Proof Nets and the Linear Substitution Calculus

Beniamino Accattoli

Proof Nets and the Linear Substitution Calculus

Beniamino Accattoli

Inria, UMR 7161, LIX, École Polytechnique
beniamino.accattoli@inria.fr

Abstract. Since the very beginning of the theory of linear logic it is known how to represent the λ-calculus as linear logic proof nets. The two systems however have different granularities, in particular proof nets have an explicit notion of sharing—the exponentials—and a micro-step operational semantics, while the λ-calculus has no sharing and a small-step operational semantics. Here we show that the *linear substitution calculus*, a simple refinement of the λ-calculus with sharing, is isomorphic to proof nets at the operational level.
Nonetheless, two different terms with sharing can still have the same proof nets representation—a further result is the characterisation of the equality induced by proof nets over terms with sharing. Finally, such a detailed analysis of the relationship between terms and proof nets, suggests a new, abstract notion of proof net, based on rewriting considerations and not necessarily of a graphical nature.

1 Introduction

Girard's seminal paper on linear logic [23] showed how to represent intuitionistic logic—and so the λ-calculus—inside linear logic. During the nineties, Danos and Regnier provided a detailed study of such a representation via proof nets [16,41,15,17], which is nowadays a cornerstone of the field. Roughly, linear logic gives first-class status to *sharing*, accounted for by the *exponential* layer of the logic, and not directly visible in the λ-calculus. In turn, cut-elimination in linear logic provides a micro-step refinement of the small-step operational semantics of the λ-calculus, that is, β-reduction.

The mismatch. Some of the insights provided by proof nets cannot be directly expressed in the λ-calculus, because of the mismatch of granularities. Typically, there is a *mismatch of states*: simulation of β on proofs passes through intermediate states / proofs that cannot be expressed as λ-terms. The mismatch does not allow, for instance, expressing fine strategies such as linear head evaluation [35,18] in the λ-calculus, nor to see in which sense proof nets quotient terms, as such a quotient concerns only the intermediate proofs. And when one starts to have a closer look, there are other mismatches, of which the lack of sharing in the λ-calculus is only the most macroscopic one.

Some minor issues are due to a *mismatch of styles*: the fact that terms and proofs, despite their similarities, have different representations of variables and notions of redexes. Typically, two occurrences of a same variable in a term are smoothly identified by simply using the same name, while for proofs there is an explicit rule, contraction, to identify them. Name identification is obviously associative, commutative, and commutes

with all constructors, while contractions do not have these properties for free[1]. For redexes, the linear logic representation of terms has many cuts with axioms that have no counterpart on terms. These points have been addressed in the literature, using for instance generalised contractions or interaction nets, but they are not devoid of further technical complications. Establishing a precise relationship between terms and proofs and their evaluations is, in fact, a very technical affair.

A serious issue is the *mismatch of operational semantics*. The two systems compute the same results, but with different rewriting rules, and linear logic is far from having the nice rewriting properties of the λ-calculus. Typically, the λ-calculus has a *residual system* [43][2], which is a strong form of confluence that allows building its famous advanced rewriting theory, given by standardisation, neededness, and Lévy's optimality [33]. In the ordinary presentations of linear logic cut-elimination is confluent but it does not admit residual systems[3], and so the advanced rewriting properties of the λ-calculus are lost. Put differently, linear logic is a structural refinement of the λ-calculus but it is far from refining it at the rewriting level.

A final point is the *mismatch of representations*: proofs in linear logic are usually manipulated in their graphical form, that is, as proof nets, and, while this is a handy formalism for intuitions, it is not amenable to formal reasoning—it is not by chance that there is not a single result about proof nets formalised in a proof assistant. And as already pointed out, the parallelism provided by proof nets, in the case of the λ-calculus, shows up only in the nets obtained as intermediate steps of the simulation of β, and so it cannot easily be seen on the λ-calculus. There is a way of expressing it, known as σ-equivalence, due to Regnier [42], but it is far from being natural.

The linear substitution calculus. The linear substitution calculus (LSC) [2,8] is a refinement of the λ-calculus with sharing, introduced by Accattoli and Kesner as a minor variation over a calculus by Milner [39], and meant to correct all these problems at once.

The LSC has been introduced in 2012 and then used in different settings—a selection of relevant studies concerning cost models, standardisation, abstract machines, intersection types, call-by-need, the π-calculus, and Lévy's optimality is [9,8,7,30,26,3,14]. The two design features of the LSC are its tight relationship with proof nets and the fact of having a residual system. The matching with proof nets, despite being one of the two reasons to be of the LSC, for some reason was never developed in detail, nor published. This paper corrects the situation, strengthening a growing body of research.

Contributions. The main result of the paper is the perfect correspondence between the LSC and the fragment of linear logic representing the λ-calculus. To this goal, the presentation of proof nets has to be adjusted, because the fault for the mismatch is not always on the calculus side. To overcome the mismatch of styles, we adopt a presentation of

[1] α-equivalence is subtle on terms, but this is an orthogonal issue, and a formal approach to proof net should also deal with α-equivalence for nodes, even if this is never done.

[2] For the unacquainted reader: having a residual system means to be a well-behaved rewriting system—related concepts are orthogonal systems, or the parallel moves or cube properties.

[3] Some presentations of proof nets (*e.g.* Regnier's in [41]) solve the operational semantics mismatch adapting proof nets to the λ-calculus, and do have residuals, but then they are unable to express typical micro-step proof nets concepts such as linear head reduction.

proof nets—already at work by the author [5]—that intuitively corresponds to interaction nets (to work modulo cut with axioms) with *hyper-wires*, that is, wires connecting more than two ports (to have smooth contractions). Our presentation of proof nets also refines the one in [5] with a micro-step operational semantics. Our exponential rewriting rules are slightly different than the others in the literature, and look more as the replication rule of the π-calculus—this is the key change for having a residual system.

Essentially, the LSC and our proof nets presentation are isomorphic. More precisely, our contribution is to establish the following tight correspondence:

1. *Transferable syntaxes*: every term translates to a proof net, and every proof net reads back to at least one term, removing the mismatch of states. We rely on a correctness criterion—Laurent's one for polarised proof nets [32,31]—to characterise proof nets and read them back. There can be many terms mapping to the same proof net, so at this level the systems are not isomorphic.
2. *Quotient*: we characterise the simple equivalence $\equiv$ on terms that is induced by the translation to proof nets. The quotient of terms by $\equiv$ is then isomorphic to proof nets, refining the previous point. The characterisation of the quotient is not usually studied in the literature on proof nets.
3. *Isomorphic micro-step operational semantics*: a term t and its associated proof net P have redexes in bijection, and such a bijection is a strong bisimulation: one step on one side is simulated by exactly one step on the other side, and vice-versa, and in both cases the reducts are still related by translation and read back. Therefore, the mismatch of operational semantics also vanishes.
   The fact that the LSC has a residual system is proved in [8], and it is not treated here. But our results allow to smoothly transfer the residual system from the LSC to our presentation of proof nets.

These features allow to consider the LSC modulo $\equiv$ as an algebraic—that is, not graphical—reformulation of proof nets for the λ-calculus, providing the strongest possible solution to the mismatch of representations. At the end of the paper, we also suggest a new perspective on proof nets from a rewriting point of view, building on our approach.

The value of this paper. This work is a bit more than the filling of a gap in the literature. The development is detailed, and so necessarily technical, and yet clean. The study of correctness and sequentialisation is stronger than in other works in the literature, because beyond sequentialising we also characterise the quotient—the proof of the characterisation is nonetheless pleasantly simple. Another unusual point is the use of *context nets* corresponding to the contexts of the calculus, that are needed to deal with the rules of the LSC. Less technically, but maybe more importantly, the paper ends with the sketch of a new and high-level rewriting perspective on proof nets.

Proofs. For lack of space, all proofs are omitted. They can be found in the technical report [6].

1.1 Historical Perspective

The fine match between the LSC and proof nets does not come out of the blue: it rather is the final product of a decades-long quest for a canonical decomposition of the λ-calculus.

At the time of the introduction of linear logic, decompositions of the λ-calculus arose also from other contexts. Abadi, Cardelli, Curien, and Lévy introduced calculi with *explicit substitutions* [1], that are refinements of the λ-calculus where meta-level substitution is delayed, by introducing explicit annotations, and then computed in a micro-step fashion. A decomposition of a different nature appeared in concurrency, with the translations of the λ-calculus to the π-calculus [37], due to Milner.

