Stream Associative Nets and Lambda-mu-calculus

Michele Pagani, Alexis Saurin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Stream Associative Nets and $\Lambda\mu$-calculus.

Michele Pagani — Alexis Saurin

N° ????

janvier 2008

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

Stream Associative Nets and $\Lambda\mu$-calculus.

Michele Pagani* , Alexis Saurin†

Thème SYM — Systèmes symboliques
Équipe-Projet PARSIFAL

Rapport de recherche n° ???? — janvier 2008 — 47 pages

Abstract: $\Lambda\mu$-calculus has been built as an untyped extension of Parigot's $\lambda\mu$-calculus in order to recover Böhm theorem which was known to fail in $\lambda\mu$-calculus. An essential computational feature of $\Lambda\mu$-calculus for separation to hold is the unrestricted use of abstractions over continuations that provides the calculus with a construction of streams.

Based on the Curry-Howard paradigm Laurent has defined a translation of $\lambda\mu$-calculus in polarized proof-nets. Unfortunately, this translation cannot be immediately extended to $\Lambda\mu$-calculus: the type system on which it is based freezes $\Lambda\mu$-calculus's stream mechanism.

We introduce *stream associative nets (SANE)*, a notion of nets which is between Laurent's polarized proof-nets and the usual linear logic proof-nets. SANE have two kinds of $\parr$ (hence of $\otimes$), one is linear while the other one allows free structural rules (as in polarized proof-nets). We prove confluence for SANE and give a reduction preserving encoding of $\Lambda\mu$-calculus in SANE, based on a new type system introduced by the second author. It turns out that the stream mechanism at work in $\Lambda\mu$-calculus can be explained by the associativity of the two different kinds of $\parr$ of SANE.

At last, we achieve a Böhm theorem for SANE. This result follows Girard's program to put into the fore the separation as a key property of logic.

Key-words: $\lambda\mu$-calculus, linear logic, Böhm theorem, proof-nets, classical logic, associativity in logic, continuations.

* PPS, CNRS & Université Paris VII – Michele.Pagani@pps.jussieu.fr
† LIX – Parsifal, INRIA & École Polytechnique – Saurin@lix.polytechnique.fr

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

SANE: des réseaux pour le $\Lambda\mu$-calcul.

Résumé : Le $\Lambda\mu$-calcul a été introduit comme une extension non-typée du $\lambda\mu$-calcul de Parigot, re manière à retrouver la propriété de séparation (ou théorème de Böhm) dont on savait qu'elle était fausse en $\lambda\mu$-calcul. Un élément essentiel en $\Lambda\mu$-calcul pour que la séparation soit valide est l'utilisation sans restriction d'abstraction sur les continuations qui donnent au calcul une construction de streams.

Fondé sur le paradigme de Curry-Howard, Olivier Laurent a défini une traduction du $\lambda\mu$-calculus dans les réseaux de preuve polariés. Malheureusement, cette traduction ne peut pas être étendue au $\Lambda\mu$-calculs: le système de typage sur lequel elle est basée désactive le mécanisme de stream du $\Lambda\mu$-calcul.

Nous introduisons les *stream associative nets (SANE)*, une variante de réseaux qui se situe entre les réseaux polarisées de Laurent et les réseaux habituels de la logique linéaire. Les SANE ont deux types de $\parr$ (et donc de $\otimes$): l'un est linéaire tandis que l'autre admet librement des règles structurelles comme dans les réseaux polarisés.

Nous prouvons la confluence pour SANE et présentons une réduction qui préserve l'encodage du $\Lambda\mu$-calcul dans SANE. Cette réduction, fondée sur un nouveau système de typage introduit par le second auteur. On s'aperçoit que le mécanisme de stream à l'œuvre en $\Lambda\mu$-calculs peut être expliqué par l'associativité des deux types de $\parr$ des SANE.

Finalement, on montre un théorème de Böhm pour les SANE. Le résultat suit le programme de Girard visant à donner une place clé à la séparation parmi les propriétés des systèmes logiques.

Mots-clés : $\lambda\mu$-calcul, logique linéaire, théorème de Böhm, réseaux de preuve, logique classique, associativité en logique, continuations.

Stream Associative Nets and Λμ-calculus.

Michele Pagani , Alexis Saurin

Λμ-calculus has been built as an untyped extension of Parigot's λμ-calculus in order to recover Böhm theorem which was known to fail in λμ-calculus. An essential computational feature of Λμ-calculus for separation to hold is the unrestricted use of abstractions over continuations that provides the calculus with a construction of streams.

Based on the Curry-Howard paradigm Laurent has defined a translation of λμ-calculus in polarized proof-nets. Unfortunately, this translation cannot be immediately extended to Λμ-calculus: the type system on which it is based freezes Λμ-calculus's stream mechanism.

We introduce *stream associative nets (SANE)*, a notion of nets which is between Laurent's polarized proof-nets and the usual linear logic proof-nets. SANE have two kinds of ⅋ (hence of ⊗), one is linear while the other one allows free structural rules (as in polarized proof-nets). We prove confluence for SANE and give a reduction preserving encoding of Λμ-calculus in SANE, based on a new type system introduced by the second author. It turns out that the stream mechanism at work in Λμ-calculus can be explained by the associativity of the two different kinds of ⅋ of SANE.

At last, we achieve a Böhm theorem for SANE. This result follows Girard's program to put into the fore the separation as a key property of logic.

1 Introduction

Curry-Howard in classical logic. Curry-Howard correspondence was first designed as a correspondence between simply typed λ-calculus and intuitionnistic logic. Basically, it expresses (i) that a type can be seen as a logical formula, (ii) that a λ-term can be seen as a proof, and (iii) that the execution of a λ-term corresponds to applying the cut-elimination procedure to the associated proof, and conversely. Indeed, this correspondence was at first limited to intuitionistic logic on the one hand and to functional programming (λ-calculus) on the other hand. Extending the correspondence to classical logic resulted in strong connections with control operators in functional programming languages as first noticed by Griffin [Gri90]. In particular, λμ-calculus [Par92] was introduced by Michel Parigot as an extension of λ-calculus isomorphic to an alternative presentation of classical natural deduction (known as free deduction) in which one can encode usual control operators and in particular the `call/cc` operator.

Polarized linear logic. Based on the extension of the Curry-Howard isomorphism to classical logic, Laurent defines a translation of Parigot's λμ-calculus in

*PPS, CNRS & Université Paris VII – Michele.Pagani@pps.jussieu.fr
†LIX – Parsifal, INRIA & École Polytechnique – Saurin@lix.polytechnique.fr

polarized linear logic: a variant of linear logic (LL), allowing free structural rules on negative formulas [Lau02]. Laurent's translation enlarges the comparison between LL and usual λ-calculus, started from Girard [Gir87], Danos [Dan90] and Regnier [Reg92]. In particular polarized LL provides a class of proof-nets (the graph-theoretical representation of LL proofs) corresponding to the $\lambda\mu$-terms, so sheding new ligth into the computation of $\lambda\mu$-calculus.

$\lambda\mu$-calculus and Separation. $\lambda\mu$-calculus became one of the most standard ways to study classical lambda-calculi. As a result, the calculus has been more and more studied and more fundamental questions arose. The best known example of separation result is Böhm's theorem for the pure λ-calculus [B$\ddot{\text{o}}$8]: if t, t' are two distinct closed $\beta\eta$-normal terms, then there exist terms $u_1, \ldots, u_n$, such that $(t)u_1 \ldots u_n \to_\beta x$ and $(t')u_1 \ldots u_n \to_\beta y$. This result has consequences both at the semantical level as well as at the syntactical one: on the one hand it entails that a model of the λ-calculus cannot identify two different $\beta\eta$-normal forms without being trivial; on the other hand it establishes a balance between syntactical constructs and β-reduction: any difference in the structure of a $\beta\eta$-normal form implies a difference in the value of that normal form on suitable arguments. In 2001 David & Py addressed the question of separation to Parigot's $\lambda\mu$-calculus and they gave a negative answer by exhibiting a counter-example [DP01]. In a previous work of 2005, the second author introduced an extension to $\lambda\mu$-calculus, $\Lambda\mu$-calculus, for which he could prove that separation holds [Sau05]. $\Lambda\mu$-calculus is fairly close to standard presentations of $\lambda\mu$-calculus (see [dG94, dG98] for instance), but is definitely a different calculus. In particular, an essential computational feature of $\Lambda\mu$-calculus for separation to hold is the unrestricted use of abstractions over continuations that provides the calculus with a construction of streams.

The logic of $\Lambda\mu$-calculus. We pursue an investigation of the logic behind $\Lambda\mu$-calculus. Our feeling is that the rules of classical logic imposes a too strict discipline over the use of streams: in Parigot's $\lambda\mu$-calculus streams represent only channels through which terms can be sent, these channels can be plugged to each other, they can be exchanged, but they do not really communicate with the terms in the course of a computation. Streams and terms live in different worlds, in particular the former ones are not first class citizens in the early versions of $\lambda\mu$-calculus. We think that the Curry-Howard isomorphism at the base of Parigot's $\lambda\mu$-calculus restricts too much the computational power of streams, a consequence of which is the failure of the separation property, as proved by David & Py. When departing from classical logic and building more freely the programs in $\Lambda\mu$-calculus, one gets back the separation property and in the same time one moves away from classical logic.

Stream Associative NEts: from the rules of classical logic to the logic of $\Lambda\mu$-calculus rules. This turning-point requires also a change of the encoding of $\lambda\mu$-calculus into proof-nets: indeed Laurent's translation is based on the Curry-Howard isomorphism with classical logic. We follow another direction, in order to have an encoding of $\Lambda\mu$-calculus which is more faithfull to the stream behaviour at the base of the separation property. We believe that it is by departing from the rules of classical logic that we will understand the real logic of $\Lambda\mu$-calculus rules.

We thus define a new class of nets, Stream Associative NEts (SANE). SANE lies in between usual linear logic proof-nets and polarized proof-nets: we have two kinds of $\parr$ (and dually of $\otimes$), one coming from LL (associated with the λ-variables) and the other one coming from polarized LL (and associated with the μ-variables). The essential ingredient is the associativity property between these two kinds of multiplicatives, which makes possible the communication between streams and λ-variables much in the same way as *fst* rule does in $\Lambda\mu$-calculus.

Better be in SANE to study $\Lambda\mu$-calculus. The correspondence between $\Lambda\mu$-calculus and SANE will allow for considerable "transfers of technologies" between the two domains, in particular proof-nets will provide powerful geometrical abstractions and a deeply symmetrical framework as well as strong dualities. In addition to a finer-grained study of the reduction rules of $\Lambda\mu$-calculus (as emphasized by our simulation result), SANE reductions will provide $\Lambda\mu$-calculus with a notion of explicit substitution. Moreover, SANE should help studying the relationships of $\Lambda\mu$-calculus with other continuation-based calculi.

Proof-nets with separation property. SANE have been designed in order to study $\Lambda\mu$-calculus, but separation property plays a key role in the theory of SANE. As in Ludics [Gir01] where Girard chose separation to be a requirement for his elementary objects, the designs, the nets we introduce in the present work have been designed with separation property to be at the heart of the theory, much in the same way as confluence does.

Structure of the Paper. The following section is dedicated to a short introduction to Parigot's $\lambda\mu$-calculus to separation related topics and to $\Lambda\mu$-calculus. A new type system for $\Lambda\mu$-calculus is provided which serves as a basis to define, in section 3, the pure Stream associative nets, their reductions, state the correctness criterion for SANE and prove an original strong normalization result of $\rightsquigarrow_{s,r,a}$ which implies the strong normalization of exponential reduction in SANE[1]. The following section is dedicated to proving confluence of SANE before going to the question of the separation property in section 5. Finally, we simulate $\Lambda\mu$-calculus in SANE in section 6.

[1] This gives as a corollary the SN of the implicit explicit substitution system

2 $\Lambda\mu$-calculus

2.1 $\lambda\mu$-calculus, streams and Separation: $\Lambda\mu$-calculus

David & Py counter-example to Separation in $\lambda\mu$-calculus. In their 2001 paper [DP01], David & Py adressed the question of separation property in $\lambda\mu$-calculus by exhibiting a counter-example to separation, the $\lambda\mu$-term $W = \lambda x.\mu\alpha.[\alpha]((x)\ \mu\beta.[\alpha](x)\ U_0\ y)\ U_0$ with $U_0 = \mu\delta.[\alpha]\lambda z_1.\lambda z_2.z_2$. Separation property fails in this setting because there is no way to put the variable y in head position. The key point is that the entire applicative context in which this term is placed is transmitted through $\mu\alpha$ to subterms; as a consequence, the usual technique (which consists in building a context that shall explore the part of the term we want) cannot be applied.

Recovering Separation in $\lambda\mu$-calculus: relaxing implicit (underlying) typing constraints. What we do by introducing $\Lambda\mu$-calculus is precisely to be more liberal with the construction of terms in order to provide the calculus with more applicative contexts and retrieve the ability to realize the needed exploration paths. In particular, Parigot's $\lambda\mu$-calculus syntax has a constraint of naming a term right before it is μ-abstracted (terms have the form $\mu\alpha.[\beta]\_$) which can actually be seen as a typing constraint directly built in the syntax of the untyped calculus. $\Lambda\mu$-calculus is basically the result of removing this constraint. By doing so, we obtain a calculus which is close to de Groote's presentation of $\lambda\mu$-calculus but it is not equivalent to this calculus since de Groote's presentation also contains a typing constraint which is built in the syntax, namely the ϵ rule that is absent from $\Lambda\mu$-calculus[2].

$\Lambda\mu$-calculus was introduced in [Sau05] as an untyped extension of Parigot's $\lambda\mu$-calculus in which separation holds. Given two infinite disjoint sets $\mathcal{V}_t$ (of term variables, denoted by $x, y, z \dots$) and $\mathcal{V}_s$ (of stream variables, denoted by $\alpha, \beta, \gamma \dots$), $\Lambda\mu$-calculus is defined by the following grammar:

$$t, u ... ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.t \mid (t)\alpha$$

An *abstraction* is a term of shape $\lambda x.t$ or $\mu\alpha.t$ and an *application* is a term of shape $(t)u$ or $(t)\alpha$. We refer to the application of an abstraction as a *cut*. There are four kinds of cuts in $\Lambda\mu$-calculus as shown in figure 1: $(\mathcal{T})\mathcal{T}$, $(\mathcal{T})\mathcal{S}$, $(\mathcal{S})\mathcal{T}$, $(\mathcal{S})\mathcal{S}$.

$\Lambda\mu$-calculus reductions.
Cuts of type $(\mathcal{T})\mathcal{T}$ and $(\mathcal{S})\mathcal{S}$ are redexes for the following rules:

$$(\lambda x.t)u \quad \rightarrow_{\beta_T} \quad t[u/x] \tag{1}$$
$$(\mu\alpha.t)\beta \quad \rightarrow_{\beta_S} \quad t[\beta/\alpha] \tag{2}$$

But cuts of type $(\mathcal{S})\mathcal{T}$ and $(\mathcal{T})\mathcal{S}$ are not redexes for these rules.

[2]The result of the ϵ rule in $\Lambda\mu$-calculus would actually be to cancel multiple stream abstractions which would be problematic with respect to separation.

