Proof nets and the call-by-value λ-calculus

Beniamino Accattoli

Proof Nets and the Call-by-Value λ-Calculus

Beniamino Accattoli
INRIA & LIX, École Polytechnique
beniamino.accattoli@inria.fr

Abstract

This paper gives a detailed account of the relationship between (a variant of) the call-by-value lambda calculus and linear logic proof nets. The presentation is carefully tuned in order to realize an isomorphism between the two systems: every single rewriting step on the calculus maps to a single step on proof nets, and viceversa. In this way, we obtain an algebraic reformulation of proof nets. Moreover, we provide a simple correctness criterion for our proof nets, which employ boxes in an unusual way, and identify a subcalculus that is shown to be as expressive as the full calculus.

Keywords: Curry-Howard isomorphism, call-by-value λ-calculus, linear logic, graphical syntaxes, proof nets, explicit substitutions, correctness criteria

1. Introduction

A key feature of linear logic (LL for short) is that it is a refinement of intuitionistic logic, *i.e.* of λ-calculus. In particular, *one* β-reduction step in the λ-calculus corresponds to the sequence of *two* cut-elimination steps in linear logic, steps which are of a very different nature: the first is multiplicative and the second is exponential. The Curry-Howard interpretation of this fact is that λ-calculus can be refined adding a constructor $t[x \leftarrow s]$ for sharing annotations called *explicit substitution*, and decomposing a β-step $(\lambda x.t)s \to_\beta t\{x \leftarrow s\}$ into the sequence $(\lambda x.t)u \to_{\mathtt{m}} t[x \leftarrow s] \to_{\mathtt{e}} t\{x \leftarrow s\}$.

Another insight due to linear logic is that proofs can be represented graphically, in a formalism called *proof nets*, and the reformulation of cut-elimination on proof nets takes a quite different flavor with respect to cut-elimination in sequent calculus. The parallel nature of the graphical syntax makes commutative cut-elimination steps—which are the annoying burden of every proof of cut-admissibility—(mostly) disappear.

These two features of LL have influenced the theory of explicit substitutions in various ways [12, 16, 23, 24], culminating in the design of *the structural λ-calculus* [8], a calculus isomorphic (more precisely *strongly bisimilar*[1]) to its

[1] A strong bisimulation between two rewriting systems S and R is a relation $\equiv$ between

representation in a variant of LL proof nets [7, 1]. Such a calculus can be seen as an algebraic reformulation of proof nets for λ-calculus [14, 36], and turned out to have a simpler meta-theory than previous calculi with explicit substitutions.

Girard's seminal paper on linear logic [19] presents two translations of λ-calculus into LL. The first one follows the typed scheme $(A \Rightarrow B)^{\mathtt{n}} = !A^{\mathtt{n}} \multimap B^{\mathtt{n}}$, and it is the one to which the previous paragraphs refer to. It represents the ordinary—or call-by-name (CBN)—λ-calculus. The second one, identified by $(A \Rightarrow B)^{\mathtt{v}} = !(A^{\mathtt{v}} \multimap B^{\mathtt{v}})$, was qualified as *boring* by Girard and received little attention in the literature [32, 35, 15, 17, 18, 31]. Usually, it is said to represent Plotkin's call-by-value (CBV) $\lambda_{\beta v}$-calculus [34]. These two representations concern typed terms only, but it is well-known that they can be extended to represent the whole untyped calculi by considering linear recursive types ($o = !o \multimap o$ for call-by-name and and $o = !(o \multimap o)$ for call-by-value).

Surprisingly, the extension of the CBV translation to the untyped $\lambda_{\beta v}$ calculus introduces a violent unexpected behavior: some normal terms in $\lambda_{\beta v}$ map to (recursively typed) proof nets without normal form, as we will point out here. A possible interpretation of this fact is that there is something inherently wrong in the CBV translation.

In this paper we show how to refine the three actors of the play (the CBV λ-calculus, the translation, and the proof nets presentation) in order to obtain a perfect match between terms and proof nets. Technically, we show that the new translation is an isomorphism of rewriting systems, and since isomorphisms preserve reductions length (in both directions), the normalization mismatch vanishes.

Interestingly, to obtain an isomorphism we have to make some radical changes to both the calculus and the presentation of proof nets. The calculus, that we call the *value substitution kernel* λ_{vker}, is a subcalculus of the *value substitution calculus* λ_{vsub} studied in [10], which is a CBV λ-calculus with explicit substitutions. Such a kernel—as we will show—is as expressive as the full calculus.

Our contributions are:

1. *Graphical Syntax and Algebraic Formalism.* It is far from easy to realize an isomorphism between terms and nets, as it is necessary to take care of many delicate details about weakenings, contractions, representation of variables, administrative reduction steps, and so on. The search for an isomorphism may seem a useless obsession, but it is not. Operational properties as confluence and termination then transfer immediately from graphs to terms, and vice versa. More generally, such a strong relationship turns the calculus into an algebraic language for proof nets, providing a handy tool to reason by structural induction over proof nets.
2. *Correctness Criterion.* We provide a characterization of the system of proof nets representing λ_{vker} based on graph-theoretical principles and which does not refer to λ_{vker}, that is, we present a *correctness criterion*.

S and R s.t. whenever $s \equiv r$ then for every step from $s \to_S s'$ there is a step $r \to_R r'$ s.t. $s' \equiv r'$, *and viceversa* (for $s, s' \in S$ and $r, r' \in R$).

Surprisingly, the known criteria for the representation of the call-by-name λ-calculus (with explicit substitutions) fail to characterize the fragment encoding the call-by-value λ-calculus. Here we present a simple and non-standard solution to this problem. We hack the usual presentation of proof nets so that Laurent's criterion for polarized nets [26, 28, 27]—the simplest known correctness criterion—captures the fragment we are interested in. The hacking is about explicit boxes: they are usually attached to !-links, while we put them on ⅋-links. An interesting point is that the fragment we deal with is not polarized in Laurent's sense, despite being polarized in the intuitionistic/Lamarche sense [25].

3. *The Kernel Calculus.* The kernel subcalculus is new. We provide a detailed study of the relationship with the value substitution calculus, showing that they are equivalent from the point of view of termination, which is the property used to define most notions of programs equivalence. Moreover, the two systems simulate each other with only a linear overhead.

In Sect. 7, we complement the results of the paper with 1) a discussion aimed to show that the use of boxes for ⅋-links is natural rather than ad-hoc, and 2) an account of the technical points concerning the representations of terms with proof nets, and how they have been treated in the literature.

This paper is a longer version of the workshop paper [3]. Apart from updating the notation of [3] to that of other recent papers using the same formalism [5, 9, 6], it extends it with the detailed relationship between the value substitution calculus and the kernel calculus in Sect. 8.

2. The Problem with Plotkin's Calculus

Plotkin's call-by-value (CBV for short) λ-calculus $\lambda_{\beta v}$ is obtained by restricting the β-rule $(\lambda x.t)s \to_\beta t\{x \leftarrow s\}$ (where $\{x \leftarrow s\}$ denotes the usual meta-level substitution) so that s is a value v, *i.e.* a variable or an abstraction. The *value substitution calculus*, similarly to other variations on the CBV λ-calculus as *e.g.* the one by Herbelin and Zimmerman [22], implements the restriction in a different way. Every β-redex $(\lambda x.t)s$, independently of the shape of s, is fired, but substitution is delayed by introducing an explicit substitution (or, equivalently, a `let` expressions), obtaining $t[x \leftarrow s]$ (aka `let` $x = s$ `in` t). Then, only explicit substitutions containing values are actually substituted, by means of an additional rewriting rule.

Before giving the details, let us point out what is the problem with Plotkin's calculus $\lambda_{\beta v}$. Essentially, it is about open terms, and shows up as soon as one does evaluate under abstraction or does not restrict to closed terms. While weak evaluation and closed terms are common practice hypothesis for an evaluation *strategy*, a *calculus* should be more liberal and not rely on them. Now, consider a term like $t := (\lambda x.\lambda y.y)sv$ where s is an irreducible open application, say, $s := zz$. Note that the redex $(\lambda x.\lambda y.y)s$ cannot be fired. Rather than a problem in itself, the issue is that such a blocked redex forbids the reduction of the future redex $(\lambda y.y)v$, that would appear if the first redex was to be somehow reduced

(*i.e.* it forbids what Lévy calls *creations of type 1* [29]). Preventing these redexes to take place ruins the relationships between the rewriting and the semantics, see [33, 10]. In particular, a term like

$$u := (\lambda z.\delta)(yy)\delta$$

(where $\delta := \lambda x..xx$ is the usual term whose auto application $\Omega := \delta\delta$ diverges) is a normal form in $\lambda_{\beta v}$, because the first argument yy of $\lambda z.\delta$ is not a value. From a semantical point of view, however, such a term is unsolvable (roughly, it is equivalent to Ω) and should diverge. A similar but different example of discrepancy between Plotkin's calculus and proof nets is given by $u' := \delta((\lambda z.\delta)(yy))$, where now it is the evaluation of the argument—rather than the function—that is stuck, forbidding the creation of a redex. This kind of redex creation is different from the previous one, and typical of the CBV λ-calculus. It can be called *creation of type 4*, as it does not appear in Lévy's classification, that is only concerned with the three kinds of creation of the CBN λ-calculus.

The value substitution calculus λ_{vsub} provides a compact solution to the problem of stuck redex and creations of type 1 and 4. It employs explicit substitutions and it adopts carefully crafted *contextual* rewriting rules, also known as *at a distance*. We will see that in λ_{vsub} the terms u and u' diverge, as required.

Another solution is provided by Carraro and Guerrieri in [11], where rather than adding explicit substitution the authors extend Plotkin's $\lambda_{\beta v}$ with two rewriting rules permuting constructors, so that blocked redexes can be enabled. Their approach essentially adapts Regnier's σ-equivalence [37] to CBV.

3. The Value Substitution Calculus and its Kernel

The language of the *value substitution calculus* λ_{vsub} is given by

$$\begin{array}{lrcl} \text{Terms} & t, s, u, r & ::= & v \mid ts \mid t[x{\leftarrow}s] \\ \text{Values} & v & ::= & x \mid \lambda x.t \end{array}$$

where $t[x{\leftarrow}s]$ is an *explicit substitution*. Both $\lambda x.t$ and $t[x{\leftarrow}s]$ bind x in t, and we silently work modulo α-equivalence. We shall use contexts (noted C, D, E) extensively, in particular *substitution contexts* (noted L, L', L'') so let us define them formally:

$$\begin{array}{rrcl} \text{Contexts} & C, D, E & ::= & \langle\cdot\rangle \mid \lambda x.C \mid vC \mid Ct \mid C[x{\leftarrow}s] \mid t[x{\leftarrow}C] \\ \text{Substitution Contexts} & L, L', L'' & ::= & \langle\cdot\rangle \mid L[x{\leftarrow}t] \end{array}$$

The *plugging* $C\langle t\rangle$ (resp. $C\langle D\rangle$) of a term t (resp. context D) in a context C is defined as $\langle t\rangle := t$ (resp. $\langle D\rangle := D$), $(\lambda x.C)\langle t\rangle := \lambda x.C\langle t\rangle$ (resp. $(\lambda x.C)\langle D\rangle := \lambda x.C\langle D\rangle$), $(Cs)\langle t\rangle := C\langle t\rangle s$ (resp. $(Cs)\langle D\rangle := C\langle D\rangle s$), and so on. Note that plugging in a context can capture variables.

As usual, the rewriting rules of λ_{vsub} are obtained by first defining the rewriting rules at top level (noted $\mapsto_{\mathtt{m}}$ and $\mapsto_{\mathtt{e}}$), and then taking their closure by

contexts (noted $\to_{\mathtt{m}}$ and $\to_{\mathtt{e}}$). It is less common, instead, that contexts are also used to specify the top-level rules, as we do here, that is the essence of rewriting *at a distance*. The multiplicative and exponential rewriting rules are given by:

	Rule at Top Level			Contextual Closure	
Multiplicative	$L\langle\lambda x.t\rangle s$	$\mapsto_{\mathtt{m}}$	$L\langle t[x\leftarrow s]\rangle$	$C\langle t\rangle \to_{\mathtt{m}} C\langle s\rangle$	iff $t \mapsto_{\mathtt{m}} s$
Exponential	$t[x\leftarrow L\langle v\rangle]$	$\mapsto_{\mathtt{e}}$	$L\langle t\{x\leftarrow v\}\rangle$	$C\langle t\rangle \to_{\mathtt{e}} C\langle s\rangle$	iff $t \mapsto_{\mathtt{e}} s$

the fact that in the lhs of $\mapsto_{\mathtt{e}}$ the context L appears inside [ ] while in the rhs it appears outside { } is not a typo. We write $\to_{\lambda_{vsub}}$ for the union of $\to_{\mathtt{m}}$ and $\to_{\mathtt{e}}$.

Theorem 3.1 ([10]). *The value substitution calculus λ_{vsub} is confluent.*

Let us show that in λ_{vsub} the two problematic terms u and u' (discussed in the previous section) diverge in λ_{vsub}, as required:

$$\begin{array}{rcll} u & := & (\lambda z.\delta)(yy)\delta & \to_{\mathtt{m}} \\ & & \delta[z\leftarrow yy]\delta & \to_{\mathtt{m}} \\ & & (xx)[x\leftarrow\delta][z\leftarrow yy] & \to_{\mathtt{e}} \\ & & (\delta\delta)[z\leftarrow yy] & \to_{\mathtt{m}} \quad \dots \end{array}$$

and

$$\begin{array}{rcll} u' & := & \delta((\lambda z.\delta)(yy)) & \to_{\mathtt{m}} \\ & & \delta(\delta[z\leftarrow yy]) & \to_{\mathtt{m}} \\ & & (xx)[x\leftarrow\delta[z\leftarrow yy]] & \to_{\mathtt{e}} \\ & & (\delta\delta)[z\leftarrow yy] & \to_{\mathtt{m}} \quad \dots \end{array}$$

Essentially, the problems with creations are solved by

1. turning *generic* β-redexes (*i.e.* not only CBV β-redexes) into ES,
2. allowing such an action to be possible *up to ES*, *i.e.* by turning to rewriting *at a distance*,
3. restricting ES to substitute when they contain a value *up to ES*.

Note that the problem illustrated by u, *i.e.* stuck creations of type 1, is solved by points 1 and 2, while the problem showcased by u', *i.e.* stuck creations of type 4, is solved by points 1 and 3.

As hinted at in [10], the value substitution calculus λ_{vsub} is designed around the CBV translation of λ-calculus to proof nets (extended to untyped terms by means of recursive types). This shows a serious mismatch between Plotkin's calculus and proof nets, with respect to termination. However, in [10] the relationship between λ_{vsub} and proof nets was not shown. One of the reasons is that such a relationship—with respect to the usual presentation of the translation—is complex, as term redexes and proof nets redexes do not match very nicely. This paper addresses this point, providing a new and simpler relationship, identifying a subcalculus λ_{vker} of λ_{vsub} which perfectly represents the image of the translation to proof nets. How λ_{vsub} is related to proof nets is studied in Sect. 8.

The relationship between the value substitution calculus λ_{vsub} and Plotkin's calculus $\lambda_{\beta v}$ has been treated indirectly in [10]. Therein, λ_{vsub} is related to another call-by-value calculus, Herbelin and Zimmermann's λ_{CBV} [22], whose equational theory is shown to be strictly contained in the theory of λ_{vsub}. In turn, λ_{CBV} is related to Plotkin's $\lambda_{\beta v}$ in [22], where it is shown that the equational theory of $\lambda_{\beta v}$ is strictly contained in the theory of λ_{CBV}.

