Proofs as terms, terms as graphs

***Abstract*—Starting from an encoding of untyped λ-terms with sharing, defined using synthetic inference rules based on a focused proof system for Gentzen's *LJ*, we introduce a graphical representation in order to capture equivalences on terms that can be described using rule permutations. On one hand, this graphical representation provides a way to remove redundancy in the syntax, and on the other hand, it allows implementing basic operations such as substitution and reduction in a more straightforward way. In the end, we study a multi-focused proof system for *LJ*, illustrate how multi-focusing is related to permutations of synthetic inference rules, and give the corresponding notion of synthetic inference rules and term annotations.**

I. Introduction

Terms (or expressions) are essential in different settings: mathematical proofs, programming languages, proof assistants, etc. To prevent redundancy in these systems, a canonical and compact syntactic structure is needed. Proof theory has been broadly used in the studies of term representation. There have been several different approaches to address the question of the canonicity of proofs, such as proof nets [1], expansion trees [2], focusing [3], combinatorial proofs [4], etc.

In this paper, we choose to use focusing as our main tool. Andreoli introduced the first focused proof system to describe proof structure in linear logic in a more structured way. Inference rules are organized into two different phases: *positive* and *negative* phases. Andreoli [5] and Chaudhuri [6] suggested that phases should be viewed as large-scale rules for proof construction. In [7], Marin et al. defined another version of large-scale rules, called *synthetic inference rule*, which is essentially composed of a negative phase and a positive phase, in order to provide more high-level descriptions of proof systems. The invention of synthetic inference rules also provides a systematic way to extend proof systems such as *LJ* and *LK*.

Various focused proof systems have been proposed for *LJ* [8], [9], [10], and *LK* [11], [10]. Several authors also designed term calculi for *LJT* [8], *LJQ* [9], and *LJF* [12]. In [13], Miller and Wu use synthetic inference rules built using the focused proof system *LJF* to study term structures. In *LJF*, different polarity assignments yield different forms of proofs and thus provide different styles of term representation. In this paper, we are interested in their encoding of untyped λ-terms using the positive bias assignment. Unlike the usual syntax of untyped λ-terms, this encoding constructs terms in a *bottom-up* style and allows explicit sharing within a term using the construct **name**. This term representation is, however, not compact, in the sense that a lot of named structures can be permuted or put in parallel. These permutations correspond, in fact, to phase permutations or phase merging in focused proof systems, and to rule permutation in terms of synthetic inference rules.

Several authors have proposed *multi-focused* proof systems to illustrate this phenomenon for *MALL* [14], [15], [16], *LK* [17] and *LJ* [18]. Moreover, in each of [16] and [17], an isomorphism has been established between the multi-focused proof system and a graphical representation of proofs (proof nets and expansion trees, respectively).

This is where our *graphical representation* for terms comes in. Though inspired by these works on multi-focusing, we choose to first put our focus on terms, i.e. (single-)focused proofs. We first study the positive bias syntax of untyped λ-terms proposed in [13] and the equivalences on terms based on different permutations of synthetic inference rules. We then present a graphical representation for untyped λ-terms, called λ-graphs with bodies, that is isomorphic to the encoding mentioned above up to permutations of named structures based on phase permutation in *LJF*. This graphical structure allows, on one hand, to represent named structures in a compact way using internal nodes, and, on the other, to deal with operations such as substitution and reduction in a rather simple way. The duplication of function bodies that appears during reduction pushes us to consider another possible operation on terms: lifting definitions (named structures) to a more global scope in order to allow more sharing. This operation can again be very easily implemented on graphs with a rewrite rule, and it is not difficult to check if a graph is normal with respect to the rewrite rule. The main results of this paper are: (1) a one-to-one correspondence between λ-graphs with bodies and terms up to permutations of independent namings (Theorem 30) and (2) a one-to-one correspondence between normal λ-graphs with bodies and terms up to permutations of independent namings and name lifting (Theorem 43).

Historically, there have been several attempts to connect λ-calculus with sharing, or explicit substitution [19], to proof nets. This connection was first proposed by Di Cosmo and Kesner [20] to study cut-elimination. Recently, proof nets have also been connected to calculi such as the call-by-value λ-calculus [21] and the linear substitution calculus [22]. In contrast to these works that often introduce boxes to deal with sharing, we decide to use the notion of *bodies*, which is closely related to the notion of *level* in these works. This choice does not make a huge difference in our theoretical results, but it provides a clear way to establish the correspondence between the usual syntax and the graphical one.

II. Preliminaries: the focused proof system LJF and synthetic inference rules

Fix a set ATOM of ***atomic formulas***. ***Formulas*** are built with implications and atomic formulas. An ***atomic bias assignment*** is a map δ from ATOM to $\{+, -\}$. A ***polarized formula*** (resp. ***polarized theory***) is a formula (resp. multiset of formulas) together with an atomic bias assignment. Implications are negative and atomic formulas can be either positive or negative following the atomic bias assignment. In *LJF*, there are two kinds of sequents: $\Uparrow$-sequents $\Gamma \Uparrow \Theta \vdash \Delta \Uparrow \Delta'$ and $\Downarrow$-sequents $\Gamma \Downarrow \Theta \vdash \Delta \Downarrow \Delta'$. Here, $\Gamma, \Theta, \Delta, \Delta'$ are multisets of formulas, and we denote by $\cdot$ the empty multiset. Γ and Δ' are called ***storage zones*** and Θ and Δ are called ***staging zones***. In a $\Downarrow$-sequent, exactly one of Θ and Δ can be non-empty and contains exactly one formula called the *focused formula*: if Θ (resp. Δ) is non-empty, the sequent is called ***left-focused*** (resp. ***right-focused***). To simplify the notation, we often drop an arrow whenever its corresponding staging zone is empty. Furthermore, $\Delta \cup \Delta'$ is a singleton, as we are in an intuitionistic setting. A ***border sequent*** is a sequent that has empty staging zones, i.e., it has no arrow after simplification. The rules of the implicational fragment of *LJF* are presented in Figure 1. *LJF* proofs have a two-phase structure: $\Uparrow$-phases and $\Downarrow$-phases.

Definition 1 (Synthetic inference rules, [7])**.** A ***left synthetic inference rule*** is a rule of the form

$$\frac{\Gamma_1 \vdash A_1 \quad \cdots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\, N \qquad \text{\textit{justified} by an \textit{LJF} derivation of the form} \qquad \frac{\begin{array}{c}\Gamma_1 \vdash A_1 \quad \cdots \quad \Gamma_n \vdash A_n \\ \Pi \\ \Gamma \Downarrow N \vdash A\end{array}}{\Gamma \vdash A}\, D_l$$

Here, N is a negative formula that appears in Γ, $n \geq 0$, and within Π, a $\Downarrow$-sequent never occurs above an $\Uparrow$-sequent. The structure of *LJF* proofs implies $N \in \Gamma_i$ for all $1 \leq i \leq n$. The negative formula N is used to name this left synthetic inference rule, and this rule is called the ***left synthetic inference rule for*** N. We can similarly define the notion of *right synthetic inference rule*. However, the only positive formulas in our setting are atomic, and the only rule that can be applied to the premise of the D_r rule is I_r. As a result, right synthetic inference rules in our setting perform in exactly the same way as the initial rule of *LJ*. In the following, we only consider left synthetic inference rules and write just ***synthetic inference rules***.

Synthetic inference rules show how a formula can be *used* in a proof and they can be used to extend sequent systems such as *LJ*. Since the synthetic inference rules for atomic formulas are exactly the same as the initial rule of *LJ*, we will only consider synthetic inference rules for non-atomic formulas. In [7], it is shown that the synthetic inference rules for certain formulas are particularly simple and can be easily used to extend *LJ* with a (polarized) theory. To express this, we first define the order of a formula.

Definition 2. The ***order*** of a formula B, written $ord(B)$, is defined as follows: $ord(A) = 0$ if A is atomic and $ord(B_1 \supset B_2) = max(ord(B_1) + 1, ord(B_2))$.

Definition 3 (Extensions of *LJ* by polarized theories, [7])**.** Let $\mathcal{T}$ be a finite multiset of formulas of order one or two, and let δ be an atomic bias assignment. We define the ***extension*** $LJ\lfloor\delta, \mathcal{T}\rfloor$ ***of LJ by the polarized theory*** $(\mathcal{T}, \delta)$ to be the two-sided proof system built as follows. The only sequents in the $LJ\lfloor\delta, \mathcal{T}\rfloor$ proof system are of the form $\Gamma \vdash A$ where A is atomic and Γ is a multiset of atomic formulas. The inference rules of $LJ\lfloor\delta, \mathcal{T}\rfloor$ include the initial rule of *LJ* and for every synthetic inference rule

$$\frac{N, \Gamma_1 \vdash A_1 \quad \ldots \quad N, \Gamma_n \vdash A_n}{N, \Gamma \vdash A}\, N$$

where $N \in \mathcal{T}$ is a negative formula, then the rule

$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\, N$$

is included in $LJ\lfloor\delta, \mathcal{T}\rfloor$.

Note that in the original version proposed in [7], $\mathcal{T}$ is a set instead of a multiset. As we will see, each formula in $\mathcal{T}$ corresponds to a combinator in annotated proofs. Different occurrences of the same formula define different combinators. This is why we consider multisets instead of sets of formulas.

The following theorem justifies Definition 3.

Theorem 4. *Let $\mathcal{T}$ be a finite multiset of formulas of order one or two, and let δ be an atomic bias assignment. Then for any atomic formula A and Γ containing atomic formulas only, $\Gamma, \mathcal{T} \vdash A$ is provable in LJ if and only if $\Gamma \vdash A$ is provable in $LJ\lfloor\delta, \mathcal{T}\rfloor$.*

Proof. By consistency and completeness of *LJF* with respect to *LJ*. □

III. Annotations, terms and permutations

The notion of terms, which is usually a primitive notion in most of the literature, is a derived notion here: they are annotations of proofs. By annotating inference rules in the proof system $LJ\lfloor\delta, \mathcal{T}\rfloor$, we obtain rules for various combinators, each one of which corresponds to a formula in $\mathcal{T}$.

In [13], Miller and Wu give an encoding of untyped λ-terms by considering the theory $\Gamma_0 = \{D \supset D \supset D, (D \supset D) \supset D\}$ where D is an atomic formula. If D is given the negative polarity, we get the ***negative bias syntax***, i.e., the usual tree structure for untyped λ-terms. If D is given the positive polarity, we get the ***positive bias syntax***, a structure where explicit sharing is possible. The annotated inference rules obtained using both polarity assignments are shown in Figure 2.

In the following, we will focus on terms built using the positive bias syntax. Since the sequents and inference rules considered only involve the atomic formula D, we choose to replace the annotated formula $x : D$ (resp. $t : D$) with simply its annotation x (resp. t).

A. An encoding of untyped λ-terms

Fix a set NAME $= \{x, y, z, \ldots\}$ of ***names*** (or ***variables***). A ***signature*** is a finite subset Σ of NAME. We write Σ, x for $\Sigma \cup \{x\}$ and this also implies that $x \notin \Sigma$.

We now give the definition of well-formed terms under a given signature Σ.