$(\mathcal{T})\mathcal{T} : (\lambda x.t)u$	$(\mathcal{T})\mathcal{S} : (\lambda x.t)\alpha$
$(\mathcal{S})\mathcal{T} : (\mu\alpha.t)u$	$(\mathcal{S})\mathcal{S} : (\mu\alpha.t)\beta$

Figure 1: Cuts in Λμ-calculus.

$$
\begin{array}{ll}
\triangleright\ (\lambda x.t)u & \longrightarrow_{\beta_T} t[u/x] \\
\triangleright\ \lambda x.(t)x & \longrightarrow_{\eta_T} t \\
\triangleright\ (\mu\alpha.t)\beta & \longrightarrow_{\beta_S} t[\beta/\alpha] \\
\triangleright\ \mu\alpha.(t)\alpha & \longrightarrow_{\eta_S} t \\
\triangleright\ \mu\alpha.t & \longrightarrow_{fst} \lambda x.\mu\beta.t[(U)x\beta/(U)\alpha]
\end{array}
$$

Proviso:
In η, fst, $x \notin FV_t(t)$; in η_s, $\alpha \notin FV_s(t)$

Figure 2: Λμ-calculus reduction rules

The following *fst*-rule relates term variables with stream variables, it is a way to access the first term of the stream and it will allow to reduce the last two types of cuts:

$$\mu\alpha.t \quad \rightarrow_{fst} \quad \lambda x.\mu\beta.t[(U)x\beta/(U)\alpha] \tag{3}$$

- Indeed the *fst*-rule makes reducible the cuts of type $(\mathcal{S})\mathcal{T}$:

$$(\mu\alpha.t)u \rightarrow_{fst} (\lambda x.\mu\beta.t[(U)x\beta/(U)\alpha])u \rightarrow_{\beta_T} \mu\beta.t[(U)u\beta/(U)\alpha]$$

- as well as those of type $(\mathcal{T})\mathcal{S}$, whenever subterms of a closed term:

$$\mu\beta.\ldots(\lambda x.t)\beta\cdots \rightarrow_{fst} \lambda x.\mu\beta.\ldots(\lambda x.t)x\beta\cdots \rightarrow_{\beta} \lambda x.\mu\beta.\ldots(t)\beta\ldots$$

The following rules defines extensional equivalences (with the usual proviso $x \notin FV_T(t)$ and $\alpha \notin FV_S(t)$):

$$\lambda x.(t)x \quad \rightarrow_{\eta_T} \quad t \tag{4}$$
$$\mu\alpha.(t)\alpha \quad \rightarrow_{\eta_S} \quad t \tag{5}$$

Λμ-calculus reduction rules are summarized in figure 2.

In Λμ-calculus, μ can be seen as an abstraction over streams of terms[3].
For instance, while $\lambda x.\lambda y.\lambda z.((z)(t)xy)(t')xy$ may duplicate two terms passed

[3] Streams as first-class citizens are consequences of more extensionality in Λμ-calculus than in λμ-calculus, due to the fact that it is possible to use the extensionality rules η and η_s where λμ-calculus syntax forbids to do so, for instance: $\mu\alpha.(t)\beta \rightarrow_{\eta} \mu\alpha.(\lambda x.(t)x)\beta$.

$$\frac{}{\Gamma, x : \mathcal{T} \vdash x \ : \mathcal{T} | \Delta} \ Var_{\mathcal{T}}$$

$$\frac{\Gamma, x : \mathcal{T} \vdash t \ : \mathcal{T}' | \Delta}{\Gamma \vdash \lambda x.t \ : \mathcal{T} \to \mathcal{T}' | \Delta} \ Abs_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ : \mathcal{T} \to \mathcal{T}' | \Delta \qquad \Gamma \vdash u \ : \mathcal{T} | \Delta}{\Gamma \vdash (t)u \ : \mathcal{T}' | \Delta} \ App_{\mathcal{T}}$$

$$\frac{\Gamma \vdash t \ : \bot | \Delta, \alpha : A}{\Gamma \vdash \mu\alpha.t \ : A | \Delta} \ \mu Abs \qquad \frac{\Gamma \vdash t \ : A | \Delta, \alpha : A}{\Gamma \vdash (t)\alpha \ : \bot | \Delta, \alpha : A} \ \mu App$$

Figure 3: $\Lambda\mu$-calculus Classical Type System.

through x and y, $\Lambda\mu$-term $\mu\alpha.\mu\beta.\lambda z.((z)(t)\alpha\beta)(t')\alpha\beta$ can duplicate two *streams* of terms, these streams being for instance applied through the applicative context: $[]t_1 \dots t_k \gamma u_1 \dots u_l \delta$.

Compared to $\lambda\mu$-calculus where the effect of μ is only to redirect the computation flow, in $\Lambda\mu$-calculus, one can manage to deal with streams as first-class citizens: for instance, $\mu\alpha.\mu\beta.\lambda x.\lambda y.x$ is a term that erases two streams of terms and returns the boolean value `true`. As previously said, $\Lambda\mu$-calculus has been designed in order to recover the separation property. The original counter-example to separation by David & Py [DP01], W, is solved by the following $\Lambda\mu$-context: $\mathcal{C} = []\mathcal{P}x_0x_1\alpha 0\alpha 1\alpha$ where $\mathcal{P} = \lambda z_0, z_1.\mu\gamma.\lambda u.((u)\mu\beta.z_1)z_0$: $\mathcal{C}(W) \to^\star y$ (see [Sau05] for more details).

2.2 Typing $\Lambda\mu$-calculus

Typing $\Lambda\mu$-calculus as $\lambda\mu$-calculus. One could think of typing $\Lambda\mu$ using a standard type system for classical lambda-calculi as shown in figure 3. However, this approach is not satisfactory considering our motivations in developping the new calculus, that is from the point of view of separation. Indeed, the main structures used in [Sau05] in order to obtain separation would not be typable in the system of figure 3 and for very fundamental reasons. Any term of the form $\mu\alpha.\lambda x.t$ would be untypable whereas this is the typical term used in the proof of separation for $\Lambda\mu$-calculus. In fact, the typing system originally introduced in order to connect the calculus with free deduction [Par92] precisely forbids such terms: $\lambda x.t$ is a λ-abstracted term and thus shall be of an $\to$-type whereas the fact that it is μ-abstracted through stream variable α forces the term to be of type $\bot$ which is incompatible (see rule μAbs in figure 3).

Making streams first-class citizens in the typed setting. The stream mechanism that was used in the untyped calculus in order to obtain separation is thus desactivated when classical types are reintroduced. We shall look for a variant of this type system that would reflect in types the stream construction. In particular, since μ is seen as a stream abstraction, one might think of a functional type for streams: if the term t is of type $\mathcal{T}$ when stream α is of stream type $\mathcal{S}$, then $\mu\alpha.t$ would be of the type of a stream function from $\mathcal{S}$ to $\mathcal{T}$ (that we write $\mathcal{S} \Rightarrow \mathcal{T}$). We can thus think of the following typing rules for

$$\frac{}{\Gamma, x:\mathcal{T} \vdash x \ :\mathcal{T}|\Delta}\ Var_{\mathcal{T}}$$

$$\frac{\Gamma, x:\mathcal{T} \vdash t \ :\mathcal{T}'|\Delta}{\Gamma \vdash \lambda x.t \ :\mathcal{T} \to \mathcal{T}'|\Delta}\ Abs_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ :\mathcal{T} \to \mathcal{T}'|\Delta \qquad \Gamma \vdash u \ :\mathcal{T}|\Delta}{\Gamma \vdash (t)u \ :\mathcal{T}'|\Delta}\ App_{\mathcal{T}}$$

$$\frac{\Gamma \vdash t \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash \mu\alpha.t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta}\ Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash (t)\alpha \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}\ App_{\mathcal{S}}$$

Figure 4: $\Lambda_{\mathcal{S}}$: a type system for $\Lambda\mu$-calculus.

μ-abstracted terms in $\Lambda\mu$-calculus:

$$\frac{\Gamma \vdash t \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash \mu\alpha.t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta}\ Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash (t)\alpha \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}\ App_{\mathcal{S}}$$

A type mismatch. Rule *fst* does complicate the definition of a type system for $\Lambda\mu$ that would take streams into account: whereas $\mu\alpha.t$ is of a stream type, say $\mathcal{S} \Rightarrow \mathcal{T}$, the term resulting from $\mu\alpha.t$ by applying the *fst* rule once (namely $\lambda x.\mu\beta.t[(U)x\beta/(U)\alpha]$) should be of a standard function type $A \to B$ (more precisely $A \to (\mathcal{S}' \Rightarrow \mathcal{T}')$). Moreover, things should not be as simple as in the previous paragraph since streams are streams of terms and thus they should be related to each other and they should not leave in distinct worlds: one should be allowed to apply a term to a stream function (for instance $(\mu\alpha.t)u$) and conversely, one might want to apply a stream to a λ-abstracted term (for instance $(\lambda x.t)\alpha$). $\Rightarrow$-types and $\to$-types should be related in some way. *fst* gives the key to this connection; we thus analyze more carefully this rule in the following paragraph.

A relation over stream types. We recall that *fst* synthesized in $\Lambda\mu$-calculus as the result of a η-expansion and a μ-reduction. In the typed case, the η-expansion can occur only on $\to$-type terms. This restriction adapted to $\Lambda\mu$-calculus results in the condition that $\mu\alpha.t$ is of a stream type of the form $(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}'$. After an application of *fst*, we have term $\lambda x.\mu\beta.t[(U)x\beta/(U)\alpha]$ that should be of type $\mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$.

Simply typed streams: $\Lambda_{\mathcal{S}}$. We now define more formally the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus. The types are produced by the following grammar of simple types:

Term types:	$\mathcal{T}, A, B, \ldots$	$::= o \mid A \to B \mid \mathcal{S} \Rightarrow \mathcal{T}$
Stream types:	$\mathcal{S}, P, Q, \ldots$	$::= \bot \mid \mathcal{T} \to \mathcal{S}$

In addition, we consider the congruence relation $\equiv_{fst}$ over Term types which is the symmetric, reflexive and transitive closure of relation $\succ_{fst}$ defined by $(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}' \succ_{fst} \mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$ and we always consider the types of $\Lambda_{\mathcal{S}}$ up to this congruence relation. We show in figure 4 the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : \mathcal{S}_\alpha \Rightarrow \mathcal{A} \vdash y \ : \mathcal{S}_\alpha \Rightarrow \mathcal{A}|}\,Var_{\mathcal{T}}}{y : \mathcal{S}_\alpha \Rightarrow \mathcal{A} \vdash (y)\alpha \ : \mathcal{A}|\alpha : \mathcal{S}_\alpha}\,App_{\mathcal{S}}}{y : \mathcal{S}_\alpha \Rightarrow \mathcal{A} \vdash \mu\beta.(y)\alpha \ : \mathcal{S}_\beta \Rightarrow \mathcal{A}|\alpha : \mathcal{S}_\alpha}\,Abs_{\mathcal{S}}}{\vdash \lambda y.\mu\beta.(y)\alpha \ : (\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})|\alpha : \mathcal{S}_\alpha}\,Abs_{\mathcal{T}} \quad \dfrac{}{x : \mathcal{T}_x \vdash x \ : \mathcal{T}_x|}\,Var_{\mathcal{T}}}{x : ((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B}) \vdash (x)\lambda y.\mu\beta.(y)\alpha \ : \mathcal{S}_\alpha \Rightarrow \mathcal{B}|\alpha : \mathcal{S}_\alpha}\,App_{\mathcal{T}}}{x : ((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B}) \vdash ((x)\lambda y.\mu\beta.(y)\alpha)\alpha \ : \mathcal{B}|\alpha : \mathcal{S}_\alpha}\,App_{\mathcal{S}}}{x : ((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B}) \vdash \mu\alpha.((x)\lambda y.\mu\beta.(y)\alpha)\alpha \ : \mathcal{S}_\alpha \Rightarrow \mathcal{B}|}\,Abs_{\mathcal{S}}}{\vdash \texttt{call/cc} \ : (((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B})|}\,Abs_{\mathcal{T}}$$

with $\mathcal{T}_x = ((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B})$.

Figure 5: $\Lambda_{\mathcal{S}}$ type derivation for `call/cc`.

In the typed case, application of the *fst* rule to term t requires that t has a type in relation with a type of shape $(\mathcal{T}_1 \to \mathcal{S}) \Rightarrow \mathcal{T}_2$ (this requirement is similar to the condition on the η-expansion application in simply typed λ-calculus).

Moving from $\Rightarrow$ to $\parr$. Contrarily to what the notation $\Rightarrow$ may suggest, no duality is involved with this connective. The rule $Abs_{\mathcal{S}}$ would rather suggest the $\Rightarrow$ connective to be related with the $\parr$ connective of linear logic. This is precisely what we evidence in the present work: when translating Λ_S into (a kind of) polarized proof nets, $T_1 \to T_2$ becomes as usual $?T_1^{\perp} \parr T_2$ while $S \Rightarrow T$ is translated into $S \parr T$. The $\equiv_{fst}$ is thus an associativity property (namely $(?T^{\perp} \parr S) \parr T \equiv_{fst} ?T^{\perp} \parr (S \parr T)$) of $\parr$ which is perfectly sound logically.

Typing `call/cc` in $\Lambda_{\mathcal{S}}$. The $\Lambda\mu$-calculus encoding of `call/cc` is the term $\lambda x.\mu\alpha.((x)\lambda y.\mu\beta.(y)\alpha)\alpha$. In the classical type system presented in figure 3, this term is typed by the Peirce's Law: $((A \to B) \to A) \to A$. In $\Lambda_{\mathcal{S}}$, `call/cc` can be assigned type $(((\mathcal{S}_\alpha \Rightarrow \mathcal{A}) \to (\mathcal{S}_\beta \Rightarrow \mathcal{A})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B})) \to (\mathcal{S}_\alpha \Rightarrow \mathcal{B})$ as shown by the type derivation in figure 5. One may notice that the structure of the the Peirce's Law is now to be found in the stream type (see the alternation of $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$ types).

3 Stream associative nets

Formulas. In order to embed Λμ-calculus is proof nets, we define the following fragment of linear logic formulas, designed thanks to type system $\Lambda_{\mathcal{S}}$:

$$\begin{array}{lllllll} \mathcal{T}\text{-formulas} & T, A, B, \ldots & := & o & | & ?Q \parr T & | & S \parr T \\ & Q, D, E, \ldots & := & o^\perp & | & !T \otimes Q & | & P \otimes Q \\ \\ \mathcal{S}\text{-formulas} & S, M, N, \ldots & := & s & | & ?Q \parr S & & \\ & P, R, U, \ldots & := & s^\perp & | & !T \otimes S & & \end{array}$$

Since we want to consider pure Λμ-calculus, we introduce the following recursive equations (in the same spirit as pure-nets [Dan90], [Reg92]):

$$o = ?o^\perp \parr o \qquad o = s \parr o \qquad s = ?o^\perp \parr s$$

$$o^\perp = !o \otimes o^\perp \qquad o^\perp = s^\perp \otimes o^\perp \qquad s^\perp = !o \otimes s^\perp$$

This gives exactly three pairs of dual formulas:

negatives:	o	$?o^\perp$	s
positives:	$o^\perp$	$!o$	$s^\perp$

We shall see that the formulas $o, o^\perp, !o, ?o^\perp$ behave as in Danos and Regnier's pure proof-nets (i.e. $o, o^\perp$ are linear and $!o, ?o^\perp$ manage duplication and erasing), but the formulas $s, s^\perp$ are of a different nature and they behave as in polarized linear logic (i.e. $s, s^\perp$ allow free structural rules on negatives). The formulas $s, s^\perp$ will be used to type streams.[4]

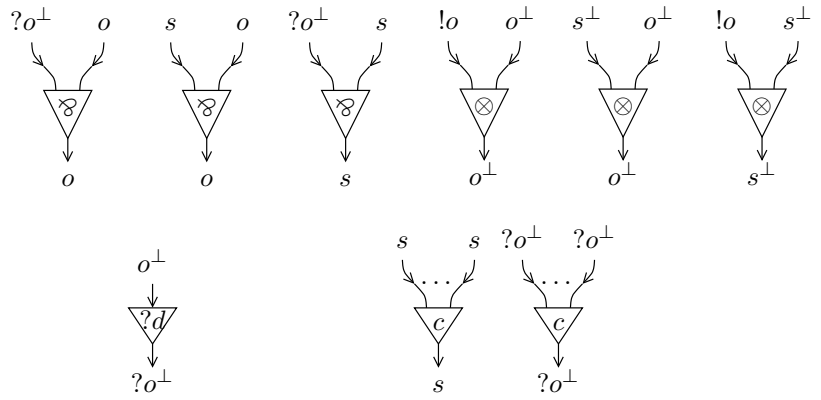
In the sequel we will use n to denote without distinction occurences of s or $?o^\perp$.

