Proof nets and the call-by-value $\lambda$-calculus
Beniamino Accattoli

To cite this version:
Beniamino Accattoli. Proof nets and the call-by-value $\lambda$-calculus. Theoretical Computer Science, Elsevier, 2015, <10.1016/j.tcs.2015.08.006>. <hal-01244842>

HAL Id: hal-01244842
https://hal.archives-ouvertes.fr/hal-01244842
Submitted on 16 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Proof Nets and the Call-by-Value $\lambda$-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 vice-versa. 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 $\lambda$-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 $\lambda$-calculus. In particular, one $\beta$-reduction step in the $\lambda$-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 $\lambda$-calculus can be refined adding a constructor $t[x\leftarrow s]$ for sharing annotations called explicit substitution, and decomposing a $\beta$-step $(\lambda x.t)s \rightarrow^\beta t[x\leftarrow s]$ into the sequence $(\lambda x.t)u \rightarrow^\alpha t[x\leftarrow s] \rightarrow^\alpha t[x\leftarrow s]$.

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 $\lambda$-calculus [8], a calculus isomorphic (more precisely strongly bisimilar) to its...
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 \((A \Rightarrow B)^* = !A^n \leadsto B^n\), 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 \((A \Rightarrow B)^v = !(A^v \leadsto B^v)\), 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 \((o = !o \leadsto o)\) for call-by-name and and \(o = !(o \leadsto o)\) 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 \(\lambda_{vker}\), is a subcalculus of the value substitution calculus \(\lambda_{vsub}\) 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.

Our contributions are:

2. **Correctness Criterion.** We provide a characterization of the system of proof nets representing \(\lambda_{vker}\) based on graph-theoretical principles and which does not refer to \(\lambda_{vker}\), that is, we present a correctness criterion.
Surprisingly, the known criteria for the representation of the call-by-name \( \lambda \)-calculus (with explicit substitutions) fail to characterize the fragment encoding the call-by-value \( \lambda \)-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 \(\exists\)-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 \(\exists\)-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) \( \lambda \)-calculus \( \lambda_{\beta v} \) is obtained by restricting the \( \beta \)-rule \((\lambda x.t)s \rightarrow_{\beta} t[x:=s]\) (where \(x:=s\) 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 \( \lambda \)-calculus as e.g. the one by Herbelin and Zimmerman \([22]\), implements the restriction in a different way. Every \( \beta \)-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 \texttt{let} expressions), obtaining \(t[x:=s]\) (aka \texttt{let } \texttt{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 \(t := (\lambda x.\lambda y.y)s\) 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 \cite{29}). Preventing these redexes to take place ruins the relationships between the rewriting and the semantics, see \cite{33,10}. In particular, a term like

\[
\begin{align*}
\lambda := (\lambda z. \delta)(yy) \delta
\end{align*}
\]

(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 \( \Omega \)) and should diverge. A similar but different example of discrepancy between Plotkin’s calculus and proof nets is given by

\[
\lambda := \delta((\lambda z. \delta)(yy)),
\]

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 \( \lambda \)-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 \( \lambda \)-calculus.

The value substitution calculus \( \lambda_{vsub} \) 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 \( \lambda_{vsub} \) the terms \( \lambda \) and \( \lambda' \) diverge, as required.

Another solution is provided by Carraro and Guerrieri in \cite{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 \( \sigma \)-equivalence \cite{37} to CBV.

3. The Value Substitution Calculus and its Kernel

The language of the value substitution calculus \( \lambda_{vsub} \) is given by

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

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 \( \alpha \)-equivalence. We shall use contexts (noted \( C, D, E \)) extensively, in particular substitution contexts (noted \( L, L', L'' \)) so let us define them formally:

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

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. \( \langle D \rangle := D \)), \( \lambda x.C\langle t \rangle \) (resp. \( \lambda x.C\langle D \rangle \) := \( \lambda x.C\langle D \rangle \)), \( C\langle t \rangle s \) (resp. \( C\langle D \rangle s \) := \( C\langle D \rangle s \)), and so on. Note that plugging in a context can capture variables.

As usual, the rewriting rules of \( \lambda_{vsub} \) are obtained by first defining the rewriting rules at top level (noted \( \rightarrow_m \) and \( \rightarrow_s \), and then taking their closure by
contexts (noted \(\rightarrow_m\) and \(\rightarrow_e\)). 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:

\[
\begin{align*}
\text{Multiplicative} & \quad L(\lambda x.t)s \rightarrow_m L(t[x\leftarrow s]) \\
\text{Exponential} & \quad t[x\leftarrow L(v)] \rightarrow_e L(t[x\leftarrow v])
\end{align*}
\]

\(C(t) \rightarrow_m C(s)\) iff \(t \rightarrow_m s\)

\(C(t) \rightarrow_e C(s)\) iff \(t \rightarrow_e s\)

the fact that in the lhs of \(\rightarrow_e\) the context \(L\) appears inside \([\ ]\) while in the rhs it appears outside \(\{\}\) is not a typo. We write \(\rightarrow_{\text{vsub}}\) for the union of \(\rightarrow_m\) and \(\rightarrow_e\).

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

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

\[
\begin{align*}
\text{u} & \quad := \quad (\lambda z.\delta)(yy)\delta \rightarrow_m \\
& \quad \delta[z\leftarrow yy]\delta \rightarrow_m \\
& \quad (xx)[x\leftarrow \delta][z\leftarrow yy] \rightarrow_e \\
& \quad (\delta\delta)[z\leftarrow yy] \rightarrow_m \ldots
\end{align*}
\]

and

\[
\begin{align*}
\text{u'} & \quad := \quad \delta((\lambda z.\delta)(yy)) \rightarrow_m \\
& \quad \delta(\delta[z\leftarrow yy]) \rightarrow_m \\
& \quad (xx)[x\leftarrow \delta[z\leftarrow yy]] \rightarrow_e \\
& \quad (\delta\delta)[z\leftarrow yy] \rightarrow_m \ldots
\end{align*}
\]

Essentially, the problems with creations are solved by

1. turning generic \(\beta\)-redexes (i.e. not only CBV \(\beta\)-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 \(\lambda_{\text{vsub}}\) is designed around the CBV translation of \(\lambda\)-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 \(\lambda_{\text{vsub}}\) 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 \(\lambda_{\text{vker}}\) of \(\lambda_{\text{vsub}}\) which perfectly represents the image of the translation to proof nets. How \(\lambda_{\text{vsub}}\) is related to proof nets is studied in Sect. 8.
The relationship between the value substitution calculus $\lambda_{vsub}$ and Plotkin’s calculus $\lambda_{\beta_v}$ has been treated indirectly in [10]. Therein, $\lambda_{vsub}$ is related to another call-by-value calculus, Herbelin and Zimmermann’s $\lambda_{CBV}$ [22], whose equational theory is shown to be strictly contained in the theory of $\lambda_{vsub}$. In turn, $\lambda_{CBV}$ 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 $\lambda_{CBV}$.