The Kernel Calculus λ_{vker}. The value substitution calculus λ_{vsub} is somewhat redundant. In fact, there is a subcalculus, *the value substitution kernel* λ_{vker}, which is as expressive as the whole of λ_{vsub}, and that is what exactly corresponds to proof nets, as we will show. Its syntax is:

$$\begin{array}{lrcl} \text{Terms} & t,s,u,r & ::= & v \mid vs \mid t[x{\leftarrow}s] \\ \text{Values} & v & ::= & x \mid \lambda x.t \end{array}$$

The distinguished feature of the value substitution kernel is that iterated applications as $(ts)u$ are not part of the language, because the left subterm of an application can only be a value. Contexts are defined similarly as for λ_{vsub} (with the obvious restriction on applications). The rewriting rules of λ_{vker} are those of λ_{vsub} restricted to the new syntax. Explicitly:

	Rule at Top Level			Contextual Closure	
Multiplicative	$(\lambda x.t)s$	$\mapsto_{\mathtt{m}}$	$t[x{\leftarrow}s]$	$C\langle t\rangle \to_{\mathtt{m}} C\langle s\rangle$	iff $t \mapsto_{\mathtt{m}} s$
Exponential	$t[x{\leftarrow}L\langle v\rangle]$	$\mapsto_{\mathtt{e}}$	$L\langle t\{x{\leftarrow}v\}\rangle$	$C\langle t\rangle \to_{\mathtt{e}} C\langle s\rangle$	iff $t \mapsto_{\mathtt{e}} s$

Note that $\mapsto_{\mathtt{m}}$ does no longer need the substitution context around the abstraction. Note also that λ_{vker} is stable under reduction because only values are substituted, and so a substitution can never turn an application of the form xu into a term of the form $(ts)u'$. Since λ_{vsub} is confluent and λ_{vker} is a closed subcalculus of λ_{vsub}, we immediately obtain

Theorem 3.2. *The value substitution kernel λ_{vker} is confluent.*

In Sect. 8.2, we will show that λ_{vsub} can be represented inside λ_{vker}. The idea is that applications as $(ts)u$ are rather represented as $(xu)[x{\leftarrow}ts]$ with x fresh. We will prove that a term t of λ_{vsub} and its representation $t^{\mathtt{k}}$ in λ_{vker} are equivalent from the point of view of termination.

In the next three sections, instead, we will show that λ_{vker} has an isomorphic representation as a variant of linear logic proof nets.

The results of this paper are complemented by those in [4], where it is shown that λ_{vker} has an isomorphic representation inside the π-calculus as well.

4. Proof Nets: Definition

Introduction. Our presentation of proof nets is nonstandard in at least four points (we suggest to have a quick look to Fig. 3):

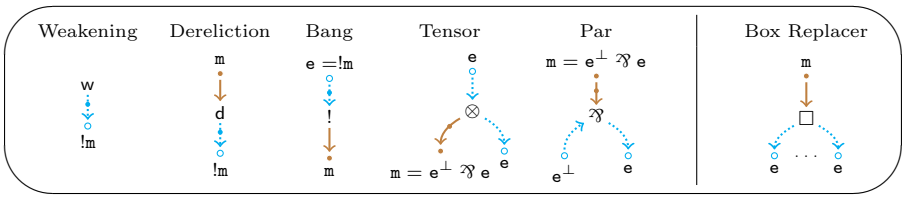

Figure 1: links.

1. *Hypergraphs*: we use directed hypergraphs (for which formulas are nodes and links—*i.e.* logical rules—are hyperedges) rather than the usual graphs with pending edges (for which formulas are edges and links are nodes). We prefer hypergraphs—that despite the scaring name are nothing else but bipartite graphs—because
   (a) 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 and axiom links are represented implicitly, collapsing them on nodes. This is analogous to what happens in interaction nets. Essentially, our multiplicative nodes are *wires*, with exponential nodes being *hyper*wires, *i.e.* wires involving an arbitrary number of ports;
   (c) subnets can be elegantly defined as subsets of links, that would not be possible by adopting other approaches as generalized ?-links.

   The choice of hypergraphs, however, has various (minor) technical consequences, and the formulation of some usual notions (*e.g.* the nesting condition for boxes) will be slightly different with respect to the literature.
2. *⅋-boxes*: we use boxes for ⅋-links and not for !-links. This choice is discussed in Sect. 7, and it allows to use a very simple correctness criterion—*i.e.* Laurent's criterion for polarized nets [28, 27]—without losing any property.
3. *Polarity*: we apply a polarized correctness criterion to a setting which is not polarized in the usual sense.
4. *Syntax tree*: since we use proof nets to represent terms, we will dispose them on the plane according to the syntax tree of the corresponding terms, and not according to the corresponding sequent calculus proof. Moreover, the orientation of the links does not reflect the usual premise-conclusion orientation of proof nets.

Nets. Nets are directed and labeled hypergraphs $G = (V(G), L(G))$, *i.e.*, graphs where $V(G)$ is a set of labeled *nodes* and $L(G)$ is a set of labeled and *directed hyperedges*, called *links*, which are edges with 0, 1, or more sources and 0, 1, or more targets[2]. Nodes are labeled with a type in $\{\mathtt{e}, \mathtt{m}\}$, where $\mathtt{e}$ stands for

[2] A hypergraph G can be understood as a bipartite graph B_G, where $V_1(B_G)$ is $V(G)$ and $V_2(B_G)$ is $L(G)$, and the edges are determined by the relations *being a source* and *being a*

exponential and m for *multiplicative*. If a node u has type e (resp. m) we say that it is a e-node (resp. m-node). The label of a node will usually be left implicit, as e and m nodes are distinguished graphically, using both colors and different shapes: e-nodes are cyan and white-filled, while m-nodes are brown and dot-like. We shall consider hypergraphs whose links are labeled from $\{!, \mathsf{d}, \mathsf{w}, ⅋, \otimes\}$. 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 (the types will be discussed later, and the figure also contains the $\square$-link, which is not used to define nets: it will be used later to define the correction graph). Similarly to nodes, we use colors and shapes for the type of the source/target connection of a link to a node: e-connections are cyan and dotted, while m-connections are brown and solid. Our choice of shapes allows to read the paper also if printed in black and white. 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.

Note that every link (except $\square$) has exactly one connection with a little circle: it denotes the *principal* node, *i.e.* the node on which the link can interact. Remark the principal node for tensor and !, which is not misplaced. Moreover, every ⅋-link has an associated *box*, *i.e.*, a sub-hypergraph of P (have a look to Fig. 3). Formally:

Definition 4.1 (Net)**.** *A* pre-net P *is a triple* $(|P|, \mathtt{fv}(P), r_P)$*, where* $|P| = (V(P), L(P))$ *is a hypergraph whose nodes are labeled with either* e *or* m *and whose hyperedges are* $\{!, \mathsf{d}, \mathsf{w}, ⅋, \otimes\}$*-links, and such that:*

Interface*:*

Root*:* $r_P \in V(P)$ *is a non-isolated initial* e*-node of* P*, called the* root *of* P*.*

Free Variables*:* $\mathtt{fv}(P)$ *is the set of terminal nodes of* P*, also called* free variables *of* P*, which are targets of* $\{\mathsf{d}, \mathsf{w}\}$*-links (*i.e. *they are not allowed to be targets of* $\otimes$*-links, nor to be isolated).*

Nodes*:*

Multiplicative*:* m*-nodes have at most one incoming and at most one outgoing link.*

Exponential*: an* e*-node has at most one outgoing link, and if it is the target of more than one link then they all are* d*-links.*

---

target of a hyperedge.

A net P *is a pre-net together with a function* $\mathtt{box}_P$ *(or simply* $\mathtt{box}$*) associating to every* $\parr$*-link* l *a subset* $\mathtt{box}(l)$ *of* $L(P) \setminus \{l\}$ *(*i.e. *the links of* P *except* l *itself) s. t.* $\mathtt{box}(l)$ *is a pre-net with a distinguished free variable called the* variable *of* l*, and verifies:*

> Border*: the root* $r_{\mathtt{box}(l)}$ *and the free variable* x *are respectively the target and source* $\mathtt{e}$*-nodes of* l*, and any free variable* $\neq x$ *of* $\mathtt{box}(l)$ *is not the target of a weakening.*
>
> Nesting*: for any* $\parr$*-box* $\mathtt{box}(h)$ *if* $\emptyset \neq I := |\mathtt{box}(l)| \cap |\mathtt{box}(h)|$*,* $|\mathtt{box}(l)| \not\subseteq |\mathtt{box}(h)|$*, and* $|\mathtt{box}(h)| \not\subseteq |\mathtt{box}(l)|$ *then all the nodes in* I *are free variables of both* $\mathtt{box}(l)$ *and* $\mathtt{box}(h)$*.*
>
> Internal Closure*:*
>
> > Contractions*:* $h \in \mathtt{box}(l)$ *for any link* h *of* P *having as target an internal* $\mathtt{e}$*-node of* $\mathtt{box}(l)$*.*
> >
> > Boxes*:* $\mathtt{box}(h) \subseteq \mathtt{box}(l)$ *for any* $\parr$*-link* $h \in \mathtt{box}(l)$*.*

Comments on the Definition.

> *Weakenings and Box Borders*: in the border condition for nets the fact that the free variables $\neq x$ 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 will have to preserve this invariant).
>
> *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.
>
> *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 morally disjoint can in our syntax share free variables, because of an implicit contraction merging two of them.
>
> *Boxes as Nets*: note that a box $\mathtt{box}(l)$ 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 $\mathtt{box}_P$ of P to the $\parr$-links of $\mathtt{box}(l)$, and see $\mathtt{box}(l)$ 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 will freely consider boxes as *nets*.

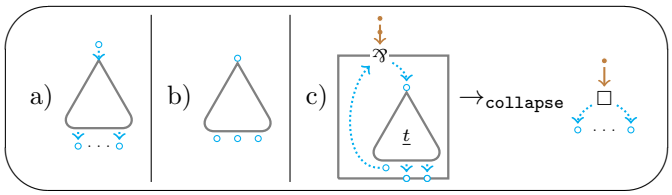

Figure 2: various images.

Terminology about Nets. The *level* of a node/link/box is the maximum number of nested boxes in which it is contained[3] (a $\parr$-link is not contained in its own box). Two links are *contracted* if they share an e-target. Note that the exponential condition states that only derelictions (*i.e.* $\mathtt{d}$-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. Sometimes (*e.g.* the bottom half of Fig. 3), the figures show a link in a box having as target a contracted $\mathtt{e}$-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.

Typing. Nets are typed using a recursive type $o = !(o \multimap o)$, that we rename $\mathtt{e} = !(\mathtt{e} \multimap \mathtt{e}) = !(\mathtt{e}^\perp \parr \mathtt{e})$ because $\mathtt{e}$ is a mnemonic for *exponential*. Let $\mathtt{m} := \mathtt{e} \multimap \mathtt{e} = \mathtt{e}^\perp \parr \mathtt{e}$, where $\mathtt{m}$ stands for *multiplicative*. Note that $\mathtt{e} = !\mathtt{m}$ and $\mathtt{m} = !\mathtt{m} \multimap !\mathtt{m}$. This shows that the two translations of λ-calculus into linear logic identified by the two recursive types $\mathtt{e} = !(\mathtt{e} \multimap \mathtt{e})$ and $\mathtt{m} = !\mathtt{m} \multimap !\mathtt{m}$ are in fact the same. 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 colors 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 have to be read bottom-up, and thus negated (to match the usual typing for links) if the conclusion of the logical rule is the bottom node of the link, as it is the case for the $\{\mathtt{w}, \mathtt{d}, \otimes\}$-links, while $!$ and $\parr$ have their logical conclusion on the top node, and so their type does not need to be negated.

Induced !*-boxes.* Note that a !-link is always applied to something ($\mathtt{m}$-nodes cannot be free variables), and there is not so much freedom for this *something*: either it is a dereliction link or a $\parr$ with its box. Note also that in these cases we obtain (what would usually be) a valid content for a !-box, *i.e.* something ending on $\mathtt{e}$-nodes. For the dereliction case it is evident, and for the $\parr$ case it is guaranteed by the border condition in the definition of nets: the content of a $\parr$-box ends on $\mathtt{e}$-nodes. Hence, any !-link has an associated box, possibly induced by a $\parr$-box, that needs not be represented explicitly. Such induced !-boxes will play a role in Sect. 6, where proof nets cut-elimination is studied.

[3]Here the words *maximum* and *nested* are due to the fact that the free variables of $\parr$-boxes may belong to two not nested boxes, because of the way we represent contraction.

Translation. Nets representing terms have the general form in Fig. 2.a, also schematized as in Fig. 2.b. The translation $\underline{\cdot}$ from terms to nets is in Fig. 3 (the original boring translation is sketched in Fig. 6, page 23). A net which is the translation of a term is a *proof net*. Note that in some cases there are various connections entering an $\mathtt{e}$-node, that is the way we represent contraction. In some cases the $\mathtt{e}$-nodes have an incoming connection with a perpendicular little bar: it represents an arbitrary number (> 0) of incoming connections. The net corresponding to a variable is given by a !-link on a dereliction and not by an (exponential) axiom, as it is sometimes the case [18]. The reason is that an axiom (in our case a node, because axioms are collapsed on nodes) would not reflect on nets some term reductions, as $x[x{\leftarrow}v] \to_{\mathtt{e}} v$, for which both the redex and the reduct would be mapped on the same net.

Extended Translation. 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. The clause defining $\underline{t}_{\Delta\cup\{y\}}$ when $y \notin \mathtt{fv}(t)$ is the second on the first line of Fig. 3, the definition is then completed by the following two clauses: $\underline{t}_\emptyset := \underline{t}$ and $\underline{t}_{\Delta\cup\{y\}} := \underline{t}_\Delta$ if $y \in \mathtt{fv}(t)$.

α-Equivalence. To circumvent an explicit and formal treatment of α-equivalence we assume that the set of $\mathtt{e}$-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.

Remark 4.2. *The translation of terms to nets is not injective. By simply applying the translation it is easily seen that the following pairs of terms are sent on the same net:*

$$\begin{array}{llll} t[x{\leftarrow}s][y{\leftarrow}u] & \sim_{vo_{CS}} & t[y{\leftarrow}u][x{\leftarrow}s] & \text{if } x \notin \mathtt{fv}(u) \ \&\ y \notin \mathtt{fv}(s) \\ vu[x{\leftarrow}s] & \sim_{vo_1} & (vu)[x{\leftarrow}s] & \text{if } x \notin \mathtt{fv}(v) \\ t[x{\leftarrow}s[y{\leftarrow}u]] & \sim_{vo_2} & t[x{\leftarrow}s][y{\leftarrow}u] & \text{if } y \notin \mathtt{fv}(t) \end{array} \tag{1}$$

Let $\equiv_{vo}$ be the reflexive, symmetric, transitive, and contextual closure of $\sim_{vo_{CS}} \cup \sim_{vo_1} \cup \sim_{vo_2}$. In the proof of Lemma 6.1, we will use the fact that if $t \equiv_{vo} s$ then t and s are mapped on the same net. We also claim—without proving it—that $\equiv_{vo}$ is exactly the quotient induced on terms by the translation to nets.

