x}\cdot u$ (linear in σ *and in* u) as follows:

$$\frac{\partial y}{\partial x}\cdot u = \begin{cases} u & \text{if } y = x, \\ 0 & \text{otherwise,} \end{cases}$$

$$\frac{\partial \lambda y\, t}{\partial x}\cdot u = \lambda y \left(\frac{\partial t}{\partial x}\cdot u\right) \quad \text{where we assume of course that } y \neq x,$$

[10] We can use arbitrary coefficients as long as normalization issues are not concerned, for our resource terms as well as for differential nets: this point is discussed in [9] and more accurately studied in [18]. As set R of coefficients, we can also take a semi-ring, which is like a ring apart from the fact that the additive monoid is not supposed to be a group; the set $\mathbb{N}$ of natural numbers is a typical semi-ring.

$$\frac{\partial \langle s\rangle S}{\partial x}\cdot u=\left\langle \frac{\partial s}{\partial x}\cdot u\right\rangle S+\langle s\rangle\left(\frac{\partial S}{\partial x}\cdot u\right),$$

$$\frac{\partial [s_1,\ldots,s_n]}{\partial x}\cdot u=\sum_{i=1}^{n}\left[s_1,\ldots,s_{i-1},\frac{\partial s_i}{\partial x}\cdot u,s_{i+1},\ldots,s_n\right].$$

Observe that with our notations for poly-terms, $\frac{\partial 1}{\partial x}\cdot u=0$ and $\frac{\partial ST}{\partial x}\cdot u=(\frac{\partial S}{\partial x}\cdot u)T+S(\frac{\partial T}{\partial x}\cdot u)$. Observe also that more generally $\frac{\partial \sigma}{\partial x}\cdot u=0$ as soon as x does not occur free in σ, and that $\frac{\partial \sigma}{\partial x}\cdot u$ is linear in u in the sense that $\frac{\partial \sigma}{\partial x}\cdot(\sum a_i u_i)=\sum a_i\frac{\partial \sigma}{\partial x}\cdot u_i$. This operation $\frac{\partial \sigma}{\partial x}\cdot u$ is then extended by linearity to arbitrary terms and poly-terms σ. The basic commutation property of this operation is the following equation, which is a syntactic version of Schwarz Lemma:

$$\frac{\partial}{\partial y}\left(\frac{\partial t}{\partial x}\cdot u\right)\cdot v=\frac{\partial}{\partial x}\left(\frac{\partial t}{\partial y}\cdot v\right)\cdot u+\frac{\partial t}{\partial x}\cdot\left(\frac{\partial u}{\partial y}\cdot v\right),$$

where we assume that x does not occur free in v. In particular, if we assume moreover that y does not occur free in u, then the two partial derivatives commute (whence the name we gave to this property).

6.2. Reduction

A *redex* is a simple term s of the shape $s=\langle\lambda x\, t\rangle T$ (so t and T are simple). A redex reduces to a term which is not simple in general. If $T=1$, then s reduces to $t[0/x]$, and if $T=uU$, then s reduces to $\langle\lambda x\,(\frac{\partial t}{\partial x}\cdot u)\rangle U$ (of course, T can be written as a product uU in various different ways in general and thus this redex can reduce to various different terms in general). This notion of reduction is then defined for arbitrary simple terms and poly-terms (and not simply redexes) by extension to contexts. We denote by $\rightsquigarrow$ the corresponding reduction relation, which is thus included in $(\Delta\times R\langle\Delta\rangle)\cup(\Delta^!\times R\langle\Delta^!\rangle)$ (and more precisely in $(\Delta\times\mathbb{N}\langle\Delta\rangle)\cup(\Delta^!\times\mathbb{N}\langle\Delta^!\rangle)$). This relation can be extended to arbitrary terms and poly-terms (with coefficients in R) as we did in [9]. But here, when we are concerned with reduction, we assume for simplicity that $R=\mathbb{N}$, and we extend $\rightsquigarrow$ to arbitrary linear combinations as we did for differential nets.

This notion of reduction enjoys confluence as well as strong normalization. Confluence results from the syntactic version of Schwarz Lemma we mentioned above. Strong normalization is essentially trivial, as reducing a simple term or poly-terms, one obtains a sum of terms or poly-terms whose all elements have less symbols than the original term.