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

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

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 $\lambda_{vsub}$ (with the obvious restriction on applications). The rewriting rules of $\lambda_{vker}$ are those of $\lambda_{vsub}$ restricted to the new syntax. Explicitly:

\[
\begin{align*}
\text{RULE AT TOP LEVEL} & \quad \text{CONTEXTUAL CLOSURE} \\
\text{Multiplicative} & \quad (\lambda x.t)s \mapsto_{\alpha} t[x\leftarrow s] \quad C(t) \rightarrow_{\alpha} C(s) \quad \text{iff } t \mapsto_{\alpha} s \\
\text{Exponential} & \quad t[x\leftarrow L(v)] \rightarrow_{\alpha} L(t[x\leftarrow v]) \quad C(t) \rightarrow_{\alpha} C(s) \quad \text{iff } t \mapsto_{\alpha} s
\end{align*}
\]

Note that $\mapsto_{\alpha}$ does no longer need the substitution context around the abstraction. Note also that $\lambda_{vker}$ 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 $\lambda_{vsub}$ is confluent and $\lambda_{vker}$ is a closed subcalculus of $\lambda_{vsub}$, we immediately obtain

\textbf{Theorem 3.2.} \textit{The value substitution kernel $\lambda_{vker}$ is confluent.}

In Sect. 8.2 we will show that $\lambda_{vsub}$ can be represented inside $\lambda_{vker}$. 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 $\lambda_{vsub}$ and its representation $t^k$ in $\lambda_{vker}$ are equivalent from the point of view of termination.

In the next three sections, instead, we will show that $\lambda_{vker}$ 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 $\lambda_{vker}$ has an isomorphic representation inside the $\pi$-calculus as well.

4. Proof Nets: Definition

\textit{Introduction.} Our presentation of proof nets is nonstandard in at least four points (we suggest to have a quick look to Fig. 3):
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 hyperwires, 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. Nodes are labeled with a type in \( \{e, m\} \), where \( e \) stands for

---

\( m \rightarrow !m \)

\( !m \rightarrow d \)

\( d \rightarrow !n \)

\( n \rightarrow e = \perp e \)

\( e = e \otimes e \)

\( e \rightarrow \perp e \)

\( e \rightarrow e \)

**Figure 1**: links.
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 \{!, d, w, N, \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 \( \Box \)-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 \( \Box \)) 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 \( \otimes \)-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 \( (|P|, fv(P), r_P) \), where \( |P| = (V(P), L(P)) \) is a hypergraph whose nodes are labeled with either e or m and whose hyperedges are \{!, d, w, N, \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:** \( fv(P) \) is the set of terminal nodes of \( P \), also called free variables of \( P \), which are targets of \{d, 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.

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

**Border**: the root $r_{\text{box}(l)}$ and the free variable $x$ are respectively the target and source $e$-nodes of $l$, and any free variable $\neq x$ of $\text{box}(l)$ is not the target of a weakening.

**Nesting**: for any $\exists$-box $\text{box}(h)$ if $\emptyset \neq I := |\text{box}(l)| \cap |\text{box}(h)|$, $|\text{box}(l)| \nsubseteq |\text{box}(h)|$, and $|\text{box}(h)| \nsubseteq |\text{box}(l)|$ then all the nodes in $I$ are free variables of both $\text{box}(l)$ and $\text{box}(h)$.

**Internal Closure**: 
- Contractions: $h \in \text{box}(l)$ for any link $h$ of $P$ having as target an internal $e$-node of $\text{box}(l)$.
- Boxes: $\text{box}(h) \subseteq \text{box}(l)$ for any $\exists$-link $h \in \text{box}(l)$.

**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 $e$-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 $\text{box}(l)$ 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 $\text{box}_P$ of $P$ to the $\exists$-links of $\text{box}(l)$, and see $\text{box}(l)$ 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.
Terminology about Nets. The level of a node/link/box is the maximum number of nested boxes in which it is contained\(^3\) (a \(\forall\)-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. \(d\)-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 \(e\)-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 \(o = !((o \rightarrow o))\), that we rename \(e = !(e \rightarrow e) = !(e^\perp \forall e)\) because \(e\) is a mnemonic for exponential. Let \(m := e \rightarrow e = e^\perp \forall e\), where \(m\) stands for multiplicative. Note that \(e = !m\) and \(m = !m \rightarrow !m\). This shows that the two translations of \(\lambda\)-calculus into linear logic identified by the two recursive types \(e = !(e \rightarrow e)\) and \(m = !m \rightarrow !m\) are in fact the same. Links are typed using \(m\) and \(e\), but the types are omitted by all figures except Fig. 1 because they are represented using colors and with different shapes (\(m\)-nodes are brown and dot-like, \(e\)-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 \(\{w, d, \otimes\}\)-links, while ! and \(\forall\) 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 (\(m\)-nodes cannot be free variables), and there is not so much freedom for this something: either it is a dereliction link or a \(\forall\) 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 \(e\)-nodes. For the dereliction case it is evident, and for the \(\forall\) case it is guaranteed by the border condition in the definition of nets: the content of a \(\forall\)-box ends on \(e\)-nodes. Hence, any !-link has an associated box, possibly induced by a \(\forall\)-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 \textit{maximum} and \textit{nested} are due to the fact that the free variables of \(\forall\)-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. 2a, also schematized as in Fig. 2b. The translation \( \tau \) 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 \( e \)-node, that is the way we represent contraction. In some cases the \( e \)-nodes have an incoming connection with a perpendicular little bar: it represents an arbitrary number (> 0) of incoming connections. 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 \( x[x \cdot v] \rightarrow_s v \), for which both the redex and the reduct would be mapped on the same net.

Extended Translation. The translation \( \tau \) is refined to a translation \( \tau_\Delta \), where \( \Delta \) 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 \( t_\emptyset \) when \( y \not\in \text{fv}(t) \) is the second on the first line of Fig. 3, the definition is then completed by the following two clauses: \( t_\emptyset := t \) and \( t_\Delta(y) := t_\Delta \) if \( y \in \text{fv}(t) \).