These settings introduce an explicit treatment of *sharing*—called *exponentials* in linear logic, or explicit substitutions, or *replication* in the π-calculus. The first calculus of explicit substitutions suffered of a design issue, as showed by Melliès in [36]. A turning point was the link between explicit substitutions and linear logic proof nets by Di Cosmo and Kesner in [19]. Kesner and co-authors then explored the connection in various directions [20,28,29]. In none of these cases, however, do terms and proof nets behave exactly the same.

The graphical representation of λ-calculus based on linear logic in [10] induced a further calculus with explicit substitutions, the *structural λ-calculus* [11], isomorphic to their presentation of proof nets. The structural λ-calculus corrects most mentioned mismatches, but it lacks a residual system.

Independently, Milner developed a graphical framework for concurrency, *bigraphs* [38], able to represent the π-calculus and, consequently, the λ-calculus. He extracted from it a calculus with explicit substitutions [39,27], similar in spirit to the structural λ-calculus. Accattoli and Kesner later realised that Milner's calculus has a residual system. In 2011-12, they started to work on the LSC, obtained as a merge of Milner's calculus and the structural λ-calculus.

At first, the LSC was seen as a minor variation over existing systems. With time, however, a number of properties arose, and the LSC started to be used as a sharp tool for a number of investigations. Two of them are relevant for our story. First, the LSC also allows refining the relationship between the λ-calculus and the π-calculus, as shown by the author in [3]. The LSC can then be taken as the harmonious convergence and distillation of three different approaches—linear logic, explicit substitutions, and the π-calculus—at decomposing the λ-calculus. Second, Lévy's optimality adapts to the LSC as shown by Barenbaum and Bonelli in [14], confirming that the advanced rewriting theory of the λ-calculus can indeed be lifted to the micro-step granularity via the LSC.

1.2 Related Work on Proof Nets

The relationship between λ-calculi and proof nets has been studied repeatedly, beyond the already cited work (Danos & Regnier, Kesner & co-authors, Accattoli & Guerrini). A nice and detailed introduction to the relationship between λ-terms and proof nets is [24].

Laurent extends the translation to represent the $\lambda\mu$-calculus in [32,31]. In this paper we use an adaptation of his correctness criterion. The translation of differential / resource calculi has also been studied at length: Ehrhard and Regnier [21] study the case without the promotion rule, while Vaux [46] and Tranquilli [44,45] include promotion. Vaux also extends the relationship to the classical case (thus encompassing a differential $\lambda\mu$-calculus), while Tranquilli refines the differential calculus into a *resource calculus* that better matches proof nets. Vaux and Tranquilli use interaction nets to circumvent the minor issue of cuts with axioms.

Strategies rather than calculi are encoded in interaction nets in [34].

None of these works uses explicit substitutions, so they all suffer of the *mismatch of states*. Explicit substitutions are encoded in proof nets in [22], but the operational semantics are not isomorphic, nor correctness is studied. An abstract machine akin to the LSC is mapped to proof nets in [40], but the focus is on cost analyses, rather than on matching syntaxes.

Other works that connect λ-calculi and graphical formalisms with some logical background are [13,25].

An ancestor of this paper is [5], that adopts essentially the same syntax for proof nets. In that work, however, the operational semantics is small-step rather than micro-step, there is no study of the quotient, and no use of contexts, nor it deals with the LSC.

2 The Linear Substitution Calculus

Expressions and terms. One of the features of the LSC is the use of contexts to define the rewriting rules. Contexts are terms with a single occurrence of a special constructor called *hole*, and often noted $\langle\cdot\rangle$, that is a placeholder for a removed subterm. To study the relationship with proof nets, it is necessary to represent both terms and contexts, and, to reduce the number of cases in definitions and proofs, we consider a syntactic category generalizing both. *Expressions* may have 0, 1, or more holes. Proof nets also require holes to carry the set Δ of variables that can appear free in any subterm replacing the hole—*e.g.* $\Delta = \{x, y, z\}$. Expressions are then defined as follows:

$$\text{Expressions} \qquad e, f, g, h ::= x \mid \langle\cdot\rangle_\Delta \mid \lambda x.e \mid ef \mid e[x{\leftarrow}f]$$

Terms are expressions without holes, noted t, s, u, and so on, and *contexts* are expressions with exactly one hole, noted C, D, E, etc.

The construct $t[x{\leftarrow}s]$ is an *explicit substitution*, shortened *ES*, of s for x in t—essentially, it is a more compact notation for $\texttt{let}\ x = s\ \texttt{in}\ t$. Both $\lambda x.t$ and $t[x{\leftarrow}s]$ bind x in t. Meta-level, capture-avoiding substitution is rather noted $t\{x{\leftarrow}s\}$. On terms, we silently work modulo α-equivalence, so that for instance $(\lambda x.((xyz)[y{\leftarrow}x])\{z{\leftarrow}xy\} = \lambda x'.((x'y'(xy))[y'{\leftarrow}x'])$. Applications associate to the left. Free variables of holes are defined by $\texttt{fv}(\langle\cdot\rangle_\Delta) := \Delta$, and for the other constructors as expected. The *multiplicity* of a variable x in a *term* t, noted $|t|_x$, is the number of free occurrences of x in t.

Contexts. The LSC uses contexts extensively, in particular *substitution contexts*:

$$\text{Substitution contexts} \qquad L, L', L'' ::= \langle\cdot\rangle_\Delta \mid L[x{\leftarrow}t]$$

Sometimes we write C_Δ for a context C whose hole $\langle\cdot\rangle_\Delta$ is annotated with Δ, and we call Δ the *interface* of C. Note that the free variables of C_Δ do not necessarily include those in its interface Δ, because the variables in Δ can be captured by the binders in C_Δ.

The basic operation over contexts is *plugging* of an expression e in the hole of the context C, that produces the expression $C\langle e\rangle$. The operation is defined only when the free variables $\texttt{fv}(e)$ of e are included in the interface of the context.

PLUGGING OF e IN C_Δ (ASSUMING $\mathtt{fv}(e) \subseteq \Delta$)

$$\begin{array}{rlrl} \langle e\rangle_\Delta &:= e & (\lambda x.C)\langle e\rangle &:= \lambda x.C\langle e\rangle \\ (Cs)\langle e\rangle &:= C\langle e\rangle s & (sC)\langle e\rangle &:= sC\langle e\rangle \\ (C[x{\leftarrow}s])\langle e\rangle &:= C\langle e\rangle[x{\leftarrow}s] & (s[x{\leftarrow}C])\langle e\rangle &:= s[x{\leftarrow}C\langle e\rangle] \end{array}$$

An example of context is $C_{\{x,y\}} := \lambda x.(y\langle\cdot\rangle_{\{x,y\}}[z{\leftarrow}x])$, and one of plugging is $C_{\{x,y\}}\langle xx\rangle = \lambda x.(y(xx)[z{\leftarrow}x])$. Note the absence of side conditions in the cases for $\lambda x.C$ and $C[x{\leftarrow}s]$—it means that plugging in a context can capture variables, as in the given example. Clearly, $C\langle e\rangle$ is a term / context if and only if e is a term / context. Note also that if t is a term and s is a subterm of t then $t = C\langle s\rangle$ for some context C. Such a context C is unique up to the annotation Δ of the hole of C, which only has to satisfy $\mathtt{fv}(s) \subseteq \Delta$, and that can always be satisfied by some Δ.

We also define *the set* $\mathtt{cv}(C_\Delta)$ *of variables captured by a context* C_Δ:

VARIABLES CAPTURED BY A CONTEXT

$$\begin{array}{rcl} \mathtt{cv}(\langle\cdot\rangle_\Delta) & := & \emptyset \\ \mathtt{cv}(\lambda x.C_\Delta) = \mathtt{cv}(C_\Delta[x{\leftarrow}t]) & := & \mathtt{cv}(C_\Delta) \cup \{x\} \\ \mathtt{cv}(tC_\Delta) = \mathtt{cv}(C_\Delta t) = \mathtt{cv}(t[x{\leftarrow}C_\Delta]) & := & \mathtt{cv}(C_\Delta) \end{array}$$

Rewriting rules for terms. The rewriting rules of the LSC concern terms only. They are unusual as they use contexts in two ways: to allow their application anywhere in the term—and this is standard—and to define the rules at top level—this is less common (note the substitution context L and the context C in rules $\to_{\mathtt{m}}$ and $\to_{\mathtt{e}}$ below). We write $C\langle\!\langle t\rangle\!\rangle$ if C does not capture any free variable of t, that is, if $\mathtt{cv}(C) \cap \mathtt{fv}(t) = \emptyset$.