Definition 5. Let Σ be a signature. We define the set of Σ-***terms*** using the following rules:

Decide, Release, and Store Rules

$$\frac{N,\Gamma \Downarrow N \vdash A}{N,\Gamma \vdash A}\, D_l \qquad \frac{\Gamma \vdash P \Downarrow}{\Gamma \vdash P}\, D_r \qquad \frac{\Gamma \Uparrow P \vdash A}{\Gamma \Downarrow P \vdash A}\, R_l \qquad \frac{\Gamma \vdash N \Uparrow}{\Gamma \vdash N \Downarrow}\, R_r \qquad \frac{\Gamma, C \Uparrow \Theta \vdash \Delta \Uparrow \Delta'}{\Gamma \Uparrow \Theta, C \vdash \Delta \Uparrow \Delta'}\, S_l \qquad \frac{\Gamma \Uparrow \Theta \vdash A}{\Gamma \Uparrow \Theta \vdash A \Uparrow}\, S_r$$

Initial Rules

$$\delta(A) = + \quad \frac{}{A, \Gamma \vdash A \Downarrow}\, I_r \qquad \delta(A) = - \quad \frac{}{\Gamma \Downarrow A \vdash A}\, I_l$$

Introduction Rules for implication

$$\frac{\Gamma \vdash B \Downarrow \qquad \Gamma \Downarrow B' \vdash A}{\Gamma \Downarrow B \supset B' \vdash A} \supset L \qquad \frac{\Gamma \Uparrow \Theta, B \vdash B' \Uparrow}{\Gamma \Uparrow \Theta \vdash B \supset B' \Uparrow} \supset R$$

Fig. 1. The implicational fragment of the focused proof system *LJF*. Here, P is positive, N is negative, A is atomic, and B, B' and C are arbitrary formulas.

$$x : D \in \Gamma \quad \frac{}{\Gamma \vdash \mathsf{nvar}\ x : D}\, nvar \qquad\qquad x : D \in \Gamma \quad \frac{}{\Gamma \vdash \mathsf{pvar}\ x : D}\, pvar$$

$$\frac{\Gamma \vdash s : D \qquad \Gamma \vdash t : D}{\Gamma \vdash \mathsf{napp}\ s\ t : D}\, napp \qquad\qquad \{y : D, z : D\} \subseteq \Gamma \quad \frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash \mathsf{papp}\ y\ z\ (\lambda x.t) : D}\, papp$$

$$\frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash \mathsf{nabs}\ (\lambda x.t) : D}\, nabs \qquad\qquad \frac{\Gamma, y : D \vdash s : D \qquad \Gamma, x : D \vdash t : D}{\Gamma \vdash \mathsf{pabs}\ (\lambda y.s)\ (\lambda x.t) : D}\, pabs$$

Fig. 2. Annotated inference rules of the systems $LJ\lfloor\delta^-, \Gamma_0\rfloor$ (left) and $LJ\lfloor\delta^+, \Gamma_0\rfloor$ (right) where $\delta^-(D) = -$, $\delta^+(D) = +$ and $\Gamma_0 = \{D \supset D \supset D, (D \supset D) \supset D\}$.

$$x \in \Sigma \quad \frac{}{\Sigma \vdash_{wf} x}\, var \qquad \{y, z\} \subseteq \Sigma \quad \frac{\Sigma, x \vdash_{wf} t}{\Sigma \vdash_{wf} \textbf{name}\ x = yz\ \textbf{in}\ t}\, app$$

$$\frac{\Sigma, y \vdash_{wf} s \qquad \Sigma, x \vdash_{wf} t}{\Sigma \vdash_{wf} \textbf{name}\ x = \lambda y.s\ \textbf{in}\ t}\, abs$$

t is a ***Σ-term*** if $\Sigma \vdash_{wf} t$. We say that t is a ***term*** if t is a Σ-term for some Σ.

Remark 6. These rules are, in fact, the same as those in $LJ\lfloor\delta^+, \Gamma_0\rfloor$ but we use the **name** construct to represent certain bindings shown in Figure 2. One should not confuse **name** with **let** which is often a syntactic sugar for β-redexes.

A term is a list of *named structures* followed by a name, which we call the ***output*** of the term. The set TERM of terms, denoted by $s, t, u, \ldots$, can actually be generated by the following grammar:

$$s, t \coloneqq x \mid \textbf{name}\ x = yz\ \textbf{in}\ t \mid \textbf{name}\ x = \lambda y.s\ \textbf{in}\ t$$

Note that names introduced by **name** are bound, so we should consider terms up to α-equivalence and assume that all the bound names are distinct.

The set $fv(t)$ of ***free variables*** of a term t is given by: $fv(x) = \{x\}$, $fv(\textbf{name}\ x = yz\ \textbf{in}\ t) = (fv(t) \setminus \{x\}) \cup \{y, z\}$ and $fv(\textbf{name}\ x = \lambda y.s\ \textbf{in}\ t) = (fv(s) \setminus \{y\}) \cup (fv(t) \setminus \{x\})$. We also define ***contexts*** by the following grammar:

$$C \coloneqq \square \mid \textbf{name}\ x = yz\ \textbf{in}\ C \mid \textbf{name}\ x = \lambda y.s\ \textbf{in}\ C$$
$$\mid \textbf{name}\ x = \lambda y.C\ \textbf{in}\ t$$

The ***plugging*** $C\langle t\rangle$ of t in the context C is obtained from C by replacing the placeholder $\square$ with t. Note that we allow plugging in a context to capture variables. A ***congruence*** on terms is an equivalence relation that is closed by context.

In the following, we often write $t[x \leftarrow p]$ for **name** $x = p$ **in** t for simplicity but one should not confuse this notation with the meta-level substitution of λ-calculus.

B. Rule permutation and equivalence on terms

Named structures can be seen as intermediate definitions within a term. If two definitions are independent of each other, we should be able to permute them. This kind of permutation can actually be described using permutations of synthetic inference rules. Consider the following derivation:

$$\{x_1, x_2\} \subseteq \Sigma \quad \frac{\{y_1, y_2\} \subseteq \Sigma, x \quad \dfrac{\Sigma, x, y \vdash_{wf} t}{\Sigma, x \vdash_{wf} t[y \leftarrow y_1 y_2]}\, app}{\Sigma \vdash_{wf} t[y \leftarrow y_1 y_2][x \leftarrow x_1 x_2]}\, app$$

If y_1 and y_2 are both distinct from x, then it is possible to permute these two rules:

$$\{y_1, y_2\} \subseteq \Sigma \quad \frac{\{x_1, x_2\} \subseteq \Sigma, y \quad \dfrac{\Sigma, y, x \vdash_{wf} t}{\Sigma, y \vdash_{wf} t[x \leftarrow x_1 x_2]}\, app}{\Sigma \vdash_{wf} t[x \leftarrow x_1 x_2][y \leftarrow y_1 y_2]}\, app$$

Likewise, we can consider the following derivation:

$$\{x_1, x_2\} \subseteq \Sigma \quad \frac{\dfrac{\Sigma, x, z \vdash_{wf} s \qquad \Sigma, x, y \vdash_{wf} t}{\Sigma, x \vdash_{wf} t[y \leftarrow \lambda z.s]}\, abs}{\Sigma \vdash_{wf} t[y \leftarrow \lambda z.s][x \leftarrow x_1 x_2]}\, app$$

If $x \notin fv(s)$, then it is possible to permute these two rules:

$$\frac{\Sigma, z \vdash_{wf} s \qquad \{x_1, x_2\} \subseteq \Sigma \quad \dfrac{\Sigma, y, x \vdash_{wf} t}{\Sigma, y \vdash_{wf} t[x \leftarrow x_1 x_2]}\, app}{\Sigma \vdash_{wf} t[x \leftarrow x_1 x_2][y \leftarrow \lambda z.s]}\, abs$$

where the left premise of the *abs* rule is obtained by strengthening. The same observation also applies to the case *abs*/*abs*.

These rule permutations induce the following equations on Σ-terms:

$$t[y \leftarrow y_1 y_2][x \leftarrow x_1 x_2] \sim_r t[x \leftarrow x_1 x_2][y \leftarrow y_1 y_2] \quad \text{if } x \notin \{y_1, y_2\} \text{ and } y \notin \{x_1, x_2\}$$

$$t[y \leftarrow \lambda z.s][x \leftarrow x_1 x_2] \sim_r t[x \leftarrow x_1 x_2][y \leftarrow \lambda z.s] \quad \text{if } x \notin fv(s) \text{ and } y \notin \{x_1, x_2\}$$

$$t[x_2 \leftarrow \lambda y_2.s_2][x_1 \leftarrow \lambda y_1.s_1] \sim_r t[x_1 \leftarrow \lambda y_1.s_1][x_2 \leftarrow \lambda y_2.s_2] \quad \text{if } x_1 \notin fv(s_2) \text{ and } x_2 \notin fv(s_1)$$

Based on these equations, we can define an equivalence relation on terms.

Definition 7 (Right equivalence)**.** We define an equivalence relation $\equiv_r$ on terms, called the ***right equivalence***, as the smallest congruence containing $\sim_r$.

We can actually express the right equivalence by using the notion of body and dependency that we define in the following.

Definition 8. Let t be a Σ-term. Let x be a name introducing an abstraction in t, i.e., we have the pattern $[x \leftarrow \lambda y.s]$ in t. We define the ***body*** of x, written $body(x)$, as the set $\{x_1, \cdots, x_k\}$ of names where s is of the form $x_{k+1}[x_k \leftarrow p_k] \cdots [x_1 \leftarrow p_1]$. We say that y is the ***bound variable*** of x, denoted by $bv(x)$. It is clear that any name introduced by **name** belongs to at most one body. We denote by $body(t)$ the set of names introduced by **name** that do not belong to any body of any name introducing an abstraction.

Definition 9. Let t be a Σ-term. Let x be a name introduced by some $[x \leftarrow p]$ in t. We define the ***dependency set*** of x, written $dep(x)$, as follows:

- If p is an abstraction $\lambda y.s$, then $dep(x) = fv(s)$. Also, note that x and y are distinct.
- If p is an application yz, then $dep(x) = \{y, z\}$.

We then define the ***dependency graphs*** of t and of its names introducing abstractions:

- The dependency graph $\mathcal{D}_t$ of t is defined as the graph
$$(body(t), \{(x, y) \mid x, y \in body(t) \text{ and } x \in dep(y)\}).$$
- The dependency graph $\mathcal{D}_x$ of a name x introducing an abstraction is defined as the graph
$$(body(x), \{(y, z) \mid y, z \in body(x) \text{ and } y \in dep(z)\}).$$

Note that if $x \in dep(y)$, then x cannot be defined later than y in the same body. This observation leads to the following proposition.

Proposition 10. *Let t be a Σ-term. Then we have:*

- *$\mathcal{D}_t$ is a DAG.*
- *$\mathcal{D}_x$ is a DAG for any x introducing an abstraction.*

The naming permutations presented earlier are applied to two consecutive namings in the same body and we can permute them if and only if neither of them is in the dependency set of the other. Thus, we have the following proposition.

Proposition 11. *Two consecutive namings can be permuted using $\sim_r$ if and only if they are not adjacent in their corresponding dependency graph.*

Theorem 12. *Let s and t be two Σ-terms. Then $s \equiv_r t$ if and only if t can be obtained from s by*

- *a sequence of naming permutations on non-adjacent nodes in $\mathcal{D}_t$, and*
- *a sequence of naming permutations on non-adjacent nodes in $\mathcal{D}_x$ for all x introducing an abstraction.*

Proof. A direct consequence of Proposition 11. □

IV. A graphical representation for terms: λ-graphs with bodies

In this section, we introduce a graphical representation for terms, called λ-graphs with bodies, that will be proved to capture the equivalence on Σ-terms given in Section III. The definition of λ-graphs with bodies is split into two parts: we first define pre-graphs, and then define λ-graphs with bodies by giving additional structures and properties to deal with abstractions.

Definition 13. A ***pre-graph*** is a directed acyclic graph built with the following three kinds of nodes:

- ***Application***: an application node is labeled with @ and has two incoming edges (left and right). An application node is also called an @-node.
- ***Abstraction***: an abstraction node is labeled with λ and has one incoming edge. Its only direct predecessor is called the ***output*** of the abstraction node. An abstraction node is also called a λ-node.
- ***Variable***: a variable node has no incoming edge.

A direct predecessor of a node is also called a ***child*** of the node.

Internal nodes (application and abstraction) of a pre-graph are used to represent intermediate expressions defined using **name** within a term. We orient edges in such a way that there is an edge from n to m if and only if the definition of m requires the definition of n. In other words, nodes are defined in a *bottom-up* fashion.

Notation 14. In the following, we denote by $\mathcal{N}_\mathcal{G}$ and $\mathcal{E}_\mathcal{G} \subseteq \mathcal{N}_\mathcal{G} \times \mathcal{N}_\mathcal{G}$, respectively, the set of nodes and the sets of edges of a graph $\mathcal{G}$.

Definition 15. A ***pre-graph with bodies*** is a pre-graph $\mathcal{G}$ together with two functions $bv : \Lambda_\mathcal{G} \to \mathcal{V}_\mathcal{G}$ and $body : \Lambda_\mathcal{G} \to 2^{\mathcal{N}_\mathcal{G} \setminus \mathcal{V}_\mathcal{G}}$ where $\Lambda_\mathcal{G}$ is the set of abstraction nodes of $\mathcal{G}$ and $\mathcal{V}_\mathcal{G}$ is the set of variable nodes of $\mathcal{G}$:

1) $body(l) \cap body(l') = \emptyset$ for $l \neq l'$.
2) The graph $\mathcal{B}_\mathcal{G} = (\Lambda_\mathcal{G}, \{(l, l') \mid l, l' \in \Lambda_\mathcal{G}, l \in body(l')\})$, called the ***scope graph*** of $\mathcal{G}$, is a DAG.
3) If a node n is $bv(l)$ or is in $body(l)$ and there is an edge $(n, m) \in \mathcal{E}_\mathcal{G}$, then we have
   - $m = l$, or

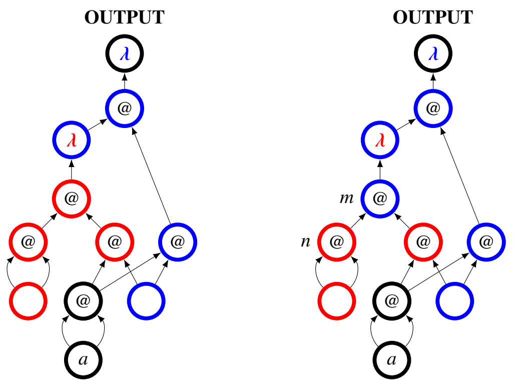


Fig. 3. A valid λ-graph with bodies and an invalid one.