\( \alpha \)-Equivalence. To circumvent an explicit and formal treatment of \( \alpha \)-equivalence we assume that the set of \( e \)-nodes and the set of variable names for terms coincide. This convention removes the need to label the free variables of \( t_\Delta \) with the name of the corresponding free variables in \( t \) or \( \Delta \). Actually, before translating a term \( t \) it is necessary to pick a well-named \( \alpha \)-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:

\[
\begin{align*}
&t[x \cdot s][y \cdot u] \sim_{\text{vocs}} t[y \cdot u][x \cdot s] & \text{if } x \notin \text{fv}(u) \land y \notin \text{fv}(s) \\
v u[x \cdot s] & \sim_{\text{vo}1} (v u)[x \cdot s] & \text{if } x \notin \text{fv}(v) \\
t[x \cdot s][y \cdot u] & \sim_{\text{vo}2} t[x \cdot s][y \cdot u] & \text{if } y \notin \text{fv}(t)
\end{align*}
\]

Let \( \equiv_{\text{vo}} \) be the reflexive, symmetric, transitive, and contextual closure of \( \sim_{\text{vocs}} \cup \sim_{\text{vo}1} \cup \sim_{\text{vo}2} \). In the proof of Lemma 6.1 we will use the fact that if \( t \equiv_{\text{vo}} s \) then \( t \) and \( s \) are mapped on the same net. We also claim—without proving it—that \( \equiv_{\text{vo}} \) is exactly the quotient induced on terms by the translation to nets.

Paths. A path \( \tau \) of length \( k \in \mathbb{N} \) from \( u \) to \( w \), noted \( \tau : u \rightarrow^k w \), 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 \( u \rightarrow^k u \) 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 □-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 □-box at level 0 in $P$ into a □-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 □-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. $t_\Delta$ is a correct net for every term $t$ and set of variables $\Delta$.

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 $\lambda$-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 $\lambda$-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: \( \text{box}(h) \subseteq Q \) for any \( \exists \)-link \( h \in Q \).

Free variables: \( \text{box}(l) \subseteq Q \) if a free variable of \( \text{box}(l) \) 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 e-node such that it is not a free variable of \( P \) (i.e. \( x \notin \text{fv}(P) \)) and it is not the variable (i.e. not the e-source) of a \( \exists \)-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 \( \text{king}_P(x) \) of \( x \) in \( P \) is the set of links defined by case analysis on the link \( l \) of source \( x \):

- \( l \) is a !-link: \( \text{king}_P(x) \) is given by \( l \) plus the \( \text{d} \)-link or the \( \exists \)-box on the \( \text{m} \)-target of \( l \).

- \( l \) is a \( \otimes \)-link: \( \text{king}_P(x) \) is given by \( l \) plus the \( \text{d} \)-link or the \( \exists \)-box on the \( \text{m} \)-target of \( l \) plus \( \text{king}_P(y) \), where \( y \) is the e-target of \( l \).

It is easily seen that \( \text{king}_P(x) \) 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 \( \text{king}_P(x) = \text{king}_Q(x) \).

**Proof.** By induction on the length of the maximum directed path from \( x \) in \( Q^0 \). \( \square \)

Therefore, most of the time we will simplify and simply write \( \text{king}(x) \) (instead of \( \text{king}_P(x) \)).

To characterize kingdoms the following definitions are required.

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

- a substitution if it is the target of a \( \{w,d\} \)-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 \( \exists \)-box).

\(^4\)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 \( \exists \)-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 \( \text{king}(x) \) is the smallest subnet of \( P \) of root \( x \). Moreover, it has no free substitutions, no free weakenings, and whenever \( y \in \text{fv}(\text{king}(x)) \)

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

**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 \( \text{king}(x) \subseteq R \) and that \( \text{king}(x) \) 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 \( \square\)-link on the \( m\)-target of \( l \). In the case of a \( \square\)-link the box root condition for subnets implies that the whole box \( B \) is in \( R \), hence \( \text{king}(x) \subseteq R \). In the case of a \( d\)-link correctness is obvious, in the case of a \( \square\)-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 \( \text{king}(x) \) (by the border condition for nets boxes cannot close on weakenings). Pick \( y \in \text{fv}(\text{king}(x)) \), which in the \( d\)-link case is the target of \( h \) and in the other case is a free variable of the \( \square\)-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 \( \square\)-box, then there is a path from \( x \) to \( y \) in \( \text{king}(x)^0 \), the one which goes through the \(!\)-link and the \( \square\)-link replacing the \( \square\)-box in \( \text{king}(x)^0 \),
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 \( \text{king}(x) \subseteq Q \).

**Inductive case:** \( l \) is a \( \otimes\)-link. As in the previous case, \( R \) has to contain the \( d\)-link or the \( \square\)-box on the \( m\)-target of \( l \) (by the free variables condition for pre-nets). Consider the \( a\)-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., \( \text{king}_R(z) \subseteq R \) and by Lemma 5.4 (stating \( \text{king}_R(z) = \text{king}_P(z) \), simply noted \( \text{king}(z) \)) it follows \( \text{king}(z) \subseteq R \), in turn implying \( \text{king}(x) \subseteq R \). By i.h., \( \text{king}(z) \) is also correct, hence \( z \) verifies the root correctness condition for \( \text{king}(z) \), and so does \( x \) for \( \text{king}(x) \). Acyclicity follows by correctness of \( P \). Recursive correctness follows from the box closure condition for subnets and correctness of \( P \). Thus \( \text{king}(x) \)
is the smallest subnet of $P$ of root $x$. Moreover by i.h., $\text{king}(z)$—and so $\text{king}(x)$—has no free substitutions and no free weakenings. Connected interface and atomicity follow from the i.h. for the free variables of $\text{king}(z)$ 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 $\text{king}(x)$ are free variables of $P$.
2. $P \setminus \text{king}(x)$ is a subnet of $P$.

**Proof.**

1. Suppose not. Then there is a free variable $y$ of $\text{king}(x)$ 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 $\text{king}(x)^0$, 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 $\mathfrak{Y}$-box $B$. Thus, $y$ is internal to some $\mathfrak{Y}$-box $B$ and so it is not a node of $P^0$. By the atomicity part of Lemma 5.6, $\text{king}(x) \subseteq B$ and so $x$ is not a node of $P^0$, against the definition of free substitution, absurd.

2. By point 1 the removal of $\text{king}(x)$ cannot create new initial nodes. Being a substitution, $x$ is the target of some link. Therefore the removal of $\text{king}(x)$ 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 $P \setminus \text{king}(x)$. □

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. \( \square \)

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 \( \Delta \) be the set of e-nodes of its free weakenings. Then there is a term \( t \) s.t. \( t_{\Delta} = P \) (and \( fv(P) = fv(t) \cup \Delta \)).