REWRITING RULES

$$\begin{array}{rlll} \text{MULTIPLICATIVE} & L\langle\lambda x.t\rangle s \to_{\mathtt{m}} L\langle t[x{\leftarrow}s]\rangle & & \\ \text{MILNER EXPONENTIAL} & C\langle\!\langle x\rangle\!\rangle[x{\leftarrow}s] \to_{\mathtt{e}} C\langle\!\langle s\rangle\!\rangle[x{\leftarrow}s] & & \\ \text{GARBAGE COLLECTION} & t[x{\leftarrow}s] \to_{\mathtt{gc}} t & & \text{if } x \notin \mathtt{fv}(t) \\ \text{CONTEXTUAL CLOSURES} & \dfrac{t \to_a t'}{C\langle t\rangle \to_a C\langle t'\rangle} & \text{for } a \in \{\mathtt{m}, \mathtt{e}, \mathtt{gc}\} & \\ \text{NOTATION} & \to_{LSC} := \to_{\mathtt{m}} \cup \to_{\mathtt{e}} \cup \to_{\mathtt{gc}} & & \end{array}$$

Note that in $\to_{\mathtt{m}}$ (resp. $\to_{\mathtt{e}}$) we assume that L (resp. C) does not capture variables in $\mathtt{fv}(s)$—this is always possible by a (on-the-fly) α-renaming of $L\langle\lambda x.t\rangle$ (resp. $C\langle\!\langle x\rangle\!\rangle$), as we work modulo α. Similarly the interface of C can always be assumed to contain $\mathtt{fv}(s)$.

Structural equivalence. The LSC is sometimes enriched with the following notion of structural equivalence $\equiv$ [8].

Definition 1 (Structural equivalence). *Structural equivalence* $\equiv$ *is defined as the symmetric, reflexive, transitive, and contextual closure of the following axioms:*

$$\begin{array}{rll} (\lambda y.t)[x{\leftarrow}s] \equiv_\lambda & \lambda y.t[x{\leftarrow}s] & \text{if } y \notin \mathtt{fv}(s) \\ (t\,u)[x{\leftarrow}s] \equiv_{@l} & t[x{\leftarrow}s]\,u & \text{if } x \notin \mathtt{fv}(u) \\ t[x{\leftarrow}s][y{\leftarrow}u] \equiv_{com} & t[y{\leftarrow}u][x{\leftarrow}s] & \text{if } y \notin \mathtt{fv}(s) \text{ and } x \notin \mathtt{fv}(u) \end{array}$$

Its key property is that it commutes with evaluation in the following strong sense.

Proposition 2 (≡ is a strong bisimulation wrt $\to_{LSC}$ [8]). *Let $a \in \{\mathtt{m}, \mathtt{e}, \mathtt{gc}\}$. If $t \equiv s \to_a u$ then exists r such that $t \to_a r \equiv u$.*

Essentially, ≡ never creates redexes, it can be postponed, and vanishes on normal forms (that have no ES). We are going to prove that ≡ is exactly the quotient induced by translation to proof nets (Theorem 17, page 15). The absence of the axiom $(t\,u)[x\leftarrow s] \equiv_{@r} t\,u[x\leftarrow s]$ if $x \notin \mathtt{fv}(t)$ is correct: the two terms do not have the same proof net representation (defined in the next section), moreover adding this axiom to ≡ breaks Proposition 2. The extension with $\equiv_{@r}$ has nonetheless been studied in [12].

3 Proof Nets

Introduction. Our presentation of proof nets, similar to the one in [5], is nonstandard in at least four points—we suggest to have a quick look to Fig. 3, page 12:

1. *Hyper-graphs*: we use directed hyper-graphs (for which formulas are nodes and links—*i.e.* logical rules—are hyper-edges) rather than the usual graphs with pending edges (for which formulas are edges and links are nodes). We prefer hyper-graphs—that despite the scaring name are nothing but bipartite graphs—because they give
   (a) *Contraction algebra for free*: contraction is represented modulo commutativity, associativity, and permutation with box borders *for free*, by admitting that exponential nodes can have more than one incoming link,
   (b) *Cut-axiom quotient for free*: cut and axiom links are represented implicitly, collapsing them on nodes. This is analogous to what happens in interaction nets. Intuitively, our multiplicative nodes are *wires*, with exponential nodes being *hyper*-wires, *i.e.* wires involving an arbitrary number of ports;
   (c) *Subnets as subsets*: subnets can be elegantly defined as subsets of links, which would not be possible when adopting other approaches such as generalized ?-links or a standard interaction nets formalism without hyper-wires.

   The choice of hyper-graphs, however, has various (minor) technical consequences, and the formulation of some usual notions (*e.g.* the nesting condition for boxes) shall be slightly different with respect to the literature.
2. *Directed links and polarity*: our links are directed and we apply a correctness criterion based on directed paths. Be careful, however, that we do not follow the usual premises-to-conclusions orientation for links, nor the input-output orientation sometimes at work for λ-calculi or intuitionistic settings. We follow, instead, the orientation induced by logical polarity according to Laurent's correctness criterion for polarised proof nets [32,31]. Let us point out that Laurent defines proof nets using the premises-to-conclusions orientation and then he switches to the polarised orientation for the correctness criterion. We prefer to adopt only one orientation, the polarised one, which we also employ to define proof nets.
3. *Syntax tree*: since we use proof nets to represent terms, we arrange them on the plane according to the syntax tree of the corresponding terms, and not according to the corresponding sequent calculus proof, analogously to the graph rewriting literature on the λ-calculus (*e.g.* [47]) but in contrast to the linear logic literature.

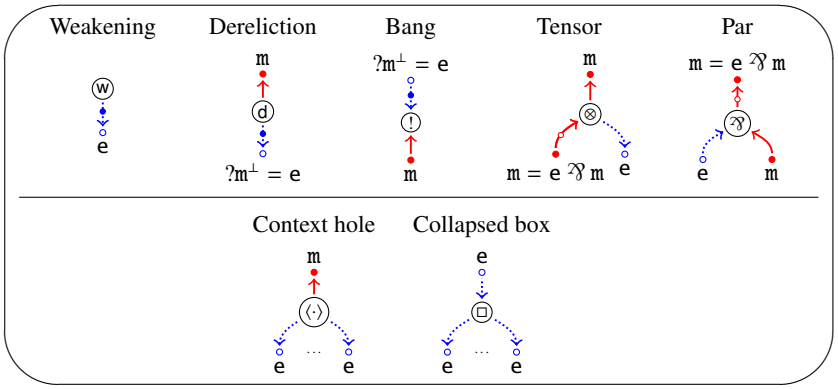

Fig. 1. Links.

4. *Contexts*: to mimic the use of contexts in the LSC rewriting rules, we need to have a notion of context net. Therefore, we have a special link for context holes.

Nets. We first overview some choices and terminology.

- *Hyper-graphs*: nets are directed and labelled hyper-graphs $G = (\mathtt{nodes}(G), \mathtt{links}(G))$, *i.e.*, graphs where $\mathtt{nodes}(G)$ is a set of labelled *nodes* and $\mathtt{links}(G)$ is a set of labelled and *directed hyper-edges*, called *links*, which are edges with 0, 1, or more sources and 0, 1, or more targets[4].
- *Nodes*: nodes are labelled with a type in $\{\mathtt{e}, \mathtt{m}\}$, where $\mathtt{e}$ stands for *exponential* and $\mathtt{m}$ for *multiplicative*. If a node u has type $\mathtt{e}$ (resp. $\mathtt{m}$) we say that it is a $\mathtt{e}$-node (resp. $\mathtt{m}$-node). The label of a node is usually left implicit, as $\mathtt{e}$ and $\mathtt{m}$ nodes are distinguished graphically, using both colours and different shapes: $\mathtt{e}$-nodes are cyan and white-filled, while $\mathtt{m}$-nodes are brown and dot-like. We come back to types below.
- *Links*: we consider hyper-graphs whose links are labelled from $\{!, \mathsf{d}, \mathsf{w}, \parr, \otimes, \langle\cdot\rangle, \Box\}$, corresponding to the promotion, dereliction, weakening, par, and tensor rules of linear logic, plus a link $\langle\cdot\rangle$ for context holes and a link $\Box$ used for defining the correction graph—contraction is hard-coded on nodes, as already explained. The label of a link l forces the number and the type of the source and target nodes of l, as shown in Fig. 1 (types shall be discussed next). Similarly to nodes, we use colours and shapes for the type of the source/target connection of a link to a node: $\mathtt{e}$-connections are blue and dotted, while $\mathtt{m}$-connections are red and solid. Our choice of shapes allows reading the paper also if printed in black and white.
- *Principal conclusions*: note that every link except $\langle\cdot\rangle$ and $\Box$ has exactly one connection with a little circle: it denotes the *principal* node, *i.e.* the node on which the link can interact. Notice the principal node for tensor and !, which is not misplaced.