- $m \in body(l')$ such that there is a path from l' to l in $\mathcal{B}_{\mathcal{G}}$. Note that this path is unique.

We call $bv(l)$ the ***bound variable node*** and $body(l)$ the ***body*** of the abstraction node l. A node that does not belong to any body is called ***body-free*** and we denote by $body(\mathcal{G})$ the set of body-free non-variable nodes in $\mathcal{G}$. A ***free variable node*** is a variable node that is not a bound variable node and a ***global node*** is a body-free node that is not a bound variable node.

Intuitively, Point 3 of Definition 15 checks that every definition in a term is used in a valid scope: a name introduced in an abstraction can only be used within the abstraction.

Definition 16. A ***λ-graph with bodies*** is a pre-graph with bodies with a unique label assigned to each free variable node, and with a global node chosen, called the ***output*** of the λ-graph with bodies. A ***Σ-λ-graph with bodies*** is a λ-graph with bodies with a free variable node labeled by each element of a signature Σ.

In order to visualize the maps $bv(\cdot)$ and $body(\cdot)$, we color the labels of abstraction nodes to distinguish them and color the frame of the nodes in their bodies with the same color. We proceed similarly for bound variables. In particular, a global node has its frame colored in black.

Figure 3(a) shows a λ-graph with bodies, while Figure 3(b) shows an example that breaks Point 3 of Definition 15. In Figure 3(b), n belongs to the red body, m belongs to the blue body and there is an edge (n, m). m is not the red λ-node and there is no path from the blue λ-node to the red λ-node in the scope graph.

V. Ordered Σ-λ-graphs with bodies and Σ-terms

In this section, we prove that there is a one-to-one correspondence between Σ-λ-graphs with bodies and Σ-terms up to $\equiv_r$. In order to establish such a correspondence, we first establish a one-to-one correspondence between ordered Σ-λ-graphs with bodies and Σ-terms where ordered Σ-λ-graphs with bodies are refinements of Σ-λ-graphs with bodies with some additional structure.

A. Dependency

Terms are expressed in a linear style. In other words, all the intermediate expressions within a term are defined in some (linear) order. However, graphs do not usually have this kind of structure. In order to establish the correspondence between terms and graphs, we have to give some more structure to our graphical representation. To do that, we first define dependency relations on internal nodes, counterparts of names in graphs.

Definition 17. Let $\mathcal{G}$ be a λ-graph with bodies and l an abstraction node. We define a relation $\prec_l$, called the ***dependency relation*** of l, on the set $body(l) \cup \{bv(l)\}$ of nodes as follows:

- $n \prec_l m$ if $(n, m) \in \mathcal{E}_{\mathcal{G}}$.
- $n \prec_l m$ if $(n, m') \in \mathcal{E}_{\mathcal{G}}$ for some $m' \in body(l')$ with $l' \neq l$, and there is a path from l' to m in $\mathcal{B}_{\mathcal{G}}$.

Likewise, we can define the dependency relation on the set of body-free non-variable nodes.

Definition 18. Let $\mathcal{G}$ be a λ-graph with bodies. We define the ***dependency relation*** $\prec_{\mathcal{G}}$ of $\mathcal{G}$ on the set $body(\mathcal{G})$ of body-free non-variable nodes of $\mathcal{G}$ as follows:

- $n \prec_{\mathcal{G}} m$ if $(n, m) \in \mathcal{E}_{\mathcal{G}}$.
- $n \prec_{\mathcal{G}} m$ if $(n, m') \in \mathcal{E}_{\mathcal{G}}$ for some $m' \in body(l)$, and there is a path from l to m in $\mathcal{B}_{\mathcal{G}}$.

Definition 19. Let $\mathcal{G}$ be a λ-graph with bodies. We define

- the ***dependency graph*** of $\mathcal{G}$, as the graph $\mathcal{D}_{\mathcal{G}} = (body(\mathcal{G}), \{(n, m) \mid n \prec_{\mathcal{G}} m\})$, and
- for all abstraction node l, the ***dependency graph*** of l, as the graph $\mathcal{D}_l = (body(l), \{(n, m) \mid n \prec_l m\})$.

Proposition 20. *Let $\mathcal{G}$ be a λ-graph with bodies. Then we have:*

- *$\mathcal{D}_{\mathcal{G}}$ is a DAG, and*
- *for all abstraction node l of $\mathcal{G}$, $\mathcal{D}_l$ is a DAG.*

Proof. This follows from the acyclicity of $\mathcal{G}$ and $\mathcal{B}_{\mathcal{G}}$. □

For example, for the λ-graph with bodies $\mathcal{G}$ in Figure 3(a), the dependency graph of the red (resp. blue) λ-node is the subgraph of $\mathcal{G}$ induced by its body, while the dependency graph $\mathcal{D}_{\mathcal{G}}$ of $\mathcal{G}$ has an edge from the application node to the blue λ-node.

As mentioned previously, our graphical representation should be equipped with some linear orderings on internal nodes. Moreover, these orderings should be compatible with the dependency relations defined above: they should be topological sorts of their corresponding dependency graphs.

Definition 21. A ***topological sort*** of a directed graph $\mathcal{G}$ is a sequence containing each of its vertices such that for every edge (n, m), n appears before m in the sequence.

Definition 22. An ***ordered λ-graph with bodies*** $\hat{\mathcal{G}}$ is a λ-graph with bodies $\mathcal{G}$ together with a topological sort $T_{\mathcal{G}}$ of the graph $\mathcal{D}_{\mathcal{G}}$ and a topological sort T_l of the graph $\mathcal{D}_l$, for each l.

Before giving a one-to-one correspondence between ordered λ-graphs with bodies and terms, we give a notion of *boxes* that is useful in the following.

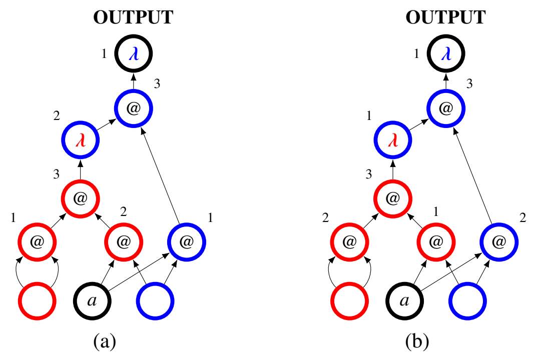


Fig. 4. Two ordered $\{a\}$-λ-graphs with bodies

Definition 23. Let $\mathcal{G}$ be a λ-graph with bodies and l an abstraction node. We define the ***box*** of l as the union of bodies together with their bound variable nodes *below* l:

$$box(l) = \bigcup_{l' \rightsquigarrow l \text{ in } \mathcal{B}_{\mathcal{G}}} (body(l') \cup \{bv(l')\})$$

where $l' \rightsquigarrow l$ in $\mathcal{B}_{\mathcal{G}}$ means that there is a path from l' to l in $\mathcal{B}_{\mathcal{G}}$.

In 3(a), the box of the red λ-node contains all the red-framed nodes while the box of the blue λ-node contains all the blue-framed and red-framed nodes.

Intuitively, for a λ-node l of a λ-graph with bodies $\mathcal{G}$, the graph obtained from the subgraph of $\mathcal{G}$ induced by $box(l)$ corresponds to the abstraction it introduces.

Ordered λ-graphs with bodies can actually be defined inductively as terms. We first give the following useful definitions.

Definition 24. Let Σ be a signature and $x \in \Sigma$. We define $(x)_\Sigma$ as the ordered Σ-λ-graph with bodies that contains a free variable node labeled by each element of Σ and has the one labeled by x as the output.

Definition 25. Let Σ and Σ' be signatures, x, y, x_1, x_2 be names such that $\{x_1, x_2\} \subseteq \Sigma$, $x \notin \Sigma$ and $y \notin \Sigma'$, $\hat{\mathcal{G}}$ an ordered (Σ, x)-λ-graph with bodies and $\hat{\mathcal{G}}'$ an ordered (Σ', y)-λ-graph with bodies. Then

- $\hat{\mathcal{G}}\{\textbf{node } x \leftarrow x_1@x_2\}$ is defined as the graph $\hat{\mathcal{H}}$ obtained from $\hat{\mathcal{G}}$ by replacing the free variable node labeled by x with an @-node whose left (resp. right) child is the variable node labeled by x_1 (resp. x_2). We then extend the topological sort $T_{\mathcal{G}}$ by having this application node as the minimal node. It is clear that $\hat{\mathcal{H}}$ is also an ordered Σ-λ-graph with bodies.
- $\hat{\mathcal{G}}\{\textbf{node } x \leftarrow \lambda y.\hat{\mathcal{G}}'\}$ is defined as the graph $\hat{\mathcal{H}}$ obtained from by merging $\mathcal{G}$ and $\mathcal{G}'$ and by replacing the free variable node labeled by x with a new abstraction node constructed as follows:
  - its only child is the output of $\mathcal{G}'$,
  - its bound variable is the free variable node labeled by y in $\mathcal{G}'$ (we erase the label y), and
  - its body contains all the body-free non-variable nodes of $\mathcal{G}'$.

  Note that $\mathcal{G}$ and $\mathcal{G}'$ can share some free variable nodes: they are merged so that there is only one free variable node labeled by each element of $\Sigma \cap \Sigma'$. In the end, we extend the topological sort $T_{\mathcal{G}}$ by having this new abstraction node as the minimal node (all the other topological sorts are inherited). It is not difficult to see that $\hat{\mathcal{H}}$ is an ordered $(\Sigma \cup \Sigma')$-λ-graph with bodies.

Note that we can also use these definitions for λ-graphs with bodies by forgetting topological sorts.

Proposition 26. *Let Σ be a signature. Then $\hat{\mathcal{G}}$ is an ordered Σ-λ-graph with bodies if and only if $\Sigma \vdash \hat{\mathcal{G}}$ where $\Sigma \vdash \hat{\mathcal{G}}$ is defined by the following rules.*

$$x \in \Sigma \ \frac{}{\Sigma \vdash (x)_\Sigma}\ var \qquad \{y, z\} \subseteq \Sigma \ \frac{\Sigma, x \vdash \hat{\mathcal{G}}}{\Sigma \vdash \hat{\mathcal{G}}\{\textbf{node } x \leftarrow y@z\}}\ @$$

$$\frac{\Sigma, y \vdash \hat{\mathcal{G}}' \qquad \Sigma, x \vdash \hat{\mathcal{G}}}{\Sigma \vdash \hat{\mathcal{G}}\{\textbf{node } x \leftarrow \lambda y.\hat{\mathcal{G}}'\}}\ \lambda$$

Proof. ($\Rightarrow$) Immediate from Definition 24 and Definition 25.

($\Leftarrow$) Let $\hat{\mathcal{G}} = (\mathcal{G}, T_{\mathcal{G}}, \{T_l \mid l \in \Lambda_{\mathcal{G}}\})$ be an ordered Σ-λ-graph with bodies.

We prove by induction on the number of non-variable nodes in $\mathcal{G}$.

- $\mathcal{G}$ contains only variable nodes and its output is labeled by x: $\hat{\mathcal{G}} = (x)_\Sigma$.
- $\mathcal{G}$ contains a non-variable node: in this case, $\mathcal{G}$ contains at least one body-free non-variable node. We consider the minimal one n with respect to $T_{\mathcal{G}}$.
  - application: both of its children are free variable nodes (otherwise, it would not have been the minimal body-free non-variable node), labeled by y and z, respectively. Let $\hat{\mathcal{G}}'$ be the graph obtained from $\hat{\mathcal{G}}$ by replacing this application node with a fresh free variable node x (with topological sorts inherited). This graph $\hat{\mathcal{G}}'$ is an ordered (Σ, x)-λ-graph with bodies and we have $\hat{\mathcal{G}} = \hat{\mathcal{G}}'\{\textbf{node } x \leftarrow y@z\}$.
  - abstraction: for every node m in $box(n)$, all the children of m in $\mathcal{G}$ are either free variable nodes or in $box(n)$. Otherwise, there would have been a body-free non-variable node m that is a direct predecessor of n in $\mathcal{D}_{\mathcal{G}}$, which leads to a contradiction. Consider the subgraph $\mathcal{H}$ of $\mathcal{G}$ induced by $box(n) \cup \Sigma$. By giving the bound variable node of n in $\mathcal{G}$ a label y, we get an ordered (Σ, y)-λ-graph with bodies $\hat{\mathcal{H}}'$ (with topological sorts inherited). Now by removing all the nodes in $box(n)$ from $\mathcal{G}$ and by replacing n with a fresh free variable node x, we get an ordered (Σ, x)-λ-graph with bodies $\hat{\mathcal{G}}'$. Then we have $\hat{\mathcal{G}} = \hat{\mathcal{G}}'\{\textbf{node } x \leftarrow \lambda y.\hat{\mathcal{H}}'\}$.