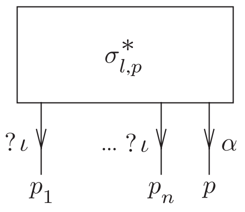
6.3. Translation

We define a translation of simple terms and poly-terms to correct simple nets which are typable in a simultaneous recursion $\mathcal{E}$ which is a loop-free recursion to which we add the two following equations involving two distinct type variables ι and o:

$$\iota={!o}\otimes\iota\quad\text{and}\quad o\simeq{?\iota}\invamp o$$

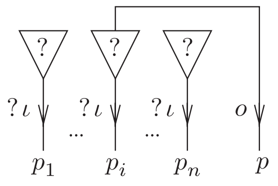
Such a net will be said to be DR-typable (in reference to the pure proof nets of Danos and Regnier, which have the same kind of typing, see [16]).

Let l be a list $((x_1,p_1),\ldots,(x_n,p_n))$ where the x_i's are pairwise distinct variables and the p_i's are pairwise distinct free ports (in other words, l is a finite partial injection from variables to free ports). Given a simple term or poly-term σ whose all free variables are contained in the domain $\{x_1,\ldots,x_n\}$ of l, and given a free port p not belonging to the image $\{p_1,\ldots,p_n\}$ of l, we define a simple differential net $\sigma^*_{l,p}$ whose free ports are $p_1,\ldots,p_n$ and p. This net will be typable as follows:

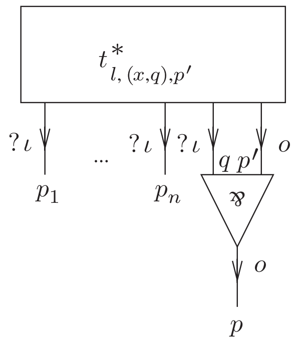


where $\alpha = o$ when σ is a simple term, and $\alpha = !o$ when σ is a simple poly-term.

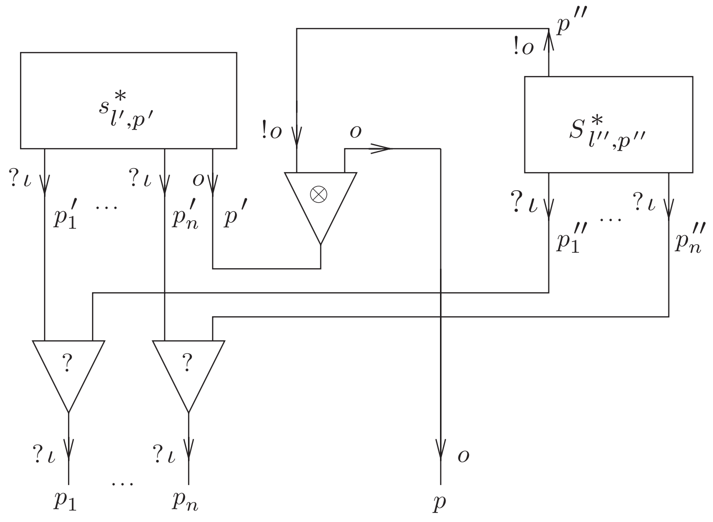
The definition is by induction on σ. If σ is a variable, and hence $\sigma = x_i$ for a uniquely determined $i \in \{1, \ldots, n\}$, then $\sigma^*_{l,p}$ is



If $\sigma = \lambda x\, t$, then $\sigma^*_{l,p}$ is

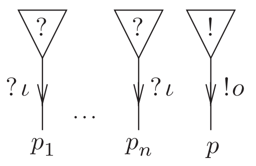


where q, p' are distinct free ports, distinct from p and from the p_i's. If $\sigma = \langle s\rangle S$, then $\sigma^*_{l,p}$ is

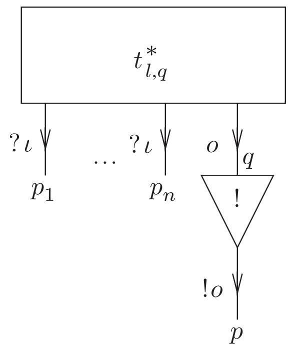


where $l' = ((x_1, p'_1), \ldots, (x_n, p'_n))$ and $l'' = ((x_1, p''_1), \ldots, (x_n, p''_n))$ are finite injections such that l, l' and l'' have disjoint images, and p, p' and p'' are distinct free ports, not belonging to the union of these images.