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

$$u = u_1, l_1, \ldots, l_k, u_{k+1} = w$$

s.t. 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 $k > 0$.

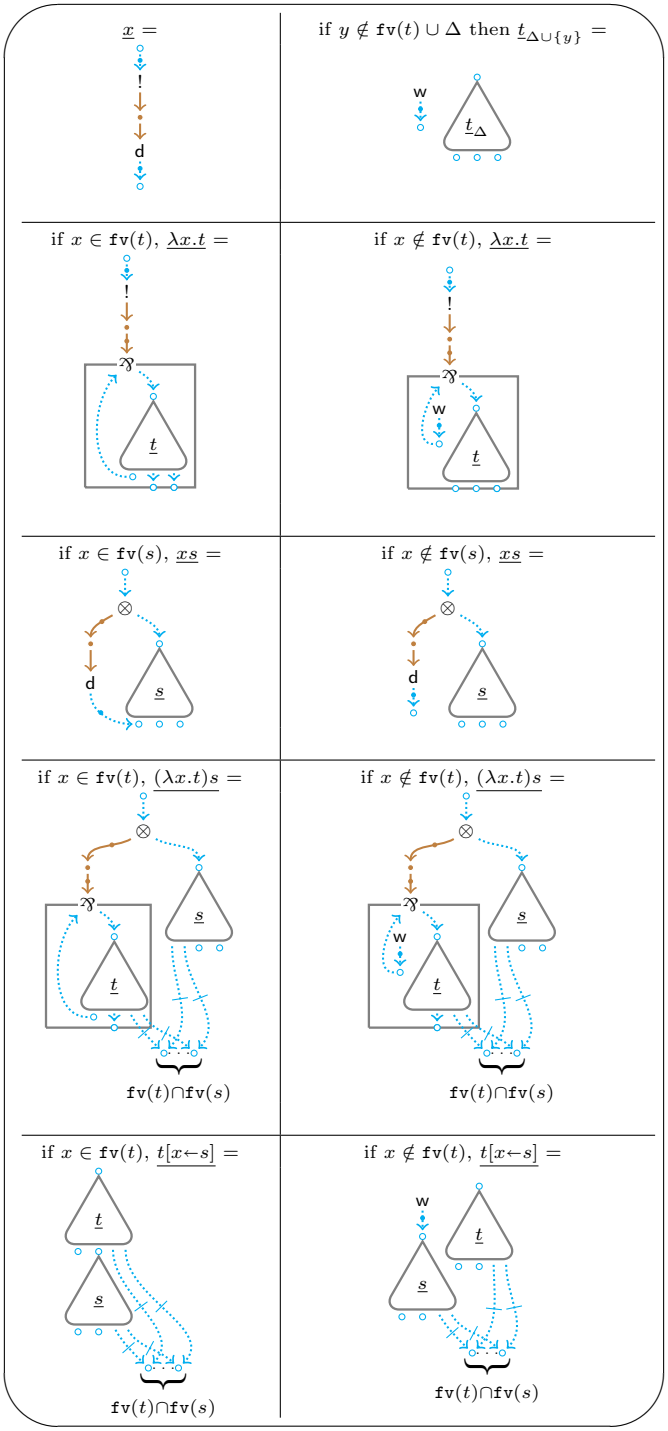

Figure 3: the translation from terms to nets.

Correctness. The correctness criterion is based on the notion of correction graph, 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 4.3 (Correction Graph)**.** *Let P be a net. The correction graph P^0 of P is the directed hypergraph obtained from P by collapsing each $\parr$-box at level 0 in P into a $\square$-link (with the same interface) by applying the rule in Fig. 2.c.*

Definition 4.4 (Correctness)**.** *A net P is* correct *if:*

Root*: the root of P induces the only initial node of P^0.*

Acyclicity*: P^0 is acyclic.*

Recursive Correctness*: the box of every $\parr$-link at level 0 is correct.*

As usual an easy induction on the translation shows that the translation of a term is correct, *i.e.* that:

Lemma 4.5. *Every proof net is correct,* i.e. *$\underline{t}_\Delta$ is a correct net for every term t and set of variables Δ.*

Note that the root correctness condition forbids the existence of isolated nodes. In logical terms, it forbids isolated axioms. Such a restriction is not *ad-hoc*, as it is also used in the call-by-name translation of λ-calculus to proof nets. Additional *shape constraints* are indeed required when proof nets are supposed to represent terms: some nets that would be considered correct proofs do not represent any λ-term and have to be rejected. Another shape constraint is the requirement that nets free variables are weakenings and derelictions (see Definition 4.1).

5. Proof Nets: Sequentialization

In this section we show how to extract a term t from every correct net P in such a way that t translates back to P, *i.e.* we show that every correct net is a proof net. The proof of such a sequentialization theorem is based on the notion of *kingdom*, along the lines of the proof for polarized nets by Olivier Laurent, see [27], pp. 57-63.

The study relies on the notion of *subnet* Q of a correct net P, that is a subset of the links of P plus some closure conditions, avoiding 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.

Definition 5.1 (Subnet)**.** *Let P be a correct net. A* subnet *Q of P is a subset of its links s.t. it is a correct net (wrt the box function inherited from P) and satisfies the following closure conditions:*

Contractions*: $l \in Q$ for any link $l \in P$ having as target an internal* e*-node of Q.*

Boxes*:*

Root*:* $\mathtt{box}(h) \subseteq Q$ *for any* $\parr$*-link* $h \in Q$*.*

Free variables*:* $\mathtt{box}(l) \subseteq Q$ *if a free variable of* $\mathtt{box}(l)$ *is internal to* Q*.*

To ease the language, we also give a name to the nodes that can be the root of a subnet.

Definition 5.2 (Royal Node)**.** *Let P be a net. A* royal node *x of P is a* $\mathtt{e}$*-node such that it is not a free variable of P (*i.e. *$x \notin \mathtt{fv}(P)$) and it is not the variable (*i.e. *not the* $\mathtt{e}$*-source) of a* $\parr$*-link, or—equivalently—such that it is the source of a* $!$*-link or of a* $\otimes$*-link.*

Next, we define the kingdom of a royal node x, that will be proved to be the *smallest* subnet of root x (forthcoming Lemma 5.6).

Definition 5.3 (Kingdom)**.** *Let P be a correct net and x one of its royal nodes. The* kingdom *$king_P(x)$ of x in P is the set of links defined by case analysis on the link l of source x:*

l is a $!$-link*: $king_P(x)$ is given by l plus the* $\mathtt{d}$*-link or the* $\parr$*-box on the* $\mathtt{m}$*-target of l.*

l is a $\otimes$-link*: $king_P(x)$ is given by l plus the* $\mathtt{d}$*-link or the* $\parr$*-box on the* $\mathtt{m}$*-target of l plus $king_P(y)$, where y is the* $\mathtt{e}$*-target of l.*

It is easily seen that $king_P(x)$ does not really depend on P, in the following sense.

Lemma 5.4. *Let P be a correct net, Q be a subnet of P, and x be a royal node of Q (and thus of P). Then $king_P(x) = king_Q(x)$.*

Proof. By induction on the length of the maximum directed path from x in Q^0. □

Therefore, most of the time we will simplify and simply write $king(x)$ (instead of $king_P(x)$).

To characterize kingdoms the following definitions are required.

Definition 5.5 ((Free/Ground) Substitution)**.** *Let P be a correct net. An* $\mathtt{e}$*-node x of P is:*

a substitution *if it is the target of a* $\{\mathtt{w}, \mathtt{d}\}$*-link (or, equivalently, if it is not the target of a* $\otimes$*-link nor the root) and the source of some link;*

a ground substitution *if it is a substitution which is a node of P^0 (*i.e. *it is not internal to any* $\parr$*-box*[4]*);*

[4]Note that our collapsed representation of contractions and cuts does not allow to simply say that x is a node at level 0: indeed the free variables of a $\parr$-box can have level > 0 and yet belong to P^0.

a free substitution *if it is a ground substitution and there is no ground substitution of P to which x has a path (in P^0).*

Now, we show that kingdoms are indeed minimal subnets. At the same time, we prove some additional properties of kingdoms to be used in the next lemmas.

Lemma 5.6 (Kingdom)**.** *Let P be a correct net and x one of its royal nodes. Then $king(x)$ is the smallest subnet of P of root x. Moreover, it has no free substitutions, no free weakenings, and whenever $y \in \mathtt{fv}(king(x))$*

1. Connected Interface*: x has a path to y in $king(x)^0$;*
2. Atomicity*: if y is internal to a subnet Q of P then $king(x) \subseteq Q$.*

Proof. Let R be a correct subnet of P rooted at x. By induction on the length of the maximum directed path from x in P^0 (that exists because by correctness P^0 is acyclic) we show that $king(x) \subseteq R$ and that $king(x)$ is correct. Let l be the link of source x. Cases of l:

Base case: l is a !-link. By the free variables condition for pre-nets R has to contain the d-link h or the ⅋-link on the m-target of l. In the case of a ⅋-link the box root condition for subnets implies that the whole box B is in R, hence $king(x) \subseteq R$. In the case of a d-link correctness is obvious, in the case of a ⅋-box it follows by the correctness of the box itself, guaranteed by the recursive correctness condition. Moreover, no free substitutions and no free weakenings belong to $king(x)$ (by the border condition for nets boxes cannot close on weakenings). Pick $y \in \mathtt{fv}(king(x))$, which in the d-link case is the target of h and in the other case is a free variable of the ⅋-box B. Now,

1. *Connected Interface*: if y is the target of the d-link then there is a path from x to y, given by the only path going through the !-link and the d-link. If y is a free variable of the ⅋-box, then there is a path from x to y in $king(x)^0$, the one which goes through the !-link and the □-link replacing the ⅋-box in $king(x)^0$.
2. *Atomicity*: if y is internal to a given subnet Q then the contractions condition for subnets guarantees that h or B are in Q. Then clearly $king(x) \subseteq Q$.

Inductive case: l is a ⊗-link. As in the previous case, R has to contain the d-link or the ⅋-box on the m-target of l (by the free variables condition for pre-nets). Consider the e-target z of l, that is a royal node of both P and R (by the free variables condition for pre-nets it cannot be a free variable). By *i.h.*, $king_R(z) \subseteq R$ and by Lemma 5.4 (stating $king_R(z) = king_P(z)$, simply noted $king(z)$) it follows $king(z) \subseteq R$, in turn implying $king(x) \subseteq R$. By *i.h.*, $king(z)$ is also correct, hence z verifies the root correctness condition for $king(z)$, and so does x for $king(x)$. Acyclicity follows by correctness of P. Recursive correctness follows from the box closure condition for subnets and correctness of P. Thus $king(x)$

is the smallest subnet of P of root x. Moreover by *i.h.*, $king(z)$—and so $king(x)$—has no free substitutions and no free weakenings. *Connected interface* and *atomicity* follow from the *i.h.* for the free variables of $king(z)$ and from the conditions for a subnet (as in the previous case) for the other free variables. □

A key property for the *read-back* of a correct net as a term is given by the following lemma, that is a sort of *splitting tensor* lemma, stating that free substitutions are splitting.

Lemma 5.7 (Substitution Splitting)**.** *Let P be a correct net with a free substitution x. Then*

1. *The free variables of $king(x)$ are free variables of P.*
2. *$P \setminus king(x)$ is a subnet of P.*

Proof.

1. Suppose not. Then there is a free variable y of $king(x)$ which is not a free variable of P. There are two possible cases:

   y is a substitution of P. By the *connected interface* part of Lemma 5.6, x has a path to y in $king(x)^0$, *i.e.* to a substitution in P^0, which is also a path in P^0. But such a path contradicts the hypothesis that x is a free substitution, absurd.

   y is the distinguished free variable of a $\parr$-box B. Thus, y is internal to some $\parr$-box B and so it is not a node of P^0. By the *atomicity* part of Lemma 5.6, $king(x) \subseteq B$ and so x is not a node of P^0, against the definition of free substitution, absurd.

2. By point 1 the removal of $king(x)$ cannot create new initial nodes. Being a substitution, x is the target of some link. Therefore the removal of $king(x)$ cannot remove the root of P. It is also clear that the removal cannot create cycles, and the box closure condition for subnets guarantees that the recursive correctness of P implies the one of $P \setminus king(x)$. □

Of course, we need to show that as long as there are substitutions out of boxes (*i.e.* ground substitutions) there are free substitutions, *i.e.* that the substitution splitting lemma can be applied.

Lemma 5.8. *Let P be a correct net with a ground substitution. Then P has a free substitution.*

Proof. Consider the following order on the elements of the set S_g of ground substitutions of P: $z \leq y$ if there is a path from z to y in P^0. Acyclicity of P^0 implies that S_g contains maximal elements with respect to $\leq$, if it is non-empty. Note that a maximal element of S_g is a free substitution in P. Now, if P has a ground substitution x then S_g is non-empty. Thus, P has a free substitution. □

The two previous lemmas together imply that ground substitutions can be removed one after the other, by repeatedly selecting a free one, until none of them are left. The next lemma show that what is left after these removals is exactly the kingdom of the root.

Lemma 5.9 (Kingdom Characterization)**.** *Let P be a correct net. Then $P = king(r_P)$ iff P has no free substitutions nor free weakenings.*

Proof. $\Rightarrow$) By Lemma 5.6. $\Leftarrow$) By Lemma 5.6, $king(r_P) \subseteq P$. If the two do not coincide then by the contractions condition for subnets, the multiplicative condition on nets, and the fact that they share the same root, it follows that P contains a ground substitution x on a free variable of $king(r_P)$. Then, P contains a free substitution by Lemma 5.8, absurd. □

We conclude the section with the sequentialization theorem, that relates terms and proof nets at the static level.

Theorem 5.10 (Sequentialization)**.** *Let P be a correct net and Δ be the set of* e*-nodes of its free weakenings. Then there is a term t s.t. $\underline{t}_\Delta = P$ (and $\mathtt{fv}(P) = \mathtt{fv}(t) \cup \Delta$).*

Proof. By induction on the number of links of P. By the root and free variables conditions the minimum number of links is 2 and the two links are necessarily a !-link on top of a d-link. Let x be the e-node of the d-link. Then $\underline{x} = P$. We now present each inductive case. After the first one we assume that the net has no free weakenings.