□

Note that the rules in Definition 5 have the same structure as those in Theorem 26.

Theorem 27. *Let Σ be a signature. Then there is a one-on-one correspondence between ordered Σ-λ-graphs with bodies and Σ-terms.*

Proof. we give translations $[\![\cdot]\!]_\Sigma$ from Σ-terms to ordered Σ-λ-graphs with bodies and $[\cdot]_\Sigma$ from ordered Σ-λ-graphs with bodies to Σ-terms by induction on the rules in Definition 5 and those in Theorem 26. For the base cases, let $[\![x]\!]_\Sigma = (x)_\Sigma$ and $[(x)_\Sigma]_\Sigma = x$. We then have $[[\![t]\!]_\Sigma]_\Sigma = t$ and $[\![[\hat{\mathcal{G}}]_\Sigma]\!]_\Sigma = \hat{\mathcal{G}}$ for all Σ-term t and ordered Σ-λ-graph with bodies $\hat{\mathcal{G}}$. □

Proposition 28. *For any Σ-term t, t and $[\![t]\!]_\Sigma$ have the same dependency graphs.*

Proof. A straightforward induction on t. □

We have established an isomorphism between Σ-terms and ordered Σ-λ-graphs with bodies. In Section III, terms are considered equivalent up to $\equiv_r$. How about ordered λ-graphs with bodies? It is natural to consider that ordered λ-graphs with bodies are equivalent if they share the same underlying λ-graph with bodies. The following proposition shows that topological sorts of a DAG can be connected to each other via *swaps*, similar to permutations for terms.

Proposition 29. *Let $\mathcal{G}$ be a DAG and S a topological sort of $\mathcal{G}$. We call swapping two non-adjacent nodes of $\mathcal{G}$ in a sequence of nodes a valid swap. Then a sequence of nodes can be obtained from S by a sequence of valid swaps if, and only if, it is a topological sort of $\mathcal{G}$.*

Proof. A proof is given in Appendix A. □

Theorem 30. *We have a one-to-one correspondence between Σ-λ-graphs with bodies and Σ-terms up to $\equiv_r$.*

Proof. Immediate from Theorem 12, Theorem 27, Proposition 28 and Proposition 29. □

VI. Substitution and Reduction

In this section, we show how substitution can be performed on λ-graphs with bodies and propose a reduction procedure for terms that can be easily implemented on this graphical representation.

A. Substitution

In [13], there is a big-step (atomic) cut-elimination procedure for proofs built using synthetic inference rules. This procedure provides a definition of substitution for terms:

Definition 31 (Substitution on terms, [13]). Let t, u be terms and x a name such that $x \notin fv(u)$. We define the result of substituting x for u in t, written $t[x/u]$, as follows:

$$t[x/\textbf{name}\ y = zw\ \textbf{in}\ s] = \textbf{name}\ y = zw\ \textbf{in}\ t[x/s]$$
$$t[x/\textbf{name}\ y = \lambda z.u\ \textbf{in}\ s] = \textbf{name}\ y = \lambda z.u\ \textbf{in}\ t[x/s]$$

Here, $t[x/y]$ is obtained from t by renaming x to y.

This definition can be expressed in a straightforward way on λ-graphs with bodies.

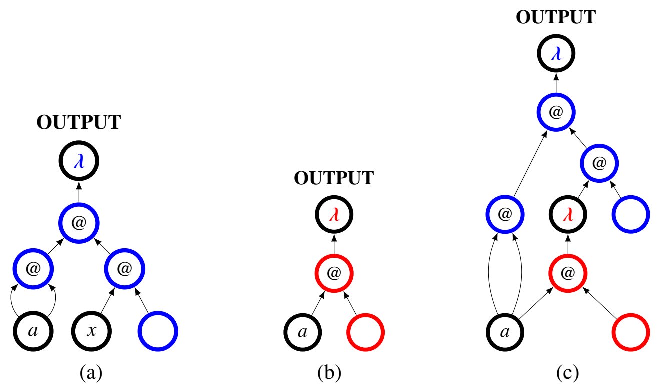


Fig. 5. An example for the substitution on λ-graphs with bodies: (c) is the result of substituting the free variable node x for (b) in (a).

Definition 32 (Substitution on λ-graphs with bodies). Let $\mathcal{G}$ be a Σ-λ-graph with bodies and $\mathcal{G}'$ a Σ'-λ-graph with bodies with $x \notin \Sigma'$. We define the substitution of x for $\mathcal{G}'$ in $\mathcal{G}$, written $\mathcal{G}[x/\mathcal{G}']$, as the $((\Sigma \setminus \{x\}) \cup \Sigma')$-$\lambda$-graph with bodies obtained from by merging $\mathcal{G}'$ into $\mathcal{G}$ and, if $x \in \Sigma$ by replacing the free variable node labeled by x with the output node of $\mathcal{G}'$. Note that we have to merge common free variable nodes labeled by elements of $(\Sigma \setminus \{x\}) \cap \Sigma'$ and the output node of $\mathcal{G}[x/\mathcal{G}']$ is that of $\mathcal{G}$.

In Figure 5, we present an example for the substitution on λ-graphs with bodies. From this example, we can clearly see that the structure of bodies is modular, i.e., kept under substitution.

The substitution on λ-graphs with bodies implements indeed the substitution on terms.

Theorem 33. *Let t be a Σ-term and u a Σ'-term such that $x \notin \Sigma'$. Then $[\![t[x/u]]\!]_{(\Sigma\setminus\{x\})\cup\Sigma'} = [\![t]\!]_\Sigma[x/[\![u]\!]_{\Sigma'}]$.*

Proof. A straightforward induction on terms. □

B. Unfolding and reduction

Term reduction is often related to cut-elimination in the literature. However, terms considered here correspond to cut-free proofs. A natural question to ask is: How should we evaluate them? Of course, we could *unfold* all the named structures of a term and get its corresponding untyped λ-term.

Definition 34. The ***unfolding*** $\underline{t}$ of a term t is the untyped λ-term defined as follows:

$$\begin{aligned} \underline{x} &= x \\ \underline{\textbf{name}\ x = yz\ \textbf{in}\ t} &= \underline{t}\{x/yz\} \\ \underline{\textbf{name}\ x = \lambda y.s\ \textbf{in}\ t} &= \underline{t}\{x/\lambda y.\underline{s}\} \end{aligned}$$

where $\{\cdot/\cdot\}$ is the meta-level substitution of untyped λ-terms.

For example, we have $\underline{\textbf{name}\ f = \lambda x.x\ \textbf{in}\ \textbf{name}\ y = fz\ \textbf{in}\ y} = (\lambda x.x)z$. This definition provides a way to translate from the positive bias syntax to the negative bias syntax. Note that the unfolding of a term is not necessarily a β-normal untyped λ-term. For a term t, we could refer to the β-normal form of its unfolding as its *meaning*. However, computing unfoldings of terms might require exponential costs.

Therefore, we proceed in a different way here: we look for a reduction procedure that only involves the positive bias syntax and is compatible with the β-reduction in the negative bias syntax (a reduction step should not change the meaning of a term). In the following, we give a notion of ***β*-redex** and its corresponding ***β*-rule**. Consider the following annotated proof corresponding to a derivation of well-formedness of a term:

$$\dfrac{\overset{\Pi_1}{\Sigma, y : D \vdash s : D} \qquad \overset{\dfrac{\overset{\Pi_2}{\Sigma', x : D, z : D \vdash t : D}}{\Sigma', x : D \vdash \textbf{name } z = xw \textbf{ in } t : D}\,app}{\vdots \atop \Sigma, x : D \vdash C\langle \textbf{name } z = xw \textbf{ in } t\rangle : D}}{\Sigma \vdash \textbf{name } x = \lambda y.s \textbf{ in } C\langle \textbf{name } z = xw \textbf{ in } t\rangle : D}\,abs$$

with $w : D \in \Sigma', x : D$ and C a context. In the term annotating the conclusion, the name x is used to introduce an abstraction $\lambda y.s$ and is later applied to an argument w. We call the named application pattern xw here a ***β*-redex**. To eliminate this β-redex, we shall consider the following proof:

$$\dfrac{\overset{\Pi_1'}{\Sigma', x : D \vdash s[y/w] : D} \qquad \overset{\Pi_2}{\Sigma', x : D, z : D \vdash t : D}}{\Sigma', x : D \vdash Cut(z.t, s[y/w]) : D}\,cut$$

where Π_1' is obtained from Π_1 by variable renaming and weakening. By eliminating this cut, we obtain a cut-free proof of the conclusion $\Sigma', x : D \vdash t[z/s[y/w]] : D$. This gives the following ***β*-rule**:

$$\begin{aligned}&\textbf{name } x = \lambda y.s \textbf{ in } C\langle \textbf{name } z = xw \textbf{ in } t\rangle \to\\ &\textbf{name } x = \lambda y.s \textbf{ in } C\langle t[z/s[y/w]]\rangle\end{aligned}$$

Intuitively, to eliminate a β-redex, it suffices to make a copy of the abstraction body, make a variable renaming within this copy and a variable renaming in the rest of the term. We illustrate these steps in the following example:

$$\begin{aligned}&\textbf{name } f = (\lambda x.\textbf{name } y = xx \textbf{ in name } z = yy \textbf{ in } z) \textbf{ in}\\ &\textbf{name } x_1 = fx_0 \textbf{ in name } x_2 = fx_1 \textbf{ in } x_2 \to_\beta\end{aligned}$$

$$\begin{aligned}&\textbf{name } f = (\lambda x.\textbf{name } y = xx \textbf{ in name } z = yy \textbf{ in } z) \textbf{ in}\\ &\textbf{name } y_1 = x_0x_0 \textbf{ in name } z_1 = y_1y_1 \textbf{ in}\\ &\textbf{name } x_2 = fz_1 \textbf{ in } x_2 \to_\beta\end{aligned}$$

$$\begin{aligned}&\textbf{name } f = (\lambda x.\textbf{name } y = xx \textbf{ in name } z = yy \textbf{ in } z) \textbf{ in}\\ &\textbf{name } y_1 = x_0x_0 \textbf{ in name } z_1 = y_1y_1 \textbf{ in}\\ &\textbf{name } y_2 = z_1z_1 \textbf{ in name } z_2 = y_2y_2 \textbf{ in } z_2\end{aligned}$$

The β-rule is *confluent* but not necessarily terminating as shown by the term:

$$\textbf{name } x = (\lambda y.\textbf{name } z = yy \textbf{ in } z) \textbf{ in name } w = xx \textbf{ in } w.$$

It captures, however, the same set of normal forms as the usual β-rule, as shown by the following propositions and theorem.

Proposition 35. *Let s and t be terms. If $s \to_\beta t$, then $\underline{s} \to_\beta^+ \underline{t}$.*

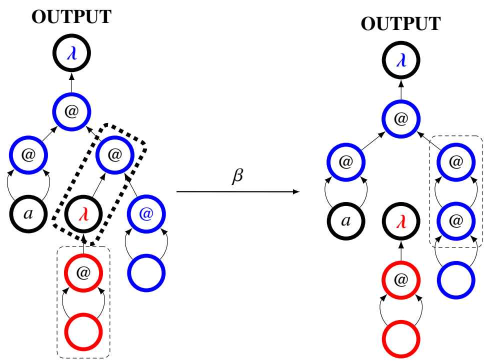


Fig. 6. An example of β-reduction on λ-graphs with bodies. The thick dashed box is a β-redex and the thin dashed boxes in two λ-graphs with bodies correspond to the box of the red λ-node and its copy, respectively.