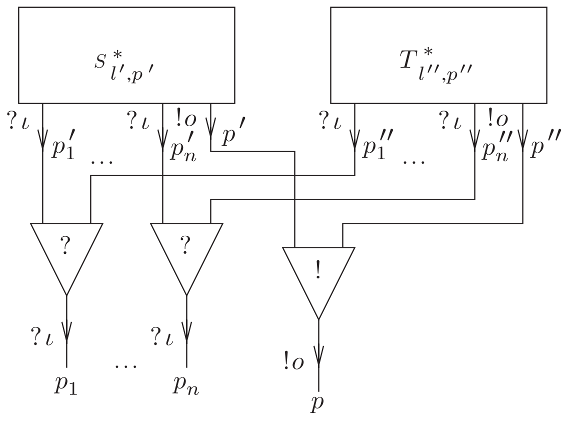
Let us turn now to the interpretation of poly-terms. If $\sigma = 1$, then $\sigma^*_{l,p}$ is



If $\sigma = [t]$ where t is a simple term, then $\sigma^*_{l,p}$ is



Last, if $\sigma = ST$, then (with the same notational conventions as in the case of an application) $\sigma^*_{l,p}$ is



Due to this last case, the translation is not exactly deterministic, because there are many ways of decomposing a poly-term as products. But the various nets obtained are equivalent (in a sense to be defined in future work) simply because all co-contraction trees having the same arity are equivalent, and similarly for contraction trees.

A straightforward induction shows that the simple differential nets obtained in that way are DR-typable as well as acyclic, and thus correct, and hence strongly normalizing.

Moreover, it can be checked that if $t \rightsquigarrow^* t'$, then the normal forms of the differential interaction nets $t^*_{l,p}$ and $t'^*_{l,p}$ are equal.

6.4. A few words about promotion

This resource calculus is essentially finitary, in the sense that it lacks the usual application operation of lambda-calculus which allows for arbitrary duplications. Duplication is present here since we have a contraction cell in our differential nets, but is not a real duplication in the sense of lambda-calculus: here, when a poly-term (multi-set of simple terms) is "duplicated" by a contraction cell, it is cut into two disjoint pieces which are communicated to the two auxiliary ports of this cell; this is done in all possible ways, whence the sums which appear during the reduction.

The finitary nature of this calculus is revealed by its strong normalization property which holds even in the untyped case. For obtaining the full power of lambda-calculus (and in particular the possibility of representing all partial recursive functions), one has to introduce a *promotion* operation (in the sense of promotion in linear logic, which turns a proof of $!A \multimap B$ into a proof of $!A \multimap !B$). In the present resource calculus, promotion is represented by a new way of constructing poly-terms, similar to the $t \mapsto t^\infty$ construction of Boudol's resource lambda-calculus [3], or the $t \mapsto !t$ of the π-calculus [17], and which plays exactly the same role of providing a potentially infinite supply of t.

So given a simple term t, we introduce a new simple poly-term $\exp_0(t)$, which corresponds to the promotion of t. The intention behind this construction is that $\exp_0(t)$ is equal to an infinite "exponential" sum, where the exponents have to be understood in the sense of the product of poly-terms:

$$\exp_0 t = \sum_{n=0}^{\infty} \frac{t^n}{n!}$$

assuming of course that this makes sense with our choice of R (take for instance R to be a field).

The reason for the index 0 in $\exp_0 t$ is that if we want to keep our calculus finite (no infinite sums of simple terms appearing during computations), then we have to equip our promotion with a rule similar to the "$!t = (t\,|\,!t)$" congruence of the π-calculus. One very natural way to proceed is to say that

$$\begin{aligned}
\exp_0 t &= 1 + t \exp_1 t,\\
\exp_1 t &= 1 + \frac{t}{2} \exp_2 t,\\
&\vdots\\
\exp_n t &= 1 + \frac{t}{n+1} \exp_{n+1}.
\end{aligned}$$

So our extension of the syntax of the resource lambda-calculus consists in introducing, for each simple term t and non-negative integer n, a new poly-term $\exp_n t$ corresponding intuitively to the series $\sum_{k=0}^{\infty} \frac{n!}{(n+k)!} t^k$.