SANE. *Stream associative nets*, or simply *nets*, are made of *cells* and *wires*. Each cell has a *type*, which is a symbol belonging to the set $\{\otimes, \parr, c, !, ?d\}$, and a number of *ports*, exactly one of which is called *principal*, or *conclusion*, while the others (if any) are called *auxiliary*, or *premises*. Cells are pictured as triangles, and ports are drawn on the border of these triangles: the principal port of a cell is seen as one of the "tips" of the triangle representing it.

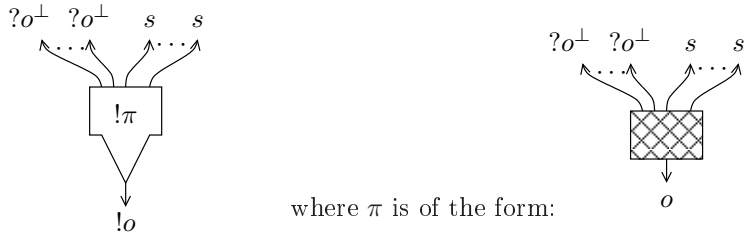
A net is a combination of cells, connected with each other by *wires*, as described in [Laf95]. More precisely, any net has a finite set of *free ports*, also called *conclusions of the net*, and has therefore a set of *ports* made of its free ports and of the ports of its cells (these sets of ports are assumed to be pairwise disjoint). The *wiring* of the net can be seen as a partition of this set of ports into sets of cardinality 2 or 0 (these latter wires are loops, they can appear during the reduction of a net).

The cells are given in figure 6, together with their typing rules. The generalized contraction has a variable arity $n \geq 0$, in case $n = 0$ it corresponds to the usual weakening rule, in case $n = 1$ we adopt the convention to consider it a simple wire, if $n > 1$ then it corresponds to a tree of usual binary contraction rules

[4] One could be tempted to set the equation $?o^\perp = s$, so reducing to only two pairs of formulas: $o, o^\perp$ and $s, s^\perp$ – in this manner however it would be allowed clashes (i.e. bad typed cuts), such as a cut between a stream $\parr$ and a promotion.



(i) generalized contraction



(j) promotion cell

Figure 6: Cells of SANE

modulo associativity. The promotion cell is a special kind of cell parametrized by a net: if π is a net with $n+1$ free ports, then $!\pi$ is a cell with one principal port and n auxiliary ports. The net π may itself contain promotion cells. The *(exponential) depth* of a net π is the maximum number of nested promotion cells in π[5]; one defines as well exponential depth of a cell or a port in a given net Π. Sometimes we will picture the net associated with a promotion cell inside the cell itself, as for example in figure 9.

A *typing* of a net π is mapping from the *oriented wires*[6] of π to the formulae $o, o^\perp, s, s^\perp, ?o^\perp, !o$, in such a way that the constraints of figure 6 be satisfied and in such a way that, if an oriented wire w is mapped to a formula A, then the oriented wire w' obtained by reversing the orientation of w be mapped to $A^\perp$.

Each free port of π is equipped with a type: the type associated to the wire connected to this port, this wire being considered as oriented towards the port under consideration. An *axiom* is a wire between two ports which are auxiliary (but not of a promotion cell) or free, a *cut* is a wire between two ports which are principal or auxiliary[7] of a promotion cell.

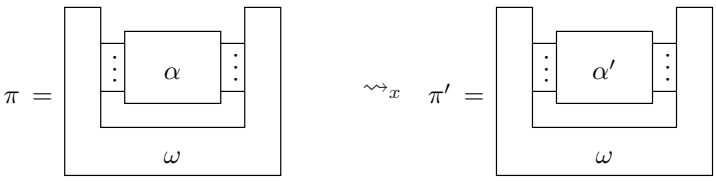
[5] To be pedantic, one should define nets by induction on the depth.

[6] An oriented wire is a wire equipped with an orientation, that is, an ordered pair of its ending ports.

[7] A cut with one extremity auxiliary port of a promotion cell is sometimes called *commutative exponential*.

3.1 Rewriting rules

A rewriting rule $\rightsquigarrow_x$ on nets is a graph transformation $\pi \rightsquigarrow_x \pi'$, consisting in taking a subnet α of π, called *redex*, and substituting it with a net α', called *contractum*, which has the same (number and type of) free ports of α:



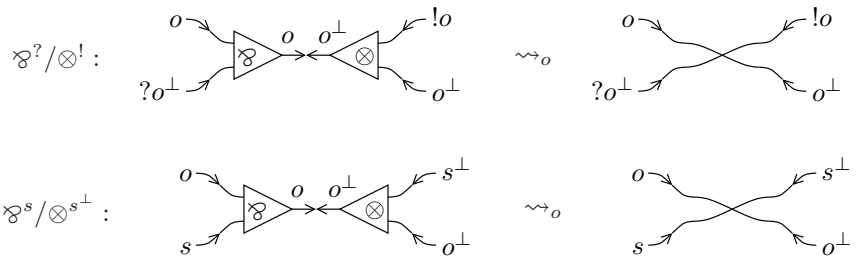
In the sequel we define the rewriting rules just by picturing their redexes and contractos (i.e. without mentioning the "context" ω).

If $\rightsquigarrow_x$ is confluent and normalizing then we denote with $\mathsf{NF}^x(\pi)$ the unique and always defined $\rightsquigarrow_x$-normal form of π.

The rewriting rule we will study in this paper is denoted by $\rightsquigarrow_{SANE}$ and it is the union of four more specific rules: the *cut-reduction* $\rightsquigarrow_{cut}$, the *Retoré reduction* $\rightsquigarrow_r$, the *wire expansion* $\rightsquigarrow_w$ and the *associativity reduction* $\rightsquigarrow_a$. These last two are the keystone of SANE, the ones which make stream and exponential to communicate.

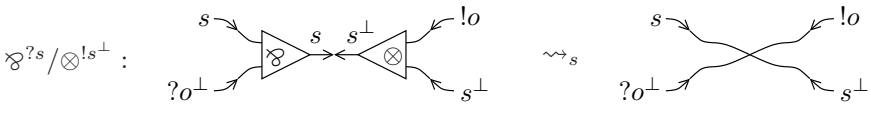
Cut-reduction. We start by recalling cut-reduction $\rightsquigarrow_{cut}$, which is the usual one of polarized linear logic (see [Lau03]). We set $\rightsquigarrow_{cut}$ as the union of two reductions $\rightsquigarrow_o$ and $\rightsquigarrow_s$, defined below.

The relation $\rightsquigarrow_o$ reduces cuts labelled by a formula o. It is defined by the following steps:



Notice that the above rules do not reduce two kinds of o labelled cuts, pictured in figure 7: these cuts play a crucial role in SANE, since they set a communication between ports labelled by exponential formulas and ports labelled by stream formulas. This communication is indeed only potential: so far those cuts stay irreducible, the two kinds of ports cannot be wired.

The relation $\rightsquigarrow_s$ reduces cuts labelled by a formula s or $?o^\perp$. It is defined by four kinds of steps: $⅋^{?s}/⊗^{!s^\perp}$, $!/?d$, $\mathcal{S}/!$ and $\mathcal{S}/c$. The first two are as follows:



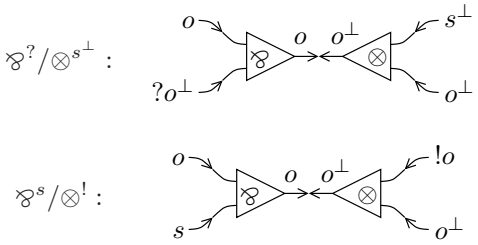


Figure 7: irreducible cuts

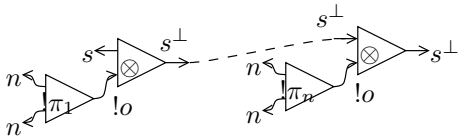
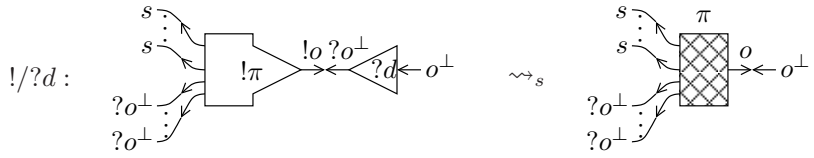


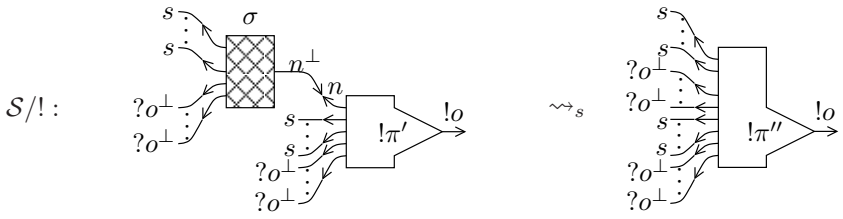
Figure 8: A generic $\otimes^{!s^\perp}$-tree



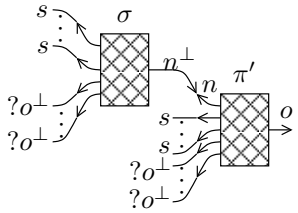
Notice that the step $!/?d$ switches the polarities of the reduced cut, i.e. it creates a wire where the positive and negative extremities are switched.

In order to define the remained steps $\mathcal{S}/!$ and $\mathcal{S}/c$, we need to introduce the notion of $\otimes^{!s^\perp}$-tree (which is a straigth adjustement to our framework of the $\otimes$-tree defined in [Lau03]). A $\otimes^{!s^\perp}$*-tree* is a connected and acyclic net which contains (at depth 0) only cells of type $\otimes^{!s^\perp}$ or promotion and which has no cuts. Note that any wire of a $\otimes^{!s^\perp}$-tree is typable only by n (or equivalently $n^\perp$) and not by $o, o^\perp$. Given a port p of type $n^\perp$ of a net π, we call *the* $\otimes^{!s^\perp}$*-tree of* p the maximal $\otimes^{!s^\perp}$-tree which is a subnet of π and which has p as a free port. One can prove by induction on the size of a $\otimes^{!s^\perp}$-tree σ that the conclusions of σ are exaclty one of type $n^\perp$, called the *root* of σ, and $m \geq 0$ of negative types $?o$ or s, called the *leaves* of σ. The general shape of a $\otimes^{!s^\perp}$-tree is pictured in figure 8.

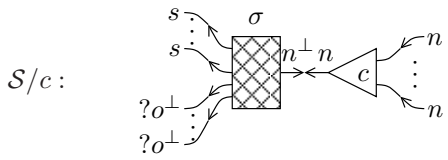
Let now $\langle p, q\rangle$ be a cut labelled by a formula n ($n \in \{s, ?o^\perp\}$), let p be its extremity labelled by $n^\perp$, σ be the $\otimes^{!s^\perp}$-tree of p. The cut $\langle p, q\rangle$ can be of two types depending on q: if q is an auxiliary port of a promotion, then we say that $\langle p, q\rangle$ has type $\mathcal{S}/!$, and we reduce it as follows:

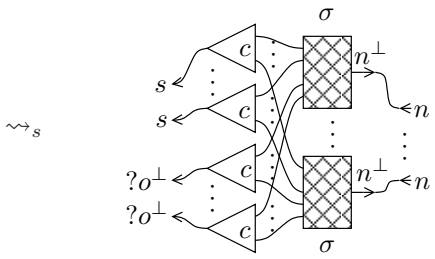


where π'' is:

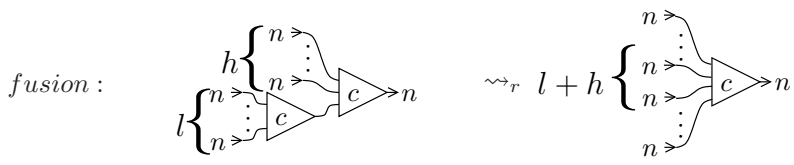


If q is a principal port of a contraction, then we say that $\langle p, q\rangle$ has type $\mathcal{S}/c$, and we reduce it as follows:

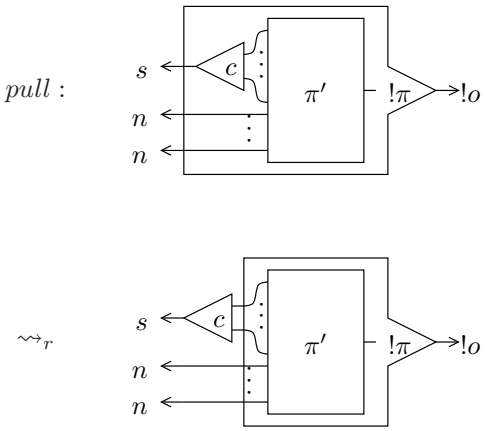




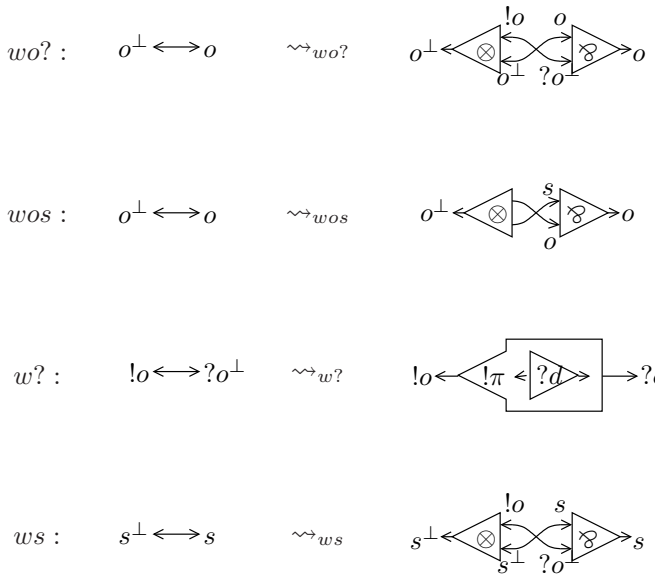
Retoré reduction. The rule $\rightsquigarrow_r$ is defined by two steps, *fusion* and *pull*: it essentially amounts to consider the contraction links as associative operators that can float freely out of a promotion cell. Different solutions have been adopted for linear logic proof nets, see for example [CK97] or [CG99].



where if $l = 0, h = 1$, then on the right-side we have a contraction cell of arity 1, that is a wire.

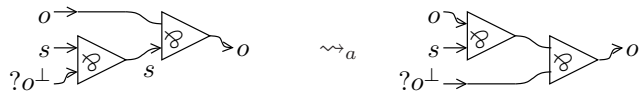


Wire expansions. The wire expansion $\rightsquigarrow_w$ corresponds to an orientation of the extensional equivalence. We define $\rightsquigarrow_w$ by four rules $wo?$, wos, $w?$ and ws:



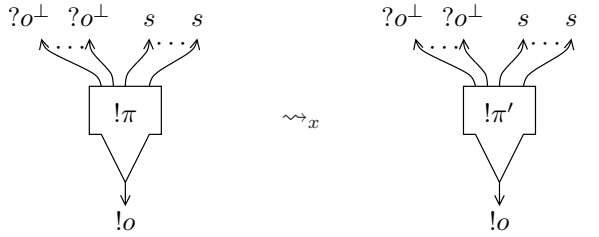
We underline that $\rightsquigarrow_w$ can be applied on every wire of type o or s, and not only on axioms as sometimes it is the case (for example in [Dan90]). Observe that ws is the only step of $\rightsquigarrow_{SANE}$ that creates cells of type $⅋^{?s}$ and $⊗^{!s^\perp}$.