Proof. A proof is given in Appendix B. □

Proposition 36. *Let s be a term. If $\underline{s} \to_\beta s'$ for some untyped λ-term s', then there exists t such that $s \to_\beta t$.*

Theorem 37. *Let s be a term. Then s is β-normal if, and only if, $\underline{s}$ is β-normal.*

Proof. Immediate from Propositions 35 and 36. □

Furthermore, $\equiv_l$ is a bisimulation with respect to $\to_\beta$.

Theorem 38. *Let t and u be two terms. If $t \equiv_l u$ and $t \to_\beta t'$, then there exists u' such that $u \to_\beta u'$ and $t' \equiv_l u'$.*

Proof. It suffices to see that $\to_l$ never creates or removes β-redexes. □

This reduction procedure does not make any copy of the argument when eliminating a named application. Instead, it keeps the sharing structure used to define the argument. On one hand, this guarantees that the result is still a valid term (in the positive bias syntax) and on the other hand avoids possible exponential blowup as shown in the example above.

This reduction procedure might seem complicated but in fact, it can be expressed in a fairly simple way on λ-graphs with bodies. A β-redex is simply an @-node that has a λ-node as its left child. To eliminate it, it suffices to duplicate the *box* of the λ-node, replace the @-node with the copy of the output node of λ-node, and then replace the bound variable in this copy with the argument, i.e., the right child of the @-node. One should be careful about the structure of bodies: in this copy, all the nodes that were in the body of the λ-node should be moved to the same body as the argument or to the corresponding body if the argument is a bound variable.

Figure 6 shows a β-reduction step on λ-graphs with bodies.

However, there may still be some redundancies in this reduction procedure. Consider the following term:

$$\begin{aligned}&\textbf{name } f = (\lambda x.\textbf{name } y = aa \textbf{ in name } z = xy \textbf{ in } z) \textbf{ in}\\ &\textbf{name } x_1 = fx_0 \textbf{ in name } x_2 = fx_1 \textbf{ in } x_2\end{aligned}$$

To eliminate the two β-redexes fx_0 and fx_1, we have to make two copies of the box of f. Thus, the structure aa is introduced twice. To solve this redundancy, a possible solution is to *lift* the named structure **name** $y = aa$ to top-level before evaluation:

$$\textbf{name } y = aa \textbf{ in name } f = (\lambda x.\textbf{name } z = xy \textbf{ in } z) \textbf{ in}$$
$$\textbf{name } x_1 = f x_0 \textbf{ in name } x_2 = f x_1 \textbf{ in } x_2$$

This operation corresponds, in fact, to a rule permutation in proofs.

VII. Lifting names and nodes

Consider the *abs* rule:

$$\frac{\Sigma, y \vdash_{wf} s \qquad \Sigma, x \vdash_{wf} t}{\Sigma \vdash_{wf} t[x \leftarrow \lambda y.s]}\ abs$$

Intuitively, the left premise is used to build another abstraction under the current body and the right premise is used to complete the current body, just as the only premise of the *app* rule. The rule permutations studied in Section III correspond to the naming permutations that take place within the same body. In fact, there exist some rule permutations that allow moving naming expressions between different bodies. To illustrate this, consider the following derivation:

$$\frac{\{z_1, z_2\} \subseteq \Sigma, y \quad \dfrac{\Sigma, y, z \vdash_{wf} s}{\Sigma, y \vdash_{wf} s[z \leftarrow z_1 z_2]}\ app \qquad \Sigma, x \vdash_{wf} t}{\Sigma \vdash_{wf} t[x \leftarrow (\lambda y.s[z \leftarrow z_1 z_2])]}\ abs$$

If $y \notin \{z_1, z_2\}$, then it is possible to permute these two rules:

$$\frac{\{z_1, z_2\} \subseteq \Sigma \quad \dfrac{\Sigma, z, y \vdash_{wf} s \qquad \Sigma, z, x \vdash_{wf} t}{\Sigma, z \vdash_{wf} t[x \leftarrow \lambda y.s]}\ abs}{\Sigma \vdash_{wf} t[x \leftarrow \lambda y.s][z \leftarrow z_1 z_2]}\ app$$

Similarly, consider the following derivation:

$$\frac{\dfrac{\Sigma, y, w \vdash_{wf} s \qquad \Sigma, y, z \vdash_{wf} s'}{\Sigma, y \vdash_{wf} s'[z \leftarrow \lambda w.s]}\ abs \qquad \Sigma, x \vdash_{wf} t}{\Sigma \vdash_{wf} t[x \leftarrow (\lambda y.s'[z \leftarrow \lambda w.s])]}\ abs$$

If $y \notin fv(s)$, then it is possible to permute these two rules:

$$\frac{\Sigma, w \vdash_{wf} s \qquad \dfrac{\Sigma, z, y \vdash_{wf} s' \qquad \Sigma, z, x \vdash_{wf} t}{\Sigma, z \vdash_{wf} t[x \leftarrow \lambda y.s']}\ abs}{\Sigma \vdash_{wf} t[x \leftarrow \lambda y.s'][z \leftarrow \lambda w.s]}\ abs$$

These two rule permutations induce the following rewrite rules on terms:

$$t[x \leftarrow (\lambda y.s[z \leftarrow z_1 z_2])] \ \to_l\ t[x \leftarrow \lambda y.s][z \leftarrow z_1 z_2] \quad \text{if } y \notin \{z_1, z_2\}$$

$$t[x \leftarrow (\lambda y.s'[z \leftarrow \lambda w.s])] \ \to_l\ t[x \leftarrow \lambda y.s'][z \leftarrow \lambda w.s] \quad \text{if } y \notin fv(s)$$

We choose to express these permutations using rewrite rules instead of equations. In both cases, we say that the name z is ***lifted*** to a higher scope. As discussed in Section VI, lifting a name will change the result of the reduction. However, the unfolding of the term remains unchanged.

Proposition 39. *Let s and t be two terms such that $s \to_l t$. Then $\underline{s} = \underline{t}$.*

Proof. It suffices to see that in λ-calculus, $t\{x/u\}\{x'/u'\} = t\{x/u\{x'/u'\}\}$ if x' is not free in t. □

Also note that with these rules only the first named structure in a body can be lifted. This is the reason why we should consider Σ-terms up to $\equiv_r$: some named structure that appears later may be permuted to the first position and then lifted. As a consequence, we write $[\![t]\!]_\Sigma$ for its underlying Σ-λ-graph with bodies.

As discussed in Section V, Σ-λ-graphs with bodies are isomorphic to Σ-terms up to $\equiv_r$, we should be able to design a rewrite system that implements $\to_l$ on Σ-λ-graphs with bodies. We first define ***contexts*** for λ-graphs with bodies using the following grammar:

$$\begin{aligned} C \coloneqq\ & \square \mid C[\textbf{node } x \leftarrow y@z] \mid C[\textbf{node } x \leftarrow \lambda y.\mathcal{G}] \\ & \mid \mathcal{G}[\textbf{node } x \leftarrow \lambda y.C] \end{aligned}$$

The ***plugging*** $C\langle\mathcal{G}\rangle$ of a λ-graph with bodies $\mathcal{G}$ in the context C is defined inductively by:

$$\begin{aligned} \square\langle\mathcal{G}\rangle &= \mathcal{G} \\ C[\textbf{node } x \leftarrow y@z]\langle\mathcal{G}\rangle &= C\langle\mathcal{G}\rangle\{\textbf{node } x \leftarrow y@z\} \\ C[\textbf{node } x \leftarrow \lambda y.\mathcal{G}']\langle\mathcal{G}\rangle &= C\langle\mathcal{G}\rangle\{\textbf{node } x \leftarrow \lambda y.\mathcal{G}'\} \\ \mathcal{G}'[\textbf{node } x \leftarrow \lambda y.C]\langle\mathcal{G}\rangle &= \mathcal{G}'\{\textbf{node } x \leftarrow \lambda y.C\langle\mathcal{G}\rangle\}. \end{aligned}$$

Definition 40 (Node lifting)**.** Let $\mathcal{G}$ be a λ-graph with bodies. Let l be a body-free abstraction node and n a node in $body(l)$ such that $m \not\prec_l n$ for all $m \in body(l) \cup \{bv(l)\}$. Then $\mathcal{G}$ can be rewritten to $\mathcal{G}'$, by moving n out of $body(l)$ (now n becomes body-free). It is clear that $\mathcal{G}'$ is also a λ-graph with bodies. In this case, we write $\mathcal{G} \to \mathcal{G}'$ and we say that the node n is ***lifted***. We define the relation $\rightsquigarrow$ as the contextual closure of $\to$.

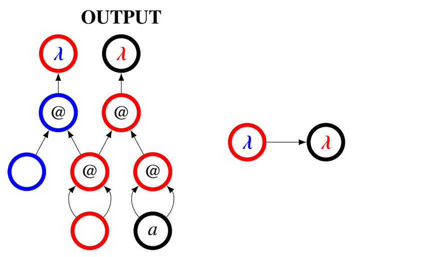
Remark 41. Alternatively, we can define $\rightsquigarrow$ directly by allowing l to be any abstraction node. In this case, if l is in $body(l')$, then n is lifted from $body(l)$ to $body(l')$.

Proposition 42. *Let s and t be two Σ-terms. Then we have $s \to_l t$ if and only if $[\![s]\!]_\Sigma \rightsquigarrow [\![t]\!]_\Sigma$.*

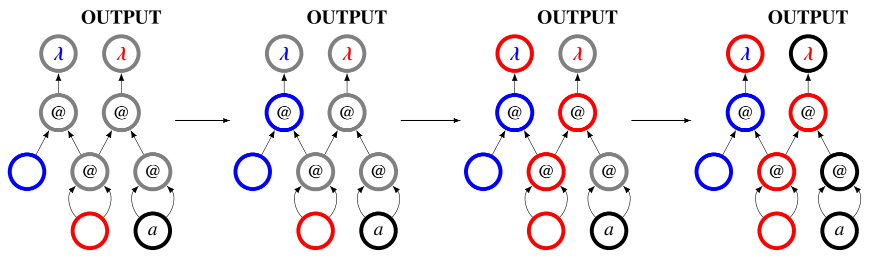
Proof. To prove this, we define a translation $[\![\cdot]\!]_\Sigma$ from contexts for Σ-terms to contexts for Σ-λ-graphs with bodies in a straightforward way. We then have $[\![C\langle t\rangle]\!]_{\Sigma\cup\Sigma'} = [\![C]\!]_{\Sigma'}\langle[\![t]\!]_\Sigma\rangle$. The rest is straightforward by definitions. □

It is not difficult to see that both $\to_l$ and $\rightsquigarrow$ are confluent and strongly normalizing. Thus, every Σ-term t (resp. Σ-λ-graph with bodies $\mathcal{G}$) has a unique normal form $t\downarrow_l$ (resp. $\mathcal{G}\downarrow$) with respect to $\to_l$ (resp. $\rightsquigarrow$). We can define their corresponding equivalences $\equiv_l$ and $\equiv$ by $s \equiv_l t \Leftrightarrow s\downarrow_l = t\downarrow_l$ and $\mathcal{G} \equiv \mathcal{G}' \Leftrightarrow \mathcal{G}\downarrow = \mathcal{G}'\downarrow$.

Theorem 43. *Let s and t be two Σ-terms. Then we have $s \equiv_l t$ if and only if $[\![s]\!]_\Sigma \equiv [\![t]\!]_\Sigma$.*



(a) A λ-graph with bodies $\mathcal{G}$ and its scope graph $\mathcal{B}_{\mathcal{G}}$.



(b) Computation of the bodies of the λ-nodes in $\mathcal{G}\downarrow$.

Fig. 7. Computation of the normal form of a λ-graph with bodies with respect to $\rightsquigarrow$.

The following proposition characterizes normal Σ-terms with respect to $\rightarrow_l$.

Proposition 44. *A Σ-term t is normal with respect to $\rightarrow_l$ if and only if for every name x introducing an abstraction in t, for every name y in $body(x)$, there exists a path $z \rightsquigarrow y$ in $\mathcal{D}_x$ such that $bv(x)$ is in $dep(z)$.*

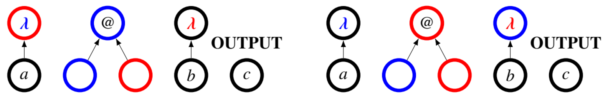
Intuitively, in a normal Σ-term, every intermediate definition in the body of a name introducing an abstraction *depends on* its bound variable. The corresponding proposition for Σ-λ-graphs with bodies can be stated as follows.