We can consider the equation $\exp_n t = 1 + \frac{t}{n+1} \exp_{n+1} t$ as part of the reduction, saying that $\exp_n t \rightsquigarrow 1 + \frac{t}{n+1} \exp_{n+1} t$ but this of course has the serious drawback of preventing strong normalization to hold, even in the simply typed case. It is certainly wiser, and fundamentally equivalent, to implement this equation by saying that

$$\langle \lambda x\, s\rangle \exp_n t \rightsquigarrow s[0/x] + \frac{1}{n+1} \left\langle \lambda x \left(\frac{\partial s}{\partial x} \cdot t \right) \right\rangle \exp_{n+1} t.$$

Of course, to complete the description of this extension of the resource lambda-calculus, one has to say what $\frac{\partial \exp_n s}{\partial x} \cdot t$ should be in general. Fortunately, this does not require the introduction of new term constructions, but surprisingly introduces negative coefficients. The reader will check easily that the following equation holds, when one expands the $\exp_n$'s into power series:

$$\frac{\partial \exp_n s}{\partial x} \cdot t = \left(\exp_n s - \frac{n}{n+1} \exp_{n+1} s \right) \frac{\partial s}{\partial x} \cdot t,$$

where, again, the product has to be taken in the sense of poly-terms multiplication (for $n = 0$, this is just the fact that the exponential is its own derivative). It can be checked that the calculus so defined still enjoys confluence, but of course, strong normalization is lost in the untyped case.

More material on the differential viewpoint on the resource lambda-calculus, and especially on the Taylor expansion of ordinary lambda-terms as infinite linear combinations of resource terms, can be found in [10].

References

[1] R. Blute, R. Cockett, R. Seely, Differential categories, unpublished article, 2005.

[2] R. Blute, P. Panangaden, R. Seely, Fock space: a model of linear exponential types, in: Proc. Ninth Conf. on Mathematical Foundations of Programming Semantics, Lecture Notes in Computer Science, Vol. 802, Springer, Berlin, 1994, pp. 1–25.

[3] G. Boudol, The lambda calculus with multiplicities, Technical Report, 2025, INRIA Sophia-Antipolis, 1993.

[4] G. Boudol, P.-L. Curien, C. Lavatelli, A semantics for lambda calculi with resource, Math. Structures Comput. Sci. 9 (4) (1999) 437–482.

[5] F. Cardone, M. Coppo, Decidability properties of recursive types, in: C. Blundo, C. Laneve (Eds.), ICTCS, Lecture Notes in Computer Science, Vol. 2841, Springer, Berlin, 2003, pp. 242–255.

[6] V. Danos, L. Regnier, The structure of multiplicatives, Arch. Math. Logic 28 (3) (1989) 181–203.

[7] T. Ehrhard, On Köthe sequence spaces and linear logic, Math. Structures Comput. Sci. 12 (2002) 579–623.

[8] T. Ehrhard, Finiteness spaces, Math. Structures Comput. Sci. 15 (4) (2005) 615–646.

[9] T. Ehrhard, L. Regnier, The differential lambda-calculus, Theoret. Comput. Sci. 309 (1–3) (2003) 1–41.

[10] T. Ehrhard, L. Regnier, Uniformity and the Taylor expansion of ordinary lambda-terms, Technical Report, Institut de mathmatiques de Luminy, 2005, accepted for publication.

[11] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1987) 1–102.

[12] J.-Y. Girard, L. Solum, Math. Structures Comput. Sci. 11 (3) (2001) 301–506.

[13] A.J. Kfoury, A linearization of the lambda-calculus, J. Logic Comput. 10 (3) (2000) 411–436.

[14] Y. Lafont, Interaction nets, in: 17th Annu. Symp. on Principles of Programming Languages, ACM Press, San Francisco, CA, 1990, pp. 95–108.

[15] Y. Lafont, From proof nets to interaction nets, in: J.-Y. Girard, Y. Lafont, L. Regnier (Eds.), Advances in Linear Logic, Cambridge University Press, Cambridge, 1995, pp. 225–247, Proc. Workshop on Linear Logic, Ithaca, New York, June 1993.

[16] L. Regnier, Lambda-calcul et réseaux, Thèse de doctorat, Université Paris, 7, January, 1992.

[17] D. Sangiorgi, D. Walker, The Pi-calculus: a Theory of Mobile Processes, Cambridge University Press, Cambridge, 2001.

[18] L. Vaux, The differential lambda-mu calculus, Technical Report, Institut de Mathématiques de Luminy, 2005, submitted for publication.