[4] A hyper-graph G can be understood as a bipartite graph B_G, where $V_1(B_G)$ is $\mathtt{nodes}(G)$ and $V_2(B_G)$ is $\mathtt{links}(G)$, and the edges are determined by the relations *being a source* and *being a target* of a hyper-edge.

- *Typing*: nets are typed using a recursive type, usually noted $o = !o \multimap o$, but that we rename $\mathtt{m} = !\mathtt{m} \multimap \mathtt{m} = ?\mathtt{m}^{\perp} \parr \mathtt{m}$ because $\mathtt{m}$ is a mnemonic for *multiplicative*. Let $\mathtt{e} := ?\mathtt{m}^{\perp}$, where $\mathtt{e}$ stands for *exponential*. Note that $\mathtt{m} = \mathtt{e}^{\perp} \multimap \mathtt{m} = \mathtt{e} \parr \mathtt{m}$. Links are typed using $\mathtt{m}$ and $\mathtt{e}$, but the types are omitted by all figures except Fig. 1 because they are represented using colours and with different shapes ($\mathtt{m}$-nodes are brown and dot-like, $\mathtt{e}$-nodes are white-filled cyan circles). Let us explain the types in Fig. 1. They may be counter-intuitive at first: note in particular the ! and $\otimes$ links, that have an unexpected type on their logical conclusion—it simply has to be negated, because the expected orientation would be the opposite one.
- *More on nodes*: a node is *initial* if it is not the target of any link; *terminal* if it is not the source of any link; *isolated* if it is initial and terminal; *internal* if it is not initial nor terminal.
- *Boxes*: every !-link has an associated *box*, *i.e.*, a sub-hyper-graph of P (have a look at Fig. 3), meant to be a sub-net.
- *Context holes and collapsed boxes*: it is natural to wonder if $\langle\cdot\rangle$ and $\square$ links can be merged into a single kind of link. They indeed play very similar roles, except that they have different polarised typings, which is why we distinguish them.

We first introduce *pre-nets*, and then add boxes on top of them, obtaining *nets*:

Definition 3 (Pre-nets). *A* pre-net *P is a triple $(|P|, \mathtt{fv}(P), r_P)$, where $|P|$ is a hyper-graph $(\mathtt{nodes}(P), \mathtt{links}(P))$ whose nodes are labelled with either $\mathtt{e}$ or $\mathtt{m}$ and whose hyper-edges are $\{!, \mathtt{d}, \mathtt{w}, \parr, \otimes, \langle\cdot\rangle, \square\}$-links, and such that:*

- Root*: $r_P \in \mathtt{nodes}(P)$ is a terminal $\mathtt{m}$-node of P, called the* root *of P.*
- Free variables*: $\mathtt{fv}(P)$ is the set of terminal $\mathtt{e}$-nodes of P, also called* free variables *of P, which are targets of $\{\mathtt{d}, \mathtt{w}, \langle\cdot\rangle, \square\}$-links (*i.e. *they are not allowed to be targets of $\otimes$-links, nor to be isolated).*
- Nodes*: every node has* at least *one incoming link and* at most one *outgoing link. Moreover,*
  - Multiplicative*: $\mathtt{m}$-nodes have* exactly one *incoming link;*
  - Exponential*: if an $\mathtt{e}$-node has more than one incoming link then they are $\mathtt{d}$-links.*

Definition 4 (Nets). *A* net *P is a pre-net together with a function $\mathtt{ibox}_P$ (or simply $\mathtt{ibox}$) associating to every !-link l a subset $\mathtt{ibox}(l)$ of $\mathtt{links}(P) \setminus \{l\}$ (*i.e. *the links of P except l itself), called the* interior of the box *of l, such that $\mathtt{ibox}(l)$ is a pre-net verifying (explanations follow):*

- Border*: the root $r_{\mathtt{ibox}(l)}$ is the source $\mathtt{m}$-nodes of l, and any free variable of $\mathtt{ibox}(l)$ is not the target of a weakening.*
- Nesting*: for any !-box $\mathtt{ibox}(h)$ if $\mathtt{ibox}(l)$ and $\mathtt{ibox}(h)$ have non-empty intersection—that is, if $\emptyset \neq I := |\mathtt{ibox}(l)| \cap |\mathtt{ibox}(h)|$—and one is not entirely contained in the other—that is, if $|\mathtt{ibox}(l)| \not\subseteq |\mathtt{ibox}(h)|$, and $|\mathtt{ibox}(h)| \not\subseteq |\mathtt{ibox}(l)|$—then all the nodes in I are free variables of both $\mathtt{ibox}(l)$ and $\mathtt{ibox}(h)$.*
- Internal closure*:*
  - Contractions*: if a contraction node is internal to $\mathtt{ibox}(l)$ then all its premises are in $\mathtt{ibox}(l)$—formally, $h \in \mathtt{ibox}(l)$ for any link h of P having as target an internal $\mathtt{e}$-node of $\mathtt{ibox}(l)$.*

- Boxes*:* $\mathtt{ibox}(h) \subseteq \mathtt{ibox}(l)$ *for any* !*-link* $h \in \mathtt{ibox}(l)$*.*

A net is

- *a* term net *if it has no* $\{\langle\cdot\rangle, \Box\}$*-links;*
- *a* context net *if it has exactly one* $\langle\cdot\rangle$*-link;*
- *a* correction net *if it has no* !*-links.*

As for the calculus, the interface *of a* $\langle\cdot\rangle$*-link is the set of its free variables, and the interface of a context net is the interface of its* $\langle\cdot\rangle$*-link.*

Remark 5. Comments on the definition of net:

1. *Weakenings and box borders*: in the border condition for nets the fact that the free variables are not the target of a weakening means that weakenings are assumed to be pushed out of boxes as much as possible—of course the rewriting rules shall have to preserve this invariant.
2. *Weakenings are not represented as nullary contractions*: given the representation of contractions, it would be tempting to define weakenings as nullary contractions. However, such a choice would be problematic with respect to correctness (to be defined soon), as it would introduce many initial $\mathtt{e}$-nodes in a correct net and thus blur the distinction between the root of the net, supposed to represent the output and to be unique (in a correct net), and substitutions on a variable with no occurrences (*i.e.* weakened subterms), that need not to be unique.
3. *Internal closure wrt contractions*: it is a by-product of collapsing contractions on nodes, which is also the reason for the unusual formulation of the nesting condition. In fact, two boxes that are intuitively disjoint can in our syntax share free variables, because of an implicit contraction merging two of them, as in the example in Fig. 3.
4. *Boxes as nets*: note that a box $\mathtt{ibox}(l)$ in a net P is only a *pre*-net, by definition. Every box in a net P, however, inherits a net structure from P. Indeed, one can restrict the box function $\mathtt{ibox}_P$ of P to the !-links of $\mathtt{ibox}(l)$, and see $\mathtt{ibox}(l)$ as a *net*, because all the required conditions are automatically satisfied by the internal boxes closure and by the fact that such boxes are boxes in P. Therefore, we freely consider boxes as *nets*.
5. *Tensors and* !*-boxes*: the requirements that the $\mathtt{e}$-target of a $\otimes$-link cannot be the free variable of a net, nor the target of more than one link force these nodes to be sources of !-links. Therefore, every $\otimes$-link is paired to a !-link, and thus a box.
6. *Acyclic nesting*: the fact that a !-link does not belong to its box, plus the internal closure condition, imply that the nesting relation between boxes cannot be cyclic, as we now show. Let l and h be !-links. If $l \in \mathtt{ibox}(h)$ then by internal closure $\mathtt{ibox}(l) \subseteq \mathtt{ibox}(h)$. It cannot then be that $h \in \mathtt{ibox}(l)$, otherwise l would belong to its own box, because $l \in \mathtt{ibox}(h) \subseteq \mathtt{ibox}(l)$ by internal closure.

Terminology about nets. Some further terminology and conventions:

- The *level* of a node/link/box is the maximum number of nested boxes in which it is contained[5] (a !-link is not contained in its own box). Note that the level is well

[5] Here the words *maximum* and *nested* are due to the fact that the free variables of !-boxes may belong to two not nested boxes, as in the example in Fig. 3, because of the way we represent contraction.