Proposition 45. *A Σ-λ-graph with bodies is normal with respect to $\rightsquigarrow$ if and only if for every abstraction node l, for every node $n \in body(l)$, there is a path $m \rightsquigarrow n$ in $\mathcal{D}_l$ such that $bv(l) <_l m$.*

Theorem 46. *The normal form $\mathcal{G}\downarrow$ of a λ-graph with bodies $\mathcal{G}$ can be entirely determined by its underlying pre-graph, bound variables, and scope graph.*

Proof. Since $\mathcal{G}\downarrow$ and $\mathcal{G}$ only differ in their bodies, we only need to show how to determine the bodies of $\mathcal{G}\downarrow$. We now determine the body of each abstraction node following a randomly chosen topological sort T of $\mathcal{B}_{\mathcal{G}}$. For each abstraction node l, we compute the set of nodes in the underlying pre-graph that are *reachable* from $bv(l)$ by a *path* not including l and are not in a body yet. Here, the notion of reachability is subtle: if we reach a node that is already assigned to the body of an abstraction node, we can *jump* to the abstraction node and continue. Let $body(l)$ be this set. It remains to prove that the graph obtained is indeed $\mathcal{G}\downarrow$. Details can be found in Appendix C. □

Figure 7 shows the computation of the normal form of a λ-graph with bodies. Note that the scope graph of $\mathcal{G}\downarrow$ is not necessarily the same as that of $\mathcal{G}$. The scope graph of $\mathcal{G}$ is, however, needed as shown by the following example:



These two distinct λ-graphs with bodies are both normal with respect to $\rightsquigarrow$ and they have the same pre-graph and bound variables.

Remark 47. Our graphical representation is closely related to *λ-graphs* used in [23]. The main difference between them is that we allow the presence of *vacuous nodes*, i.e, nodes that are not used to define the output of a term (or an abstraction). Also, note that every λ-graph can be represented by a λ-graph with bodies. There can be several λ-graphs with bodies that represent the same λ-graph but they all have the same normal form with respect to $\rightsquigarrow$.

Another key property is that normal λ-graphs with bodies with respect to $\rightsquigarrow$ are preserved under β-reduction.

Theorem 48. *Let $\mathcal{G}$ be a normal λ-graph with bodies with respect to $\rightsquigarrow$. If $\mathcal{G} \rightarrow_\beta \mathcal{G}'$, then $\mathcal{G}'$ is also normal with respect to $\rightsquigarrow$.*

VIII. Multi-focusing and synthetic inference rules

In Sections III and VII, we show that synthetic inference rules can sometimes be permuted. In this case, the order in which they are applied is not important. In other words, the actions corresponding to these synthetic inference rules can be seen *in parallel*. This observation results in the design of various *multi-focused* proof systems.

In this section, we consider *mLJF*, a multi-focused proof system for the implicational fragment of *LJ*, shown in Figure 8. This system is the implicational fragment of *mLJF* proposed in [18]. In this system, unfocused sequents are the same as in *LJF* while focused sequents have the following forms:

- left-focused: $\Gamma \Downarrow \Theta \vdash B$
- right-focused: $\Gamma \Downarrow \Theta \vdash B \Downarrow$ with $\Gamma \Downarrow \cdot \vdash B \Downarrow$ simplified as $\Gamma \vdash B \Downarrow$

with Θ a multiset of formulas. Note that the rules of *mLJF* resemble those of *LJF*. In particular, any *LJF* proof is an *mLJF* proof.

Remark 49. Note that we allow multi-focusing on the left-hand side and focusing on both sides of a sequent. However, in the decide rules of *mLJF*, we only allow putting formula(s) under focus on a single side. In fact, in the D_r rule of *mLJF* proposed in [18], it is possible to put formulas under focus on both sides. Suppose that we have a proof:

$$\dfrac{\begin{array}{c}\Pi\\ \Gamma \Downarrow N \vdash P \Downarrow\end{array}}{\Gamma \vdash P}\ D_r$$

Decide, Release, and Store Rules

$$Supp(\mathcal{N}) \subseteq \Gamma \;\; \dfrac{\Gamma \Downarrow \mathcal{N} \vdash A}{\Gamma \vdash A}\,D_l \qquad \dfrac{\Gamma \vdash P \Downarrow}{\Gamma \vdash P}\,D_r \qquad \dfrac{\Gamma \Uparrow \mathcal{P} \vdash A}{\Gamma \Downarrow \mathcal{P} \vdash A}\,R_l \qquad \dfrac{\Gamma \Uparrow \mathcal{P} \vdash N \Uparrow}{\Gamma \Downarrow \mathcal{P} \vdash N \Downarrow}\,R_r \qquad \dfrac{\Gamma, C \Uparrow \Theta \vdash \Delta \Uparrow \Delta'}{\Gamma \Uparrow \Theta, C \vdash \Delta \Uparrow \Delta'}\,S_l \qquad \dfrac{\Gamma \Uparrow \Theta \vdash A}{\Gamma \Uparrow \Theta \vdash A \Uparrow}\,S_r$$

Initial Rules | Introduction Rules for implication

$$\delta(A) = - \;\; \dfrac{}{\Gamma \Downarrow A \vdash A} \qquad \delta(A) = + \;\; \dfrac{}{\Gamma, A \vdash A \Downarrow} \qquad \dfrac{\Gamma \Downarrow \Theta_1 \vdash B \Downarrow \quad \Gamma \Downarrow C, \Theta_2 \vdash \mathcal{R}}{\Gamma \Downarrow B \supset C, \Theta_1, \Theta_2 \vdash \mathcal{R}}\,\supset L \qquad \dfrac{\Gamma \Uparrow \Theta, B \vdash B' \Uparrow}{\Gamma \Uparrow \Theta \vdash B \supset B' \Uparrow}\,\supset R$$

Fig. 8. A multi-focused proof system for *LJ*. Here, Θ_1 and Θ_2 are multisets of formulas. $\mathcal{P}$ is a multiset of positive formulas and $\mathcal{N}$ is a non-empty multiset of negative formulas. $\mathcal{R}$ is of the form B or $B \Downarrow$. $Supp(M)$ denotes the support of the multiset M.

with $\mathcal{N}$ non-empty. In the implicational fragment, as the formula P put under right focus is positive, it is atomic. Note that the application of $\supset L$ always keeps a branch (the right one) with P under focus and its left staging zone is non-empty. It is clear that this branch cannot be finished with an initial rule or a release rule, which leads to a contradiction.

We now illustrate how *mLJF* captures rule permutations described in Sections III and VII. As an example, we use the rules for the positive bias syntax, as shown in Figure 2, and consider the following *LJF* derivations justifying the *app* and *abs* rules for the positive bias syntax.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash D \Downarrow}\,I_r \qquad \dfrac{\dfrac{}{\Gamma \vdash D \Downarrow}\,I_r \qquad \dfrac{\Gamma, D \vdash D}{\Gamma \Downarrow D \vdash D}\,S_l/R_l}{\Gamma \Downarrow D \supset D \vdash D}\,\supset L}{\Gamma \Downarrow D \supset D \supset D \vdash D}\,\supset L}{\Gamma \vdash D}\,D_l$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, D \vdash D}{\Gamma \Uparrow D \vdash D \Uparrow}\,S_l/S_r}{\Gamma \vdash D \supset D \Uparrow}\,\supset R}{\Gamma \vdash D \supset D \Downarrow}\,R_r \qquad \dfrac{\Gamma, D \vdash D}{\Gamma \Downarrow D \vdash D}\,S_l/R_l}{\Gamma \Downarrow (D \supset D) \supset D \vdash D}\,\supset L}{\Gamma \vdash D}\,D_l$$

Each of the two derivations introduces a new occurrence (right-premise occurrence) of D in the right branch and the derivation corresponding to the *abs* rule also introduces a new occurrence (left-premise occurrence) of D in the left branch.

We now rephrase the conditions for permuting rules described in Sections III and VII. For a rule r to permute upward over *app*, it is necessary and sufficient to have the right-premise occurrence of r (or the left-premise occurrence, if r is *abs* and *app* is in the left branch) not be used in the initial rules in the derivation justifying *app*. For a rule r to permute upward over *abs*, it is necessary and sufficient to have the right-premise occurrence of r not be used in at least one branch so that we can permute r into the other branch.

Definition 50. A ***totally permutable*** derivation is a derivation built with *app* and *abs* in which any two consecutive rules can be permuted.

The following proposition is a consequence of the observations above.

Proposition 51. *A derivation (or partial proof) Π is totally permutable if, and only if,*

1) *every new occurrence of D introduced in Π is not used in any initial rule in the derivation justifying Π, and*
2) *for every right-premise occurrence o of D introduced in Π, there is a map $unused_o : \Lambda_\Pi \to \{left, right\}$, where Λ_Π is the set of abs applications in Π such that for every abs application r in Π, $unused_o(r)$ branch of r does not use o (even for the complete proof).*

We now show how to construct a multi-focused derivation in *mLJF* from a totally permutable derivation Π.

Start with the conclusion of Π and the D_l rule by putting all the focused formulas in Π under focus at the same time. We now apply the $\supset L$ rule to each focused formula (and its subformula) as in the derivation Π so that the multi-focused proof can have the same tree structure as Π.

It remains to determine how to split the left staging zone. A natural choice is to split it following the tree structure of the derivation Π. If a formula is put under focus in the left (resp. right) branch in Π, then we put it in the left (resp. right) premise of $\supset L$. There can be some right-premise occurrences in the left staging zone: they are from those focused formulas that are already treated. These right-premise occurrences are distributed following their *unused* functions mentioned in Proposition 51: for a right-premise occurrence o, if $unused_o(r)$ is $left$ (resp. $right$), then it is put in the right (resp. left) branch (r is the *abs* application in Π that corresponds to the focused formula considered).

Actions on focused formulas within the same $\Downarrow$-phase should be considered in parallel. This is often described using *local permutations* in corresponding unfocused proof systems as in [16] and [17]. In our case, it is possible to describe this phenomenon in the focused system.

Proposition 52. *Consider the following derivation:*

$$\dfrac{\Gamma \Downarrow \Theta_1 \vdash B'_1 \Downarrow \quad \cdots \quad \Gamma \Downarrow \Theta_l \vdash B'_l \Downarrow \quad \Gamma \Downarrow \Theta_{l+1} \vdash \mathcal{R}}{\Gamma \Downarrow F_1, \ldots, F_k \vdash \mathcal{R}}\,\Pi$$

where for $1 \le j \le l+1$, Θ_j contains atomic formulas only, and Π contains the $\supset L$ rule only. Then we can reorganize the applications of $\supset L$ such that

1) *the applications of $\supset L$ to F_i and its subformulas are connected and thus form a* sub-focused *phase,*
2) *the actions on $F_1, \ldots, F_k$ can be taken in any order, and*

$$\{x_i : D, y_i : D\} \subseteq \Gamma \text{ for all } i \quad \frac{\overbrace{\Gamma, [S_j] : D^{p_j}, z_j : D \vdash t_j : D}^{\text{for } 1 \le j \le l} \qquad \Gamma, [S_{l+1}] : D^{p_{l+1}} \vdash t_{l+1} : D}{\Gamma \vdash papp_k\ pabs_l\ x_1y_1 \cdots x_ky_k([S_1]^\#.\lambda z_1.t_1) \cdots ([S_l]^\#.\lambda z_l.t_l)([S_{l+1}]^\#.t_{l+1}) : D}$$

Fig. 9. Annotated rule scheme for $LJ\lfloor\delta^+, \Gamma_0\rfloor^*$.

3) *the endsequents remain the same.*

Proof. It suffices to observe that any two consecutive applications of $\supset L$ on independent formulas (one is not a subformula of the other) can be permuted. Details can be found in Appendix D. □

Instead of treating the left staging zone as a multiset of formulas, we can treat it as a list of formulas and take actions on each formula (and its subformulas) from left to right. By making this adjustment, the $\supset L$ rule becomes:

$$\frac{\Gamma \Downarrow \mathcal{P}_1, \Theta_1 \vdash B \Downarrow \qquad \Gamma \Downarrow \mathcal{P}_2, C, \Theta_2 \vdash \mathcal{R}}{\Gamma \Downarrow \mathcal{P}, B \supset C, \Theta \vdash \mathcal{R}} \supset L$$

such that $\mathcal{P}_1 + \mathcal{P}_2 = \mathcal{P}$ and $\Theta_1 + \Theta_2 = \Theta$ where $\mathcal{P}, \mathcal{P}_1$ and $\mathcal{P}_2$ contain positive formulas only, and $\Delta_1 + \Delta_2$ is a list of composed of all the elements of Δ_1 and Δ_2 and the order in which they appear in Δ_1 (resp. Δ_2) is preserved.

By proceeding this way, we obtain a derivation that resembles the concatenation of the synthetic inference rules of all the focused formulas. This leads to the definition of *synthetic inference rules by multi-focusing*.