> *There is a free weakening l of* e*-node x.* Then $P' = P \setminus \{l\}$ is still a correct net and by *i.h.* there exist t s.t. $\underline{t}_{\Delta\setminus\{x\}} = P'$. Then $\underline{t}_\Delta = P$.
>
> *There is a free substitution x.* Then by Lemma 5.6 and Lemma 5.7 $king(x)$ and $P \setminus king(x)$ are correct subnets of P. By the *i.h.* there exist s and u s.t. $\underline{s} = king(x)$ and $\underline{u}_{\{x\}} = P \setminus king(x)$ (note that if $x \in \mathtt{fv}(u)$ then $\underline{u}_{\{x\}} = \underline{u}_\emptyset = \underline{u}$). Then $\underline{u[x \leftarrow s]} = P$.
>
> *No free substitutions*: by Lemma 5.9 $P = king(r_P)$. For the root link l of P, of source r_P, there are three cases:
>
> > *l is a* !*-link over a* d*-link*: base case, already treated.
> >
> > *l is a* !*-link over a* ⅋*-link*: let Q be the box of the ⅋-link and x its distinguished free variable. By the border condition for a net the set of free weakenings of Q either is empty or it contains only x. If x is (resp. is not) the node of a free weakening then by *i.h.* there exists t s.t. $\underline{t}_{\{x\}} = Q$ (resp. $\underline{t} = Q$). Then $\underline{\lambda x.t} = P$.
> >
> > *l is a $\otimes$-link*: let x be its e-target and a its m-target. Note that $P = king(r_P)$ implies that P is composed by l, $king(x)$ and either the d-link or the ⅋-link (plus its box) on a. By *i.h.* there exists s s.t. $\underline{s} = king(x)$. Now, if a is the source of a d-link of e-node y we

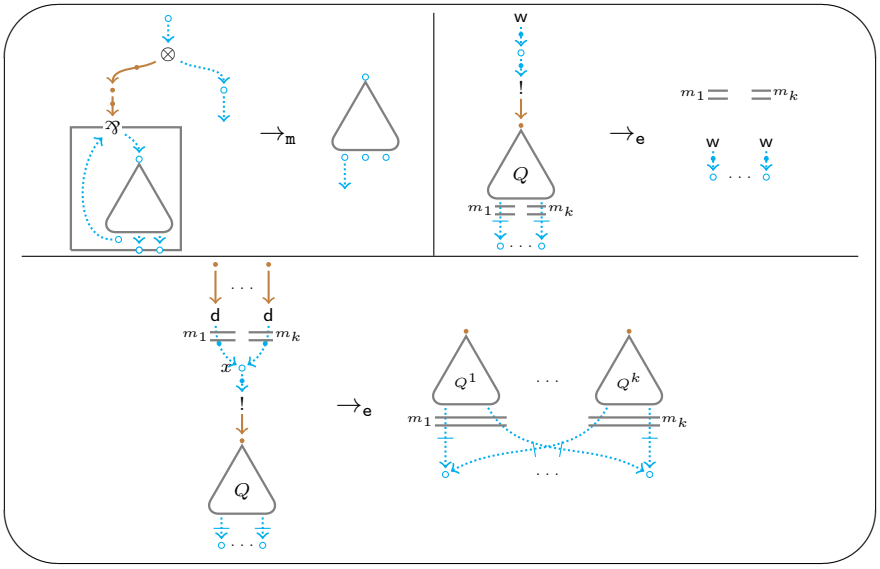

Figure 4: proof nets cut-elimination rules

conclude, since $\underline{ys} = P$. Otherwise, s is the source of a ⅋ of box Q and the *i.h.* gives a term u and a set of variables Γ s.t. $\underline{u}_\Gamma = Q$. Let us prove that Q and $king(x)$ can only share free variables, as the translation prescribes: no link at level 0 of $king(x)$ can be in Q, and no box at level 0 of $king(x)$ can intersect Q other than on free variables, by the nesting condition. By reasoning about the distinguished free variable of Q as in the previous case we then obtain $\underline{(\lambda y.u)s} = P$. □

6. Proof Nets: Dynamics

The rewriting rules are in Fig. 4. 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).

Rule $\to_{\mathtt{m}}$ is nothing but the usual elimination of a multiplicative cut, except that the step also opens the box associated with the ⅋-link.

The two $\to_{\mathtt{e}}$ rules reduce the exponential redexes. Let us explain how to read them. The graph noted Q in Fig. 4 is the induced box of the !-link (see the paragraph *induced* !*-boxes* in Sect. 4). There are in fact two possibilities: either it is simply a d-link or it is a ⅋-link with its box, so there is no ambiguity on what to duplicate/erase. Every pair of short gray lines denotes the sequence (of length m_i, with $i \in \{1, \ldots, k\}$) of boxes closing on the corresponding links. The rule has two cases, one where !-link is cut with $k \in \{1, 2, \ldots\}$ derelictions and one where it is cut with a weakening. In the first case the subgraph Q is

copied k times (if $k = 1$ no copy is done) as $Q^1, \dots Q^k$ and each copy enters the m_i boxes enclosing the corresponding (and removed) dereliction. Moreover, the k copies of each free variable of Q are contracted together, *i.e.* the nodes are merged. In the case of a cut with a weakening, Q is erased and replaced by a set of weakenings, one for every free variable of Q. Note that the weakenings are also pushed out of all boxes closing on the free variables of Q[5]. This is done to preserve the invariant that weakenings are always 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.

Now, we establish the relationship between terms and nets at the level of reduction. Essentially, there is only one fact which is not immediate, namely that $\to_{\mathtt{e}}$ actually implements the $\to_{\mathtt{e}}$ rule on terms, as it is proved by the following lemma.

Lemma 6.1 (Substitution)**.** *Let* $t = s[x{\leftarrow}L\langle v\rangle]$ *then* $\underline{t}_\Delta \to_{\mathtt{e}} \underline{L\langle s\{x{\leftarrow}v\}\rangle}_\Delta$ *for any set of names* $\Delta \supseteq \mathtt{fv}(t)$.

Proof. First of all observe that $t = s[x{\leftarrow}L\langle v\rangle]$ and $L\langle s[x{\leftarrow}v]\rangle$ both reduce to $L\langle s\{x{\leftarrow}v\}\rangle$ and by Remark 4.2 both translate to the same net. Hence it is enough to prove that $\underline{L\langle s[x{\leftarrow}v]\rangle}_\Delta \to_{\mathtt{e}} \underline{L\langle s\{x{\leftarrow}v\}\rangle}_\Delta$. We prove it by induction on L. For $L = \langle\cdot\rangle$ the proof is by induction on the number n of free occurrences of x in s. Cases:

> $n = 0$) In $\underline{s[x{\leftarrow}L\langle v\rangle]}_\Delta$ the !-link associated to v is cut with a weakening. The elimination of the cut produces a net P' without the !-link and the ⅋-box associated to v, leaving a free weakening for every free variable of the box, *i.e.* of every free variable of v: then P' is exactly $\underline{s\{x{\leftarrow}v\}}_{\Delta\cup\mathtt{fv}(v)} = \underline{s}_{\Delta\cup\mathtt{fv}(v)}$.
>
> $n > 1$) Write $s = C\langle x\rangle$ for some occurrence of x. Now, consider $u = C\langle y\rangle[y{\leftarrow}v][x{\leftarrow}v]$ and note that:
>
> $$u \to C\langle v\rangle[x{\leftarrow}v] \to C\langle v\rangle\{x{\leftarrow}v\} = s\{x{\leftarrow}v\}$$
>
> The difference between $P' = \underline{u}_\Delta$ and $P = \underline{s[x{\leftarrow}v]}_\Delta$ is that one of the occurrences of x in P has been separated from the others and cut with a copy of $\underline{v}$. Consider the step $P \to Q$ which reduces the cut on x in P and the sequence $P' \to Q'_y \to Q'_{y,x}$ which first reduces the cut on y in P' and then reduces in Q' the (unique) residual of the cut on x in P'. By the definition of reduction in nets $Q = Q'_{y,x}$. Now, the *i.h.* applied to u

[5]Note that, for the sake of a simple representation, the figure of the weakening cut-elimination rule is slightly wrong: given free a variable x_i of Q, the links $l_1, \dots, l_j$ having x_i as target (*i.e.* $l_1, \dots, l_j$ are all contracted together) are not necessarily all inside m_i boxes, as each one can be inside a different number of boxes.

and y gives $\underline{C\langle v\rangle[x{\leftarrow}v]}_\Delta = Q'_y$ and the *i.h.* applied to $C\langle v\rangle[x{\leftarrow}v]$ and x provides $\underline{C\langle v\rangle\{x{\leftarrow}v\}}_\Delta = Q'_{y,x}$. From $Q = Q'_{y,x}$ and $C\langle v\rangle\{x{\leftarrow}v\} = s\{x{\leftarrow}v\}$ we obtain $\underline{s\{x{\leftarrow}v\}}_\Delta = Q$ and conclude.

$n = 1$) By induction on s. Some cases:

Abstraction. If $s = \lambda y.u$ then by *i.h.* $\underline{u[x{\leftarrow}v]}_{\Delta\cup\{y\}} \to_{\mathtt{e}} \underline{u\{x{\leftarrow}v\}}_{\Delta\cup\{y\}}$ and so we obtain $\underline{\lambda y.(u[x{\leftarrow}v])}_{\Delta\cup\{y\}} \to_{\mathtt{e}} \underline{\lambda y.(u\{x{\leftarrow}v\})}_{\Delta\cup\{y\}}$. Now, observe that $\lambda y.(u\{x{\leftarrow}v\}) = (\lambda y.u)\{x{\leftarrow}v\} = t\{x{\leftarrow}v\}$ and that the two nets $\underline{\lambda y.(u[x{\leftarrow}v])}_{\Delta\cup\{y\}}$ and $\underline{(\lambda y.u)[x{\leftarrow}v]}_{\Delta\cup\{y\}}$ have the same reduct after firing the exponential cut on x, and so we obtain

$$\underline{(\lambda y.u)[x{\leftarrow}v]}_{\Delta\cup\{y\}} \to_{\mathtt{e}} \underline{(\lambda y.u)\{x{\leftarrow}v\}}_{\Delta\cup\{y\}}$$

Explicit Substitution. If $s = r[y{\leftarrow}u]$ then either $x \in u$ or $x \in r$. In the first case by Remark 4.2 we obtain that $\underline{s[x{\leftarrow}v]}_\Delta = \underline{r[y{\leftarrow}u][x{\leftarrow}v]}_\Delta = \underline{r[y{\leftarrow}u[x{\leftarrow}v]]}_\Delta$. Now by *i.h.* $\underline{u[x{\leftarrow}v]} \to_{\mathtt{e}} \underline{u\{x{\leftarrow}v\}}$. Then we have $\underline{s[x{\leftarrow}v]}_\Delta \to_{\mathtt{e}} \underline{r[y{\leftarrow}u\{x{\leftarrow}v\}]}_\Delta = \underline{r[y{\leftarrow}u]\{x{\leftarrow}v\}}_\Delta = \underline{s\{x{\leftarrow}v\}}_\Delta$. The second case is analogous.

Application which is an $\mathtt{m}$*-redex.* If $s = (\lambda y.r)u$. The case $x \in u$ uses Remark 4.2 and the *i.h.* as in the $s = r[y{\leftarrow}u]$ case. The case $x \in r$ is slightly different. As before $((\lambda y.r)u)[x{\leftarrow}v]$ and $((\lambda y.r[x{\leftarrow}v])u)$ have the same reduct. By *i.h.* hypothesis $\underline{r[x{\leftarrow}v]} \to_{\mathtt{e}} \underline{r\{x{\leftarrow}v\}}$ and thus $\underline{(\lambda y.r[x{\leftarrow}v])u}_\Delta \to_{\mathtt{e}} \underline{(\lambda y.r\{x{\leftarrow}v\})u}_\Delta$. We conclude since

$$\underline{((\lambda y.r)u)[x{\leftarrow}v]}_\Delta \to_{\mathtt{e}} \underline{((\lambda y.r\{x{\leftarrow}v\})u)}_\Delta = \underline{((\lambda y.r)u)\{x{\leftarrow}v\}}_\Delta$$

For $L = L'[y{\leftarrow}r]$ the *i.h.* gives $\underline{L'\langle s[x{\leftarrow}v]\rangle}_\Delta \to_{\mathtt{e}} \underline{L'\langle s\{x{\leftarrow}v\}\rangle}_\Delta$. By definition of the translation and of graph reduction it follows that $\underline{L'\langle s[x{\leftarrow}v]\rangle[y{\leftarrow}r]}_\Delta \to_{\mathtt{e}} \underline{L'\langle s\{x{\leftarrow}v\}\rangle[y{\leftarrow}r]}_\Delta$. □

The next theorem expresses the perfect match between rewriting steps on terms and rewriting steps on proof nets, and it is the main result of the paper.

Theorem 6.2 (Strong Bisimulation)**.** *Let t be a term and Δ a set of variables containing $\mathtt{fv}(t)$. The translation is a strong bisimulation between t and $\underline{t}_\Delta$,* i.e. *$t \to_a t'$ if and only if $\underline{t}_\Delta \to_a \underline{t'}_\Delta$, for $a \in \{\mathtt{m}, \mathtt{e}\}$.*

Proof. By induction on the translation. Cases:

1. *Variable, i.e.* $t = x$. Trivial, as both t and $\underline{t}_\Delta$ have no redexes.
2. *Abstraction or variable application, i.e.* $t = \lambda x.s$ or $t = xs$. The statement immediately follows by the *i.h.*, since all the redexes of t and $\underline{t}_\Delta$ are contained in s and $\underline{s}_\Delta$, respectively.

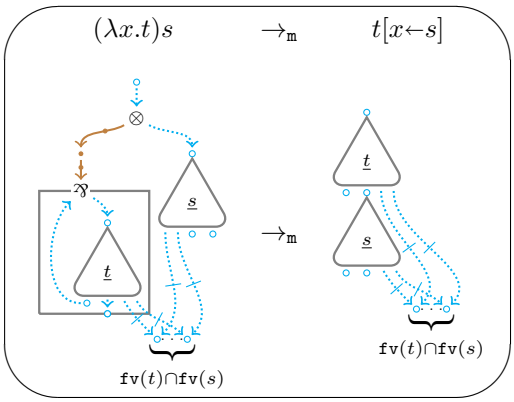

Figure 5: A $\to_{\mathtt{m}}$-step on terms and on nets.

3. $\mathtt{m}$-*redex at the root, i.e.* $t = (\lambda x.s)u$. For the redexes in s, u, $\underline{s}_{\Delta\cup\{x\}}$, and $\underline{u}_{\Delta}$ just use the *i.h.*. The only redex of t out of its subterms is the root $\mathtt{m}$-redex $t = (\lambda x.s)u \to_{\mathtt{m}} s[x{\leftarrow}u] = t'$ to which it corresponds a $\mathtt{m}$-redex in $\underline{t}_{\Delta}$ that is not contained in $\underline{s}_{\Delta\cup\{x\}}$ nor $\underline{u}_{\Delta}$. Look at Fig. 5: clearly $t \to_{\mathtt{m}} t'$ iff $\underline{t}_{\Delta} \to_{\mathtt{m}} \underline{t'}_{\Delta}$.
4. *Explicit substitution, i.e.* $t = s[x{\leftarrow}u]$. For the redexes in subterm/subnets we reason as in the previous points. Now, if $u = L\langle v\rangle$ then t has the following $\mathtt{e}$-redex out of subterms $s[x{\leftarrow}L\langle v\rangle] \to_{\mathtt{e}} L\langle s\{x{\leftarrow}v\}\rangle$. Then apply Lemma 6.1. Note moreover that $\underline{t}_{\Delta}$ has a redex out of subnets only if $u = L\langle v\rangle$, and that in such a case the redex is the one we just treated. □

Strong bisimulations preserve reduction lengths, so they preserve divergent/normalizing reductions, and termination properties in general.

Corollary 6.3. *Let $t \in \lambda_{vker}$ and Δ a set of variables. Then t is weakly normalizing/strongly normalizing/a normal form/weakly divergent/strongly divergent iff $\underline{t}_{\Delta}$ is.*

Actually, the translation is more than a strong bisimulation: the reduction graphs[6] of t and $\underline{t}$ are *isomorphic*, not just strongly bisimilar. An easy but tedious refinement of the proof of Theorem 6.2 (consisting in specifying the bijection of redexes) proves:

Theorem 6.4 (Dynamic Isomorphism)**.** *Let t be a term and Δ a set of variables containing $\mathtt{fv}(t)$. The translation induces a bijection ϕ between the redexes of t and the redexes of $\underline{t}_{\Delta}$ s.t. $R : t \to_a t'$ if and only if $\phi(R) : \underline{t}_{\Delta} \to_a \underline{t'}_{\Delta}$, where $a \in \{\mathtt{m}, \mathtt{e}\}$.*

[6] *Reduction graphs*, which are the graphs obtained considering all reductions starting from a given object, *are not nets*.