Associativity reduction. Until now, everything is quite standard, following the lines of polarized proof-nets ([Lau03]). Here we introduce the real novelty of SANE, which is the rewriting rule $\rightsquigarrow_a$, based on the associativity between ⅋'s (and dually between ⊗'s). This rule corresponds to the relation $\succ_{fst}$ of the type system $\Lambda_{\mathcal{S}}$ defined in section 2:



The same holds for the corresponding cells of type ⊗.

Promotion closure. The presence of promotion cells requires to close every rewriting rule until now defined to the promotion cells. That is, to every rule $\rightsquigarrow_x$, we add the following case:

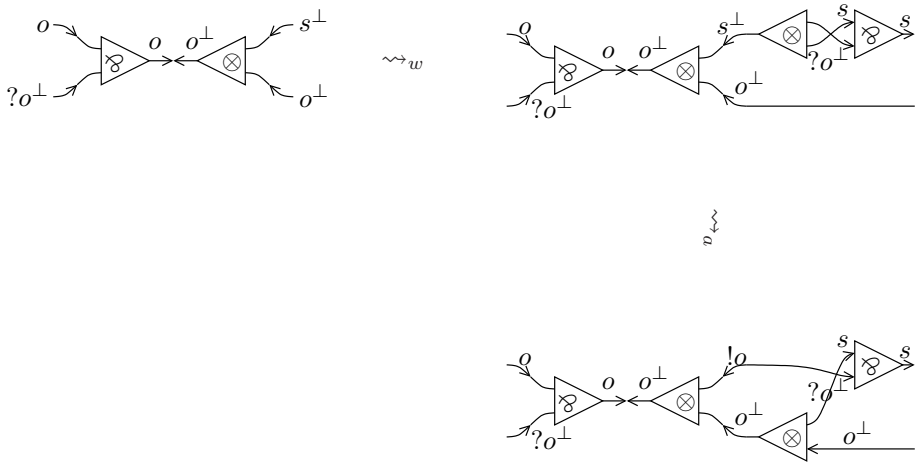


where $\pi \rightsquigarrow_x \pi'$.

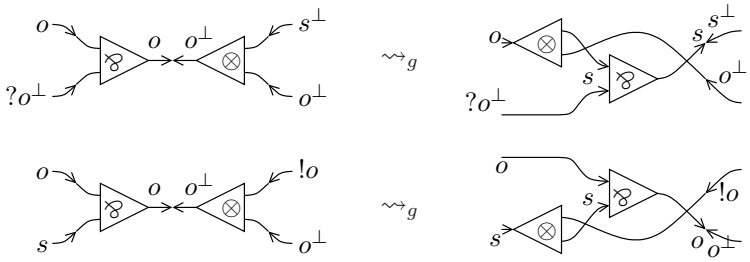
Commutativity equivalence. Moreover, we add an equivalence $\sim_{comm}$ on nets generating by the following basic equation, for every permutation σ:

Notice that this equation does not interact with the previous rewriting relations: let π_1, π_1' and π_2 be nets such that $\pi_1 \sim_{comm} \pi_2$ and $\pi_1 \rightsquigarrow_{SANE} \pi_1'$, then there is a net π_2' s.t. $\pi_1' \sim_{comm} \pi_2'$ and $\pi_2 \rightsquigarrow_{SANE} \pi_2'$. This means that $\rightsquigarrow_{SANE}$ is compatible with the $\sim_{comm}$ equivalence: from now on we will thus consider nets up to $\sim_{comm}$.

Concluding remark on associativity. The key point with the associativity reduction is that $\rightsquigarrow_a + \rightsquigarrow_{ws}$ transforms the irreducible cuts of figure 7 to reducible ones, for example:



In the sequel it will be usefull to consider the following derived rule $\rightsquigarrow_g$:



Notice that $\rightsquigarrow_g$ is derivable from $\rightsquigarrow_{SANE}$, precisely $\rightsquigarrow_g = \rightsquigarrow_w \rightsquigarrow_a \rightsquigarrow_o$. Therefore $\rightsquigarrow_{SANE} + \rightsquigarrow_g$ has the same transitive closure as $\rightsquigarrow_{SANE}$.

3.2 Correctness criterion

A *path* in a net is a sequence of ports $\langle p_1, \ldots, p_n \rangle$ at exponential depth 0, such that:

- for every $i, j \leq n$, $i \neq j$ implies $p_i \neq p_j$;
- for every $i \leq n-1$, p_i, p_{i+1} are ports of the same cell or of the same wire;
- for every $i \leq n-2$, p_i, p_{i+1}, p_{i+2} are not ports of the same cell;

Paths will be denoted by greek letters $\phi, \psi \ldots$

A path ϕ *crosses* an oriented wire $\langle p, q \rangle$ (p, q being the two ports of the wire)) if $\langle p, q \rangle$ is a subsequence of ϕ. A path ϕ is *negative* whenever every oriented wire crossed by ϕ has a negative type (i.e. type o, $?o^\perp$ or s). A *negative cycle* is a negative path $\langle p_1, \ldots, p_n \rangle$ s.t. $\langle p_{1+_n 1}, \ldots, p_{n+_n 1}, \rangle$ is a negative path (where $+_n$ denotes the sum modulo n).

Definition 1 (Correctness, [Lau03]) *A net is* correct *if:*

- *it does not contain negative cycles;*

- *the number of positive conclusions plus ?d-cells at depth* 0 *is one;*
- *and recursively the nets associated with the promotion cells are correct.*

Proposition 2 (Stability of correctness) *Correctness of nets is preserved by* $\rightsquigarrow_{SANE}$*: for every correct net* π*, if* $\pi \rightsquigarrow_{SANE} \pi'$ *then* π' *is correct.*

Proof. Completely standard, see [Lau03]. ■

The following proposition 3 states that $\rightsquigarrow_{s,r,a}$ is strong normalizing on correct nets. This property is false for generalized MELL pure nets (see [PTdF07]) but it holds for pure SANE (as well as for the pure nets fragment encoding the λ-calculus), because of the recursive types we have defined, specifically because we avoid formulas of type $!n$ or $?n^{\perp}$, for $n \in \{s, ?o^{\perp}\}$. Proposition 3 will play a crucial role in the proof of $\rightsquigarrow_{SANE}$ confluence (theorem 10, precisely see lemma 6), and in the proof of the simulation theorem (theorem 26).

Proposition 3 (SN of $\rightsquigarrow_{s,r,a}$**)** *Let* π *be a correct net, every sequence of* $\rightsquigarrow_{s,r,a}$*-steps starting from* π *is finite.*

Proof. Under the hypothesis that π is correct, we define a degree[8] of π and we check that this degree decreases (w.r.t. a well-founded ordering) after a $\rightsquigarrow_{s,r,a}$-step: this of course proves proposition 3. More precisely, the degree of π will be a triplet of multisets $\langle |\pi|_{cut}, s(\pi), c(\pi)\rangle$: the multiset $|\pi|_{cut}$ plays the most delicate role, decreasing under $\rightsquigarrow_s$; the other two multisets $s(\pi)$ and $c(\pi)$ instead decrease under $\rightsquigarrow_a$ and $\rightsquigarrow_r$ respectively. We consider multisets ordered by the multiset order, and triplets ordered lexicographically.

We start defining $|\pi|_{cut}$: to achieve this goal we introduce a partial order $\geq_{n^{\perp}}$ on ports, which is the keystone of $|\pi|_{cut}$ definition, and two numbers $\#^{\pi}(p), l^{\pi}(p)$ associated with every port at depth 0 of π.

An $n^{\perp}$*-path* is a path which crosses only wires of type $n^{\perp}$. Given two ports p, q, we write $p \geq_{n^{\perp}} q$ whenever there is an $n^{\perp}$-path from p to q. Notice that $\geq_{n^{\perp}}$ is a partial order when π is correct. Given a port p we define the set $Pred(p)$ of the *immediate predecessors of* p as the set of those ports q, s.t. $p >_{n^{\perp}} q$ and there is no q', $p >_{n^{\perp}} q' >_{n^{\perp}} q$.

For every port at depth 0 of π, we simultaneously define $l^{\pi}(p)$ and $\#^{\pi}(p)$, by induction on the depth of π and on $\geq_{n^{\perp}}$, which is a well-order:

$$(\#^{\pi}(p), l^{\pi}(p)) = \begin{cases} (1,1) & -\text{If } Pred(p) = \emptyset. \\ (\#^{\rho}(p^{\rho})\#^{\pi}(q), \\ l^{\rho}(p^{\rho}) + l^{\pi}(q) + 1) & \begin{array}{l}-\text{If } p \text{ is an auxiliary port of a} \\ \text{promotion } !\rho \text{ with principal} \\ \text{port } q, \text{ and } p^{\rho} \text{ is the free port} \\ \text{of } \rho \text{ corresponding to } p.\end{array} \\ (\sum_{q \in Pred(p)} \#^{\pi}(q), \\ 1 + max_{q \in Pred(p)}\{l^{\pi}(q)\}) & -\text{Otherwise.} \end{cases}$$

Often we will simply write $l(p)$ or $\#(p)$, when it is clear which net π they refer to. As the reader will convince himself in the progress of the proof, for

[8] This degree is indeed very general and can be adapted to other net-based systems, such as linear logic proof-nets (see [PTdF07]) and differential interaction nets (see [Pag07], [Tra07]). Its definition is the result of several discussions of the first author with Paolo Tranquilli.

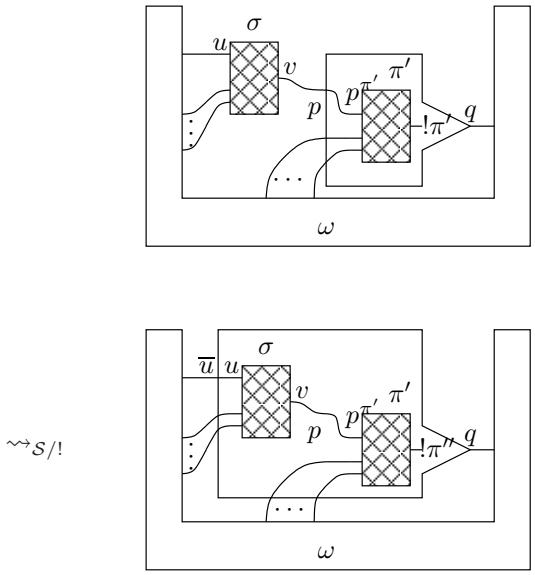


Figure 9: Reduction $\pi \rightsquigarrow_{\mathcal{S}/!} \overline{\pi}$

every port p of type $n^{\perp}$, $\#^{\pi}(p)$ is a maximum to the number of times p can be duplicated by a sequence of $\rightsquigarrow_{s,r}$ steps starting from π. Notice that if p is a port of an $\otimes^{!s^{\perp}}$-tree of π with root r, then either p is a leaf, or $\#^{\pi}(p) = \#^{\pi}(r)$.

We now define $|\pi|_{cut}$ by induction on the depth of π. Let $\mathcal{S}(\pi)$ (resp. $!(\pi)$) be the set of roots of the maximal $\otimes^{!s^{\perp}}$-trees (resp. of promotions) at depth 0 of π; for every promotion $!\rho \in !(\pi)$ let $p_{!\rho}$ be the principal port of $!\rho$; we set:

$$|\pi|_{cut} = \sum_{r \in \mathcal{S}(\pi)} [l(r)] + \sum_{!\rho \in !(\pi)} \#(p_{!\rho}) \cdot |\rho|_{cut}$$

As for $\rightsquigarrow_a$ and $\rightsquigarrow_r$: we define $s(\pi)$ to be the number of cell of $\parr^{s?}$ or $\otimes^{!s^{\perp}}$ in π and we denote by $c(\pi)$ the multiset of the depths of the cells of type c in π. Finally we define:

$$|\pi| = \langle |\pi|_{cut}, s(\pi), c(\pi) \rangle$$

We thus prove that $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ implies $|\overline{\pi}| < |\pi|$. The proof is by induction on the depth of the net π. What we exactly prove is: i) $|\overline{\pi}| < |\pi|$, ii) for every free-port of π, $l^{\pi}(p) = l^{\overline{\pi}}(p), \#^{\pi}(p) = \#^{\overline{\pi}}(p)$.

Base of induction. If the step $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ is not a promotion closure, i.e. the redex reduced is not in a promotion of π, then we split in several cases, depending on the type of the $\rightsquigarrow_{s,r,a}$ step applied to π.

Case $\mathcal{S}/!$. If the step $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ reduces a cut $\langle v, p \rangle$ of type $\mathcal{S}/!$, let v be the root of the maximal $\otimes^{!s^{\perp}}$-tree σ involved in the reduction, p be the auxiliary port of the promotion $!\pi'$ wired to v, and let q be the principal port of $!\pi'$ (see figure 9). We will prove that $|\overline{\pi}|_{cut} < |\pi|_{cut}$, which implies $|\overline{\pi}| < |\pi|$.

We set $\mathcal{S}(\pi) = \{v\} \cup \mathcal{S}^{\omega}(\pi)$ (notice every root in $\mathcal{S}^{\omega}(\pi)$ is a root in ω or it is q) and we split $!(\pi)$ in three disjoint disjoint sets: the set $!^{\sigma}(\pi)$ of the $!(\pi)$ promotions which are in σ, the set $!^{\omega}(\pi)$ of the $!(\pi)$ promotions which are in ω and $\{!\pi'\}$. Observe that:

$$\begin{array}{rcl} \mathcal{S}(\pi) & = & \{v\} \cup \mathcal{S}^{\omega}(\pi) \\ \mathcal{S}(\overline{\pi}) & = & \mathcal{S}^{\omega}(\pi) \end{array}$$

$$\begin{array}{rcl} !(\pi) & = & \{!\pi'\} \cup !^{\sigma}(\pi) \cup !^{\omega}(\pi) \\ !(\overline{\pi}) & = & \{!\pi''\} \cup !^{\omega}(\pi) \end{array}$$

We start computing $|\pi''|_{cut}$. As an easy consequence of the definition we have: $|\pi''|_{cut} = |\pi'|_{cut} + \left[l^{\pi''}(v)\right] + \sum_{!\rho \in !^{\sigma}(\pi)} \#^{\pi''}(p_{!\rho})|\rho|_{cut}$. For every $!\rho \in !^{\sigma}(\pi)$, v is the root of the (maximal) $\otimes^{!s^{\perp}}$-tree containing $p_{!\rho}$, so (as we noticed before) $\#^{\pi''}(p_{!\rho}) = \#^{\pi''}(v)$. Moreover, remark that $\#^{\pi''}(v) = \#^{\pi'}(p^{\pi'})$, thus:

$$|\pi''|_{cut} = |\pi'|_{cut} + \left[l^{\pi''}(v)\right] + \#^{\pi'}(p^{\pi'}) \left(\sum_{!\rho \in !^{\sigma}(\pi)} |\rho|_{cut} \right)$$

Now we prove that for every port t in ω, $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$. In fact, for every leaf u of σ, let $\overline{u}$ be the corresponding auxiliary port of $!\pi''$ in $\overline{\pi}$ (see figure 9): we prove that $\#^{\overline{\pi}}(\overline{u}) = \#^{\pi}(u)$. This easily implies $\forall t \in \omega$, $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$. By definition $\#^{\overline{\pi}}(\overline{u}) = \#^{\pi''}(u)\#^{\overline{\pi}}(q)$; of course, $\#^{\overline{\pi}}(q) = \#^{\pi}(q)$, since no predecessor of q is involved in the reduction step $\pi \rightsquigarrow_s \overline{\pi}$. As for $\#^{\pi''}(u)$, we set $x = \#^{\gamma}(u^{\gamma})$ if u is an auxiliary port of a promotion $!\gamma$, otherwise let $x = 1$: in this way we can say $\#^{\pi''}(u) = x\#^{\pi''}(v)$. Since $\#^{\pi''}(v)\#^{\pi'}(p^{\pi'})$, we have: $\#^{\pi''}(u) = x\#^{\pi'}(p^{\pi'})$. To sum up: $\#^{\overline{\pi}}(\overline{u}) = \#^{\pi''}(u)\#^{\overline{\pi}}(q) = x\#^{\pi'}(p^{\pi'})\#^{\pi}(q) = x\#^{\pi}(p) = \#^{\pi}(u)$.