**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 \( \overline{x} = P \). 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. \( t_{\Delta \setminus \{ x \}} = P' \). Then \( t_{\Delta} = P \).
- **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. \( s = king(x) \) and \( u_{\{ x \}} = P \setminus king(x) \) (note that if \( x \in fv(u) \) then \( u_{\{ x \}} = u_{\emptyset} = u \)). Then \( u_{\{ x \}} = P \).
- **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 \( \exists \)-link: let \( Q \) be the box of the \( \exists \)-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. \( t_{\{ x \}} = Q \) (resp. \( \overline{t} = Q \)). Then \( \lambda x.t = P \).
  - \( l \) 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 \( \exists \)-link (plus its box) on \( a \). By i.h. there exists \( s \) s.t. \( \overline{s} = king(x) \). Now, if \( a \) is the source of a d-link of e-node \( y \) we
conclude, since $ys = P$. Otherwise, $s$ is the source of a $\mathcal{N}$ of box $Q$ and the i.h. gives a term $u$ and a set of variables $\Gamma$ s.t. $u_F = Q$. Let us prove that $Q$ and $\text{king}(x)$ can only share free variables, as the translation prescribes: no link at level 0 of $\text{king}(x)$ can be in $Q$, and no box at level 0 of $\text{king}(x)$ 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 $(\lambda y. u)s = P$.

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 $\rightarrow_m$ is nothing but the usual elimination of a multiplicative cut, except that the step also opens the box associated with the $\otimes$-link.

The two $\rightarrow_e$ 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 $\mathcal{N}$-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
Lemma 6.1 (Substitution). Let \( t = s[x\leftarrow L(v)] \) then \( t_{\Delta} \rightarrow_{\Delta} L(s[x\leftarrow v])_{\Delta} \) for any set of names \( \Delta \supseteq \text{fv}(t) \).

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

1. \( n = 0 \): In \( s[x\leftarrow L(v)]_{\Delta} \) the !-link associated to \( v \) is cut with a weakening. The elimination of the cut produces a net \( P' \) without the !-link and the \( \mathcal{N} \)-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 \( s[x\leftarrow v]_{\Delta} = s_{\Delta \cup \text{tv}(v)} \).

2. \( n > 1 \): Write \( s = C(x) \) for some occurrence of \( x \). Now, consider \( u = C(y)[y\leftarrow v][x\leftarrow v] \) and note that:

\[
u \rightarrow C\langle v \rangle[x\leftarrow v] \rightarrow C\langle v \rangle\{x\leftarrow v\} = s[x\leftarrow v]\]

The difference between \( P' = u_{\Delta} \) and \( P = s[x\leftarrow v]_{\Delta} \) is that one of the occurrences of \( x \) in \( P \) has been separated from the others and cut with a copy of \( v \). Consider the step \( P \rightarrow Q \) which reduces the cut on \( x \) in \( P \) and the sequence \( P' \rightarrow Q'_{y} \rightarrow Q'_{y,x} \) 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 \( u \)

\footnote{Note that, for the sake of a simple representation, the figure of the weakening cut-elimination rule is slightly wrong: given free a variable \( x \), the links \( l_{1}, \ldots, l_{j} \) having \( x \) as target (i.e. \( l_{1}, \ldots, l_{j} \) 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 $C(v)[x\leftarrow v]_\Delta = Q'_y$ and the i.h. applied to $C(v)[x\leftarrow v]$ and $x$ provides $C(v)[x\leftarrow v]_\Delta = Q'_{y,x}$. From $Q = Q'_{y,x}$ and $C(v)[x\leftarrow v] = s\{x\leftarrow v\}$ we obtain $s\{x\leftarrow v\}_\Delta = Q$ and conclude.

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

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

$$(\lambda y.u)[x\leftarrow v]_\Delta \Rightarrow (\lambda y.u)[x\leftarrow v]_\Delta$$

**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 $s[x\leftarrow v]_\Delta = r[y\leftarrow u][x\leftarrow v]_\Delta = r[y\leftarrow u][x\leftarrow v]_\Delta$. Now by i.h. $u[x\leftarrow v] \Rightarrow u\{x\leftarrow v\}$. Then we have $s[x\leftarrow v]_\Delta \Rightarrow r[y\leftarrow u[x\leftarrow v]]_\Delta = r[y\leftarrow u[x\leftarrow v]]_\Delta = s[x\leftarrow v]_\Delta$. The second case is analogous.

**Application which is an 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 $r[x\leftarrow v] \Rightarrow r[x\leftarrow v]$ and thus $(\lambda y.r[x\leftarrow v])u_\Delta \Rightarrow (\lambda y.r[x\leftarrow v])u_\Delta$. We conclude since

$$((\lambda y.r)u)[x\leftarrow v]_\Delta \Rightarrow (\lambda y.r[x\leftarrow v])u_\Delta = ((\lambda y.r)u)[x\leftarrow v]_\Delta$$

For $L = L'[y\leftarrow r]$ the i.h. gives $L'[s[x\leftarrow v]]_\Delta \Rightarrow L'[s[x\leftarrow v]]_\Delta$. By definition of the translation and of graph reduction it follows that $L'[s[x\leftarrow v]][y\leftarrow r]_\Delta \Rightarrow L'[s[x\leftarrow v]][y\leftarrow r]_\Delta$.

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 $\Delta$ a set of variables containing $fv(t)$. The translation is a strong bisimulation between $t$ and $t_\Delta$, i.e. $t \Rightarrow t'$ if and only if $t_\Delta \Rightarrow t'_\Delta$, for $a \in \{m, e\}$.

**Proof.** By induction on the translation. Cases:

1. **Variable**, i.e. $t = x$. Trivial, as both $t$ and $t_\Delta$ 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 $t_\Delta$ are contained in $s$ and $s_\Delta$, respectively.
3. **m-redex at the root**, i.e. \( t = (\lambda x.s)u \). For the redexes in \( s, u, s_{\Delta \cup \{x\}} \) and \( u_\Delta \) just use the i.h.. The only redex of \( t \) out of its subterms is the root 
\( \text{m-redex } t = (\lambda x.s)u \rightarrow_\text{m} s[x\leftarrow u] = t' \) to which it corresponds a \( \text{m}-\text{redex in } t_\Delta \) that is not contained in \( s_{\Delta \cup \{x\}} \) nor \( u_\Delta \). Look at Fig. 5 clearly \( t \rightarrow_\text{m} t' \) iff \( t_\Delta \rightarrow_\text{m} t'_\Delta \).