A nice by-product of our approach is that preservation of correctness by reduction *comes for free*, since any reduct of a proof net is the translation of a term.

Corollary 6.5 (Preservation of Correctness)**.** *Let P be a proof net and $P \to P'$. Then P' is correct.*

Note that there is no contradiction between 1) the fact that the translation is not injective, and 2) the dynamic isomorphism between terms and proof nets. As pointed out in Remark 4.2, the lack of injectivity of the translation induces a quotient on terms: each equivalence class contains all and only the different sequentialization of a proof net, or, equivalently, all the terms translating to the same proof net. This does not forbid every term in the class to behave isomorphically to the corresponding proof net (and thus, by transitivity, isomorphically to any other term in its class).

A strong bisimulation is enough for transporting termination properties between the two systems. For confluence, instead, a priori the situation is slightly different. In general, both strong bisimulations and dynamic isomorphisms transport confluence only modulo the quotient induced by the translation. Mild additional hypothesis allow to transfer plain confluence, see [1] (pp. 83-86) for more details. Our case however is simpler, as we want to transport confluence from terms to proof nets and the quotient induced on proof nets is the identity, *i.e.* here confluence modulo actually coincides with plain confluence. Then from confluence of λ_{vker} it follows that:

Theorem 6.6. *Proof nets are confluent.*

The Original Boring Translation. For the sake of completeness, Fig. 6 sketches the ordinary CBV translation from λ-terms (possibly with iterated applications) to proof nets (including the case for explicit substitutions and using a traditional syntax with boxes on !-links). An easy computation shows that the term $t = (\lambda z.\delta)(yy)\delta$, where $\delta = \lambda x.xx$ maps to a net without normal form, while t is a $\lambda_{\beta v}$-normal form. As already mentioned, this mismatch is the motivation behind our work.

7. Proof Nets: Comments and Literature

7.1. Motivating ⅋-boxes

The call-by-name (CBN) and call-by-value (CBV) encodings of λ-calculus into linear logic (LL) can be seen as fragments of Intuitionistic Multiplicative and Exponential LL (IMELL). Let us stress that in IMELL what we noted $\otimes$ and ⅋ correspond to the right and left rules for the linear implication $\multimap$, and not to the left and right rules for $\otimes$. The four rules for $\otimes$ and $\multimap$ are collapsed in LL but not in Intuitionistic LL, in particular our ⅋ acts on the output of the term, *i.e.* on the right of the sequent, and corresponds to the right rule for $\multimap$.

Our argument is that in IMELL there is no correctness criterion unless the syntax is extended with boxes for both ! *and* $\multimap$ (our ⅋), as we shall explain in

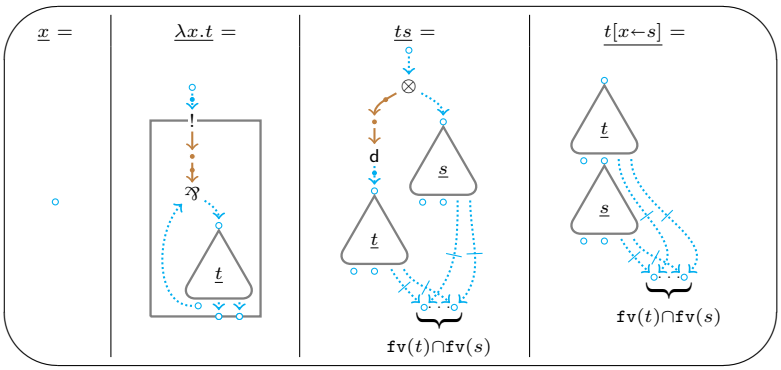

Figure 6: the ordinary CBV translation from terms to nets.

the next paragraphs. In particular, the two encodings of λ-calculus turn out to be *dual special cases* with respect to the use of boxes, in the following sense

- The fragment of IMELL encoding the CBN λ-calculus is a special case where the box for $\multimap$ needs not be represented.
- The fragment of IMELL encoding the CBV λ-calculus is a special case where the box for ! needs not be represented.

Therefore, our use of $\parr$-boxes actually unveils a nice symmetry, rather than being ad-hoc. In the literature there are occurrences of explicit boxes for abstractions (at least in [30, 21]), but none of these works studies correctness.

The difficulty of designing a correctness criterion for IMELL is given by the presence of weakenings, that break connectedness. In most cases weakenings simply prevent the possibility of a correctness criterion. The fragment encoding the CBN λ-calculus, and more generally Polarized Linear Logic, are notable exceptions. For the encoding of the CBN λ-calculus there exist two correctness criteria. Let us show that none of them works for the CBV λ-calculus:

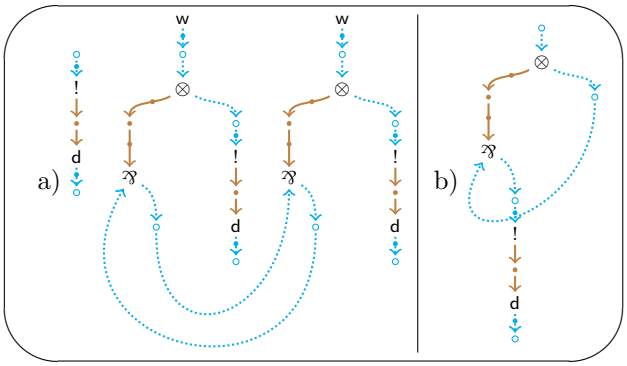

Figure 7: Counter-examples to correctness without $\parr$-boxes.

1. *Danos and Regnier's criterion*, in the variant replacing connectedness with the requirement that the number of connected components of every switching graph is $1 + \#w$, where $\#w$ is the number of weakenings at level 0 (after the collapse of boxes) [36]. In our case this criterion does not work: the net in Fig. 7.a verifies the requirement while it does not represent any proof or term. Note that adding boxes to the !-links of the example does not change anything.
2. *Laurent's polarized criterion* [28, 27], because the CBN encoding is polarized. In its original formulation it cannot be applied to the encoding of the CBV λ-calculus, because such a fragment is not polarized (there can be a weakening as a premise of a tensor, which is forbidden in polarized logic). Our re-formulation of Laurent's criterion (Definition 4.4, page 13) rejects the net in Fig. 7.a (because the two $⅋$-links form a cycle), but without using $⅋$-boxes it would accept the net in Fig. 7.b, which is not correct[7].

Thus, the known criteria do not work and—more generally—there is no criteria for IMELL. The usual way to circumvent problems about correctness is to add some information to the graphical representation, under the form of boxes (as in this paper) or jumps (*i.e.* additional connections). It is well known that in these cases various criteria can be used, but this extra information either is not canonical or limits the degree of parallelism. Another possible solution is to modify the logical system adding the mix rules. However, such rules are debatable, and also give rise to a bad notion of subnet (for details see [1], pp. 199-201).

Let us stress that our counterexamples to the known criteria do not rely on the exponentials (*i.e.* non-linearity): it is easy to reformulate them in Intuitionistic Multiplicative Linear Logic (IMLL) with unit(s)[8], for which then there is no correctness criterion.

In the case studied in this paper the use of $⅋$-boxes does not affect the level of parallelism in a considerable way. Indeed, in IMELL the parallelism of proof nets concerns the left rules (of $\otimes$ and $\multimap$, plus contractions and weakenings) and cuts: in our case there is no $\otimes$ (remember our $\otimes$ and $⅋$ rather correspond to the rules for $\multimap$), our technical choices for variables keep the parallelism for contraction and weakenings, and the parallelism of the left rule for $\multimap$ (our $\otimes$) and cuts is preserved (it is given by the equations in (1), page 11).

To best of our knowledge, the observations reported here—in particular those about the lack of correctness criteria—are an original contribution of this work.

[7] The net in Fig. 7.b would be rejected by the original version of the criterion, which is based on a different orientation. But the original orientation cannot be applied to the fragment under study.

[8] Just replace each sequence of a ! over a dereliction with an axiom, and the weakenings with $\bot$-links.

7.2. Literature on Term Representations

The relationship between λ-terms and proof nets presents a number of technical choices:

1. *Explicit Substitutions*, or *Sharing*: proof nets implement a β-step by (at least) two cut-elimination steps. This refined evaluation can be seen on the calculus only if the syntax is extended with explicit substitutions, that are nothing else but explicit sharing annotations.
2. *Variables*: to properly represent variables it is necessary to work modulo associativity and commutativity of contractions, neutrality of weakening with respect to contraction, and permutations of weakenings and contractions with box-borders. In the literature there are two approaches: to explicitly state all these additional congruences or to use a syntax naturally quotienting with respect to them. Such a syntax uses n-ary ?-links collapsing weakening, dereliction and contractions and delocalizing them out of boxes. It is sometimes called *nouvelle syntaxe*.
3. *Axioms*: various complications arise if proof nets are presented with explicit axiom and cut links. They can be avoided by working modulo cuts on axioms, which is usually done by employing an interaction nets presentation of proof nets.
4. *Exponential Cut-Elimination*: cut-elimination rules for the exponentials admit many presentations. Most of the time, either they are small-step, *i.e.* an exponential cut is eliminated in one shot (making many copies of the !-premise of the cut), or they are micro-step, with a rule for each possible ?-premise (weakening, dereliction, contraction, axiom, box auxiliary port).

Let us classify our study along these axes. We employ explicit substitutions and small-step exponential cut-elimination. For the representation of variables and axioms, instead, we depart from the literature and propose an alternative approach, the use of hypergraphs. Hypergraphs allow at the same time to mimic n-ary ?-links and collapse axioms and cut links as if we were using interaction nets. More precisely, we represent n-ary ?-links by allowing e-nodes to have more than one incoming link. This choice overcomes some technicalities about *gluing* and *degluing* of ?-links. Such technicalities are always omitted, but they are in fact necessary to properly define subnets and cut-elimination. In particular, with n-ary ?-links subnets cannot be defined as subsets of links (because of (de)gluing), that turns out to be technically quite annoying (unless one is deliberately less precise and forgets to deal with (de)gluing altogether). Of course, everything we did here could be also done using n-ary ?-links and interaction nets, it is rather a matter of taste.

We now list the works in the literature which are closer in spirit to ours, *i.e.* focusing on the representation of λ-calculi into proof nets (and thus we omit many other interesting works, as for instance [31], which studies the representation of *strategies*, not of *calculi*).

The first such works were the PhD thesis of Vincent Danos [14] and Laurent Regnier [36], that focused on the call-by-name (CBN) translation. Danos and

Regnier avoid explicit substitutions, use n-ary contractions, explicit axioms, and small-step exponential rules, see also [13]. They characterize the image of the translation using the variant of the Danos-Regnier correctness criterion requiring that any switching graph has $\#w + 1$ connected components, where $\#w$ is the number of weakenings. In [15], Danos and Regnier use the CBV translation[9]. Both translations are injective.

In [28, 27] Olivier Laurent extends the CBN translation to represent (the CBN) $\lambda\mu$-calculus. He does not use explicit substitutions nor n-ary ?-links, while he employs explicit axiom links and micro-step exponential rules. His work presents two peculiar points. First, the translation of $\lambda\mu$-terms is not injective, because—depending on the term—the μ-construct may have no counterpart on proof nets. This induces some mismatches at the dynamic level. Second, Laurent finds a simpler criterion (which is the one we used—adapted—in this paper), exploiting the fact that the fragment encoding (the CBN) $\lambda\mu$-calculus is polarized. In [27] Laurent also show how to represent the CBV $\lambda\mu$-calculus. However, such a representation does not use the same types as the boring translation, since $A \to B$ maps to $?!(A \multimap B)$, and not to $!(A \multimap B)$.

Lionel Vaux [40] and Paolo Tranquilli [38, 39] study the relationship between the differential λ-calculus and differential proof nets. 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. They do not use explicit substitutions, nor n-ary contractions, while they use interaction nets (so no explicit axioms and cut link) and micro-step exponential rules. Both Tranquilli and Vaux rely on the Danos-Regnier criterion, despite the fragment encoding their calculi is polarized and can be captured using Laurent's criterion by using boxes for coderelictions; in the context of λ-calculus such boxes do not reduce the parallelism of the representation.

Delia Kesner and co-authors [12, 16, 23] study the relationship with explicit substitutions in the CBN case. The main idea here is that explicit substitutions correspond to exponential cuts. They use explicit axiom links and micro-step exponential rules, but they do not employ n-ary contractions (and so they need additional rules and congruences). Because of explicit substitutions the translation is not injective: now different terms may map to the same proof net, as in this paper. They do not deal with correctness criteria.

In none of these works the translation is an isomorphism of rewriting systems nor a strong bisimulation. In [7] Stefano Guerrini and the author use a syntax inspired by proof nets (and extended with jumps) to represent the CBN λ-calculus with explicit substitutions. That work is the only one employing at the same time (the equivalent of) n-ary ?-links and (the equivalent of) micro-step

[9]Let us point out that [15] presents an oddity that we believe deserves to be clarified. The authors show that an optimized geometry of interaction for the proof nets of the CBV-translation is isomorphic to Krivine Abstract Machine (KAM): this is quite puzzling, because the KAM is CBN, while they use the CBV translation.

exponential rules. In [7] the correctness criterion is a variation over Lamarche's criterion for essential nets [25], which relies in an essential way on the use of jumps. A reformulation in the syntactic style of this paper of both [7] and of Danos and Regnier's proof nets for the CBN λ-calculus can be found in [1], together with a detailed account of the strong bisimulation.

Last, a nice and detailed introduction to the relationship between λ-terms and proof nets is Guerrini's [20].

8. Terms: Relating λ_{vsub} and λ_{vker}

In this section we study the relationship between λ_{vsub} and λ_{vker}. The results of this paper actually allow to read this section as if we were relating λ_{vsub} and proof nets, given that λ_{vker} and proof nets are isomorphic. In particular, by composing the (forthcoming) translation from λ_{vsub} to λ_{vker} and the translation from λ_{vker} to proof nets (Sect. 4) one obtains a translation from λ_{vsub} to proof nets. We believe, actually, that our isomorphism between terms and proof nets shines at its best here: having an algebraic language for proof nets we can relate them to λ_{vsub} at a level of detail and rigor that would be impossible if we had to deal directly with the graphical syntax.

8.1. Projecting λ_{vsub} on λ_{vker}

This subsection shows that λ_{vsub} has an interpretation inside λ_{vker}.

Notation: when we want to stress that a term is in λ_{vker} we use capital letters.

Any term $M \in \lambda_{vker}$ is a term of λ_{vsub}. Moreover, on λ_{vker} terms the rules of λ_{vsub} collapse on the rules of λ_{vker}. Then, $M \to_{\lambda_{vker}} N$ iff $M \to_{\lambda_{vsub}} N$, for any $M \in \lambda_{vker}$. Hence, λ_{vker} is a subcalculus of λ_{vsub} which is stable by reduction, and so

Lemma 8.1. *$M \in \lambda_{vker}$ is λ_{vker}-strongly (resp. weakly) normalizing iff λ_{vsub}-strongly (resp. weakly) normalizing.*

However, there are terms of λ_{vsub} which are not terms of λ_{vker}. The purpose of this section is to show that these terms can be faithfully simulated in λ_{vker} via an appropriate translation.