In a similar way, we prove that for every port t in ω, $l^{\overline{\pi}}(t) = l^{\pi}(t)$. This reduces to verify $l^{\overline{\pi}}(\overline{u}) = l^{\pi}(u)$. By definition $l^{\overline{\pi}}(\overline{u}) = l^{\pi''}(u) + l^{\overline{\pi}}(q) + 1$. Set $l^{\pi''}(u) = x + l^{\pi''}(p^{\pi'})$, where x only depends on the $\otimes^{!s^{\perp}}$-tree σ; notice that $l^{\pi''}(p^{\pi'}) = l^{\pi'}(p^{\pi'})$. So we have: $l^{\overline{\pi}}(u) = l^{\pi''}(u) + l^{\overline{\pi}}(q) + 1 = x + l^{\pi'}(p^{\pi'}) + l^{\overline{\pi}}(q) + 1 = x + l^{\pi}(p) = l^{\pi}(u)$.

I changed the two occurrences of inequality: $\#^{\pi}(q)\left[l^{\pi''}(v)\right] < l^{\pi}(v)$ into $\#^{\pi}(q)\left[l^{\pi''}(v)\right] < [l^{\pi}(v)]$.]

We conclude that condition ii) holds: for every free port t of π, $l^{\overline{\pi}}(t) = l^{\pi}(t)$, $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$. As for the condition i): by collecting all the results, the reader can check that the inequality $|\overline{\pi}|_{cut} < |\pi|_{cut}$ can be reduced to $\#^{\pi}(q)\left[l^{\pi''}(v)\right] < [l^{\pi}(v)]$. By definition of l, we have: $l^{\pi}(v) = 1 + l^{\pi}(p) = 2 + l^{\pi'}(p^{\pi'}) + l^{\pi}(q) = 1 + l^{\pi''}(v) + l^{\pi}(q)$, i.e $l^{\pi''}(v) < l^{\pi}(v)$. This implies, by definition of the multiset order: $\#^{\pi}(q)\left[l^{\pi''}(v)\right] < [l^{\pi}(v)]$. We conclude $|\overline{\pi}|_{cut} < |\pi|_{cut}$, which implies $|\overline{\pi}| < |\pi|$.

Case $\mathcal{S}/c$. If the step $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ reduces a cut $\langle v, p\rangle$ of type $\mathcal{S}/c$, let v be the root of the maximal $\otimes^{!s^{\perp}}$-tree σ involved in the reduction, let p be the principal port of the cell of type c wired to v, and $p_1, \ldots, p_n$ ($n = 0$ or $n > 1$)

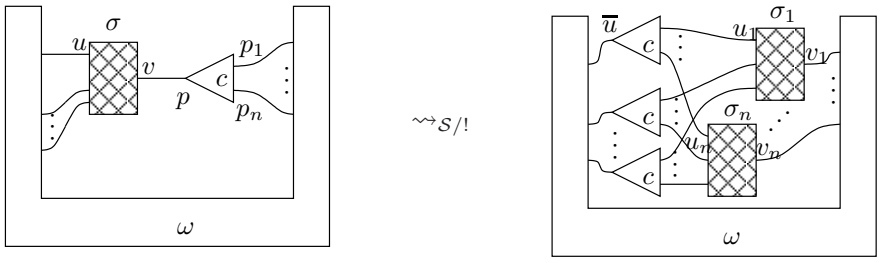


Figure 10: Reduction $\pi \rightsquigarrow_{\mathcal{S}/c} \overline{\pi}$

be its auxiliary ports. Moreover let $\sigma_1, \ldots, \sigma_n$ (resp. $v_1, \ldots, v_n$) be the copies of σ (resp. of the root v) in $\overline{\pi}$ (see figure 10). As in the former case we shall show that $|\overline{\pi}|_{cut} < |\pi|_{cut}$, which implies $|\overline{\pi}| < |\pi|$.

We set $\mathcal{S}(\pi) = \{v\} \cup \mathcal{S}^{\omega}(\pi)$, and we split $!(\pi)$ in two disjoint sets: the set $!^{\sigma}(\pi)$ of the $!(\pi)$ promotions which are in σ, and the set $!^{\omega}(\pi)$ of the $!(\pi)$ promotions which are in ω. Notice that:

$$\begin{array}{rcl} \mathcal{S}(\pi) & = & \{v\} \cup \mathcal{S}^{\omega}(\pi) \\ \mathcal{S}(\overline{\pi}) & = & \{v_1, \ldots, v_n\} \cup \mathcal{S}^{\omega}(\pi) \end{array}$$

$$\begin{array}{rcl} !(\pi) & = & !^{\sigma}(\pi) \cup !^{\omega}(\pi) \\ !(\overline{\pi}) & = & !^{\sigma_1}(\overline{\pi}) \cup \cdots \cup !^{\sigma_n}(\overline{\pi}) \cup !^{\omega}(\pi) \end{array}$$

where $!^{\sigma_i}(\overline{\pi})$ denotes the set of the $!(\overline{\pi})$ promotions which are in the i-th copy of σ.

We start noticing that $\#^{\pi}(v) = \#^{\pi}(p_1) + \cdots + \#^{\pi}(p_n) = \#^{\overline{\pi}}(v_1) + \cdots + \#^{\overline{\pi}}(v_n)$, and $l^{\pi}(v) > l^{\overline{\pi}}(v_i)$ for every $i \leq n$.

As in the former case, we prove that for every port t in ω, $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$ as well as $l^{\overline{\pi}}(t) = l^{\pi}(t)$. We check only $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$, the other verification being a straightforward variant. Indeed it is enough to prove that: for every leaf u of σ, by denoting with $\overline{u}$ the corresponding principal port of the cell c created in $\overline{\pi}$ (see figure 10), we have $\#^{\overline{\pi}}(\overline{u}) = \#^{\pi}(u)$. By definition one easily infers that: $\#^{\overline{\pi}}(\overline{u}) = \#^{\overline{\pi}}(u_1) + \cdots + \#^{\overline{\pi}}(u_n)$. As in the former case, we set $x = \#^{\gamma}(u^{\gamma})$ if u is an auxiliary port of a promotion $!\gamma$, otherwise let $x = 1$; in this way we can say $\#^{\overline{\pi}}(u_i) = x\#^{\overline{\pi}}(v_i)$, so that $\#^{\overline{\pi}}(\overline{u}) = x(\#^{\overline{\pi}}(v_1) + \cdots + \#^{\overline{\pi}}(v_n)) = x\#^{\pi}(v) = \#^{\pi}(u)$.

We conclude that condition ii) holds: for every free port t of π, $l^{\overline{\pi}}(t) = l^{\pi}(t)$, $\#^{\overline{\pi}}(t) = \#^{\pi}(t)$. As for the condition i): by collecting all the results, the reader can check that the inequality $|\overline{\pi}|_{cut} < |\pi|_{cut}$ can be reduced to $\left[l^{\overline{\pi}}(v_1) + \cdots + l^{\overline{\pi}}(v_n)\right] < [l^{\pi}(v)]$. Since we have noticed $l^{\pi}(v) > l^{\overline{\pi}}(v_i)$ for every $i \leq n$, the inequality holds by definition of the multiset order.

Case $!/?d$**.** The case where $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ reduces a cut of type $!/?d$, is an easier variant of the preceding ones, and we left it to the reader.

Case $\rightsquigarrow_r$ or $\rightsquigarrow_a$. The cases where $\pi \rightsquigarrow_r \overline{\pi}$ or $\pi \rightsquigarrow_a \pi'$ can be easily solved by showing that $|\overline{\pi}|_{cut} = |\pi|_{cut}$ and, for $\rightsquigarrow_a$ simply by proving $s(\overline{\pi}) < s(\pi)$, for $\rightsquigarrow_r$ by proving $s(\overline{\pi}) = s(\pi)$ and $c(\overline{\pi}) < c(\pi)$.

Inductive step. If the step $\pi \rightsquigarrow_{s,r,a} \overline{\pi}$ is a promotion closure, then $\overline{\pi}$ is obtained by replacing in π a promotion $!\pi'$ with a promotion $!\overline{\pi'}$, and $\pi \rightsquigarrow_{s,r} \pi'$. By induction hypothesis we know that i) $|\overline{\pi'}| < |\pi'|$, ii) for every free-port of π', $l^{\pi'}(p) = l^{\overline{\pi'}}(p), \#^{\pi'}(p) = \#^{\overline{\pi'}}(p)$. This easily implies i), ii) for $\pi, \overline{\pi}$. ■

4 Confluence theorem

In this section we prove the confluence of $\rightsquigarrow_{SANE}$ (theorem 10). The proof follows these points. First, we consider $\rightsquigarrow_{SANE,g}$ and not $\rightsquigarrow_{SANE}$: indeed the confluence of the former implies that of the latter, since the two reductions have the same transitive closure. Second, we split $\rightsquigarrow_{SANE,g}$ in three disjoint subreductions, $\rightsquigarrow_{o,g}$, $\rightsquigarrow_w$ and $\rightsquigarrow_{s,r,a}$ and we prove the confluence of each of them (lemmas 4, 5 and 6). Then, we prove the commutation of $\rightsquigarrow_{o,g}$ and $\rightsquigarrow_{s,r,a}$ (lemma 7), which implies the confluence of $\rightsquigarrow_{cut,g,r,a}$ (proposition 8). Finally, we prove the commutation of $\rightsquigarrow_{cut,g,r,a}$ and $\rightsquigarrow_w$ (lemma 9), so concluding the confluence of $\rightsquigarrow_{SANE,g}$ (theorem 10).

Lemma 4 *The rule $\rightsquigarrow_{o,g}$ is confluent.*

Lemma 5 *The rule $\rightsquigarrow_w$ is confluent.*

Proof. The above lemmas are immediate, since there are no critical pairs. ■

Lemma 6 *Any union of the rules $\rightsquigarrow_a, \rightsquigarrow_s, \rightsquigarrow_r$ is confluent on correct nets.*

Proof. It is known that $\rightsquigarrow_{s,r}$ is local confluent on correct nets: there are several critical pairs that we omitted here because their solution is standard (see [PTdF07] for the details). It is then straight to deduce the local confluence of any union of the rules $\rightsquigarrow_a, \rightsquigarrow_s, \rightsquigarrow_r$. Since $\rightsquigarrow_{s,r,a}$ is also strong normalizing on correct nets (see proposition 3), the statement follows by the Newman lemma. ■

Lemma 7 *The two rules $\rightsquigarrow_{o,g}$ and $\rightsquigarrow_{s,r,a}$ commute.*

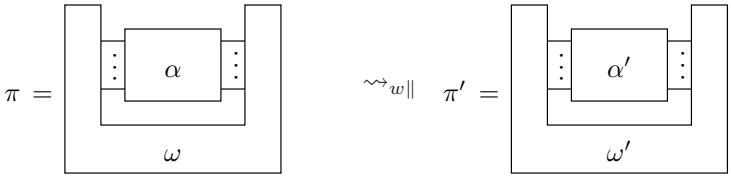
Proof. The reader can check that if $\pi \rightsquigarrow_{o,g} \pi'$ and $\pi \rightsquigarrow_{s,r,a} \pi''$ then there is π''' s.t. $\pi' \overset{=}{\rightsquigarrow}_{s,r,a} \pi'''$ and $\pi'' \rightsquigarrow^*_{o,g} \pi'''$. Precisely the only critical pairs are those produced by $\rightsquigarrow_{o,g}$ and $\rightsquigarrow_a$, which are four, two produced by applying $\rightsquigarrow_a$ to $\otimes$ cells, and other two completely symmetric, produced by applying $\rightsquigarrow_a$ to $\parr$ cells. In figure 11 you find the solution for these last ones. ■

Notice that the topmost diagram of figure 11 shows that rule $\rightsquigarrow_o$ does not commute with $\rightsquigarrow_a$ (and in general with $\rightsquigarrow_{s,r,a}$). This is the main reason we introduce the reduction $\rightsquigarrow_g$.

Proposition 8 *The rule $\rightsquigarrow_{cut,g,r,a}$ is confluent on correct nets.*

Proof. This is an immediate consequence of Lemmas 4, 6, 7 and the Hindley-Rosen lemma. ■

The last step to achieve the confluence of $\rightsquigarrow_{SANE}$ is to add $\rightsquigarrow_w$. For this task it is convenient introducing the parallel $\rightsquigarrow_w$: we define $\pi \rightsquigarrow_{w\|} \pi'$ by induction on the size of π, as follows.



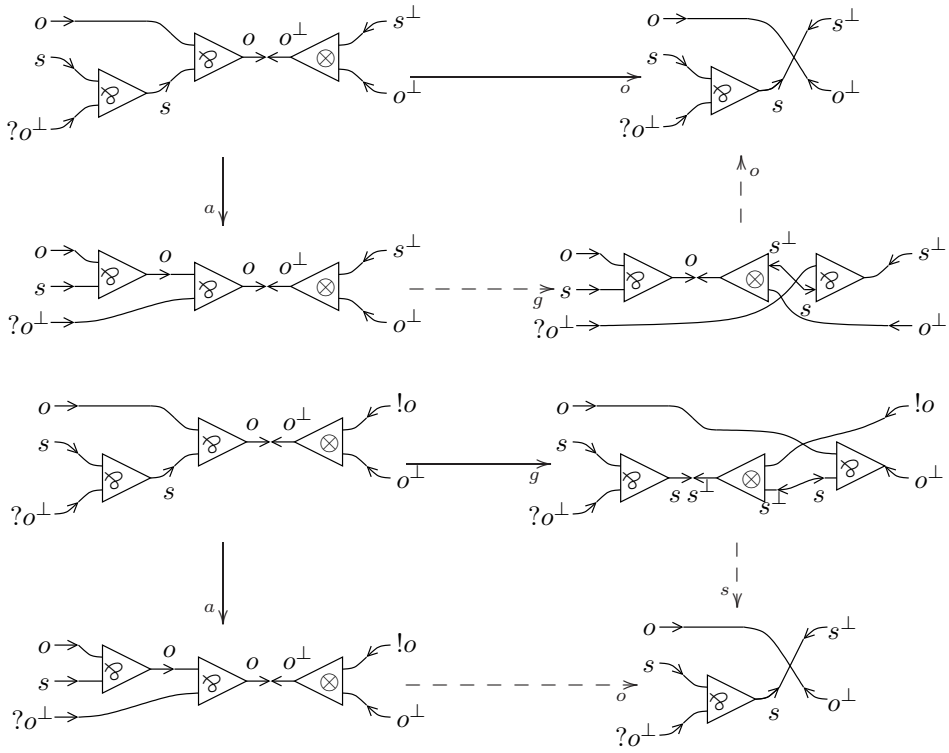


Figure 11: Critical pairs for lemma 7

where $\alpha \rightsquigarrow_w \alpha'$ and either $\omega = \omega'$ or $\omega \rightsquigarrow_{w\|} \omega'$ (which is defined, since the size of ω is less than that of π). As expected, the relations $\rightsquigarrow_{w\|}$ and $\rightsquigarrow_w$ have the same transitive closure.