Definition 53. A *synthetic inference rule by multi-focusing* is a rule of the form

$$\frac{\Gamma_1 \vdash A_1 \cdots \Gamma_n \vdash A_n}{\Gamma \vdash A} \mathcal{N} \quad \text{*justified* by an *mLJF* derivation of the form} \quad \frac{\begin{array}{c}\Gamma_1 \vdash A_1 \cdots \Gamma_n \vdash A_n \\ \Pi \\ \Gamma \Downarrow \mathcal{N} \vdash A\end{array}}{\Gamma \vdash A} D_l$$

Here, $Supp(\mathcal{N}) \subseteq \Gamma$ where $Supp(M)$ denotes the support of the multiset M, $n \ge 0$, and within Π, a $\Downarrow$-sequent never occurs above an $\Uparrow$-sequent. The structure of *mLJF* proofs also forces $Supp(\mathcal{N}) \subseteq \Gamma_i$ for all $1 \le i \le n$.

Similarly, we can define the ***extension of LJ by the polarized theory*** $(\mathcal{T}, \delta)$ ***using multi-focusing***, written $LJ\lfloor\delta, \mathcal{T}\rfloor^*$. It is clear that $LJ\lfloor\delta, \mathcal{T}\rfloor^*$ contains all the rules of $LJ\lfloor\delta, \mathcal{T}\rfloor$. Hence, by giving annotations to rules of $LJ\lfloor\delta, \mathcal{T}\rfloor^*$, we obtain a more expressive term structure than the one built from $LJ\lfloor\delta, \mathcal{T}\rfloor$.

We give in Figure 9 the annotated rule scheme of $LJ\lfloor\delta^+, \Gamma_0\rfloor^*$. The full analysis on how this rule scheme is built can be found in Appendix E. Here, $([S_j])_{1\le j\le l+1}$ is a family of lists of variables recording the choices of splitting. It is defined by two lists of variables $(w_i)_{1\le i\le k}$ and $(v_j)_{1\le j\le l}$, and two families $(I_j)_{1\le j\le l+1}$ and $(J_j)_{1\le j\le l+1}$ such that:

- $(I_j)_{1\le j\le l+1}$ is a partition of $\{1, \dots, k\}$. This corresponds to the occurrences of D coming from $(D \supset D \supset D)^k$.
- $(J_j)_{1\le j\le l+1}$ is a partition of $\{1, \dots, l\}$. Moreover, for all j, $J_{j+1} \subseteq \{1, \dots, j\}$. This corresponds to the occurrences of D coming from $((D \supset D) \supset D)^l$.

We should also keep these splitting choices in the annotation by defining $[S_j] = \{w_i, i \in I_j\} \cup \{v_{j'}, j' \in J_j\}$, $[S_j]^\# = \{(i, w_i), i \in I_j\} \cup \{(k + j', v_{j'}), j' \in J_j\}$ for $1 \le j \le l + 1$.

This annotation should be understood as follows:

- For all i, w_i is a name introducing the application x_iy_i.
- For all j, v_j is a name introducing the abstraction $\lambda z_j.t_j$.
- All the w_i and v_j are *bound* in exactly one of the abstractions $\lambda z_{j'}.t_{j'}$ or in the continuation t_{l+1}.
- For all j, only $v_{j'}$ such that $j' < j$ could be bound in $\lambda z_j.t_j$. This condition is not trivial and is shown in the construction. See Appendix E for details.

This new term representation allows expressing certain parallelism in terms and provides more flexibility compared to the term representation built with single-focusing. Technical development involving this term representation is, however, beyond the scope of this paper.

IX. Generalization

In Definition 3, *LJ* can be extended by any polarized theory of order one or two. In our study of untyped λ-terms, we use exactly one formula of order one ($D \supset D \supset D$) and one formula of order two ($(D \supset D) \supset D$). Our graphical representation can be generalized to any *positively polarized* theory of order one or two in the following way.

Each node comes with a *type* which is an atomic formula.

A formula of order one or two can be written as $B_1 \supset \cdots \supset B_n \supset A$ with $n \ge 1$, B_i of order at most one and A atomic. This formula corresponds to a node *of type* A that has n incoming edges, each of which corresponds to one B_i. For $1 \le i \le n$, if B_i is atomic, then it corresponds to simply a node, and if B_i is of order one, then it comes with a notion of *body*.

X. Conclusion

We introduce a graphical representation for the untyped λ-terms and prove that it captures certain interesting equivalences on terms. We propose a reduction procedure for terms that can be implemented on graphs easily and does not remove any sharing structure.

We study the multi-focused proof system *mLJF* and describe its relation with permutations of synthetic inference rules. We also define synthetic inference rules by multi-focusing and show that they can provide a simple syntax that allows expressing certain permutations/parallelism in terms. In the future, we hope to gain more insights into term representation by studying cut-elimination in this multi-focused system.

Finally, the developments in this paper are restricted to $\Gamma_0 = \{D \supset D \supset D, (D \supset D) \supset D\}$ but can be generalized to any theory of at most second order.

References

[1] J.-Y. Girard, "Linear logic," *Theoretical Computer Science*, vol. 50, no. 1, pp. 1–102, 1987.

[2] D. Miller, "A compact representation of proofs," *Studia Logica*, vol. 46, no. 4, pp. 347–370, 1987.

[3] J.-M. Andreoli, "Logic programming with focusing proofs in linear logic," *J. of Logic and Computation*, vol. 2, no. 3, pp. 297–347, 1992.

[4] D. J. D. Hughes, "Proofs without syntax," *Annals of Mathematics*, vol. 143, no. 3, pp. 1065–1076, Nov. 2006.

[5] J.-M. Andreoli, "Focussing and proof construction," *Annals of Pure and Applied Logic*, vol. 107, no. 1, pp. 131–163, 2001.

[6] K. Chaudhuri, "Focusing strategies in the sequent calculus of synthetic connectives," in *LPAR: International Conference on Logic, Programming, Artificial Intelligence and Reasoning*, ser. LNCS, I. Cervesato, H. Veith, and A. Voronkov, Eds., vol. 5330. Springer, Nov. 2008, pp. 467–481.

[7] S. Marin, D. Miller, E. Pimentel, and M. Volpe, "From axioms to synthetic inference rules via focusing," *Annals of Pure and Applied Logic*, vol. 173, no. 5, pp. 1–32, 2022.

[8] H. Herbelin, "A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure," in *Computer Science Logic, 8th International Workshop, CSL '94*, ser. LNCS, vol. 933. Springer, 1995, pp. 61–75.

[9] R. Dyckhoff and S. Lengrand, "LJQ: a strongly focused calculus for intuitionistic logic," in *Computability in Europe 2006*, ser. LNCS, A. Beckmann and *et al.*, Eds., vol. 3988. Springer, 2006, pp. 173–185.

[10] C. Liang and D. Miller, "Focusing and polarization in linear, intuitionistic, and classical logics," *Theoretical Computer Science*, vol. 410, no. 46, pp. 4747–4768, 2009, abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi.

[11] V. Danos, J.-B. Joinet, and H. Schellinx, "LKT and LKQ: sequent calculi for second order logic based upon dual linear decompositions of classical implication," in *Advances in Linear Logic*, ser. London Mathematical Society Lecture Note Series, J.-Y. Girard, Y. Lafont, and L. Regnier, Eds. Cambridge University Press, 1995, no. 222, pp. 211–224.

[12] T. Brock-Nannestad, N. Guenot, and D. Gustafsson, "Computation in focused intuitionistic logic," in *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14–16, 2015*, M. Falaschi and E. Albert, Eds. ACM, 2015, pp. 43–54.

[13] D. Miller and J.-H. Wu, "A positive perspective on term representation," in *CSL 2023: Computer Science Logic*, 2023.

[14] D. Miller and A. Saurin, "From proofs to focused proofs: a modular proof of focalization in linear logic," in *CSL 2007: Computer Science Logic*, ser. LNCS, J. Duparc and T. A. Henzinger, Eds., vol. 4646. Springer, 2007, pp. 405–419.

[15] O. Delande and D. Miller, "A neutral approach to proof and refutation in MALL," in *23th Symp. on Logic in Computer Science*, F. Pfenning, Ed. IEEE Computer Society Press, 2008, pp. 498–508.

[16] K. Chaudhuri, D. Miller, and A. Saurin, "Canonical sequent proofs via multi-focusing," in *Fifth International Conference on Theoretical Computer Science*, ser. IFIP, G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, Eds., vol. 273. Springer, Sep. 2008, pp. 383–396.

[17] K. Chaudhuri, S. Hetzl, and D. Miller, "A multi-focused proof system isomorphic to expansion proofs," *Journal of Logic and Computation*, vol. 26, no. 2, pp. 577–603, 2016.

[18] E. Pimentel, V. Nigam, and J. Neto, "Multi-focused proofs with different polarity assignments," in *Proc. of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015)*, ser. ENTCS, M. Benevides and R. Thiemann, Eds., vol. 323, Jul. 2016, pp. 163–179.

[19] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy, "Explicit substitutions," *Journal of Functional Programming*, vol. 1, no. 4, pp. 375–416, Oct. 1991.

[20] R. Di Cosmo and D. Kesner, "Strong normalization of explicit substitutions via cut elimination in proof nets," in *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, 1997, pp. 35–46.

[21] B. Accattoli, "Proof nets and the call-by-value λ-calculus," *Journal of Theoretical Computer Science (TCS)*, 2015.

[22] ——, "Proof nets and the linear substitution calculus," in *International Colloquium on Theoretical Aspects of Computing*. Springer, 2018, pp. 37–61.

[23] A. Condoluci, B. Accattoli, and C. S. Coen, "Sharing equality is linear," in *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, 2019, pp. 1–14.

Appendix

A. Proof of Proposition 29

We only prove the converse implication here since the direct implication is trivial. Assume that $S = S_1, \dots, S_k$. Let $T = T_1, \dots, T_k$ be a topological sort of $\mathcal{G}$. Suppose that i is the minimal integer such that $S_i \neq T_i$. We have thus $S_1 = T_1, \dots, S_{i-1} = T_{i-1}$. Now we construct a sequence S' from applying a sequence of valid swaps to S such that

$$S'_1 = T_1, \dots, S'_i = T_i. \tag{1}$$

Let j be the unique integer such that $S_j = T_i$. It is clear that $j > i$. We now swap S_j with S_{j-1}. This is a valid swap since S is a topological sort and the sequence obtained is still a topological sort. By repeating this step, we can obtain a topological sort S' by moving S_j to the i-th position and S' satisfies clearly (1).

Now repeat the step by considering S' instead of S, and so on. We can eventually reach T by a sequence of valid swaps.

B. Proof of Proposition 35

To prove this, we define the unfolding $\underline{C}$ of a context C inductively:

$$\underline{\square} = \diamond$$
$$\underline{\textbf{name } x = yz \textbf{ in } C} = \underline{C}\{x/yz\}$$
$$\underline{\textbf{name } x = \lambda y.s \textbf{ in } C} = \underline{C}\{x/\lambda y.\underline{s}\}$$
$$\underline{\textbf{name } x = \lambda y.C \textbf{ in } s} = \underline{s}\{x/\lambda y.\underline{C}\}$$

Here $\diamond$ is a variable and $\underline{C}$ is an untyped λ-term. We have $\underline{C\langle t\rangle} = \underline{C}\{\diamond/\underline{t}\}$ by a straightforward induction. It suffices now to prove

$$\underline{\textbf{name } x = \lambda y.s \textbf{ in } C\langle\textbf{name } z = xw \textbf{ in } t\rangle} \to_\beta^+ \underline{\textbf{name } x = \lambda y.s \textbf{ in } C\langle t\{z/s[y/w]\}\rangle}.$$

We have

$$\begin{aligned} & \underline{\textbf{name } x = \lambda y.s \textbf{ in } C\langle\textbf{name } z = xw \textbf{ in } t\rangle} \\ = {} & \underline{C\langle\textbf{name } z = xw \textbf{ in } t\rangle}\{x/\lambda y.\underline{s}\} \\ = {} & \underline{C}\{\diamond/\underline{\textbf{name } z = xw \textbf{ in } t}\}\{x/\lambda y.\underline{s}\} \\ = {} & \underline{C}\{\diamond/\underline{t}\{z/xw\}\}\{x/\lambda y.\underline{s}\} \end{aligned}$$

and

$$\begin{aligned} & \underline{\textbf{name } x = \lambda y.s \textbf{ in } C\langle t\{z/s[y/w]\}\rangle} \\ = {} & \underline{C\langle t\{z/s[y/w]\}\rangle}\{x/\lambda y.\underline{s}\} \\ = {} & \underline{C}\{\diamond/\underline{t\{z/s[y/w]\}}\}\{x/\lambda y.\underline{s}\} \\ = {} & \underline{C}\{\diamond/\underline{t}\{z/\underline{s}[y/w]\}\}\{x/\lambda y.\underline{s}\}. \end{aligned}$$

It is clear that

$$\underline{C}\{\diamond/\underline{t}\{z/xw\}\}\{x/\lambda y.\underline{s}\} \to_\beta^+ \underline{C}\{\diamond/\underline{t}\{z/\underline{s}[y/w]\}\}\{x/\lambda y.\underline{s}\}.$$

C. Proof of Theorem 46

Let $\mathcal{G}'$ be the graph obtained by construction. In order to distinguish the bodies of an abstraction l in $\mathcal{G}$ and $\mathcal{G}'$, we will denote them by $body_{\mathcal{G}}(l)$ and $body_{\mathcal{G}'}(l)$, respectively. We first make a claim that connects the bodies of $\mathcal{G}'$ with those of $\mathcal{G}$.