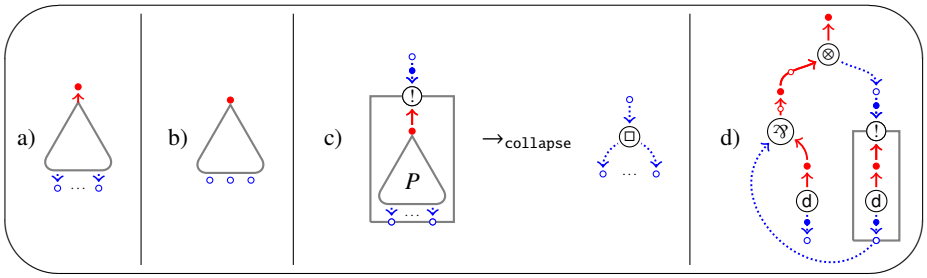

Fig. 2. Various images.

defined by the acyclicity of nesting just pointed out. In particular, if a net has !-links then it has at least one !-link at level 0.
- A *variable* x is a e-node that is the target of a $\{\mathsf{d},\mathsf{w}\}$-link—equivalently, that is not the target of a $\otimes$-link.
- Two links are *contracted* if they share an e-target. Note that the exponential condition states that only derelictions (*i.e.* d-links) can be contracted. In particular, no link can be contracted with a weakening.
- A *free weakening* in a net P is a weakening whose node is a free variable of P.
- The *multiplicity* of a variable x in P, noted $|P|_x$, is 0 if x is the target of a weakening, and $n \geq 1$ if it is the target of n derelictions.
- Sometimes (*e.g.* the bottom half of Fig. 3), the figures show a link in a box having as target a contracted e-node x which is outside the box: in those cases x is part of the box, it is outside of the box only in order to simplify the representation.

Translation. Nets representing terms have the general form in Fig. 2.a, also represented as in Fig. 2.b. The translation $\underline{\cdot}$ from expression to nets is in Fig. 3.

A net which is the translation of an expression is a *proof net*. Note the example in Fig. 3: two different terms translate to the same proof net, showing that proof nets quotient LSC terms.

The translation $\underline{\cdot}$ is refined to a translation $\underline{\cdot}_\Delta$, where Δ is a set of variables, in order to properly handle weakenings during cut-elimination. The reason is that an erasing step on terms simply erases a subterm, while on nets it also introduces some weakenings: without the refinement the translation would not be stable by reduction.

Note that in some cases there are various edges entering an e-node, that is the way we represent contraction. In some cases the e-nodes have an incoming connection with a perpendicular little bar: it represents an arbitrary number (> 0) of incoming connections. Structurally equivalent terms are translated to the same proof net, see Fig. 4 at page 15.

α-Equivalence. To circumvent an explicit and formal treatment of α-equivalence we assume that the set of e-nodes and the set of variable names for terms coincide. This convention removes the need to label the free variables of $\underline{t}_\Delta$ with the name of the corresponding free variables in t or Δ. Actually, before translating a term t it is necessary to pick a *well-named* α-equivalent term t', *i.e.* a term such that any two different variables (bound or free) have different names.

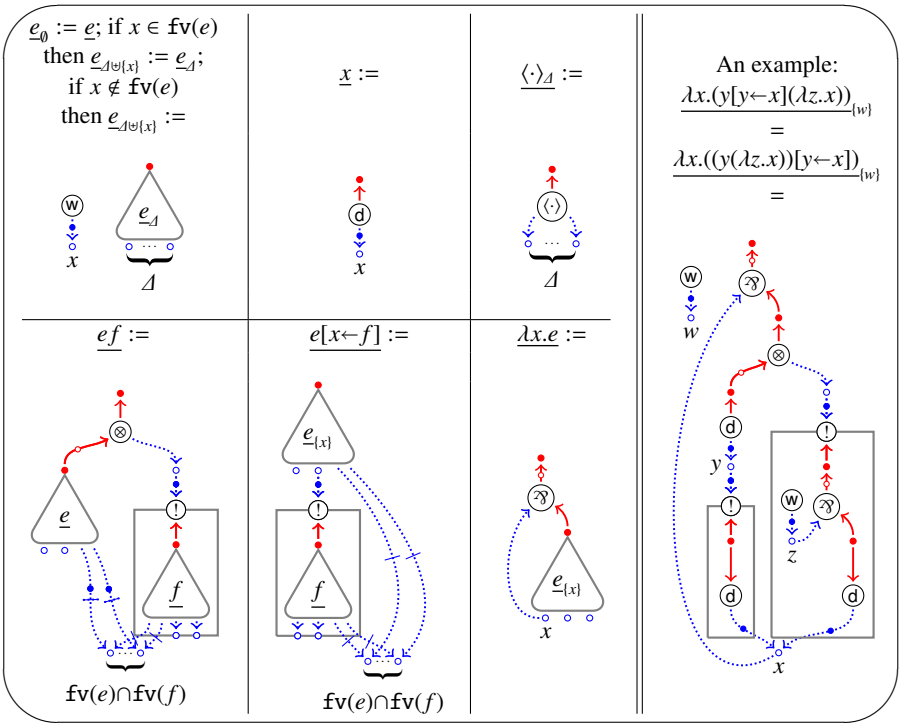

Fig. 3. Translation of expressions to nets, plus an example of translation.

Paths. A *path* τ of length $k \in \mathbb{N}$ from u to w, noted $\tau : u \to^k w$, is an alternated sequence of nodes and links $u = u_1, l_1, \ldots, l_k, u_{k+1} = w$ such that link l_i has source u_i and target u_{i+1} for $i \in \{1, \ldots, k\}$. A *cycle* is a path $u \to^k u$ with $k > 0$.

Correctness. The correctness criterion is an adaptation of Laurent's criterion for polarized nets, and it is the simplest known criterion for proof nets. It is based on the notion of correction net, which—as usual for nets with boxes—is obtained by collapsing boxes into generalized axiom links, *i.e.* our $\square$-links (see Fig. 1).

Definition 6 (Correction net). *Let P be a net. The* correction net P^0 *of P is the net obtained from P by collapsing each* !*-box at level 0 in P into a* $\square$*-link with the same interface, by applying the rule in Fig. 2.c.*

Definition 7 (Correctness). *A net P is* correct *if:*

- Root*: the root of P induces the only terminal* $\mathtt{m}$*-node of P^0.*
- Acyclicity*: P^0 is acyclic.*
- Recursive correctness*: the box of every* !*-link at level 0 is correct.*

An example of net that is not correct is in Fig. 2.d: the correction net obtained by collapsing the box indeed has a cycle.

Note that acyclicity provides an induction principle on correct nets, because it implies that there is a maximal length for paths in the correction net associated to the net.

Proof nets are correct. As usual, an easy and omitted induction on the translation shows that the translation of an expression is correct, *i.e.* that:

Proposition 8 (Proof nets are correct). *Let e be an expression and Δ a set of variables. Then $\underline{e}_\Delta$ is a correct net of free variables $\mathtt{fv}(e) \cup \Delta$. Moreover,*

1. *if e is a term then $\underline{e}_\Delta$ is a term net and their variables have the same multiplicity, that is, $|e|_x = |\underline{e}_\Delta|_x$ for every variable x.*
2. *if e is a context then $\underline{e}_\Delta$ is a context net.*

Linear skeleton. We have the following strong structural property.

Lemma 9 (Linear skeleton). *Let P be a correct net. The* linear skeleton *of P^0, given by* m*-nodes and the red (or linear) paths between them, is a linear order.*

4 Sequentialisation and Quotient

In this section we prove the sequentialisation theorem and the fact that the quotient induced by the translation on terms is exactly the structural equivalence $\equiv$ of the LSC.

Subnets. The first concept that we need is the one of *subnet* Q of a correct net P, that is a subset of the links of P plus some closure conditions. These conditions avoid that Q prunes the interior of a box in P, or takes part of the interior without taking the whole box, or takes only some of the premises of an internal contraction.

For the sake of simplicity, in the following we specify sub-hyper-graphs of a net by simply specifying their set of links. This is an innocent abuse, because—by definition of (pre-)net—there cannot be isolated nodes, and so the set of nodes is retrievable from the set of links. Similarly, the boxes of !-links are inherited from the net.

Definition 10 (Subnet). *Let P be a correct net. A* subnet *Q of P is a subset of its links such that it is a correct net (with respect to the* ibox *function inherited from P) and satisfies the following closure conditions:*

- Contractions*: $l \in Q$ for any link l of P having as target an internal* e*-node of Q.*
- Box interiors*: $\mathtt{ibox}(h) \subseteq Q$ for any !-link $h \in Q$.*
- Box free variables*: $\mathtt{ibox}(l) \subseteq Q$ if a free variable of $\mathtt{ibox}(l)$ is internal to Q.*

Decomposing correct nets. Sequentialisation shall read back an expression by progressively decomposing a correct net. We first need some terminology about boxes.