Lemma 9 *The rule $\rightsquigarrow_w$ commutes with $\rightsquigarrow_{cut,a,g,r}$.*

PROOF. The reader can check that if $\pi \rightsquigarrow_{cut,a,g,r} \pi'$ and $\pi \rightsquigarrow_{w\|} \pi''$ then there is π''' s.t. $\pi' \overset{=}{\rightsquigarrow}_{w\|} \pi'''$ and $\pi'' \rightsquigarrow^*_{cut,a,g,r} \pi'''$. This implies by a simple diagram chase that $\rightsquigarrow_{cut,a,g,r}$ and $\rightsquigarrow_{w\|}$ commute. Hence $\rightsquigarrow_{cut,a,g,r}$ and $\rightsquigarrow_w$ commute, since $\rightsquigarrow_w$ and $\rightsquigarrow_{w\|}$ have the same transitive closure. ■

Theorem 10 (Confluence theorem) *The reduction $\rightsquigarrow_{SANE}$ (as well as $\rightsquigarrow_{SANE,g}$) is confluent on correct nets.*

PROOF. The confluence of $\rightsquigarrow_{SANE,g}$ is an immediate consequence of Lemmas 9, 5 and the Hindley-Rosen lemma. The confluence of $\rightsquigarrow_{SANE}$ follows, since $\rightsquigarrow_{SANE,g}$ and $\rightsquigarrow_{SANE}$ have the same transitive closure. ■

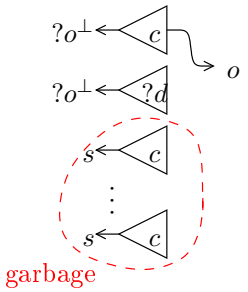
5 Separation theorem

Definition 11 (SANE Value) *A net is a value when it does not contain cuts nor redexes for any rules but the axiom expansion rules.*

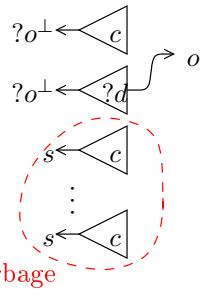
The SANE values correspond in this framework to Λμ-calculus canonical normal forms defined in [Sau05].

Definition 12 (Nets 0 and 1) *We define two particular values,* **1** *and* **0**, *that will be used to separate nets. The values will be defined modulo a certain number of* 0*-ary contraction cells, playing the role of garbage:*

1 =



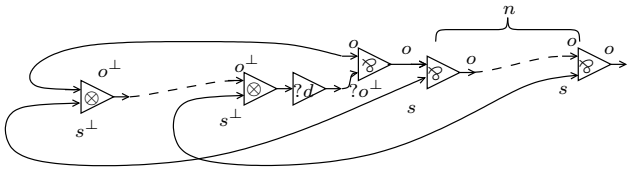
0 =



In the sequel the garbage of **1** and **0** will never disturb the proof of separation theorem, hence we will omit to mention.

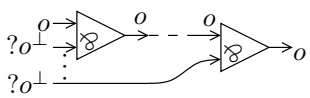
Definition 13 (ϕ^n) *Let n be an integer, one defines the net ϕ^n as follows:*

ϕ^n =



Definition 14 (Contexts) *The contexts of type I are defined as nets with an additional cell, namely $[]$ which has ports I. Correct nets are simply correct nets with this additional cell.*
Let π be a net of conclusions I, then $\mathcal{C}(\pi)$ is defined to be the net obtained from $\mathcal{C}$ by replacing all occurrencs of $[]$ in $\mathcal{C}$ by π. The correction of π and $\mathcal{C}$ implies the correction of $\mathcal{C}(\pi)$ (notice that the converse is not true).
A particularly important class of correct contexts are the head contexts: $\mathcal{H} =< []|x_1 : \Psi_1, \ldots, x_n : \Psi_n >$ where $x_1, \ldots x_n$ are conclusions of the hole (when it will not be ambiguous we shall often write simply $\mathcal{H} =< []|\Psi_1, \ldots, \Psi_n >$). In the case of head contexts, one also writes $\mathcal{H}(\pi)$ as $\pi^{\vec{\Psi}}$.

Definition 15 ($\mathcal{A}(-,-)$) *Let n be an integer, we define the net $\mathcal{A}\left(\parr^?, n\right)$ as the tree composed by n cells of type $\parr^?$:*



Similarly one has $\mathcal{A}(x,n)$, for $x \in \{\parr^s, \parr^{?,s}, \otimes^!, \otimes^{s^\perp}, \otimes^{!,s^\perp}\}$.

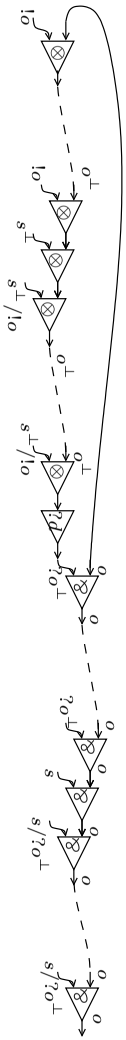


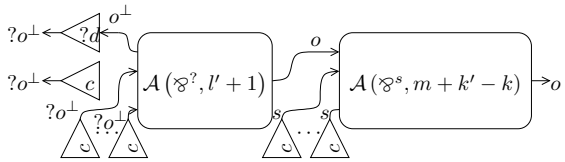
Figure 12: *klmn*-nets

Definition 16 (*klmn*-nets) *klmn-nets are the correct nets such that one finds the subnet in figure 12 starting from their o conclusion.*

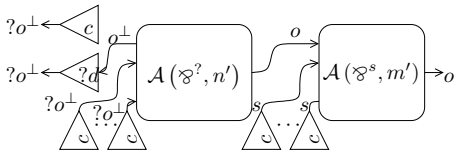
One associates a mesure (k,l,m,n) to these nets as follows: entering the conclusion o, there is one maximal $o^\perp$-path: it crosses $⅋$ cells and then $\otimes$ cells and finally reaches the dereliction at depth 0:
(i) *k is the number of $⅋^s$ cells that are crossed on the $o^\perp$-path;*
(ii) *l is the number of $⅋^?$ cells that are crossed after the last $⅋^s$ cell is passed and before reaching the $⅋^?$ connected to the dereliction;*
(iii) *n is the number of $\otimes^!$ that are crossed before the first $\otimes^{s^\perp}$ is reached;*
(iv) *m is the number of $\otimes^{s^\perp}$ that are crossed before reaching the dereliction.*

Lemma 17 *Given two klmn-nets π and π' having mesures (k,l,m,n) and (k',l',m',n') respectively, if $(k,l,m,n) \neq (k',l',m',n')$ then there is a head context $\mathcal{C}$ such that $\mathcal{C}(\pi) \rightsquigarrow^\star_{SANE} \mathbf{0}$ and $\mathcal{C}(\pi') \rightsquigarrow^\star_{SANE} \mathbf{1}$.*

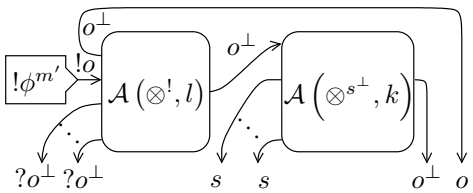
Proof. **(i) if $k < k'$.** Let $\mathcal{C}$ be the context in figure 13.
where π_1 and π_2 are respectively:



and



One can check that if one cut $\mathcal{C}$ with π on conclusion of type respectively $o^\perp - o$, one gets $\mathcal{C}(\pi) \rightsquigarrow^\star_{SANE} \mathbf{0}$ and $\mathcal{C}(\pi') \rightsquigarrow^\star_{SANE} \mathbf{1}$.
(ii) if $k = k'$ and $l < l'$. This case is solved similarly by using slight variants of $\mathcal{C}, \pi_1$ and π_2.
(iii) if $k = k'$, $l = l'$ and $m < m'$. Let $\mathcal{C}$ be the following context (recall that $\phi^{m'}$ is the net defined in definition 13):



then $\mathcal{C}(\pi)$ and $\mathcal{C}(\pi')$ reduce to nets π_0 and π'_0 which are *klmn*-nets with $k_{\pi_0} \neq k_{\pi'_0}$ and one can conclude thanks to (i).
(iv) if $k = k'$, $l = l'$, $m = m'$ and $n < n'$. The case is treated similarly to case (iii) with a context differing from the above $\mathcal{C}$ by an easy variant of ϕ^n which reduces the problem to case (ii). ■

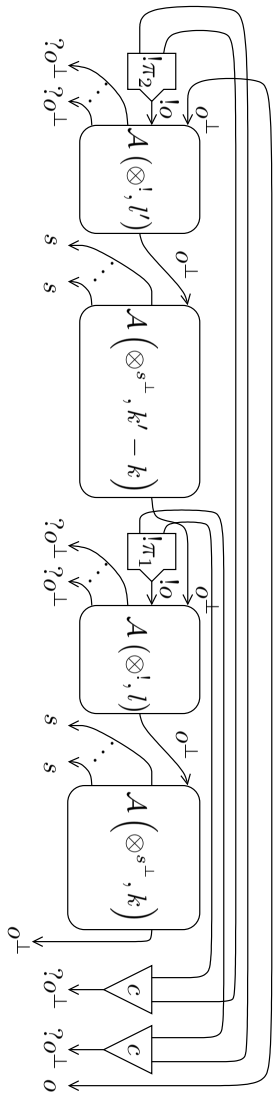


Figure 13: Context used in case (i) of lemma 17.

Lemma 18 *If π and π' are non equivalent values with the same conclusions and such that there are $\parr$ cells at depth 0, then there is a correct head context $\mathcal{C}$ such that $\mathcal{C}(\pi) \leadsto^{\star}_{SANE} \pi_1$ and $\mathcal{C}(\pi') \leadsto^{\star}_{SANE} \pi'_1$ where π_1 and π'_1 are non equivalent values with the same depths as π and π' and such that π_1 and π'_1 have no $\parr$ cell at depth 0.*

Proof. The proof is easy by an easy induction on the number of $\parr$ cells at depth 0. One studies the negative conclusions of π and π' that have cells $\parr^{?}$, $\parr^{s}$ or $\parr^{?s}$ at depth 0, those conclusions are necessarily of type o or s. The removal of cells of kind $\parr$ is done by repeatedly cutting π and π' with cells of type $\otimes$ ($\otimes^{!}$, $\otimes^{s^{\perp}}$ or $\otimes^{!s^{\perp}}$) depending on what in π, π' is above the conclusion. We only study one case which is the most complex: there is a conclusion of type o above which there is a $\parr^{?}$ in π and a $\parr^{s}$ in π'. Then one cuts this conclusion of the nets with a $\otimes^{!}$. The cut in π is reducible but the one in π' is not. We reduce the cut in π and obtain a net for which the number of $\parr$ cells has decreased by one. Concerning π' one first expand (thanks to $\leadsto_{ws}$) the wire of type s which is auxiliary port of the cell $\parr^{s}$, then applies an associativity rule which replaces an irreducible cut $\parr^{s}/\otimes^{!}$ by a reducible cut $\parr^{?}/\otimes^{!}$. After reducing the cut, we obtain a pair of nets having one $\parr$ cell less than then original nets.

The process ends only when there is no $\parr$ cell a depth 0 anymore and it does terminate since each of these steps reduces the number of $\parr$ cells at depth 0 by at least one. The process preserves equivalence as well as the depths of the nets. ■

Definition 19 (Maximal positive sub-nets) *Let π be a correct net. One defines π_+, the maximal positive subnet of π, and we distinguish a particular conclusion of the net, x_π, as follows:*

- *If π has a positive conclusion x, then $x_\pi = x$ and π_+ is the maximal $\otimes$-tree which is a sub-net of π with conclusion x.*
- *If π has a dereliction at depth 0, then x_π is the unique conclusion which is below the dereliction and π_+ is the sub-net containing the cells from x_π to the dereliction and the maximal $\otimes$-tree.*

Theorem 20 (Separation for SANE) *Let π and π' be two correct nets with the same conclusions that are non equivalent values ($\pi \not\sim_{SANE} \pi'$). There exists a (head) context $\mathcal{C}$ such that $\mathcal{C}(\pi^{\vec{\phi}}) \leadsto^{\star}_{SANE} \mathbf{0}$ and $\mathcal{C}(\pi'^{\vec{\phi}}) \leadsto^{\star}_{SANE} \mathbf{1}$.*

Proof. Actually, we prove the following stronger statement:

(*) let π and π' be two correct nets with the same conclusions $I = \{x_1^{t_1}, \ldots, x_k^{t_k}\}$ that are non equivalent values ($\pi \not\sim_{SANE} \pi'$).
For any $J = \{x_{i_1}^{?\,o^{\perp}}, \ldots, x_{i_l}^{?\,o^{\perp}}\} \subseteq I$, if $\mathcal{N} = \{n_1, \ldots, n_l\}$ is a family of distinct integers that are large enough and if $\vec{\phi} =< []|x_{i_1} : \phi^{n_1}, \ldots, x_{i_l} : \phi^{n_l} >$, then, there exists a head context $\mathcal{C}$ such that $\mathcal{C}(\pi^{\vec{\phi}}) \leadsto^{\star}_{SANE} \mathbf{0}$ and $\mathcal{C}(\pi'^{\vec{\phi}}) \leadsto^{\star}_{SANE} \mathbf{1}$.

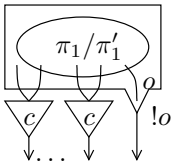
We do not provide a fully detailed proof which would be too long but we tried to treat the main and most complex cases. The proof goes by induction on the sum of the depth of π and π'. Let J be a subset of the conclusions of type

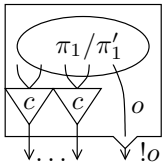
$?o^\perp$. Thanks to lemma 18, one may suppose that neither π nor π' have any $\parr$ cells at depth 0: there is a context in which the nets reduce to non-equivalent values of identical the depths and with no $\parr$ cells at depths 0. Let $x_i = x_\pi$ and $x_j = x_{\pi'}$.

One shall reason on the structure of the maximal positive subnets π_+ and π'_+:

1. **If there is a positive conclusion** x (and thus $x = x_i = x_j$), we reason on the type of x.

   (a) *If the positive conclusion is of type* $!\,o$, we have:

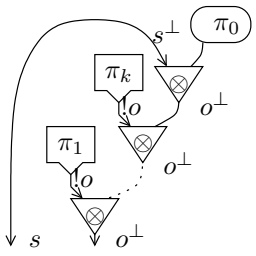
   $\pi/\pi' =$ 

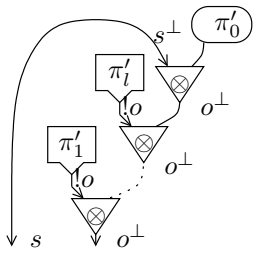
   $\sim_r$ 

   $= \tilde{\pi}/\tilde{\pi}'$

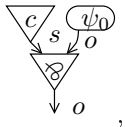
   We can thus apply the induction hypothesis to nets π_1/π'_1 with the appropriate contraction so that they have the same conclusion except for the $!o$ which is now an o. One finds a separating context for ${\pi_1}^{\vec{\phi}}/{\pi'_1}^{\vec{\phi}}$: $\mathcal{C} =< []|\psi_1^{n^\perp}, \ldots, \psi_{l'}^{n^\perp}, \psi_{l'+1}^{o^\perp} >$ and adding a dereliction cell to the conclusion of type $o^\perp$ of $\psi_{l'+1}^{o^\perp}$ makes a separating context for $\tilde{\pi}^{\vec{\phi}}/\tilde{\pi'}^{\vec{\phi}}$ and thus for $\pi^{\vec{\phi}}/{\pi'}^{\vec{\phi}}$.

   (b) *If the positive conclusion is of type* $o^\perp$, we study the shape of the $\otimes$-trees rooted in x and compare the trees, proving separation by induction on the number of $\otimes^{s^\perp}$ in the tree. The first case is when both $\otimes$ trees contain $\otimes^{s^\perp}$:

   $\pi =$ 

   $\pi' =$ 

If $\pi_0 \not\sim \pi'_0$, one separates $\pi_0{}^{\vec{\phi}}$ and $\pi'_0{}^{\vec{\phi}}$, with $\mathcal{C} =< []|\psi_0^o, \psi_1^{n^\perp}, \psi_{l'}^{n^\perp} >$



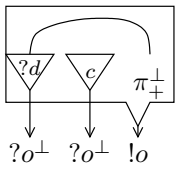
and by changing ψ_0 into , we separate $\pi^{\vec{\phi}}$ and $\pi'^{\vec{\phi}}$.
If $\pi_0 \sim \pi'_0$, either there is $i \leq k, l$ such that $\pi_i \not\sim \pi'_i$ or $k \neq l$ or the wires typed by s reach two different conclusions of I. In any of these cases, one can separate: by using induction in the first case or by using a wire expansion of the conclusions of type s, an associativity reduction and then the method of the first case.