The translation $\cdot^{\mathtt{k}} : \lambda_{vsub} \to \lambda_{vker}$ is defined as follows:

$$
\begin{array}{lcll}
x^{\mathtt{k}} & := & x & \\
(\lambda x.t)^{\mathtt{k}} & := & \lambda x.t^{\mathtt{k}} & \\
t[x{\leftarrow}s]^{\mathtt{k}} & := & t^{\mathtt{k}}[x{\leftarrow}s^{\mathtt{k}}] & \\
(L\langle v\rangle s)^{\mathtt{k}} & := & L^{\mathtt{k}}\langle v^{\mathtt{k}} s^{\mathtt{k}}\rangle & \\
(L\langle ts\rangle u)^{\mathtt{k}} & := & (xu^{\mathtt{k}})[x{\leftarrow}L\langle ts\rangle^{\mathtt{k}}] & \text{where } x \text{ is fresh} \\
\langle\cdot\rangle^{\mathtt{k}} & := & \langle\cdot\rangle & \\
L[x{\leftarrow}t]^{\mathtt{k}} & := & L^{\mathtt{k}}[x{\leftarrow}t^{\mathtt{k}}] &
\end{array}
$$

There is a slight mismatch between reductions in λ_{vsub} and in λ_{vker}. Sometimes a $\to_{\mathtt{m}}$ step in λ_{vsub} is simulated by a single $\to_{\mathtt{m}}$ step in λ_{vker}, while sometimes it requires an additional $\to_{\mathtt{e}}$ step. Consider $t := (\lambda x.\lambda y.t)su$, that translates to $t^{\mathtt{k}} = (zu^{\mathtt{k}})[z{\leftarrow}(\lambda x.\lambda y.t^{\mathtt{k}})s^{\mathtt{k}}]$ with z fresh. We have:

$$t \quad \to_{\mathtt{m}} \quad (\lambda y.t)[x{\leftarrow}s]u \quad =: \quad t'$$

But

$$t^{\mathtt{k}} \quad \to_{\mathtt{m}} \quad (zu^{\mathtt{k}})[z{\leftarrow}(\lambda y.t^{\mathtt{k}})[x{\leftarrow}s^{\mathtt{k}}]] \quad \to_{\mathtt{e}} \quad ((\lambda y.t^{\mathtt{k}})u^{\mathtt{k}})[x{\leftarrow}s^{\mathtt{k}}] = t'^{\mathtt{k}}$$

It is quite obvious that λ_{vker} can simulate λ_{vsub} with only a linear overhead (in terms of number of rewriting steps), because every step in λ_{vsub} can be simulated by at most two steps in λ_{vker}. We shall however prove this formally in the rest of this section. It is less obvious that any reduction sequence in λ_{vker} can be extended with a linear number of steps so that it represents a reduction sequence in λ_{vsub}. This will be shown in the next section, and will require some commutation properties. The two results together will provide the equivalence with respect to termination of λ_{vsub} and λ_{vker}, as we will show in Sect. 8.3.

Lemma 8.2 (Properties of $\cdot^{\mathtt{k}}$)**.** *The translation $\cdot^{\mathtt{k}}$ has the following properties:*

1. Values*: t is a value iff $t^{\mathtt{k}}$ is a value.*
2. Substitutivity*: $t\{x{\leftarrow}v\}^{\mathtt{k}} = t^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}$.*

Proof.

1. It is immediate that $v^{\mathtt{k}}$ is a value. If $t^{\mathtt{k}}$ is a value then either $t^{\mathtt{k}} = x$ and thus $t = x$, or $t^{\mathtt{k}} = \lambda x.N$, but by definition of the translation this is only possible if $t = \lambda x.s$ and $s^{\mathtt{k}} = N$.
2. By induction on t. The variable case is trivial and the abstraction case follows from the *i.h.*. The application case $t = su$ splits into two subcases. *Notation*: given L we define $L\{x{\leftarrow}v\}$ as follows $\langle\cdot\rangle\{x{\leftarrow}v\} := \langle\cdot\rangle$ and $L[y{\leftarrow}t] := L\{x{\leftarrow}v\}[y{\leftarrow}t\{x{\leftarrow}v\}]$. Moreover, to diminish the use of parentheses, we use $t\{x{\leftarrow}s\}^{\mathtt{k}}$ for $(t\{x{\leftarrow}s\})^{\mathtt{k}}$ (as in the statement of Point 2). Subcases:

   The left subterm is a value in a substitution context, i.e. $s = L\langle v'\rangle$. Then:

$$\begin{array}{lcll} t\{x{\leftarrow}v\}^{\mathtt{k}} & = & (L\langle v'\rangle u)\{x{\leftarrow}v\}^{\mathtt{k}} & \text{let } L' := L\{x{\leftarrow}v\} \\ & = & (L'\langle v'\{x{\leftarrow}v\}\rangle u\{x{\leftarrow}v\})^{\mathtt{k}} & \text{let } L'' := L^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\} \\ & = & L''\langle v'\{x{\leftarrow}v\}^{\mathtt{k}}u\{x{\leftarrow}v\}^{\mathtt{k}}\rangle & \\ & =_{i.h.} & L''\langle v'^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}u^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}\rangle & \\ & = & L^{\mathtt{k}}\langle v'^{\mathtt{k}}u^{\mathtt{k}}\rangle\{x{\leftarrow}v^{\mathtt{k}}\} & \\ & = & t^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\} & \end{array}$$

The left subterm is an application in a substitution context, i.e. $s = L\langle rp\rangle$. Then:

$$
\begin{array}{lll}
t\{x{\leftarrow}v\}^{\mathtt{k}} & = & \\
(L\langle rp\rangle u)\{x{\leftarrow}v\}^{\mathtt{k}} & = & \text{let } L' := L\{x{\leftarrow}v\} \\
(L'\langle r\{x{\leftarrow}v\}p\{x{\leftarrow}v\}\rangle u\{x{\leftarrow}v\})^{\mathtt{k}} & =_{\text{def. of } \cdot^{\mathtt{k}}} & \text{let } y \text{ be } \textit{fresh} \\
(yu\{x{\leftarrow}v\}^{\mathtt{k}})[y{\leftarrow}L'\langle r\{x{\leftarrow}v\}p\{x{\leftarrow}v\}\rangle^{\mathtt{k}}] & =_{\textit{i.h.}} & \text{let } L'' := L^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\} \\
(yu^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\})[y{\leftarrow}L''\langle r^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}p^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}\rangle] & = & \\
(yu^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\})[y{\leftarrow}L\langle rp\rangle^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}] & = & \\
(yu^{\mathtt{k}})[y{\leftarrow}L\langle rp\rangle^{\mathtt{k}}]\{x{\leftarrow}v^{\mathtt{k}}\} & = & \\
(L\langle rp\rangle u)^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\} & = & \\
t^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\} & &
\end{array}
$$

□

The next theorem relates the reductions of t and $t^{\mathtt{k}}$.

Theorem 8.3 ($(\cdot)^{\mathtt{k}}$ and Reductions)**.**

1. Projection of a $\to_{\mathtt{m}}$ step*: $t \to_{\mathtt{m}} s$ implies $t^{\mathtt{k}} \to_{\mathtt{m}} s^{\mathtt{k}}$ or $t^{\mathtt{k}} \to_{\mathtt{m}}\to_{\mathtt{e}} s^{\mathtt{k}}$.*
2. Reflection of a $\to_{\mathtt{m}}$ step*: if $t^{\mathtt{k}} \to_{\mathtt{m}} N$ then there exists s s.t. $t \to_{\mathtt{m}} s$ and either $s^{\mathtt{k}} = N$ or $N \to_{\mathtt{e}} s^{\mathtt{k}}$.*
3. Projection and reflection of a $\to_{\mathtt{e}}$ step*: $t \to_{\mathtt{e}} s$ iff $t^{\mathtt{k}} \to_{\mathtt{e}} s^{\mathtt{k}}$.*

Proof. By induction on $(\cdot)^{\mathtt{k}}$. The variable case is trivial and the abstraction case follows from the *i.h.*. The other cases:

Explicit substitution: $t = u[x{\leftarrow}r]$ and $t^{\mathtt{k}} = u^{\mathtt{k}}[x{\leftarrow}r^{\mathtt{k}}]$. If the reduction takes place in u, r, $u^{\mathtt{k}}$, or $r^{\mathtt{k}}$ then it follows from the *i.h.*. Note that $\to_{\mathtt{m}}$ steps can only take place in subterms of t, so we have proved Point 1 and Point 2. Point 3:

$\Rightarrow$) If $r = L\langle v\rangle$ then $t = u[x{\leftarrow}L\langle v\rangle] \to_{\mathtt{e}} L\langle u\{x{\leftarrow}v\}\rangle = s$ and $t^{\mathtt{k}} = u^{\mathtt{k}}[x{\leftarrow}L^{\mathtt{k}}\langle v^{\mathtt{k}}\rangle]$. By Lemma 8.2.1 $v^{\mathtt{k}}$ is a value and $t^{\mathtt{k}} \to_{\mathtt{e}} L^{\mathtt{k}}\langle u^{\mathtt{k}}\{x{\leftarrow}v^{\mathtt{k}}\}\rangle = N$. Now, Lemma 8.2.2 gives us $s^{\mathtt{k}} = N$.

$\Leftarrow$) If $r^{\mathtt{k}} = L\langle v\rangle$ then r has the form $L'\langle v'\rangle$ with $v'^{\mathtt{k}} = v$ and $L'^{\mathtt{k}} = L$, and we conclude using the reasoning for the other direction.

Application: two subcases, depending on the shape of the left subterm:

The left subterm is a value in a substitution context: $t = L\langle v\rangle r$ and $t^{\mathtt{k}} = L^{\mathtt{k}}\langle v^{\mathtt{k}}r^{\mathtt{k}}\rangle$. If the reduction takes place in v, r, L, $v^{\mathtt{k}}$, $r^{\mathtt{k}}$, or $L^{\mathtt{k}}$ then it follows from the *i.h.*. Otherwise:

1. *Projection of a $\to_{\mathtt{m}}$ step.* If $v = \lambda x.u$ then $t = L\langle\lambda x.u\rangle r \to_{\mathtt{m}} L\langle u[x{\leftarrow}r]\rangle = s$ and $t^{\mathtt{k}} = L^{\mathtt{k}}\langle\lambda x.u^{\mathtt{k}}r^{\mathtt{k}}\rangle \to_{\mathtt{e}} L^{\mathtt{k}}\langle u^{\mathtt{k}}[x{\leftarrow}r^{\mathtt{k}}]\rangle = s^{\mathtt{k}}$.
2. *Reflection of a $\to_{\mathtt{m}}$ step.* If $v^{\mathtt{k}} = \lambda x.u$ then $v = \lambda x.u'$ with $u'^{\mathtt{k}} = u$, and we repeat the same reasoning.

3. *Projection and reflection of a $\to_{\mathtt{e}}$ step.* The reasoning is analogous to the case $t = u[x \leftarrow r]$.

The left subterm is an application in a substitution context: $t = L\langle up\rangle r$ and $t^{\mathtt{k}} = (yr^{\mathtt{k}})[y \leftarrow L\langle up\rangle^{\mathtt{k}}]$, with y fresh. If the reduction takes place in u, p, r, L, $u^{\mathtt{k}}$, $p^{\mathtt{k}}$, $r^{\mathtt{k}}$, or $L^{\mathtt{k}}$ then it follows from the *i.h.*. Otherwise:

1. *Projection of a $\to_{\mathtt{m}}$ step.* If $u = L'\langle \lambda x.q\rangle$ then $t \to_{\mathtt{m}} L\langle L'\langle q[x \leftarrow p]\rangle\rangle r = s$ and

$$\begin{array}{rcll} t^{\mathtt{k}} & = & (yr^{\mathtt{k}})[y \leftarrow L\langle L'\langle \lambda x.q\rangle p\rangle^{\mathtt{k}}] & = \\ & & (yr^{\mathtt{k}})[y \leftarrow L^{\mathtt{k}}\langle L'^{\mathtt{k}}\langle (\lambda x.q^{\mathtt{k}})p^{\mathtt{k}}\rangle\rangle] & \to_{\mathtt{m}} \\ & & (yr^{\mathtt{k}})[y \leftarrow L'^{\mathtt{k}}\langle L^{\mathtt{k}}\langle q^{\mathtt{k}}[x \leftarrow p^{\mathtt{k}}]\rangle\rangle] & = \quad N \end{array}$$

   Now, there are two cases, depending on the shape of q:
   (a) *q is a value in a substitution context, i.e. $q = L''\langle v'\rangle$.* Then $s = L\langle L'\langle q[x \leftarrow p]\rangle\rangle r = L\langle L'\langle L''\langle v'\rangle[x \leftarrow p]\rangle\rangle r$ and

$$\begin{array}{rcll} N & = & (yr^{\mathtt{k}})[y \leftarrow L^{\mathtt{k}}\langle L'^{\mathtt{k}}\langle L''\langle v'\rangle^{\mathtt{k}}[x \leftarrow p^{\mathtt{k}}]\rangle\rangle] & \\ & = & (yr^{\mathtt{k}})[y \leftarrow L^{\mathtt{k}}\langle L'^{\mathtt{k}}\langle L''^{\mathtt{k}}\langle v'^{\mathtt{k}}\rangle[x \leftarrow p^{\mathtt{k}}]\rangle\rangle] & \\ & \to_{\mathtt{e}} & L^{\mathtt{k}}\langle L'^{\mathtt{k}}\langle L''^{\mathtt{k}}\langle v'^{\mathtt{k}}r^{\mathtt{k}}\rangle[x \leftarrow p^{\mathtt{k}}]\rangle\rangle & = \quad s^{\mathtt{k}} \end{array}$$

   (b) *q is an application in a substitution context, i.e. $u = L''\langle qm\rangle$:* then

$$s = L\langle L'\langle L''\langle qm\rangle[x \leftarrow p]\rangle\rangle r$$

   and

$$N = (yr^{\mathtt{k}})[y \leftarrow L'^{\mathtt{k}}\langle L^{\mathtt{k}}\langle L''\langle qm\rangle^{\mathtt{k}}[x \leftarrow p^{\mathtt{k}}]\rangle\rangle] = s^{\mathtt{k}}$$

2. *Reflection of a $\to_{\mathtt{m}}$ step.* If $t^{\mathtt{k}} = (yr^{\mathtt{k}})[y \leftarrow L^{\mathtt{k}}\langle u^{\mathtt{k}}p^{\mathtt{k}}\rangle]$ has a $\mathtt{m}$-redex not contained in one of the subterms then necessarily $u^{\mathtt{k}} = L'\langle \lambda x.Q\rangle$ and so $u = L''\langle \lambda x.q\rangle$ with $q^{\mathtt{k}} = Q$ and $L''^{\mathtt{k}} = L'$. Then the proof repeats the reasoning of the previous point.
3. *Projection and reflection of a $\to_{\mathtt{e}}$ step.* The reasoning is analogous to the case $t = u[x \leftarrow r]$.