Claim. *For every node n in $\mathcal{G}$, if $n \in body_{\mathcal{G}}(l) \cup \{bv(l)\}$ and $n \in body_{\mathcal{G}'}(l') \cup \{bv(l')\}$, then we have $l \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$.*

To prove this claim, the following lemma is useful and is not difficult to prove.

Lemma. *Let n_0 and n_1 be two nodes in a λ-graph with bodies $\mathcal{G}$ such that $n_0 \in body_{\mathcal{G}}(l_0) \cup \{bv_{\mathcal{G}}(l_0)\}$, $n_1 \in body_{\mathcal{G}}(l_1)$, and n_1 is reachable from n_0 in the underlying pre-graph. Then we have $l_1 \rightsquigarrow l_0$ in $\mathcal{B}_{\mathcal{G}}$.*

We now prove the claim by induction on the construction of $\mathcal{G}'$ (on the topological sort T of $\mathcal{B}_{\mathcal{G}}$):

- Base case: let l' be the minimal abstraction node with respect to T. Assume that $n \in body_{\mathcal{G}'}(l')$. Then n is reachable from $bv(l')$ in the underlying pre-graph of $\mathcal{G}$. By the lemma, we have: if $n \in body_{\mathcal{G}}(l)$, then $l \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$.
- For the inductive cases, we proceed by induction on the path reaching the node n from $bv(l')$ in Theorem 46.
  - Base case: trivial.
  - Inductive case: n is reached by a (possibly empty) sequence of *jumps* from n_0 such that the path reaching n_0 from $bv(l')$ does not end with a jump. n is reached from n_0 by jumps, which means that there exist k and a sequence $n_0, \dots, n_k = n$ such that $n_i \in body_{\mathcal{G}'}(n_{i+1})$ for all i. Suppose that $n_i \in body_{\mathcal{G}}(l_i)$. Then by I.H., we have $l_i \rightsquigarrow n_{i+1}$ in $\mathcal{B}_{\mathcal{G}}$. Therefore, $l_0 \rightsquigarrow n_1 \to l_1 \rightsquigarrow l_1 \cdots l_{k-1} \rightsquigarrow n_k = n \to l$ in $\mathcal{B}_{\mathcal{G}}$. Let m be the node from which n is reached by an edge of $\mathcal{G}$. We have $m \in body_{\mathcal{G}'}(l') \cup \{bv(l')\}$. Suppose that $m \in body_{\mathcal{G}}(l_m)$. Then by I.H. and by the lemma, we have $l_0 \rightsquigarrow l_m \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$. Now we have two paths $l_0 \rightsquigarrow l$ and $l_0 \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$. Since bodies are disjoint and $\mathcal{B}_{\mathcal{G}}$ is acyclic, we should have $l' \rightsquigarrow l$ or $l \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$. Note that we have $n_{k-1} \in body_{\mathcal{G}'}(n_k)$, which means n_k appears earlier than l' in T. As a result, we have $l \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$.

We now show that $\mathcal{G}'$ is indeed a λ-graph with bodies.

- It is clear that the bodies of $\mathcal{G}'$ are disjoint.
- We now prove that the scope graph $\mathcal{B}_{\mathcal{G}'}$ of $\mathcal{G}'$ is acyclic. Let l_0 and l' be two abstraction nodes such that $l_0 \in body_{\mathcal{G}'}(l')$. Suppose that $l_0 \in body_{\mathcal{G}}(l)$. Then by the claim, we have $l_0 \to l \rightsquigarrow l'$ in $\mathcal{B}_{\mathcal{G}}$. Therefore, the acyclicity of $\mathcal{B}_{\mathcal{G}}$ implies that of $\mathcal{B}_{\mathcal{G}'}$.
- Point 3 of Definition 15 is trivial by construction.

It remains to prove that $\mathcal{G}'$ can be reached from $\mathcal{G}$ by $\rightsquigarrow$. We give the following procedure that lifts nodes from the body of each abstraction node l following the topological sort T. For each abstraction node l, we lift all the nodes in $body_{\mathcal{G}}(l)$ that are not in $body_{\mathcal{G}'}(l)$ and *update* $\mathcal{G}$. It is not difficult to see these lifting steps are feasible by inspecting the (updated) dependency graphs of $\mathcal{G}$.

Remark. As stated by the claim, for a node $n \in body_{\mathcal{G}}(l) \cap body_{\mathcal{G}'}(l')$, we have a path from $l = l_0, \dots, l_k = l'$ in $\mathcal{B}_{\mathcal{G}}$. In fact, during this procedure, the node is lifted from $body(l_0)$ to $body(l_1)$, …, and finally, from $body(l_{k-1})$ to $body(l_k)$.

D. *Proof of Proposition 52*

Let $F_i = B_{i1} \supset \cdots \supset B_{im_i} \supset A_i$ with A_i atomic and $\mathcal{F}_i = \{A_i, B_{im_i} \supset A_i, \dots, B_{i1} \supset \cdots \supset B_{im_i} \supset A_i\}$. Observe that for every sub-derivation of Π that has the same conclusion as Π, the left staging zones of its endsequents contain exactly one formula from each $\mathcal{F}_i$. In particular, $l = \sum_{1 \le i \le k} m_i$, and $(\Theta_j)_{1 \le j \le l+1}$ forms a partition of $\{A_1, \dots, A_k\}$. Moreover, $(B'_j)_{1 \le j \le l}$ forms a permutation of $\bigcup_{1 \le i \le k} \{B_{i1}, \dots, B_{im_i}\}$.

We now show that we can permute rules in Π such that the $\supset L$ rule is first applied to non-atomic formulas in $\mathcal{F}_{\sigma(1)}$, and then $\mathcal{F}_{\sigma(2)}$, …, until $\mathcal{F}_{\sigma(k)}$ where σ is any permutation of $\{1, \dots, k\}$. To show this, it suffices to show that two consecutive $\supset L$ applications to (non-atomic) formulas from $\mathcal{F}_i$ and $\mathcal{F}_j$ with $i \neq j$ can always be permuted:

1)

$$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B' \Downarrow\end{matrix} \quad \begin{matrix}\Pi_2\\ \Gamma \Downarrow C', \Theta_2 \vdash B \Downarrow\end{matrix}}{\Gamma \Downarrow B' \supset C', \Theta_1, \Theta_2 \vdash B \Downarrow} \supset L \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C, \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

$$\rightsquigarrow$$

$$\dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B' \Downarrow\end{matrix} \quad \dfrac{\begin{matrix}\Pi_2\\ \Gamma \Downarrow C', \Theta_2 \vdash B \Downarrow\end{matrix} \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C, \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow B \supset C, C', \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

2)

$$\dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B \Downarrow\end{matrix} \quad \dfrac{\begin{matrix}\Pi_2\\ \Gamma \Downarrow C, \Theta_2 \vdash B' \Downarrow\end{matrix} \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C', \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow C, B' \supset C', \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B \Downarrow\end{matrix} \quad \begin{matrix}\Pi_2\\ \Gamma \Downarrow C, \Theta_2 \vdash B' \Downarrow\end{matrix}}{\Gamma \Downarrow B \supset C, \Theta_1, \Theta_2 \vdash B' \Downarrow} \supset L \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C', \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

3)

$$\dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B \Downarrow\end{matrix} \quad \dfrac{\begin{matrix}\Pi_2\\ \Gamma \Downarrow \Theta_2 \vdash B' \Downarrow\end{matrix} \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C, C', \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow C, B' \supset C', \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

$$\rightsquigarrow$$

$$\dfrac{\begin{matrix}\Pi_2\\ \Gamma \Downarrow \Theta_2 \vdash B' \Downarrow\end{matrix} \quad \dfrac{\begin{matrix}\Pi_1\\ \Gamma \Downarrow \Theta_1 \vdash B \Downarrow\end{matrix} \quad \begin{matrix}\Pi_3\\ \Gamma \Downarrow C, C', \Theta_3 \vdash \mathcal{R}\end{matrix}}{\Gamma \Downarrow B \supset C, C', \Theta_1, \Theta_3 \vdash \mathcal{R}} \supset L}{\Gamma \Downarrow B \supset C, B' \supset C', \Theta_1, \Theta_2, \Theta_3 \vdash \mathcal{R}} \supset L$$

E. *Construction of the annotated rule scheme for* $LJ\lfloor\delta^+, \Gamma_0\rfloor^*$ *presented in Figure 9*

We start with the following derivation:

$$\dfrac{\begin{matrix}\Pi\\ \Gamma_0 \Downarrow (D \supset D \supset D)^k, ((D \supset D) \supset D)^l \vdash D\end{matrix}}{\Gamma_0 \vdash D} D_l$$

where $B^k = \overbrace{B, \dots, B}^{k \text{ times}}$. Assume $k \ge 1$ and $l \ge 1$. We have thus

$$\Pi = \dfrac{\Gamma_0 \Downarrow \Theta_1 \vdash D \Downarrow \quad \dfrac{\Gamma_0 \Downarrow \Theta_2 \vdash D \Downarrow \quad \begin{matrix}\Pi'\\ \Gamma_0 \Downarrow D, \Theta_3 \vdash D\end{matrix}}{\Gamma_0 \Downarrow D \supset D, \Theta_2 + \Theta_3 \vdash D} \supset L}{\Gamma_0 \Downarrow (D \supset D \supset D)^k, ((D \supset D) \supset D)^l \vdash D} \supset L$$

with $\Theta_1 + \Theta_2 + \Theta_3 = (D \supset D \supset D)^{k-1}, ((D \supset D) \supset D)^l$.

By using the same argument as the one used in Remark 49, Θ_1 has to be empty, and so does Θ_2. By repeating this step for all the occurrences of $D \supset D \supset D$, and by omitting all the intermediate steps, we have $\Pi =$

$$\dfrac{\overbrace{\Gamma_0 \vdash D \Downarrow \quad \Gamma_0 \vdash D \Downarrow}^{\times k} \quad \begin{matrix}\Pi'\\ \Gamma_0 \Downarrow D^k, ((D \supset D) \supset D)^l \vdash D\end{matrix}}{\Gamma_0 \Downarrow (D \supset D \supset D)^k, ((D \supset D) \supset D)^l \vdash D}$$

Π' contains l applications of $\supset L$, so we have l right-focused endsequents and one left-focused endsequent. The left staging zones of these endsequents contain l occurrences of D from the l occurrences of $(D \supset D) \supset D$ and the left-focused endsequent contains at least one of them. As a result, the l right-focused endsequents share the remaining (at most $l-1$) occurrences of D. By the pigeonhole principle, there is one right-focused endsequents whose left staging zone does not contain any occurrence of D from the l occurrences of $(D \supset D) \supset D$ (it might contain some occurrences of D that are already in the conclusion). Consider the occurrence of $(D \supset D) \supset D$ that is the origin of the right-hand side of this right-focused sequent. We now reorganize Π' so that we first apply $\supset L$ to this occurrence. Π' becomes:

$$\dfrac{\begin{matrix}\Pi'_1\\ \Gamma_0 \Downarrow D^{k_1} \vdash D \supset D \Downarrow\end{matrix} \quad \begin{matrix}\Pi'_2\\ \Gamma_0 \Downarrow D^{k_2}, D, ((D \supset D) \supset D)^{l-1} \vdash D\end{matrix}}{\Gamma_0 \Downarrow D^k, ((D \supset D) \supset D)^l \vdash D} \supset L$$

with $k_1 + k_2 = k$.

By repeating the same argument, Π can be organized in such a way that whenever $\supset L$ is applied, the left staging zone of the left premise contains only occurrences of D that are already present in the sequent and no occurrence of $(D \supset D) \supset D$. This leads to the annotated rule scheme shown in Figure 9.