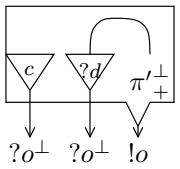
The cases when some of the trees only contain $\otimes^!$ cells or when there is an axiom link but no $\otimes$ at all are treated fairly similarly.

(c) *If the positive conclusion is of type* $s^\perp$, we proceed essentially as in the previous case by induction on the number of $\otimes^{!s^\perp}$ on the $\otimes$-trees.

2. **If there is a dereliction at depth 0 and** $x_i, x_j \notin J$. We know that x_i and x_j are of type $?\,o^\perp$ (indeed, we supposed that there is no $⅋$ cell at depth 0).

(a) *if* $i \neq j$, one defines two contexts ψ_i and ψ_j as follows:

$\psi_i =$ 

$\psi_j =$ 

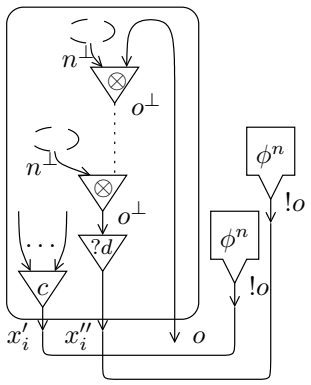
where $\pi^\perp_+$ is an $⅋$ tree dual to the $\otimes$-tree composing the maximal positive subnet of π where all left auxiliary ports are weakened[9].

Thus one has $< \pi^{\vec{\phi}} | \psi_i, \psi_j > \rightsquigarrow^\star_{SANE} \mathbf{0}$ and $< \pi'^{\vec{\phi}} | \psi_i, \psi_j > \rightsquigarrow^\star_{SANE} \mathbf{1}$.

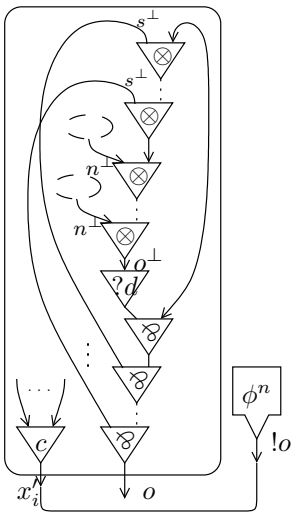
(b) *if* $i = j$, then we are essentially in the same case as with a positive conclusion, however there is a slight subtility because the conclusion x_i is of type $?\,o^\perp$ and thus it can be contracted. This contraction could interfere with an induction step and to cope with this problem, we shall use the nets ϕ^n.
Let p, q be the numbers of $\otimes^{s^\perp}$ in the $\otimes$-trees of π and π' respectively. One cuts the conclusion x_i of π/π' with ϕ^n with $n > p, q$ (and n being distinct from all the indices in $\mathcal{N}$).

[9] This subnet $\pi^\perp_+$ plays the role of an eraser of nets.

$< \pi^{\vec{\phi}} | \phi^n > \sim$



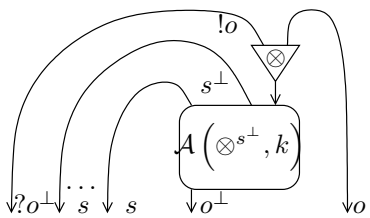
$\rightsquigarrow^\star_{SANE}$



$= \tilde{\pi}^{\vec{\phi},\phi^n}$

The net pictured as equivalent to $< \pi^{\vec{\phi}} | \phi^n >$ is indeed equivalent since they both reduce to the same net when reducing the cut $\mathcal{S}/c$. $\tilde{\pi}^{\vec{\phi},\phi^n}$ is obtained by reducing the cut on the derelicted conclusion x''_i in the above figure. $\tilde{\pi}^{\vec{\phi},\phi^n}$ (and its counterpart $\tilde{\pi'}^{\vec{\phi},\phi^n}$) are $klmn$-nets.

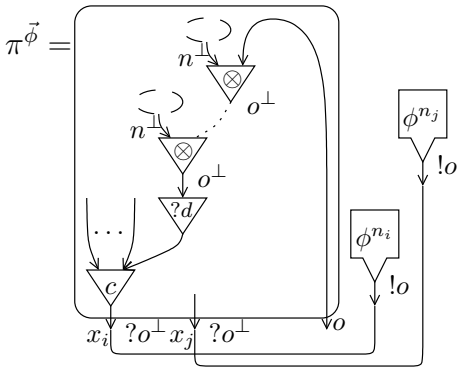
If $p \neq q$ then the lemma 17 can be applied (since $k = n - p$ and $k' = n - q$) in order to have separation. Otherwise, by cutting the conclusion of type o with a net of shape:

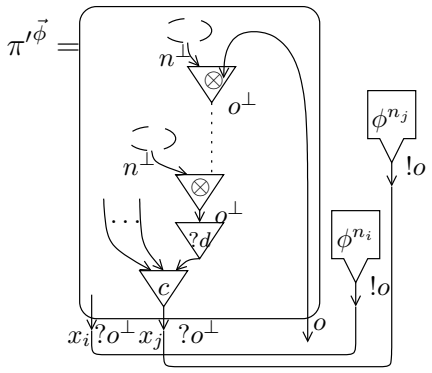


After reduction we obtain a net with dereliction as conclusion cell (there is no contraction anymore). If we consider the nets without the derelictions, then they have a positive conclusion, have the same depth as the original nets and now J has become $J \cup x'_i$ (the head context $\vec{\phi}$ is replaced by $\vec{\phi}, \phi^n$), this can be separated by head context $\mathcal{C} =< []|\psi_0, \psi_1, \ldots, \psi_{l'} >$. Let ψ_0^o be the element of the separating context which is cut with the conclusion of type $o^\perp$. ψ_0^o has a conclusion of type o its other conclusions have type n, as a consequence ψ_0^o can be enclosed in a promotion box leading to net ${\psi'_0}^{!o}$ and to a separating context for $\pi^{\vec{\phi}}/{\pi'}^{\vec{\phi}}$.

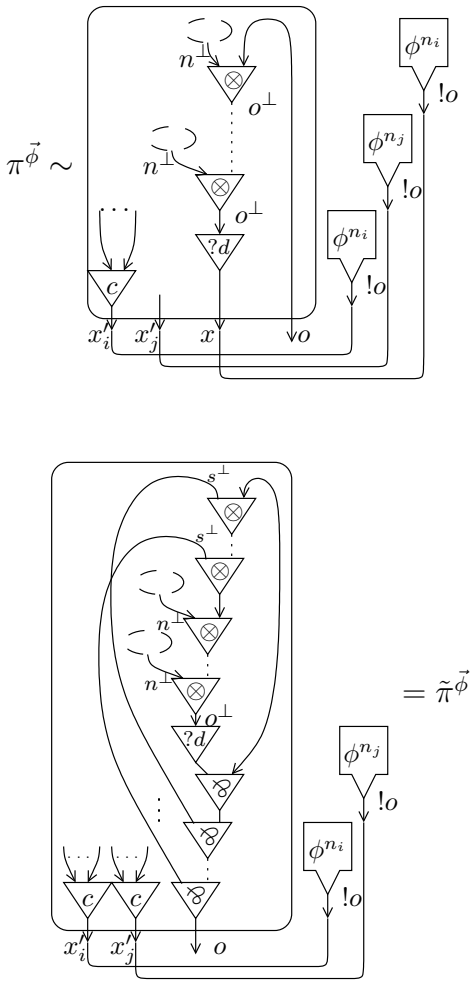
3. **Finally, if there is a dereliction at depth 0 and if** $\{x_i, x_j\} \cap J \neq \emptyset$. Let k, k' be the numbers of $\otimes^{s^\perp}$ in the $\otimes$ maximal positive trees of π/π'.

   (a) *if* $i \neq j$, there are two cases: either $x_i \in J$ and $x_j \notin J$ or $\{x_i, x_j\} \subseteq J$. We only consider the case when they both are in J since the other case can be treated similarly as we shall see.
   By hypothesis, we know that we have $n_i \neq n_j$ and that we can choose n_i and n_j to be as large as we want. Consider $n_i > k$ and $n_j > k'$. $\pi^{\vec{\phi}}/{\pi'}^{\vec{\phi}}$ have the following structure:





And thus we have:

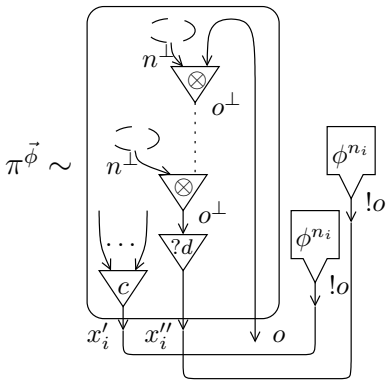


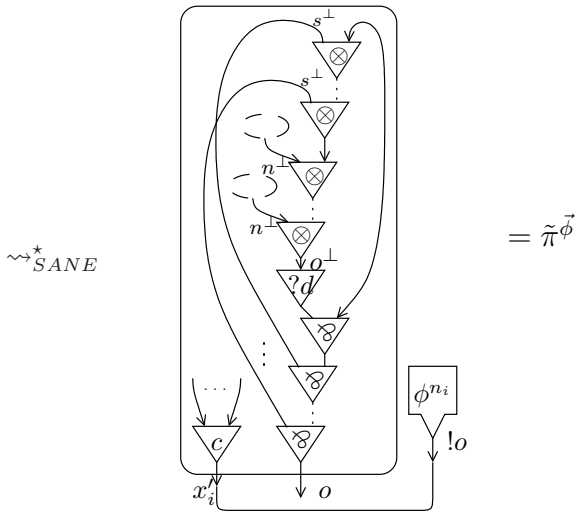
By renaming x_j as x_j' and x_i as x_i' or x in $\tilde{\pi}^{\vec{\phi}}$ (we do the same thing with $\pi'^{\vec{\phi}}$: x_i becomes x_i' and x_j becomes x_j' or x in $\tilde{\pi'}^{\vec{\phi}}$).

And if $k = k'$ then $n_i - k \neq n_j - k'$ so that the lemma 17 can be applied and if $k \neq k'$ then the lemma can also be applied since the $m = n_i$ and $m' = n_j$ so that we can find a separating context.

The case where $x_j \in J$ and $x_i \notin J$ is treated similarly: in this case we can choose what to cut x_i with and in particular we can choose a ϕ^{n_i}.

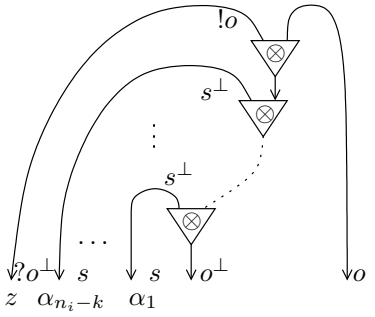
(b) *if* $\{x_i, x_j\} \subseteq J$ *and* $i = j$. In that case, let us consider $n_i > k, k'$. Then we have:



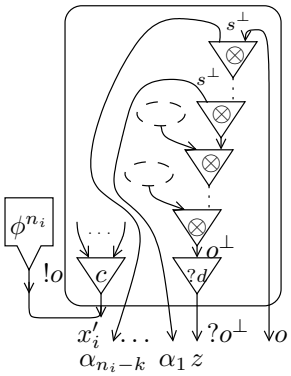


If $k \neq k'$ we can again conclude thanks to lemma 17.

If $k = k'$, then it is possible to cut the conclusion of type o of $\tilde{\pi}^{\vec{\phi}}$ with:



and it reduces to:



These $\tilde{\pi}_0/\tilde{\pi'}_0$ have same depths as π/π' and have now conclusions in $J \cup \{z^{?\,o^\perp}, \alpha^s_1, \ldots, \alpha^s_{n_i-k}\}$ and a dereliction at depth 0 with $x_{\tilde{\pi}_0} = x_{\tilde{\pi'}_0} \notin J$, which is treated by the second case of the proof, which ensure that there exists a separating context for $\pi^{\vec{\phi}}/\pi'^{\vec{\phi}}$ and which concludes the proof.

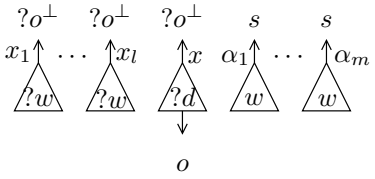
■

6 Simulation theorem

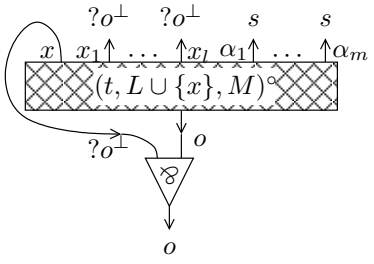
We now give a translation, denoted by $(\,)^\circ$, of $\Lambda\mu$-terms into correct stream nets. Actually what we translate is a triplet (t, L, M), where t is a term, L is a set of λ-variables, M is a set of μ-variables and $FV(t) \subseteq L \cup M$; moreover the translation brings a one-to-one correspondence between the conclusions of the net $(t, L, M)^\circ$ and the set $L \cup M \cup \{t\}$, in such a way that: i) t is associated with the unique conclusion of type o, ii) every variable of L is associated with a conclusion of type $?o^\perp$ and iii) every variable of M is associated with a conclusion of type s. For the purpose of the translation, we label the free ports of $(t, L, M)^\circ$ of type $?o^\perp$ or s with the corresponding variable in $L \cup M$.

Let $L = \{x_1, \ldots, x_l\}$ and $M = \{\alpha_1, \ldots, \alpha_m\}$, then:

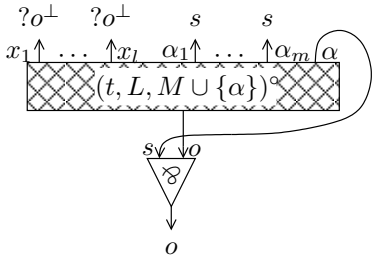
- $(x, L \cup \{x\}, M)^\circ$



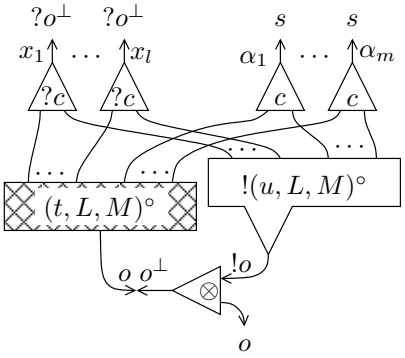
- $(\lambda x.t, L, M)^\circ$



- $(\mu\alpha.t, L, M)^\circ$

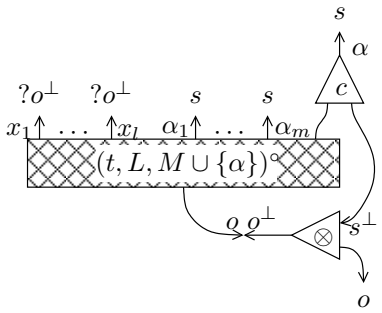


- $((t)u, L, M)^\circ$, let π be the following net:



then $((t)u, L, M)^\circ = \mathsf{NF}^r(\pi)$.

- $((t)\alpha, L, M \cup \{\alpha\})^\circ$, let π be the following net:



then $((t)\alpha, L, M \cup \{\alpha\})^\circ = \mathsf{NF}^r(\pi)$.