□

An immediate consequence is:

Corollary 8.4 (λ_{vker} Linearly Simulates λ_{vsub})**.** *Let $t \in \lambda_{vsub}$ and $t \to^{k}_{\lambda_{vsub}} s$. Then there exists j s.t. $k \leq j \leq 2k$ and $t^{\mathtt{k}} \to^{j}_{\lambda_{vker}} s^{\mathtt{k}}$.*

8.2. Reflecting λ_{vker} reductions in λ_{vsub}

Corollary 8.4 projects reduction sequences in λ_{vsub} on sequences in λ_{vker}. Here we study the converse simulations, *i.e.* how to simulate an evaluation sequence in λ_{vker} in λ_{vsub}, which is more complex because the relation between $\mathtt{m}$-steps may involve an additional step in λ_{vker}. Let us give a status to such additional steps.

Definition 8.5 (Adjusting Reduction)**.** *The* adjusting rewriting relation $t \to_{\mathtt{adj}} s$ *is defined as follows:*

$$\begin{array}{c}\textsc{Rule at Top Level}\\ (xu)[x{\leftarrow}L\langle v\rangle] \quad \mapsto_{\mathtt{adj}} \quad L\langle vu\rangle \quad \textit{with } x \notin \mathtt{fv}(u)\end{array}$$

$$\begin{array}{c}\textsc{Contextual Closure}\\ C\langle t\rangle \quad \to_{\mathtt{adj}} \quad C\langle s\rangle \qquad \textit{iff } t \mapsto_{\mathtt{adj}} s\end{array}$$

The relationship between $\to_{\mathtt{m}}$ steps in the two calculi (given by Point 1 and Point 2 of Theorem 8.3) then becomes:

1. If $t \to_{\mathtt{m}} s$ then either $t^{\mathtt{k}} \to_{\mathtt{m}} s^{\mathtt{k}}$ or $t^{\mathtt{k}} \to_{\mathtt{m}} \to_{\mathtt{adj}} s^{\mathtt{k}}$.
2. If $t^{\mathtt{k}} \to_{\mathtt{m}} N$ then there exists s s.t. $t \to_{\mathtt{m}} s$ and either $s^{\mathtt{k}} = N$ or $N \to_{\mathtt{adj}} s^{\mathtt{k}}$.

Let us explain the technical difficulty in reflecting reductions from λ_{vker} to λ_{vsub}. Suppose that:

1. $t^{\mathtt{k}} \to_{\lambda_{vker}} s_1 \to^+_{\lambda_{vker}} s$ is a reduction sequence whose first step $t^{\mathtt{k}} \to_{\lambda_{vker}} s_1$ requires an adjusting step $s_1 \to_{\mathtt{adj}} u$ in order to be the simulation of a step $t \to_{\mathtt{m}} r$ on λ_{vsub} (with $r^{\mathtt{k}} = u$), and that
2. this adjusting step $s_1 \to_{\mathtt{adj}} u$ is not the first step of the suffix $s_1 \to^+_{\lambda_{vker}} s$. In terms of diagrams we have:

$$\begin{array}{ccccc} t^{\mathtt{k}} & \longrightarrow_{\mathtt{m}} & s_1 & \longrightarrow^+ & s \\ & & \downarrow_{\mathtt{adj}} & & \\ & & u & & \end{array}$$

Then we have to show that the reduction sequence $s_1 \to^+_{\lambda_{vker}} s$ and the adjusting step $s_1 \to_{\mathtt{adj}} u$ *commute*, *i.e.* that there exists p s.t.

$$\begin{array}{ccccc} t^{\mathtt{k}} & \longrightarrow_{\mathtt{m}} & s_1 & \longrightarrow^+ & s \\ & & \downarrow_{\mathtt{adj}} & & \vdots \\ & & u & \dashrightarrow^* & \downarrow_{\mathtt{adj}^*} \\ & & & & p \end{array}$$

i.e. that doing first the adjusting step we do not forbid the computation in the suffix $s_1 \to^+_{\lambda_{vker}} s$ to take place (translated as a reduction sequence $u \to^*_{\lambda_{vker}} p$).

However, to later relate termination properties of λ_{vsub} and λ_{vker} we need a bit more than that. In fact, commutation of $\to_{\mathtt{adj}}$ and $\to_{\lambda_{vker}}$ is in itself trivial, because $\to_{\mathtt{e}}$ commutes with $\to_{\lambda_{vker}}$ and a $\to_{\mathtt{adj}}$ step is a $\to_{\mathtt{e}}$ step. What we need to show is that this commutation does not change the length of reduction sequences in a way which may affect termination. Technically, this will be given

by the fact that commutation with $\to_{\mathtt{adj}}$ does not change the number of $\mathtt{m}$-steps (in contrast to commutation with $\to_{\mathtt{e}}$).

Lemma 8.6 (Commutations of $\to_{\mathtt{adj}}$)**.** *Let $t \in \lambda_{vker}$. Then:*

1. Span of $\to_{\mathtt{e}}$ and $\to_{\mathtt{adj}}$*:*

$$\begin{array}{ccc} t & \longrightarrow_{\mathtt{e}} & s_1 \\ \downarrow_{\mathtt{adj}} & & \\ s_2 & & \end{array} \qquad \textit{implies that either } s_1 = s_2 \textit{ or exists } u \textit{ s.t.} \qquad \begin{array}{ccc} t & \longrightarrow_{\mathtt{e}} & s_1 \\ \downarrow_{\mathtt{adj}} & & \downarrow_{\mathtt{adj}^*} \\ s_2 & \longrightarrow_{\mathtt{e}} & u \end{array}$$

2. Span of $\to_{\mathtt{e}}$ and $\to^*_{\mathtt{adj}}$*:*

$$\begin{array}{ccc} t & \longrightarrow_{\mathtt{e}} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \\ s_2 & & \end{array} \qquad \textit{implies that either } s_1 = s_2 \textit{ or exists } u \textit{ s.t.} \qquad \begin{array}{ccc} t & \longrightarrow_{\mathtt{e}} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \downarrow_{\mathtt{adj}^*} \\ s_2 & \longrightarrow_{\mathtt{e}} & u \end{array}$$

3. Span of $\to_{\mathtt{m}}$ and $\to_{\mathtt{adj}}$*:*

$$\begin{array}{ccc} t & \longrightarrow_{\mathtt{m}} & s_1 \\ \downarrow_{\mathtt{adj}} & & \\ s_2 & & \end{array} \qquad \textit{implies that exists } u \textit{ s.t.} \qquad \begin{array}{ccc} t & \longrightarrow_{\mathtt{m}} & s_1 \\ \downarrow_{\mathtt{adj}} & & \downarrow_{\mathtt{adj}} \\ s_2 & \longrightarrow_{\mathtt{m}} & u \end{array}$$

4. Span of $\to_{\mathtt{m}}$ and $\to^*_{\mathtt{adj}}$*:*

$$\begin{array}{ccc} t & \longrightarrow_{\mathtt{m}} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \\ s_2 & & \end{array} \qquad \textit{implies that exists } u \textit{ s.t.} \qquad \begin{array}{ccc} t & \longrightarrow_{\mathtt{m}} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \downarrow_{\mathtt{adj}^*} \\ s_2 & \longrightarrow_{\mathtt{m}} & u \end{array}$$

Proof.

1. By induction on $t \to_{\mathtt{e}} s_1$. The case $s_1 = s_2$ is given by the case in which the $\to_{\mathtt{e}}$ step is $t \to_{\mathtt{adj}} s_2$.
2. By induction on the length of $t \to^*_{\mathtt{adj}} s_2$, using Point 1.
3. By induction on $t \to_{\mathtt{m}} s_1$.
4. By induction on the length of $t \to^*_{\mathtt{adj}} s_2$, using Point 3.

□

We can now show the required commutation property.

Lemma 8.7 (Commutation of $\to_{\lambda_{vker}}$ and $\to_{\mathtt{adj}}$)**.** *Let $t \in \lambda_{vker}$. Then*

$$\begin{array}{ccc} t & \overset{k}{\longrightarrow} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \\ s_2 & & \end{array} \qquad \textit{implies that exists } u \textit{ s.t.} \qquad \begin{array}{ccc} t & \overset{k}{\longrightarrow} & s_1 \\ \downarrow_{\mathtt{adj}^*} & & \downarrow_{\mathtt{adj}^*} \\ s_2 & \overset{\leq k}{\longrightarrow} & u \end{array}$$

Moreover, the two derivations $t \to^k s_1$ and $s_2 \to^{\leq k} u$ have the same number of $\mathtt{m}$*-steps.*

Proof. Simple induction on k, using Lemma 8.6. □

Now, we are ready to lift reduction sequences from $t^{\mathtt{k}}$ to t. As expected they are lifted only modulo an extension by adjusting steps.

Lemma 8.8 (Lifting Reduction Sequences from $t^{\mathtt{k}}$ to t)**.** *Let $t \in \lambda_{vsub}$ and $t^{\mathtt{k}} \to^{k}_{\lambda_{vker}} N$. Then there exists s s.t. $t \to^{\leq k}_{\lambda_{vsub}} s$ and $N \to^{*}_{\mathtt{adj}} s^{\mathtt{k}}$. Moreover, the derivations $t^{\mathtt{k}} \to^{k}_{\lambda_{vker}} N$ and $t \to^{\leq k}_{\lambda_{vsub}} s$ have the same number of $\mathtt{m}$-steps.*

Proof. By induction on k. If $k = 0$ there is nothing to prove. Then let $k > 0$, *i.e.* $t^{\mathtt{k}} \to_{\lambda_{vker}} u_1 \to^{k-1}_{\lambda_{vker}} N$. By Theorem 8.3.2 there is a term s_1 s.t. $t \to_{\lambda_{vsub}} s_1$ and either $u_1 = s_1^{\mathtt{k}}$ or $u_1 \to_{\mathtt{adj}} s_1^{\mathtt{k}}$. Cases:

> *No need of an adjusting step,* i.e. $u_1 = s_1^{\mathtt{k}}$. Then by *i.h.* applied to $s_1^{\mathtt{k}} = u_1 \to^{k-1} N$ there exists s s.t. $s_1 \to^{\leq k-1} s$, $N \to^{*}_{\mathtt{adj}} s^{\mathtt{k}}$, and the two derivations $s_1 \to^{\leq k-1} s$ and $s_1^{\mathtt{k}} = u_1 \to^{k-1} N$ have the same number of $\mathtt{m}$-steps. Then $t \to_{\lambda_{vsub}} s_1 \to^{\leq k-1} s$ has length $\leq k$ and we conclude, because the step $t^{\mathtt{k}} \to_{\lambda_{vker}} u_1$ is a $\mathtt{m}$-step iff $t \to_{\lambda_{vsub}} s_1$ is.
>
> *An adjusting step is required,* i.e. $u_1 \to_{\mathtt{adj}} s_1^{\mathtt{k}}$. Then both $t^{\mathtt{k}} \to_{\lambda_{vker}} u_1$ and $t \to_{\lambda_{vsub}} s_1$ are $\mathtt{m}$-steps. By Lemma 8.7 applied to $u_1 \to_{\mathtt{adj}} s_1^{\mathtt{k}}$ and the suffix $u_1 \to^{k-1}_{\lambda_{vker}} N$ there exists Q s.t. $s_1^{\mathtt{k}} \to^{\leq k-1}_{\lambda_{vker}} Q$ and $N \to^{*}_{\mathtt{adj}} Q$, and moreover both derivations $s_1^{\mathtt{k}} \to^{\leq k-1}_{\lambda_{vker}} Q$ and $u_1 \to^{k-1}_{\lambda_{vker}} N$ have the same number of $\mathtt{m}$-steps. Now, let's apply the *i.h.* to $s_1^{\mathtt{k}} \to^{\leq k-1}_{\lambda_{vker}} Q$: we obtain s s.t. $s_1 \to^{\leq k-1}_{\lambda_{vsub}} s$ and $Q \to^{*}_{\mathtt{adj}} s^{\mathtt{k}}$, and moreover both derivations $s_1^{\mathtt{k}} \to^{\leq k-1}_{\lambda_{vker}} Q$ and $s_1 \to^{\leq k-1}_{\lambda_{vsub}} s$ have the same number of $\mathtt{m}$-steps, which is the same of $u_1 \to^{k-1}_{\lambda_{vker}} N$. We have proved the statement, because
>
> 1. $N \to^{*}_{\mathtt{adj}} Q$ and $Q \to^{*}_{\mathtt{adj}} s^{\mathtt{k}}$, *i.e.* $N \to^{*}_{\mathtt{adj}} s^{\mathtt{k}}$;
> 2. $t \to_{\mathtt{m}} s_1 \to^{\leq k-1}_{\lambda_{vsub}} s$ has length $\leq k$, and
> 3. this derivation has the same number of $\mathtt{m}$-steps of $t^{\mathtt{k}} \to_{\mathtt{m}} u_1 \to^{k-1}_{\lambda_{vker}} N$.
>
> □

8.3. Relating Normalization Properties

Here we use the results of this section to relate termination properties in λ_{vsub} and λ_{vker}, in particular we show that the translation $\cdot^{\mathtt{k}}$ of λ_{vsub} to λ_{vker} preserves λ_{vsub}-strong normalization.

Theorem 8.9 ($\cdot^{\mathtt{k}}$ and Normalization)**.** *Let $t \in \lambda_{vsub}$.*

1. *t is a normal form iff $t^{\mathtt{k}}$ is a normal form.*
2. *t is strongly (resp. weakly) normalizing iff $t^{\mathtt{k}}$ is strongly (resp. weakly) normalizing.*

Notations: we use d, e, f, g for *derivations* (*i.e.* reduction sequences), and note $|d|_{\mathtt{m}}$ (resp. $|d|$) the number of $\mathtt{m}$-steps (resp. steps) in d.

Proof.

1. By Theorem 8.3 if t has a redex then $t^{\mathtt{k}}$ has a redex, and viceversa.
2. $\Rightarrow$) we split strong and weak normalization:

   Strong Normalization. Suppose that $t^{\mathtt{k}}$ is not strongly normalizing. Then there is an infinite derivation from $t^{\mathtt{k}}$, *i.e.* there is a sequence of derivations $d_k : t^{\mathtt{k}} \to^k_{\lambda_{vker}} N_k$ for any $k \in \mathbb{N}$. By Lemma 8.8 there is a sequence of derivations $e_k : t \to^k_{\lambda_{vsub}} u_k$ from t s.t. $|e_k|_{\mathtt{m}} = |d_k|_{\mathtt{m}}$ for $k \in \mathbb{N}$. Since $\to_{\mathtt{e}}$ is strongly normalizing we obtain that the sequence $\{|d_i|_{\mathtt{m}}\}_{i\in\mathbb{N}}$ cannot be bounded. But $|e_i| \geq |e_i|_{\mathtt{m}} = |d_i|_{\mathtt{m}}$, and so there is an infinite derivation from t, absurd.

   Weak Normalization. Let $t \to^*_{\lambda_{vsub}} u$ and u be a normal form. By Corollary 8.4 there is a reduction $t^{\mathtt{k}} \to^*_{\lambda_{vker}} u^{\mathtt{k}}$, and by Point 1 $u^{\mathtt{k}}$ is a normal form.

   $\Leftarrow$) Analogously:

   Strong Normalization. By Corollary 8.4 any derivation d from t in λ_{vsub} induces a derivation e from $t^{\mathtt{k}}$ in λ_{vker} s.t. $|d| \leq |e|$, so there cannot be infinite derivations from t.