Definition 11 (Kinds of boxes). *Let P be a correct net. A !-link l of P is:*

- free *if it is at level 0 in P and its free variables are free variables of P.*
- *an* argument *if its* e*-node is the target of a $\otimes$-link;*

– *a* substitution *if its* e*-node is the target of a* $\{\mathsf{w}, \mathsf{d}, \langle\cdot\rangle\}$*-link (or, equivalently, if it is not the target of a* $\otimes$*-link).*

The following lemma states that, in correct nets whose root structure is similar to the translation of an expression, it is always possible to decompose the net in correct subnets. The lemma does not state the correctness of the interior of boxes because they are correct by definition of correctness.

Lemma 12 (Decomposition). *Let P be a correct net.*

1. Free weakening*: if P has a free weakening l then* $\mathtt{links}(P) \setminus l$ *is a subnet of P.*
2. Root abstraction*: if the root link l of P is a* $\parr$*-link then* $\mathtt{links}(P) \setminus l$ *is a subnet of P.*
3. Free substitution*: if P has a free substitution l then* $\mathtt{links}(P) \setminus (\{l\} \cup \mathtt{ibox}(l))$ *is a subnet of P.*
4. Root application with free argument*: if the root link l of P is a* $\otimes$*-link whose argument is a free* !*-link h then* $\mathtt{links}(P) \setminus (\{l, h\} \cup \mathtt{ibox}(h))$ *is a subnet of P.*

Definition 13 (Decomposable net). *A correct net P is decomposable if it is in one of the hypothesis of the decomposition lemma (Lemma 12), that is, if it has a free weakening, a root abstraction, a free substitution, or a root application with free argument.*

The last bit is to prove that every correct net is decomposable, and so, essentially corresponds to the translation of an expression.

Lemma 14 (Correct nets are decomposable). *Let P be a correct net with more than one link. Then P is decomposable.*

We now introduce the read back of correct net as expressions, which is the key notion for the sequentialisation theorem. Its definition relies, in turn, on the various ways in which a correct net can be decomposed, when it has more than one link.

Definition 15 (Read back). *Let P be a correct net and e be an expression. The relation e* is a read back of *P, noted* $P \rhd e$*, is defined by induction on the number of links in P:*

– One link term net*:* P *is a* d*-link of* e*-node* x*. Then* $P \rhd x$*;*
– One link context net*:* P *is a* $\langle\cdot\rangle$*-link of* e*-nodes* Δ*. Then* $P \rhd \langle\cdot\rangle_\Delta$*;*
– Free weakening*:* P *has a free weakening* l *and* $P \setminus l \rhd e$*. Then* $P \rhd e$*;*
– Root abstraction*: the root link* l *of* P *is a* $\parr$*-link of* e*-node* x *and* $P \setminus l \rhd e$*. Then* $P \rhd \lambda x.e$*;*
– Free substitution*:* P *has a free substitution* l *of* e*-node* x*,* $P \setminus (\{l\} \cup \mathtt{ibox}(l)) \rhd e$*, and* $\mathtt{ibox}(l) \rhd f$*. Then* $P \rhd e[x \leftarrow f]$*.*
– Root application with free argument*: the root link* l *of* P *is a* $\otimes$*-link whose argument is a free* !*-link* h*,* $P \setminus (\{l, h\} \cup \mathtt{ibox}(h)) \rhd e$*, and* $\mathtt{ibox}(h) \rhd f$*. Then* $P \rhd e\,f$*.*

We conclude the section with the sequentialisation theorem, that relates terms and proof nets at the static level. Its formulation is slightly stronger than similar theorems in the literature, that usually do not provide completeness.

Theorem 16 (Sequentialisation). *Let P be a correct net and* Δ *be the set of* e*-nodes of its free weakenings.*

1. Read backs exist*: there exists e such that* $P \rhd e$ *with* $\mathtt{fv}(e) = \mathtt{fv}(P)$*.*
2. The read back relation is correct*: for all expressions e,* $P \rhd e$ *implies* $\underline{e}_\Delta = P$ *and* $\mathtt{fv}(P) = \mathtt{fv}(e) \cup \Delta$*.*
3. The read back relation is complete*: if* $\underline{e}_\Gamma = P$ *then* $P \rhd e$ *and* $\Gamma \subseteq \mathtt{fv}(P) \cup \Delta$*.*

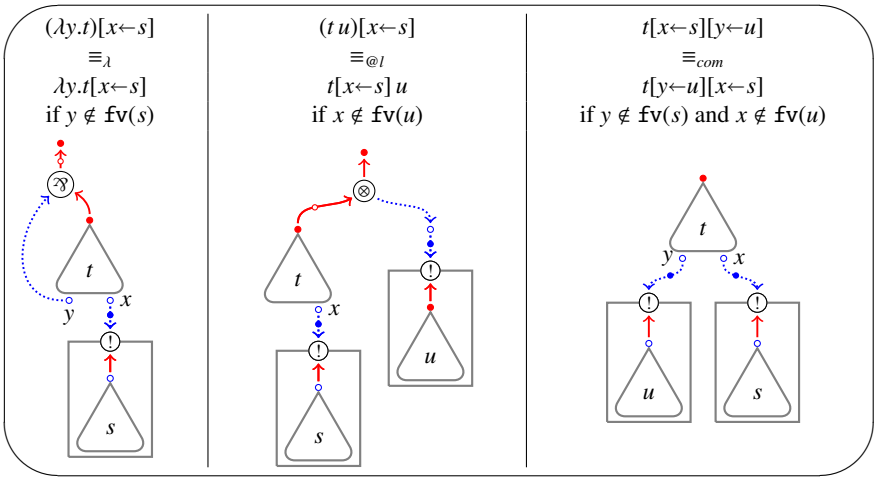

Fig. 4. Structural equivalent terms translate to the same proof nets (contractions of common variables are omitted).

Quotient. Next we prove that structural equivalence on the LSC is exactly the quotient induced by proof nets. We invite the reader to look at the proof of the following quotient theorem. The $\Leftarrow$ direction essentially follows from figure Fig. 4, where for simplicity we have omitted the contractions of common variables for the subnets. The $\Rightarrow$ direction is the tricky point. Note that $\equiv$-classes do not admit canonical representantives, because the $\equiv_{com}$ axiom is not orientable, and so it is not possible to rely on some canonical read back. The argument at work in the proof is however pleasantly simple.

Theorem 17 (Quotient). *Let P be correct term net. Then, $\underline{t} = P$ and $\underline{s} = P$ if and only if $t \equiv s$.*

5 Contexts

This short section develops a few notions about relating contexts in the two frameworks. We only deal with what is strictly needed to relate rewriting steps on terms and on term nets—a more general treatment is possible, but not explored here, for the sake of simplicity.

The plugging operation can also be done on context nets.

Definition 18 (Plugging on context nets). *Let P be a context net and let Δ be the free variables of its $\langle\cdot\rangle$-link l. The* plugging *of a net Q with free variables $\Gamma \subseteq \Delta$ in P is the net $P\langle Q\rangle$ obtained by*

- *if l is at level 0:*
  - Replacement*: replacing l with Q;*
  - Weakening unused variables in the interface*: adding a weakening h on every variable $x \in (\Delta \setminus \Gamma)$ not shared in P (or whose only incoming link in P is l).*

– *if l is in* $\texttt{ibox}(h)$ *for a* !*-link h at level 0 then:*
  - Recursive plugging*: replacing the links of* $\texttt{ibox}(h)$ *with those in* $\texttt{ibox}(h)\langle Q\rangle$*, inheriting the boxes;*
  - Pushing weakenings out of the box*: redefining* $\texttt{ibox}(h)$ *as* $\texttt{ibox}(h)\langle Q\rangle$ *less its free weakenings, if any.*

The next lemma relies plugging in context nets with the corresponding read backs.

Lemma 19 (Properties of context nets plugging). *Let P be a context net of interface Δ, Q a correct net with free variables $\Gamma \subseteq \Delta$. Then*

1. Correctness*: $P\langle Q\rangle$ is correct;*
2. Read back*: if $P \rhd C_\Delta$ and $Q \rhd e$ then $P\langle Q\rangle \rhd C_\Delta\langle e\rangle$.*

From the read back property, a dual property follows for the translation.