Proposition 21 (Injectivity of $(\,)^\circ$) *The translation $(\,)^\circ$ is injective: if $t \neq u$ then $t^\circ \neq u^\circ$.*

Proof. By induction on t. ■

With the following definition we characterize those nets which translate $\Lambda\mu$-terms (theorem 23):

Definition 22 ($\Lambda\mu$-net) *A* $\Lambda\mu$-net *is a net π s.t.:*

1. *π is correct;*
2. *π does not contain cells of type $\parr^{?,s}$ or $\otimes^{!,s^\perp}$;*
3. *π is a normal form w.r.t. $\rightsquigarrow_{s,r,a}$;*
4. *every free port of π is negative;*
5. *and recursively the nets associated with promotion cells are $\Lambda\mu$-nets.*

Theorem 23 (Sequentialization) *Let π be a net, π is a $\Lambda\mu$-net iff there is a $\Lambda\mu$-term t s.t. $\pi = t^\circ$.*

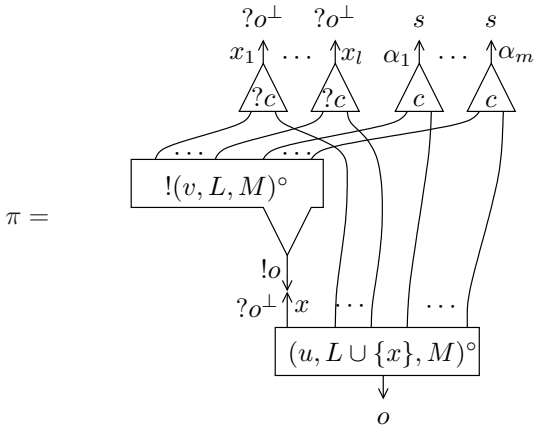
Moreover, there is a natural $1-1$ correspondence between the cuts of t° and the $\Lambda\mu$-cuts of t s.t. a cut of type $\parr^?/\otimes^!$ (resp. $\parr^?/\otimes^{s^\perp}, \parr^s/\otimes^!, \parr^s/\otimes^{s^\perp}$) corresponds to a $\Lambda\mu$-cut of type $(\mathcal{T})\mathcal{T}$ (resp. $(\mathcal{T})\mathcal{S}$, $(\mathcal{S})\mathcal{T}$, $(\mathcal{S})\mathcal{S}$). In particular, t° is cut-free (hence a $\rightsquigarrow_{cut,g,a,r}$ normal form) iff t is canonical.

Proof. The direction $\Rightarrow$ is an easy inspection of the case definition of the translation $(\,)^\circ$. The direction $\Leftarrow$ is a simple variant of the proof of the sequentialization theorem, see [Dan90],[Reg92],[Lau03]. ■

Let us turn our attention to the dynamics of $\rightsquigarrow_{\Lambda\mu}$. In what follows we will prove that $\rightsquigarrow_{SANE}$ simulates $\rightsquigarrow_{\Lambda\mu}$: if $t \rightsquigarrow_{\Lambda\mu} u$, then $t^\circ \rightsquigarrow_{\Lambda\mu} u^\circ$ (theorem 26).

The following two lemmas are easy variants of the corresponding ones in [Lau03].

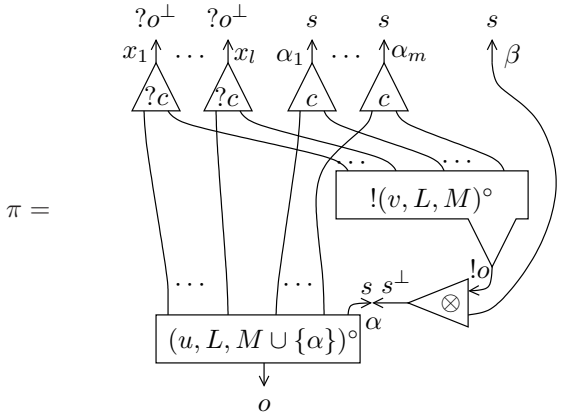
Lemma 24 (λ-substitution) *Let π be the following net:*



then $\mathsf{NF}^{s,r}(\pi) = (u[v/x], L, M)^\circ$.

Proof. By induction on u. ■

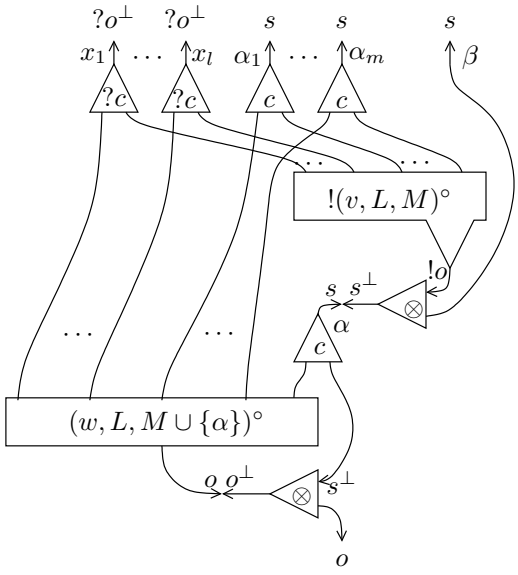
Lemma 25 (μ-substitution) *Let π be the following net:*



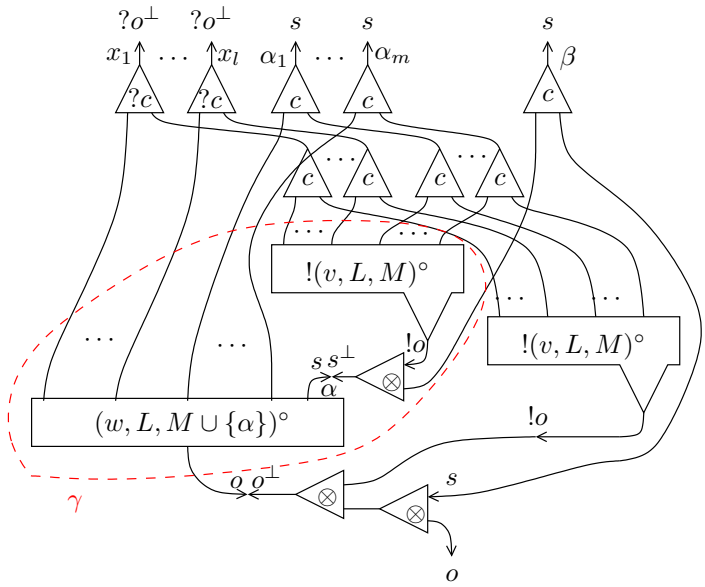
then $\mathsf{NF}^{s,r,a}(\pi) = (u[(t)v\beta/(t)\alpha], L, M \cup \{\beta\})^\circ$.

Proof. By induction on u. We consider only one case (the one where $\rightsquigarrow_a$ play a crucial role), leaving the other cases to the reader.

If $u = (w)\alpha$, then π is the $\rightsquigarrow_r$-normal form of this net:



Which by one $\rightsquigarrow_s$ step and one $\rightsquigarrow_a$ step reduces to the following net π'



By induction hypthesis $\mathsf{NF}^{s,r,a}(\gamma) = (w[(t)v\beta/(t)\alpha], L, M \cup \{\beta\})^\circ$, so we conclude that $\mathsf{NF}^{s,r,a}(\pi) = ((w[(t)v\beta/(t)\alpha])v\beta, L, M \cup \{\beta\})^\circ$. ■

Theorem 26 (Simulation) *Let t, u be two $\Lambda\mu$-terms, then:*

1. $t \to_{\beta_T} u$ *implies* $t^\circ \rightsquigarrow_o \cdot \rightsquigarrow^*_{s,r} u^\circ$
2. $t \to_{\beta_S} u$ *implies* $t^\circ \rightsquigarrow_o u^\circ$
3. $t \to_{fst} u$ *implies* $t^\circ \rightsquigarrow_{ws} \cdot \rightsquigarrow^*_{s,r,a} u^\circ$
4. $t \to_{\eta_T} u$ *implies* $u^\circ \rightsquigarrow_{wo?} \cdot \rightsquigarrow_{w?} t^\circ$

5. *$t \to_{\eta_S} u$ implies $u^\circ \rightsquigarrow_{wos} \cdot \rightsquigarrow_{w?} t^\circ$*

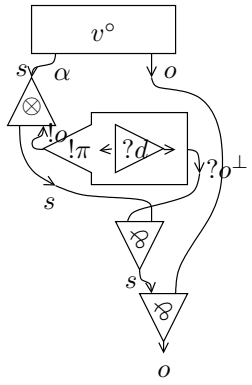
PROOF. One can restrict to the case the redex reduced in $t \rightsquigarrow_{\Lambda\mu} u$ is the head-redex of t, the more general case will follow by a straight induction on the complexity of t.

If $t \rightsquigarrow_{\beta_T} u$, then the redex reduced corresponds to a cut of type $⅋^?/⊗^!$ of t° (theorem 23): by reducing this redex we obtain a net π as that pictured in lemma 24; by this lemma we conclude $\mathsf{NF}^{s,r}(\pi) = u^\circ$.

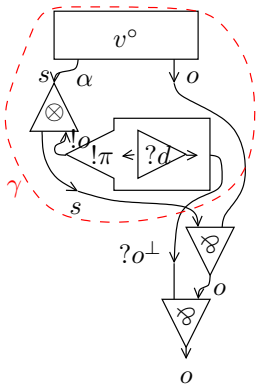
If $t \rightsquigarrow_{\beta_S} u$, then the redex reduced corresponds to a cut of type $⅋^s/⊗^{s^\perp}$ of t° (theorem 23): by reducing this redex we obtain straight u°.

If $t \rightsquigarrow_{fst} u$, let $t = \mu\alpha.v$. We have:

$t^\circ \quad \rightsquigarrow_{ws} \cdot \rightsquigarrow_{w?}$



$\rightsquigarrow_a$



then by lemma 25 we have that $\mathsf{NF}^{s,r,a}(\gamma) = v[(w)x\beta/(w)\alpha]^\circ$, so we conclude: $\lambda x.\mu\beta.v[(w)x\beta/(w)\alpha]$.

The cases $t \rightsquigarrow_\eta u$ and $t \rightsquigarrow_{\eta_s} u$ are directly simulated by $\rightsquigarrow_w$.

■

Concluding remark. The relashionships between Λμ-calculus and SANE are deep as the previous simulation theorem makes it clear. We close this final section by few additional remarks related with the question of simulation.

- Other $\Lambda\mu$-based rules can be simulated, in particular the μ rule $((\mu\alpha.t)u \to \mu\alpha.t[(v)u\alpha/(v)\alpha])$ which is simulated by $\rightsquigarrow_g \cdot \rightsquigarrow^*_{s,r,a}$. Since μ rule can itself be simulated by $\Lambda\mu$-calculus rules, there is no surprise about simulating it in SANE, but it is interesting to see that μ can be simulated using $\rightsquigarrow_g$.

- There seems to be another interesting underlying calculus in SANE that we would hope to simulate thanks to SANE: this would be a real stream language where the main structure would not correspond to $\parr^?$ and $\otimes^!$ but to $\parr^{?s}$ and $\otimes^{!s^\perp}$. This would probably be the stream-calculus $\Lambda_{\mathcal{S}}$ already suggested in [Sau05].

- A notion of explicit substitutions seems to be underlying in the proof of simulation. This would correspond toreductions $\rightsquigarrow_{s,r}$ in SANE. This explicit substitution has good properties since it is strongly normalizing (proposition 3) and confluent (lemma 6).

- Beyond simulation, bisimulation: we guess that one can have even a bisimulation result: if $t^\circ \rightsquigarrow^*_{SANE} u^\circ$ then $t \rightsquigarrow^*_{\Lambda\mu} u$. However the proof of bisimulation is very delicate, since $\rightsquigarrow^*_{SANE}$ allows much more reductions between t° and u°, than those used to simulate $\rightsquigarrow_{\Lambda\mu}$.

7 Conclusion

We introduced SANE, a new class of nets which lies in between usual linear logic proof-nets and polarized proof-nets for which we proved strong normalization for the exponential cut-elimination, confluence and separation:

- The strong normalization is proved by induction on a mesure which is indeed very general and can be adapted to other net-based systems.
- The confluence proof is original in the sense it is not a direct consequence of the proof of confluence for MELL proof-nets, in fact $\rightsquigarrow_{SANE}$ has $\rightsquigarrow_a$ and $\rightsquigarrow_w$ in addition to $\rightsquigarrow_{cut,r}$ (confluence is already hard to prove even in the multiplicative fragment in presence of $\rightsquigarrow_a$). Moreover, it is an interesting result since we were able to prove confluence for all kinds of correct nets, whereas in Λμ-calculus confluence holds only for μ-closed terms.
- We were especially interested in having separation since we considered it as a design requirement for our nets and since this is one of the few separation results that exist for proof-nets (other know results are [MP07, MP94]).

Our initial aim was to study Λμ-calculus thanks to the powerful techniques of proof-nets. In particular we obtained a simulation of Λμ-calculus. Moreover we have, as a by-product of the simulation theorem, a notion of explicit substitutions for Λμ-calculus which is the one simulated by the $\rightsquigarrow_{s,r}$ of SANE. This explicit substitution has good properties since it is strongly normalizing (proposition 3) and confluent (lemma 6). The encoding of Λμ-calculus in SANE and the way Λμ-calculus reduction rules are simulated shed an interesting light on the reduction rules, in particular with respect to the *fst*-rule that relates λ-variables with μ-variables.

In addition, the simulation result suggests that there exists another calculus hidden in SANE in which the associativity rule would go in the other direction. For this reason we are optimistic about SANE being a platform in which to study continuation calculi.

Future works. This work is being pursued in two directions:

- We are investigating an extension of the simulation result in the form of a bisimulation. Indeed, in addition to the simulation of Λμ-calculus by SANE we hope to obtain a converse result.
- A natural developement of the study of separation would be to look for a typed result since our theorem only deals with pure nets.

References

[Bö68] Corrado Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel λk-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.

[CG99] Roberto Di Cosmo and Stefano Guerrini. Strong normalization of proof nets modulo structural congruences. In *RtA '99: Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, pages 75–89, London, UK, 1999. Springer-Verlag.

[CK97] Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 35, Washington, DC, USA, 1997. IEEE Computer Society.

[Dan90] Vincent Danos. *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)*. Thèse de doctorat, Université Paris VII, 1990.

[dG94] Philippe de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *LPAR'94*, volume 822 of *LNAI*, 1994.

[dG98] Philippe de Groote. An environment machine for the lambda-mu-calculus. *Mathematical Structures in Computer Science*, 8:637–669, 1998.

[DP01] René David and Walter Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 66(1):407–413, March 2001.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir01] Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, June 2001.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *Proceedings of the 1990 Principles of Programming Languages Conference*, pages 47–58. IEEE Computer Society Press, 1990.

[Jol00] Thierry Joly. *Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types.* PhD thesis, Université Paris VII, 2000.

[Laf95] Yves Lafont. From proof nets to interaction nets. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 225–247. Cambridge University Press, 1995.

[Lau02] Olivier Laurent. *Étude de la polarisation en logique.* Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau03] Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, January 2003.

[MP94] Gianfranco Mascari and Marco Pedicini. Head linear reduction and pure proof net extraction. *Theoret. Comput. Sci.*, 135(1):111–137, 1994.

[MP07] Damiano Mazza and Michele Pagani. The separation theorem for differential interaction nets. Lecture Notes in Artificial Intelligence, 2007.

[Pag07] Michele Pagani. Between interaction and semantics: visible acyclic nets. In preparation, 2007.

[Par92] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

[PTdF07] Michele Pagani and Lorenzo Tortora de Falco. Strong normalization property for second order linear logic. Pre-print, IAC, Roma, 2007. Submitted to Theor. Comput. Sci.

[Reg92] Laurent Regnier. *Lambda-Calcul et Réseaux.* Thèse de doctorat, Université Paris VII, 1992.

[Sau05] Alexis Saurin. Separation with streams in the $\Lambda\mu$-calculus. In Prakash Panangaden, editor, *Proceedings of the Twentieth Annual IEEE Symp. on Logic in Computer Science, LICS 2005*, pages 356–365. IEEE Computer Society Press, June 2005.

[Tra07] Paolo Tranquilli. Translating differential lambda calculus into differential interaction nets. Submitted to: *Theoretical Computer Science*, 2007.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399