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 \( u = L(v) \) then \( t \) has the following \( e\)-redex out of subterms \( s[x\leftarrow L(v)] \rightarrow_\text{e} L(s[x\leftarrow v]) \). Then apply Lemma 6.1. Note moreover that \( t_\Delta \) has a redex out of subnets only if \( u = L(v) \), and that in such a case the redex is the one we just treated. \( \square \)

![Figure 5: A \( \rightarrow_\text{m} \)-step on terms and on nets.](image)

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

**Corollary 6.3.** Let \( t \in \lambda_{\ker} \) and \( \Delta \) a set of variables. Then \( t \) is weakly normalizing/strongly normalizing/a normal form/weakly divergent/strongly divergent iff \( t_\Delta \) is.

Actually, the translation is more than a strong bisimulation: the reduction graphs of \( t \) and \( t_\Delta \) 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 \( \Delta \) a set of variables containing \( \text{fv}(t) \). The translation induces a bijection \( \phi \) between the redexes of \( t \) and the redexes of \( t_\Delta \) s.t. \( R : t \rightarrow_\alpha t' \) if and only if \( \phi(R) : t_\Delta \rightarrow_\alpha t'_\Delta \), where \( \alpha \in \{m, e\} \).

---

6Reduction 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 \( P \rightarrow P' \). 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. every confluence modulo actually coincides with plain confluence. Then from confluence of \( \lambda_{\text{oker}} \) 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 \( \lambda \)-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 \( \mathcal{N} \)-boxes

The call-by-name (CBN) and call-by-value (CBV) encodings of \( \lambda \)-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 \( \mathcal{N} \) 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 \( \rightarrow \) are collapsed in LL but not in Intuitionistic LL, in particular our \( \mathcal{N} \) 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 \( \rightarrow \) (our \( \mathcal{N} \)), as we shall explain in
the next paragraphs. In particular, the two encodings of \( \lambda \)-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 \( \lambda \)-calculus is a special case where the box for \( \to \) needs not be represented.
- The fragment of IMELL encoding the CBV \( \lambda \)-calculus is a special case where the box for \( ! \) needs not be represented.

Therefore, our use of %-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 \( \lambda \)-calculus, and more generally Polarized Linear Logic, are notable exceptions. For the encoding of the CBN \( \lambda \)-calculus there exist two correctness criteria. Let us show that none of them works for the CBV \( \lambda \)-calculus:

![Figure 6: the ordinary CBV translation from terms to nets.](image)

![Figure 7: Counter-examples to correctness without %-boxes.](image)
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 \(\lambda\)-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 \(\otimes\)-links form a cycle), but without using \(\otimes\)-boxes it would accept the net in Fig. 7.b, which is not correct.

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 [11], 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) [11] for which then there is no correctness criterion.

In the case studied in this paper the use of \(\otimes\)-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 \(\rightarrow\), plus contractions and weakenings) and cuts: in our case there is no \(\otimes\) (remember our \(\otimes\) and \(\rightarrow\) rather correspond to the rules for \(\rightarrow\)), our technical choices for variables keep the parallelism for contraction and weakenings, and the parallelism of the left rule for \(\rightarrow\) (our \(\otimes\)) and cuts is preserved (it is given by the equations in [11], 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 \(\perp\)-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. 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 \( \mu \)-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 \( A \rightarrow B \) maps to \( ?!(A \rightarrow B) \), and not to \( !(A \rightarrow B) \).

Lionel Vaux \[40\] and Paolo Tranquilli \[38, 39\] study the relationship between the differential \( \lambda \)-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 \textit{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 \( \lambda \)-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 \( \lambda \)-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 exponential rules.

\[\text{\footnotesize Let us point out that }13\text{\footnotesize presents an oddity that we believe deserves to be clarified. The authors show that an optimized geometry of interaction for the proof nets of the CBV-translation is isomorphic to Krivine Abstract Machine (KAM): this is quite puzzling, because the KAM is CBN, while they use the CBV translation.}\]
8. Terms: Relating $\lambda_{vsub}$ and $\lambda_{vker}$

In this section we study the relationship between $\lambda_{vsub}$ and $\lambda_{vker}$. The results of this paper actually allow to read this section as if we were relating $\lambda_{vsub}$ and proof nets, given that $\lambda_{vker}$ and proof nets are isomorphic. In particular, by composing the (forthcoming) translation from $\lambda_{vsub}$ to $\lambda_{vker}$ and the translation from $\lambda_{vker}$ to proof nets (Sect. 4) one obtains a translation from $\lambda_{vsub}$ 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 $\lambda_{vsub}$ at a level of detail and rigor that would be impossible if we had to deal directly with the graphical syntax.

8.1. Projecting $\lambda_{vsub}$ on $\lambda_{vker}$

This subsection shows that $\lambda_{vsub}$ has an interpretation inside $\lambda_{vker}$.

**Notation:** when we want to stress that a term is in $\lambda_{vker}$ we use capital letters.

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

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

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

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

\[
\begin{align*}
x^k & := x \\
(\lambda x.t)^k & := \lambda x.t^k \\
t[x \leftarrow s]^k & := t[x \leftarrow s^k] \\
(L(v)s)^k & := L^k(v^k s^k) \\
(L(ts)u)^k & := (x u^k)[x \leftarrow L^k(ts)] & \text{where } x \text{ is fresh} \\
(\cdot)^k & := (\cdot) \\
L[x \leftarrow t]^k & := L^k[x \leftarrow t^k]
\end{align*}
\]
There is a slight mismatch between reductions in $\lambda_{vsub}$ and in $\lambda_{vker}$. Sometimes a $\rightarrow_n$ step in $\lambda_{vsub}$ is simulated by a single $\rightarrow_n$ step in $\lambda_{vker}$, while sometimes it requires an additional $\rightarrow_\ast$ step. Consider $t := (\lambda x. \lambda y. t)su$, that translates to $t^k = (zu^k)[z \leftarrow (\lambda x. \lambda y. t^k)s^k]$ with $z$ fresh. We have:

$$t \rightarrow_n (\lambda y. t)[x \leftarrow s]u =: t'$$

But

$$t^k \rightarrow_n (zu^k)[z \leftarrow (\lambda y. t^k)[x \leftarrow s^k]] \rightarrow_\ast ((\lambda y. t^k)u^k)[x \leftarrow s^k] = t'^k$$

It is quite obvious that $\lambda_{vker}$ can simulate $\lambda_{vsub}$ with only a linear overhead (in terms of number of rewriting steps), because every step in $\lambda_{vsub}$ can be simulated by at most two steps in $\lambda_{vker}$. We shall however prove this formally in the rest of this section. It is less obvious that any reduction sequence in $\lambda_{vker}$ can be extended with a linear number of steps so that it represents a reduction sequence in $\lambda_{vsub}$. 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 $\lambda_{vsub}$ and $\lambda_{vker}$, as we will show in Sect. 8.3.