Lemma 20 (Context-free translation). *Let C_Δ a context, e an expression such that $\texttt{fv}(e) \subseteq \Delta$, and Γ a set of variables. Then $\underline{C_\Delta\langle e\rangle}_\Pi = \underline{C_\Delta}_\Gamma\langle \underline{e}\rangle$ where $\Pi = \Gamma \cup (\Delta \setminus \texttt{cv}(C_\Delta))$.*

The following lemma shall be used to relate the exponential steps in the two systems. The proof is a straightforward but tedious induction on $P \rhd C_\Delta$, which is omitted.

Lemma 21 (Read back and free variable occurrences). *Let $P \rhd t$ be a term net with a fixed read back, l be a* $\texttt{d}$*-link of P whose* $\texttt{e}$*-node x is a free variable of P. Then for every set of variable names Δ there are a context C and a context net Q, both of interface $\Delta \cup \{x\}$, such that*

1. Net factorisation*: $Q\langle l\rangle = P$;*
2. Term factorisation*: $C\langle\!\langle x\rangle\!\rangle = t$; and*
3. Read back*: $Q \rhd C$.*

6 Micro-Step Operational Semantics

Here we define the rewriting rules on proof nets and prove the isomorphism of rewriting systems with respect to the LSC. Since the rules of the LSC and those of proof nets match perfectly, we use the same names and the same notations for them.

The rules. The rewriting rules are in Fig. 5. Let us explain them. First of all, note that the notion of cut in our syntax is implicit, because cut-links are not represented explicitly. A cut is given by a node whose incoming and outgoing connections are principal (*i.e.* with a little dot on the line).

The multiplicative rule $\to_{\texttt{m}}$ is nothing but the usual elimination of a multiplicative cut, adapted to our syntax. The matching with the rule on terms is shown in Fig. 5.

The garbage collection rule $\to_{\texttt{gc}}$ corresponds to a cut with a weakening. It is mostly as the usual rule, the only difference is with respect to the reduct. The box of the !-link is erased and replaced by a set of weakenings, one for every free variable of Q—this is standard. Each one of these new weakenings is also pushed out of all the m_i boxes closing on its $\texttt{e}$-node. This is done to preserve the invariant that weakenings are always

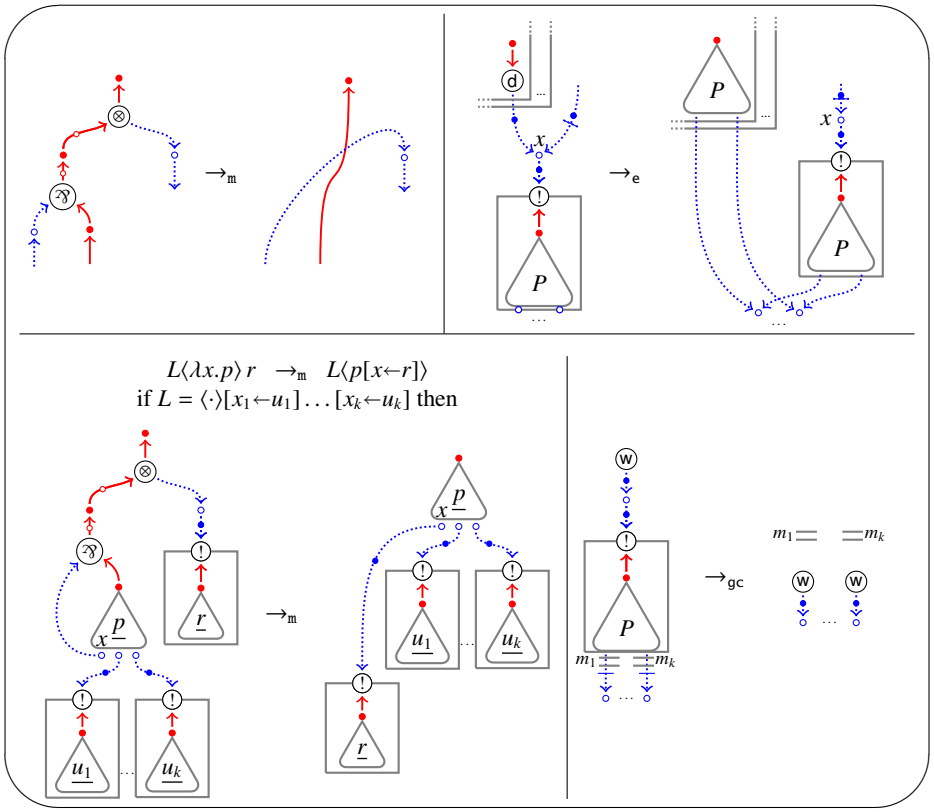

Fig. 5. Proof nets cut-elimination rules, plus—in the bottom-left corner—the matching of the multiplicative rule on terms and on term nets (forgetting, for simplicity, about the contraction of common variables for the boxes, and the fact that x_j can occur in u_i for $i < j$).

pushed out of boxes as much as possible. Such an invariant is also used in the rule: note that the weakening is at the same level of Q. Last, if the weakenings created by the rule are contracted with any other link then they are removed on the fly, because by definition weakenings cannot be contracted.

The Milner exponential rule $\to_{\mathtt{e}}$ is the most unusual rule, and—to our knowledge—it has never been considered before on proof nets. There are two unusual points about it. The first one is that the redex crosses box borders, as the d-link is potentially inside many boxes, while the !-link is out of those boxes. In the literature, this kind of rules is usually paired with a small-step operational semantics (*e.g.* in [41]), that is, all the copies of the box are done in a single shot. Here instead we employ a micro-step semantics, as also done in [4]—that paper contains a discussion about this *box-crossing principle* and its impact on the rewriting theory of proof nets.

The second unusual point is the way the cut is eliminated. Roughly, it corresponds to a duplication of the box (so a contraction cut-elimination) immediately followed by commutation with all the boxes and opening of the box (so a dereliction cut-elimination).

We say *roughly*, because there is a difference: the duplication happens also if the d-link is not contracted. Exactly as in the LSC, indeed, the $\to_{\mathsf{e}}$ rule duplicates the ES even if there are no other occurrences of the replaced variable. In case the d-link is not contracted, the rule puts a weakening on the e-node source of the !-link.

The isomorphism. Finally, we relate the evaluation of proof nets and of the LSC.

Theorem 22 (Dynamic isomorphism). *Let $P \rhd t$ be a correct net with a fixed read back, and $a \in \{\mathtt{m}, \mathtt{e}, \mathtt{gc}\}$. There is a bijection ϕ between a-redexes of t and P such that:*

1. Terms to proof nets*: given a redex $\gamma : t \to_a s$ then there exists Q such that $\phi(\gamma) : P \to_a Q$ and $Q \rhd s$.*
2. Proof nets to terms*: given a redex $\gamma : P \to_a Q$ then there exists s such that $\phi^{-1}(\gamma) : t \to_a s$ and $Q \rhd s$.*

From Theorem 22 it immediately follows that cut-elimination preserves correctness, because the reduct of a correct net is the translation of a term, and therefore it is correct.

Corollary 23 (Preservation of correctness). *Let P be a term net and $P \to Q$. Then Q is correct.*

The perfect matching also transfers to proof nets the residual system of the LSC defined in [8]. Finally, the dynamic isomorphism (Theorem 17) combined with the quotient theorem (Theorem 17) also provides a new proof of the strong bisimulation property of structural equivalence (Proposition 2).

7 Abstracting Proof Nets From a Rewriting Point of View

In this section we provide a new, rewriting-based perspective on proof nets.

Cut commutes with cut. One of the motivations for proof nets is the fact that cut-elimination in the sequent calculus has to face commutative cut-elimination cases. They are always a burden, but most of them are harmless. There is however at least one very delicate case, the commutation of cut with itself, given by:

$$\dfrac{\dfrac{\gamma\ \vdots\ \vdash \Gamma, B \qquad \dfrac{\pi\ \vdots\ \vdash \Gamma, A \qquad \theta\ \vdots\ \vdash \Gamma, A, B}{\vdash \Gamma, B}\ \text{cut}}{\vdash \Gamma}\ \text{cut} \quad\to\quad \dfrac{\pi\ \vdots\ \vdash \Gamma, A \qquad \dfrac{\gamma\ \vdots\ \vdash \Gamma, B \qquad \theta\ \vdots\ \vdash \Gamma, A, B}{\vdash \Gamma, A}\ \text{cut}}{\vdash \Gamma}\ \text{cut}$$

Such a commutation is delicate because it can be iterated, creating silly loops. If one studies weak normalisation (*i.e.* the *existence* of a normalising path) then it is enough to design a cut-elimination strategy that never commutes cut with itself—this is what is done in the vast majority of cut-elimination theorems. But if one is interested in strong normalisation (*i.e.*, *all* paths eventually normalise), then this is a serious issue. Morally, this is the conceptual problem behind proof nets and also behind the design of good explicit substitution calculi—it could be said that it is *the* rewriting issue of the Curry-Howard correspondence at the micro-step granularity.