   Weak Normalization. Let $t^{\mathtt{k}} \to^*_{\lambda_{vker}} N$ a reduction to normal form. By Lemma 8.8 there is s s.t. $t \to^*_{\lambda_{vsub}} s$ and $N \to^*_{\mathtt{adj}} s^{\mathtt{k}}$. But N is a normal form, and so $N = s^{\mathtt{k}}$. By Point 1 s is a normal form.

□

Weak Reduction. In CBV an important notion of evaluation is weak reduction, which contracts only the redexes which are not under abstractions. We now show that Theorem 8.9 holds also for weak reduction. Formally, a weak context is given by:

$$W ::= \langle\cdot\rangle \mid Wt \mid tW \mid W[x{\leftarrow}t] \mid [x{\leftarrow}W]$$

Weak reduction $\to_{\mathtt{w}}$ is defined as the union of $\to_{\mathtt{wm}}$ and $\to_{\mathtt{we}}$, which are obtained as the closure by *weak* contexts of $\mapsto_{\mathtt{m}}$ and $\mapsto_{\mathtt{e}}$. Weak reduction $\to_{\mathtt{w}}$ has the *diamond property* [10], *i.e.* if $s_1 \,{}_{\mathtt{w}}\!\leftarrow t \to_{\mathtt{w}} s_2$ and $s_1 \neq s_2$ then there exists u s.t. $s_1 \to_{\mathtt{w}} u \,{}_{\mathtt{w}}\!\leftarrow s_2$.

By looking at its proof it is easy to see that Theorem 8.3 holds also if any step in the statements is a weak step, because the map $\cdot^{\mathtt{k}}$ is defined on abstraction as $(\lambda x.s)^{\mathtt{k}} := \lambda x.s^{\mathtt{k}}$ and so any step under a λ in a term t is mapped to a step under a λ in $t^{\mathtt{k}}$, and viceversa. In particular, t is a weak normal form iff $t^{\mathtt{k}}$ is (so also Theorem 8.9.1 adapts to weak reduction). The lifting lemma (Lemma 8.8) is based on Theorem 8.3 and the commutation property of Lemma 8.7. The diamond property of weak reduction is stronger than the property in Lemma 8.7, thus the lifting lemma holds also with respect to weak reductions.

Then, all the reasoning in Theorem 8.9.2 is also valid with respect to weak reductions and gives the following theorem (whose statement is simpler than Theorem 8.9.2 because the diamond property implies that strong and weak normalization are equivalent for $\to_{\mathtt{w}}$):

Theorem 8.10. *Let $t \in \lambda_{vsub}$. Then t has a weak normal form iff $t^{\mathtt{k}}$ has a weak normal form.*

Stratified Weak Reduction. The main result of [10] is that a term of λ_{vsub} is solvable if and only if it has a normal form with respect to a notion of reduction therein called *stratified weak*. A natural question is if such a characterization is stable by $\cdot^{\mathtt{k}}$. Let us define stratified weak reduction. Define *head contexts* as:

$$H ::= \langle\cdot\rangle \mid Ht \mid \lambda x.H \mid H[x{\leftarrow}t]$$

Note that head contexts are not weak, as they go under abstractions. Now, define *stratified weak reduction* $\to_{\mathtt{sw}}$ as the union of $\to_{\mathtt{swm}}$ and $\to_{\mathtt{swe}}$, defined as follows

Rule at Top Level			Contextual Closure	
$L\langle\lambda x.t\rangle s$	$\mapsto_{\mathtt{m}}$	$L\langle t[x{\leftarrow}s]\rangle$	$H\langle W\langle t\rangle\rangle \to_{\mathtt{m}} H\langle W\langle s\rangle\rangle$	iff $t \mapsto_{\mathtt{m}} s$
$t[x{\leftarrow}L\langle v\rangle]$	$\mapsto_{\mathtt{e}}$	$L\langle t\{x{\leftarrow}v\}\rangle$	$H\langle W\langle t\rangle\rangle \to_{\mathtt{e}} H\langle W\langle s\rangle\rangle$	iff $t \mapsto_{\mathtt{e}} s$

The idea is that stratified weak reduction allows weak reduction to go under abstraction, but only under head abstractions.

While $\to_{\mathtt{sw}}$-steps are reflected by $\cdot^{\mathtt{k}}$, it is easily seen that they are not projected. The $\cdot^{\mathtt{k}}$ translation may in fact place head abstractions into explicit substitutions. For instance,

$$t \;:= (\lambda x.((\lambda y.t)s))ur \;\to_{\mathtt{swm}}\; (\lambda x.(t[y{\leftarrow}s]))ur \;=: t'$$

But (with z fresh)

$$t^{\mathtt{k}} \;=\; (zr^{\mathtt{k}})[z{\leftarrow}\lambda x.((\lambda y.t^{\mathtt{k}})s^{\mathtt{k}})] \;\not\to_{\mathtt{swm}}\; (zr^{\mathtt{k}})[z{\leftarrow}\lambda x.(t^{\mathtt{k}}[y{\leftarrow}s^{\mathtt{k}}])] \;=\; t'^{\mathtt{k}}$$

However, it is true that $t \in \lambda_{vsub}$ has a $\to_{\mathtt{sw}}$-normal form iff $t^{\mathtt{k}} \in \lambda_{vker}$ has a $\to_{\mathtt{sw}}$-normal form. The point is that in order to prove this result one needs to pick a particular $\to_{\mathtt{sw}}$-evaluation to normal form. Let us sketch the idea.

By using the axioms of [2], one can easily show that $\to_{\mathtt{sw}}$ factors *by levels*: any derivation $t \to^*_{\mathtt{sw}} s$ can be reorganized so that one first reduces all the $\to_{\mathtt{sw}}$-redexes out of all abstractions, then those under the first head abstraction, and so on. In particular, if there is a derivation to $\to_{\mathtt{sw}}$-normal form then there is one of this shape. Now, $\to_{\mathtt{sw}}$-derivations by levels to normal form *do project* via $\cdot^{\mathtt{k}}$, roughly because the counterexample we showed is ruled out (note that it lies under a $\to_{\mathtt{swm}}$ redex at a lower level).

Acknowledgements. To Stefano Guerrini, for introducing me to proof nets, correctness, and the representation of λ-terms; to Giulio Guerrieri, who took a very close look to this work and pointed out many inaccuracies; to the reviewers, who provided useful comments and corrections.

[1] Beniamino Accattoli (2011): *Jumping around the box: graphical and operational studies on λ-calculus and Linear Logic*. PhD thesis, *La Sapienza* University of Rome.

[2] Beniamino Accattoli (2012): *An Abstract Factorization Theorem for Explicit Substitutions*. In: *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, pp. 6–21. Available at http://dx.doi.org/10.4230/LIPIcs.RTA.2012.6.

[3] Beniamino Accattoli (2012): *Proof nets and the call-by-value lambda-calculus*. In: *Proceedings Seventh Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2012, Rio de Janeiro, Brazil, September 29-30, 2012.*, pp. 11–26. Available at http://dx.doi.org/10.4204/EPTCS.113.5.

[4] Beniamino Accattoli (2013): *Evaluating functions as processes*. In: *Proceedings 7th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2013, Rome, Italy, 23th March 2013*, *EPTCS* 110, pp. 41–55. Available at http://dx.doi.org/10.4204/EPTCS.110.

[5] Beniamino Accattoli, Pablo Barenbaum & Damiano Mazza (2014): *Distilling abstract machines*. In: *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pp. 363–376. Available at http://doi.acm.org/10.1145/2628136.2628154.

[6] Beniamino Accattoli & Claudio Sacerdoti Coen (2014): *On the Value of Variables*. In: *Logic, Language, Information, and Computation - 21st International Workshop, WoLLIC 2014, Valparaíso, Chile, September 1-4, 2014. Proceedings*, pp. 36–50. Available at http://dx.doi.org/10.1007/978-3-662-44145-9_3.

[7] Beniamino Accattoli & Stefano Guerrini (2009): *Jumping Boxes*. In: *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, pp. 55–70. Available at http://dx.doi.org/10.1007/978-3-642-04027-6_7.

[8] Beniamino Accattoli & Delia Kesner (2010): *The Structural λ-Calculus*. In: *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pp. 381–395. Available at http://dx.doi.org/10.1007/978-3-642-15205-4_30.

[9] Beniamino Accattoli & Ugo Dal Lago (2014): *Beta reduction is invariant, indeed*. In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, p. 8. Available at http://doi.acm.org/10.1145/2603088.2603105.

[10] Beniamino Accattoli & Luca Paolini (2012): *Call-by-Value Solvability, Revisited*. In: *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, pp. 4–16. Available at http://dx.doi.org/10.1007/978-3-642-29822-6_4.

[11] Alberto Carraro & Giulio Guerrieri (2014): *A Semantical and Operational Account of Call-by-Value Solvability*. In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pp. 103–118. Available at http://dx.doi.org/10.1007/978-3-642-54830-7_7.

[12] Roberto Di Cosmo & Delia Kesner (1997): *Strong Normalization of Explicit Substitutions via Cut Elimination in Proof Nets (Extended Abstract)*. In: *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pp. 35–46. Available at http://dx.doi.org/10.1109/LICS.1997.614927.

[13] V. Danos & L. Regnier (1995): *Proof-nets and the Hilbert Space*. In: *Proceedings of the Workshop on Advances in Linear Logic*, Cambridge University Press, New York, NY, USA, pp. 307–328. Available at http://dl.acm.org/citation.cfm?id=212876.212903.

[14] Vincent Danos (1990): *La Logique Linéaire appliqué à l'étude de divers processus de normalisation (principalement du λ-calcul)*. Phd thesis, Université Paris 7.

[15] Vincent Danos & Laurent Regnier (1999): *Reversible, Irreversible and Optimal lambda-Machines*. *Theor. Comput. Sci.* 227(1-2), pp. 79–97. Available at http://dx.doi.org/10.1016/S0304-3975(99)00049-3.

[16] Roberto Di Cosmo, Delia Kesner & Emmanuel Polonovski (2003): *Proof Nets And Explicit Substitutions*. *Math. Str. in Comput. Sci.* 13(3), pp. 409–450. Available at http://dx.doi.org/10.1017/S0960129502003791.

[17] Maribel Fernández & Ian Mackie (2002): *Call-by-Value lambda-Graph Rewriting Without Rewriting*. In: *Graph Transformation, First International Conference, ICGT 2002, Barcelona, Spain, October 7-12, 2002, Proceedings*, pp. 75–89. Available at http://dx.doi.org/10.1007/3-540-45832-8_8.

[18] Maribel Fernández & Nikolaos Siafakas (2009): *Labelled Lambda-calculi with Explicit Copy and Erase*. In: *Proceedings First International Workshop on Linearity, LINEARITY 2009, Coimbra, Portugal, 12th September 2009.*, pp. 49–64. Available at http://dx.doi.org/10.4204/EPTCS.22.5.

[19] Jean-Yves Girard (1987): *Linear Logic*. *Theoretical Computer Science* 50, pp. 1–102. Available at http://dx.doi.org/10.1016/0304-3975(87)90045-4.

[20] Stefano Guerrini (2004): *Proof nets and the lambda-calculus*. In: *Linear Logic in Computer Science*, Cambridge University Press, pp. 65–118.

[21] Dimitri Hendriks & Vincent van Oostrom (2003): *adbmal*. In: *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, pp. 136–150. Available at `http://dx.doi.org/10.1007/978-3-540-45085-6_11`.

[22] Hugo Herbelin & Stéphane Zimmermann (2009): *An Operational Account of Call-by-Value Minimal and Classical lambda-Calculus in "Natural Deduction" Form*. In: *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, pp. 142–156. Available at `http://dx.doi.org/10.1007/978-3-642-02273-9_12`.

[23] Delia Kesner & Stéphane Lengrand (2007): *Resource operators for lambda-calculus*. *Inf. Comput.* 205(4), pp. 419–473. Available at `http://dx.doi.org/10.1016/j.ic.2006.08.008`.

[24] Delia Kesner & Fabien Renaud (2009): *The Prismoid of Resources*. In: *MFCS*, pp. 464–476. Available at `http://dx.doi.org/10.1007/978-3-642-03816-7_40`.

[25] François Lamarche (2008): *Proof Nets for Intuitionistic Linear Logic: Essential Nets*. Research Report. Available at `http://hal.inria.fr/inria-00347336/PDF/prfnet1.pdf`.

[26] Olivier Laurent (1999): *Polarized Proof-Nets: Proof-Nets for LC*. In: *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, pp. 213–227. Available at `http://dx.doi.org/10.1007/3-540-48959-2_16`.

[27] Olivier Laurent (2002): *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II.

[28] Olivier Laurent (2003): *Polarized proof-nets and $\lambda\mu$-calculus*. *Theor. Comput. Sci.* 290(1), pp. 161–188. Available at `http://dx.doi.org/10.1016/S0304-3975(01)00297-3`.

[29] Jean-Jacques Lévy (1978): *Réductions correctes et optimales dans le lambda-calcul*. Thése d'Etat, Univ. Paris VII, France.

[30] Ian Mackie (2004): *Efficient lambda-Evaluation with Interaction Nets*. In: *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, pp. 155–169. Available at `http://dx.doi.org/10.1007/978-3-540-25979-4_11`.

[31] Ian Mackie (2005): *Encoding Strategies in the Lambda Calculus with Interaction Nets*. In: *Implementation and Application of Functional Languages, 17th International Workshop, IFL 2005, Dublin, Ireland, September 19-21, 2005, Revised Selected Papers*, pp. 19–36. Available at `http://dx.doi.org/10.1007/11964681_2`.

[32] John Maraist, Martin Odersky, David N. Turner & Philip Wadler (1999): *Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus*. *Theor. Comput. Sci.* 228(1-2), pp. 175–210. Available at `http://dx.doi.org/10.1016/S0304-3975(98)00358-2`.

[33] Luca Paolini & Simona Ronchi Della Rocca (1999): *Call-by-value Solvability*. *ITA* 33(6), pp. 507–534. Available at `http://dx.doi.org/10.1051/ita:1999130`.

[34] Gordon D. Plotkin (1975): *Call-by-Name, Call-by-Value and the lambda-Calculus*. *Theor. Comput. Sci.* 1(2), pp. 125–159. Available at `http://dx.doi.org/10.1016/0304-3975(75)90017-1`.

[35] Alberto Pravato, Simona Ronchi Della Rocca & Luca Roversi (1999): *The call-by-value λ-calculus: a semantic investigation*. *Math. Str. in Comput. Sci.* 9(5), pp. 617–650. Available at `http://dx.doi.org/10.1017/S0960129598002722`.

[36] Laurent Regnier (1992): *Lambda-calcul et réseaux*. PhD thesis, Université Paris VII.

[37] Laurent Regnier (1994): *Une équivalence sur les lambda-termes*. *Theor. Comput. Sci.* 126(2), pp. 281–292. Available at `http://dx.doi.org/10.1016/0304-3975(94)90012-4`.

[38] Paolo Tranquilli (2009): *Nets Between Determinism and Nondeterminism*. Ph.D. thesis, Università degli Studi Roma Tre/Université Paris Diderot (Paris 7).

[39] Paolo Tranquilli (2011): *Intuitionistic differential nets and lambda-calculus*. *Theor. Comput. Sci.* 412(20), pp. 1979–1997. Available at `http://dx.doi.org/10.1016/j.tcs.2010.12.022`.

[40] Lionel Vaux (2007): *λ-calcul différentiel et logique classique: interactions calculatoires*. Ph.D. thesis, Université Aix-Marseille II.