**Lemma 8.2** (Properties of $^k$). The translation $^k$ has the following properties:

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

**Proof.**

1. It is immediate that $v^k$ is a value. If $t^k$ is a value then either $t^k = x$ and thus $t = x$, or $t^k = \lambda x. N$, but by definition of the translation this is only possible if $t = \lambda x. s$ and $s^k = N$.

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 $\langle v \rangle \{x \leftarrow v\} := \langle v \rangle$ and $L[y \leftarrow t] := L\{x \leftarrow v\}[y \leftarrow t\{x \leftarrow v\}]$. Moreover, to diminish the use of parentheses, we use $t\{x \leftarrow s\}^k$ for $(t\{x \leftarrow s\})^k$ (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:

   \[
   \begin{align*}
   t\{x \leftarrow v\}^k &= (L\langle v'\rangle u)\{x \leftarrow v\}^k \\
   &= (L'\langle v'\{x \leftarrow v\}u\{x \leftarrow v\}\rangle)^k \\
   &= L''\langle v'\{x \leftarrow v\}u\{x \leftarrow v\}\rangle^k \\
   &= i.h. \quad L''\langle v'^k\{x \leftarrow v^k\}u^k\{x \leftarrow v^k\}\rangle^k \\
   &= L^k\langle v'^k u^k\rangle^k \{x \leftarrow v^k\} \\
   &= t^k \{x \leftarrow v^k\}
   \end{align*}
   \]
Theorem 8.3 (Projection and Reflections).

1. Projection of a \( \lambda \) step: \( t \leadsto \lambda s \) implies \( t^k \rightarrow_s s^k \) or \( t^k \rightarrow_s \varphi^k s^k \).
2. Reflection of a \( \lambda \) step: if \( t \leadsto \lambda N \) then there exists \( s \) s.t. \( t \rightarrow \lambda s \) and either \( s^k = N \) or \( N \rightarrow \lambda s^k \).
3. Projection and reflection of a \( \rightarrow \lambda \) step: \( t \rightarrow \lambda s \) iff \( t^k \rightarrow \lambda s^k \).

Proof. By induction on \( (\cdot)^k \). 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 \( t^k = u^k[x \leftarrow r^k] \). If the reduction takes place in \( u, r, u^k \), or \( r^k \) then it follows from the \( i.h. \). Note that \( \rightarrow \lambda \) steps can only take place in subterms of \( t \), so we have proved Point 1 and Point 2. Point 3:

\[ \Rightarrow \) If \( r = L(v) \) then \( t = u[x \leftarrow L(v)] \rightarrow \lambda L(u[x \leftarrow v]) = s \) and \( t^k = u^k[x \leftarrow L^k(v^k)] \). By Lemma 8.2 1 \( u^k \) is a value and \( t^k \rightarrow \lambda L^k(u^k[x \leftarrow v^k]) = N \). Now, Lemma 8.2 2 gives us \( s^k = N \).

\( \Leftarrow \) If \( v^k = L(v) \) then \( r \) has the form \( L'(v') \) with \( v' = v \) and \( L^k = \lambda \), 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: \( t = L(v)r \) and \( t^k = L^k(u^k r^k) \). If the reduction takes place in \( v, r, v^k, v^k, \) or \( L^k \) then it follows from the \( i.h. \). Otherwise:

1. **Projection of a \( \rightarrow \lambda \) step.** If \( v = \lambda x.u \) then \( t = L(\lambda x.u)r \rightarrow \lambda L(u[x \leftarrow r]) = s \) and \( t^k = L^k(\lambda x.u^k r^k) \rightarrow \lambda L^k(u^k[x \leftarrow v^k]) = s^k \).
2. **Reflection of a \( \rightarrow \lambda \) step.** If \( v^k = \lambda x.u \) then \( v = \lambda x.u' \) with \( u^k = u \), and we repeat the same reasoning.
3. **Projection and reflection of a $\rightarrow_e$ step.** The reasoning is analogous to the case $t = u[x\leftarrow r]$.  

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

1. **Projection of a $\rightarrow_n$ step.** If $u = L'(\lambda x.q)$ then $t \rightarrow_n L'(q[x\leftarrow p]) r = s$ and

   \[
   t^k = (y^k)[y \leftarrow L'(\lambda x.q)p^k] = \begin{cases} 
   (y^k)[y \leftarrow L^k(\lambda x.q)p^k] & \rightarrow_e N \\
   (y^k)[y \leftarrow L^k(q[x\leftarrow p]^k)] & \end{cases}
   \]

   Now, there are two cases, depending on the shape of $q$:

   (a) $q$ is a value in a substitution context, i.e. $q = L''(v')$. Then $s = L'(q[x\leftarrow p]) r = L(L''(v')[x\leftarrow p]) r$ and

   \[
   N = (y^k)[y \leftarrow L^k(\lambda x.q)p^k](L''(v'[x\leftarrow p]) r) = (y^k)[y \leftarrow L^k(\lambda x.q)p^k](L''(v'[x\leftarrow p]) r) 
   \]

   \[
   \rightarrow_e L^k(L''(v'[x\leftarrow p]) r) = s^k
   \]

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

   then

   \[
   s = L'(L''(qm)[x\leftarrow p]) r
   \]

   and

   \[
   N = (y^k)[y \leftarrow L^k(\lambda x.q)p^k](L''(qm)[x\leftarrow p]) r = s^k
   \]

2. **Reflection of a $\rightarrow_n$ step.** If $t^k = (y^k)[y \leftarrow L^k(u^k p^k)]$ has a $\eta$-redex not contained in one of the subterms then necessarily $u^k = L'(\lambda x.Q)$ and so $u = L''(\lambda x.q)$ with $q^k = Q$ and $L''^k = L'$. Then the proof repeats the reasoning of the previous point.

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

\[\square\]

An immediate consequence is:

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

30
8.2. Reflecting \( \lambda_{vker} \) reductions in \( \lambda_{vsub} \)

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

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

**Rule at Top Level**

\[(xu)[x \leftarrow L(v)] \rightarrow_{adj} L(vu) \text{ with } x \notin \text{fv}(u)\]

**Contextual Closure**

\[C(t) \rightarrow_{adj} C(s) \quad \text{iff} \quad t \rightarrow_{adj} s\]

The relationship between \( \rightarrow_\alpha \) steps in the two calculi (given by Point 1 and Point 2 of Theorem 8.3) then becomes:

1. If \( t \rightarrow_\alpha s \) then either \( t^k \rightarrow_\alpha s^k \) or \( t^k \rightarrow_\alpha \rightarrow_{adj} s^k \).
2. If \( t^k \rightarrow_\alpha N \) then there exists \( s \) s.t. \( t \rightarrow_\alpha s \) and either \( s^k = N \) or \( N \rightarrow_{adj} s^k \).