One way to address this problem is to introduce an equivalence relation $\sim$ on proofs including the commutation of cut with itself, and then to switch to eliminate cuts modulo $\sim$. Rewriting modulo is a studied but technical and subtle topic, see [43] chapter 14.3. The problem is that cut-elimination $\to$ and $\sim$ in general do not interact nicely, in particular $\sim$ cannot be postponed, because it *creates* $\to$-redexes.

Proof nets are a different, more radical solution: a change of syntax in which $\sim$-classes collapse on a single object, the proof net, so that the problem of the interaction between $\to$ and $\sim$ disappears. Proof nets seem, at first, elegant objects, and certainly a brilliant solution to the problem, providing many new intuitions about proofs. They are however heavy to manipulate formally, and it would be often preferable to have an alternative, more traditional syntax with similar properties.

Structural rewriting systems. The LSC is the prototype of a finer solution to the problem of commuting cut with itself. In general, we said, $\to$ and $\sim$ do not interact nicely. However, it is sometimes possible to *redefine* $\to$ so as to interact nicely with $\sim$. Typically, the contextual rules of the LSC interact nicely with $\equiv$ ($\equiv$ is the equivalence $\sim$ of the LSC, note in particular that axiom $\equiv_{com}$ is exactly commutation of cut with itself)—this is the motivation behind contextual rules, sometimes also called *at a distance*. This suggests the following notion, which is a special case of rewriting modulo an equivalence relation.

Definition 24 (Structural rewriting system). *Let T be a set of objects, $\to$ a rewriting relation and $\sim$ an equivalence relation over T. The triple $(T, \to, \sim)$ is a structural rewriting system (modulo) if $\sim$ is a strong bisimulation with respect to $\to$.*

Note that the definition does not mention graphs. We can then see proof nets and the LSC as instances of a single concept.

Proposition 25. *let $\to_{PN}$ be the union of rules $\to_{\mathtt{m}}$, $\to_{\mathtt{e}}$, and $\to_{\mathtt{gc}}$ on proof nets.*

1. *Proof nets with $\to_{PN}$ are a structural rewriting sytem, by taking $\sim$ to be the identity.*
2. *The LSC with $\to_{LSC}$ and $\equiv$ is a structural rewriting sytem.*

Structural rewriting sytems can be exported to different settings, with no need to bother about correctness criteria or graphical presentations, or the existence of a logical interpretation. For instance, in [3] there is a structural presentation of a fragment of the π-calculus based on contextual rules, independently of any logical interpretation.

8 Conclusions

This paper provides a perfect matching between the LSC and a certain presentation of the fragment of linear logic representing the λ-calculus. In particular, we prove that proof nets can be identified with the LSC up to structural equivalence $\equiv$, enabling one to reason about proof nets by means of a non-graphical language.

We also discuss our approach with respect to the basic proof theoretical problem of the cut rule commuting with itself. We try to suggest that the idea behind our result goes beyond proof nets and the LSC, as it also applies to other settings where rewriting has to interact with a notion of structural equivalence such as the π-calculus.

Acknowledgments. To the reviewers, for useful comments. This work has been partially funded by the ANR JCJC grant COCA HOLA (ANR-16-CE40-004-01).

References

1. Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.
2. Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In *RTA*, pages 6–21, 2012.
3. Beniamino Accattoli. Evaluating functions as processes. In *TERMGRAPH*, pages 41–55, 2013.
4. Beniamino Accattoli. Linear logic and strong normalization. In *RTA*, pages 39–54, 2013.
5. Beniamino Accattoli. Proof nets and the call-by-value λ-calculus. *Theor. Comput. Sci.*, 606:2–24, 2015.
6. Beniamino Accattoli. Proof Nets and the Linear Substitution Calculus. *CoRR*, abs/1808.03395, 2018. `https://arxiv.org/abs/1808.03395`.
7. Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In *ICFP 2014*, pages 363–376, 2014.
8. Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In *POPL*, pages 659–670, 2014.
9. Beniamino Accattoli and Ugo Dal Lago. (Leftmost-Outermost) Beta-Reduction is Invariant, Indeed. *LMCS*, 12(1), 2016.
10. Beniamino Accattoli and Stefano Guerrini. Jumping boxes. In *CSL*, pages 55–70, 2009.
11. Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In *CSL*, pages 381–395, 2010.
12. Beniamino Accattoli and Delia Kesner. Preservation of strong normalisation modulo permutations for the structural λ-calculus. *Logical Methods in Computer Science*, 8(1), 2012.
13. Andrea Asperti and Cosimo Laneve. Comparing lambda-calculus translations in sharing graphs. In *TLCA '95*, pages 1–15, 1995.
14. Pablo Barenbaum and Eduardo Bonelli. Optimality and the linear substitution calculus. In *FSCD 2017*, pages 9:1–9:16, 2017.
15. V. Danos and L. Regnier. Proof-nets and the Hilbert space. In *Proceedings of the Workshop on Advances in Linear Logic*, pages 307–328, New York, NY, USA, 1995. Cambridge University Press.
16. Vincent Danos. *La Logique Linéaire appliqué à l'étude de divers processus de normalisation (principalement du λ-calcul)*. Phd thesis, Université Paris 7, 1990.
17. Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.
18. Vincent Danos and Laurent Regnier. Head linear reduction. Technical report, 2004.
19. Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *LICS*, pages 35–46, 1997.
20. Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. Proof nets and explicit substitutions. *Math. Str. in Comput. Sci.*, 13(3):409–450, 2003.
21. Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Electr. Notes Theor. Comput. Sci.*, 123:35–74, 2005.
22. Maribel Fernández and Nikolaos Siafakas. Labelled calculi of resources. *J. Log. Comput.*, 24(3):591–613, 2014.
23. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
24. Stefano Guerrini. Proof nets and the lambda-calculus. In *Linear Logic in Computer Science*, pages 65–118. Cambridge University Press, 2004.
25. Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda calculus: A typed lambda-calculus with explicit sharing. In *LICS*, pages 311–320, 2013.

26. Delia Kesner. Reasoning about call-by-need by means of types. In *FOSSACS 2016*, pages 424–441, 2016.
27. Delia Kesner and Shane Ó Conchúir. Milner's lambda calculus with partial substitutions. Technical report, Paris 7 University, 2008. `http://www.pps.univ-paris-diderot.fr/~kesner/papers/shortpartial.pdf`.
28. Delia Kesner and Stéphane Lengrand. Extending the explicit substitution paradigm. In *RTA*, pages 407–422, 2005.
29. Delia Kesner and Fabien Renaud. The prismoid of resources. In *MFCS*, pages 464–476, 2009.
30. Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In *IFIP TCS 2014*, pages 296–310, 2014.
31. Olivier Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.
32. Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theor. Comput. Sci.*, 290(1):161–188, 2003.
33. Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France, 1978.
34. Ian Mackie. Encoding strategies in the lambda calculus with interaction nets. In *IFL*, pages 19–36, 2005.
35. Gianfranco Mascari and Marco Pedicini. Head linear reduction and pure proof net extraction. *Theor. Comput. Sci.*, 135(1):111–137, 1994.
36. Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *TLCA*, pages 328–334, 1995.
37. Robin Milner. Functions as processes. *Math. Str. in Comput. Sci.*, 2(2):119–141, 1992.
38. Robin Milner. Bigraphical reactive systems. In *CONCUR*, pages 16–35, 2001.
39. Robin Milner. Local bigraphs and confluence: Two conjectures. *Electr. Notes Theor. Comput. Sci.*, 175(3):65–73, 2007.
40. Koko Muroya and Dan R. Ghica. The dynamic geometry of interaction machine: A call-by-need graph rewriter. In *CSL 2017*, pages 32:1–32:15, 2017.
41. Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Univ. Paris VII, 1992.
42. Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 2(126):281–292, 1994.
43. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
44. Paolo Tranquilli. *Nets Between Determinism and Nondeterminism*. Ph.D. thesis, Università degli Studi Roma Tre/Université Paris Diderot (Paris 7), 2009.
45. Paolo Tranquilli. Intuitionistic differential nets and lambda-calculus. *Theor. Comput. Sci.*, 412(20):1979–1997, 2011.
46. Lionel Vaux. *λ-calcul différentiel et logique classique: interactions calculatoires*. Ph.D. thesis, Université Aix-Marseille II, March 2007.
47. Christopher P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD Thesis, Oxford, 1971.