Let us explain the technical difficulty in reflecting reductions from \( \lambda_{vker} \) to \( \lambda_{vsub} \). Suppose that:

1. \( t^k \rightarrow_{\lambda_{vker}} s_1 \rightarrow_{\lambda_{vker}}^+ s \) is a reduction sequence whose first step \( t^k \rightarrow_{\lambda_{vker}} s_1 \) requires an adjusting step \( s_1 \rightarrow_{adj} u \) in order to be the simulation of a step \( t \rightarrow_\alpha r \) on \( \lambda_{vsub} \) (with \( r^k = u \)), and that
2. this adjusting step \( s_1 \rightarrow_{adj} u \) is not the first step of the suffix \( s_1 \rightarrow_{\lambda_{vker}}^+ s \).

In terms of diagrams we have:

\[
\begin{array}{c}
\text{[diagram)}
\end{array}
\]

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

\[
\begin{array}{c}
\text{[diagram)}
\end{array}
\]

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

However, to later relate termination properties of \( \lambda_{vsub} \) and \( \lambda_{vker} \) we need a bit more than that. In fact, commutation of \( \rightarrow_{adj} \) and \( \rightarrow_{\lambda_{vker}} \) is in itself trivial, because \( \rightarrow_e \) commutes with \( \rightarrow_{\lambda_{vker}} \) and a \( \rightarrow_{adj} \) step is a \( \rightarrow_e \) 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 $\rightarrow_{\mathbf{adj}}$ does not change the number of $m$-steps (in contrast to commutation with $\rightarrow_e$).

**Lemma 8.6** (Commutations of $\rightarrow_{\mathbf{adj}}$). Let $t \in \lambda_{\mathbb{ker}}$. Then:

1. **Span of $\rightarrow_e$ and $\rightarrow_{\mathbf{adj}}$:**

   $$
   t \xrightarrow{k} s_1 \quad \text{implies that either } s_1 = s_2 \text{ or there exists } u \text{ s.t. } t \xrightarrow{\mathbf{adj}^*} u
   $$

2. **Span of $\rightarrow_e$ and $\rightarrow_{\mathbf{adj}}^*$:**

   $$
   t \xrightarrow{k} s_1 \quad \text{implies that either } s_1 = s_2 \text{ or there exists } u \text{ s.t. } t \xrightarrow{\mathbf{adj}^*} u
   $$

3. **Span of $\rightarrow_m$ and $\rightarrow_{\mathbf{adj}}$:**

   $$
   t \xrightarrow{k} s_1 \quad \text{implies that there exists } u \text{ s.t. } t \xrightarrow{\mathbf{adj}^*} u
   $$

4. **Span of $\rightarrow_m$ and $\rightarrow_{\mathbf{adj}}^*$:**

   $$
   t \xrightarrow{k} s_1 \quad \text{implies that there exists } u \text{ s.t. } t \xrightarrow{\mathbf{adj}^*} u
   $$

**Proof.**

1. By induction on $t \rightarrow_e s_1$. The case $s_1 = s_2$ is given by the case in which the $\rightarrow_e$ step is $t \rightarrow_{\mathbf{adj}} s_2$.
2. By induction on the length of $t \rightarrow_{\mathbf{adj}}^* s_2$, using Point 1.
3. By induction on $t \rightarrow_m s_1$.
4. By induction on the length of $t \rightarrow_{\mathbf{adj}}^* s_2$, using Point 3.

We can now show the required commutation property.

**Lemma 8.7** (Commutation of $\rightarrow_{\lambda_{\mathbb{ker}}}$ and $\rightarrow_{\mathbf{adj}}$). Let $t \in \lambda_{\mathbb{ker}}$. Then

$$
\begin{align*}
   t \xrightarrow{k} s_1 \quad &\text{implies that there exists } u \text{ s.t. } t \xrightarrow{\mathbf{adj}^*} u
   \end{align*}
$$

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

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

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

Proof. By induction on $k$. If $k = 0$ there is nothing to prove. Then let $k > 0$, i.e. $t^k \rightarrow_{\lambda_{vker}} u_1 \rightarrow_{\lambda_{vker}}^k N$. By Theorem 8.3 there is a term $s_1$ s.t. $t \rightarrow_{\lambda_{vsub}} s_1$ and either $u_1 = s_1^k$ or $u_1 \rightarrow_{adj} s_1^k$. Cases:

No need of an adjusting step, i.e. $u_1 = s_1^k$. Then by i.h. applied to $s_1^k = u_1 \rightarrow_{\lambda_{vker}}^k N$ there exists $s$ s.t. $s_1 \rightarrow_{\lambda_{vker}}^{\leq k-1} s$, $N \rightarrow_{adj}^* s^k$, and the two derivations $s_1 \rightarrow_{\lambda_{vker}}^{\leq k-1} s$ and $s_1^k = u_1 \rightarrow_{\lambda_{vker}}^{k-1} N$ have the same number of $m$-steps. Then $t \rightarrow_{\lambda_{vsub}} s_1 \rightarrow_{\lambda_{vker}}^{\leq k-1} s$ has length $\leq k$ and we conclude, because the step $t^k \rightarrow_{\lambda_{vker}} u_1$ is a $m$-step iff $t \rightarrow_{\lambda_{vsub}} s_1$ is.

An adjusting step is required, i.e. $u_1 \rightarrow_{adj} s_1^k$. Then both $t^k \rightarrow_{\lambda_{vker}} u_1$ and $t \rightarrow_{\lambda_{vker}} s_1$ are $m$-steps. By Lemma 8.7 applied to $u_1 \rightarrow_{adj} s_1^k$ and the suffix $u_1 \rightarrow_{\lambda_{vker}}^k N$ there exists $Q$ s.t. $s_1^k \rightarrow_{\lambda_{vker}}^k Q$ and $N \rightarrow_{adj}^* Q$, and moreover both derivations $s_1^k \rightarrow_{\lambda_{vker}}^{\leq k-1} Q$ and $u_1 \rightarrow_{\lambda_{vker}}^{k-1} N$ have the same number of $m$-steps. Now, let’s apply the i.h. to $s_1^k \rightarrow_{\lambda_{vker}}^k Q$: we obtain $s$ s.t. $s_1 \rightarrow_{\lambda_{vker}}^{\leq k-1} s$ and $Q \rightarrow_{adj}^* s^k$, and moreover both derivations $s_1^k \rightarrow_{\lambda_{vker}}^{\leq k-1} Q$ and $s_1 \rightarrow_{\lambda_{vker}}^{\leq k-1} s$ have the same number of $m$-steps, which is the same of $u_1 \rightarrow_{\lambda_{vker}}^{k-1} N$. We have proved the statement, because

1. $N \rightarrow_{adj}^* Q$ and $Q \rightarrow_{adj}^* s^k$, i.e. $N \rightarrow_{adj}^* s^k$;
2. $t \rightarrow_{\lambda_{vsub}} s \rightarrow_{\lambda_{vker}}^{\leq k-1} s$ has length $\leq k$, and
3. this derivation has the same number of $m$-steps of $t^k \rightarrow_{\lambda_{vker}} u_1 \rightarrow_{\lambda_{vker}}^{k-1} N$.

\[ \square \]

### 8.3. Relating Normalization Properties

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

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

1. $t$ is a normal form iff $t^k$ is a normal form.
2. $t$ is strongly (resp. weakly) normalizing iff $t^k$ is strongly (resp. weakly) normalizing.

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

33
Proof.

1. By Theorem 8.3 if \( t \) has a redex then \( t^k \) has a redex, and viceversa.

2. \( \Rightarrow \) we split strong and weak normalization:

   **Strong Normalization.** Suppose that \( t^k \) is not strongly normalizing. Then there is an infinite derivation from \( t^k \), i.e. there is a sequence of derivations \( d_k : t^k \rightarrow_{\lambda \text{ker}} N_k \) for any \( k \in \mathbb{N} \). By Lemma 8.8 there is a sequence of derivations \( e_k : t \rightarrow_{\lambda \text{sub}} u_k \) from \( t \) s.t. \( |e_k| = |d_k| \) for \( k \in \mathbb{N} \). Since \( \rightarrow_{\text{e}} \) is strongly normalizing we obtain that the sequence \( \{|d_k|\}_{k \in \mathbb{N}} \) cannot be bounded. But \( |e_k| \geq |d_k| \), and so there is an infinite derivation from \( t \), absurd.

   **Weak Normalization.** Let \( t \rightarrow_{\lambda \text{sub}}^* u \) and \( u \) be a normal form. By Corollary 8.4 there is a derivation \( t^k \rightarrow_{\lambda \text{ker}} u^k \), and by Point 1 \( u^k \) is a normal form.

\( \Leftarrow \) Analogously:

   **Strong Normalization.** By Corollary 8.4 any derivation \( d \) from \( t \) in \( \lambda \text{sub} \) induces a derivation \( e \) from \( t^k \) in \( \lambda \text{ker} \) s.t. \( |d| \leq |e| \), so there cannot be infinite derivations from \( t \).

   **Weak Normalization.** Let \( t \rightarrow_{\lambda \text{sub}}^* N \) a reduction to normal form. By Lemma 8.8 there is \( s \) s.t. \( t \rightarrow_{\lambda \text{sub}}^* s \) and \( N \rightarrow_{\lambda \text{adj}}^* s^k \). But \( N \) is a normal form, and so \( N = s^k \). By Point 1 \( s \) is a normal form.

\[ \square \]

**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:

\[ W ::= \langle \cdot \rangle | Wt | tW | W[x \leftarrow t] | [x \leftarrow W] \]

Weak reduction \( \rightarrow_{w} \) is defined as the union of \( \rightarrow_{m} \) and \( \rightarrow_{e} \), which are obtained as the closure by weak contexts of \( \rightarrow_{n} \) and \( \rightarrow_{a} \). Weak reduction \( \rightarrow_{w} \) has the diamond property [10], i.e. if \( s_1 \leftarrow_{w} t \rightarrow_{w} s_2 \) and \( s_1 \neq s_2 \) then there exists \( u \) s.t. \( s_1 \rightarrow_{w} u \leftarrow_{w} s_2 \).

By looking at its proof it is easy to see that Theorem 8.3 holds also for weak reduction. Formally, the weak context of a weak step under a \( \lambda \) in a term \( t \) is mapped to a step under a \( \lambda \) in \( t^k \), and viceversa. In particular, \( t \) is a weak normal form iff \( t^k \) 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 \( \rightarrow_{w} \)):
Theorem 8.10. Let \( t \in \lambda_vsub \). Then \( t \) has a weak normal form iff \( t^k \) has a weak normal form.

Stratified Weak Reduction. The main result of [10] is that a term of \( \lambda_vsub \) 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 \( \cdot^k \). Let us define stratified weak reduction. Define head contexts as:

\[
H ::= \langle \cdot \rangle | Ht | \lambda x.H | H[t \leftarrow x] \]

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

\[
\text{Rule at Top Level} \quad \text{Contextual Closure} \\
L(\lambda x.t)s \rightarrow_\mathfrak{m} L(t[x \leftarrow s]) \\
t[x \leftarrow L(v)] \rightarrow_\mathfrak{e} L(t[x \leftarrow v]) \\
H(W(t)) \rightarrow_\mathfrak{m} H(W(s)) \quad \text{iff} \quad t \rightarrow_\mathfrak{m} s \\
H(W(t)) \rightarrow_\mathfrak{e} H(W(s)) \quad \text{iff} \quad t \rightarrow_\mathfrak{e} s
\]

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

While \( \rightarrow_{sv} \)-steps are reflected by \( .^k \), it is easily seen that they are not projected. The \( .^k \) translation may in fact place head abstractions into explicit substitutions. For instance,

\[
t := (\lambda x.((\lambda y.t)s))ur \rightarrow_{sv} (\lambda x.(t[y \leftarrow s]))ur =: t'
\]

But (with \( z \) fresh)

\[
t^k = (zr^k)[z \leftarrow \lambda x.((\lambda y.t^k)s^k)] \not\rightarrow_{sv} (zr^k)[z \leftarrow \lambda x.(t^k[y \leftarrow s^k])] = t'^k
\]

However, it is true that \( t \in \lambda_vsub \) has a \( \rightarrow_{sv} \)-normal form iff \( t^k \in \lambda_vker \) has a \( \rightarrow_{sv} \)-normal form. The point is that in order to prove this result one needs to pick a particular \( \rightarrow_{sv} \)-evaluation to normal form. Let us sketch the idea.

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

Acknowledgements. To Stefano Guerrini, for introducing me to proof nets, correctness, and the representation of \( \lambda \)-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.


