# Nested Proof Search as Reduction in the $\lambda$-calculus

Nicolas Guenot
LIX, École Polytechnique
rue de Saclay, 91128 Palaiseau, France
nguenot@lix.polytechnique.fr

## ABSTRACT

We present here a proof system called JS for purely implicative intuitionistic logic at the propositional level in the formalism of the *calculus of structures*, which is a generalisation of the sequent calculus implementing the *deep inference* methodology. We show that this system is sound and complete with respect to the usual sequent calculus LJ, and consider a restricted system JLSd for a restricted class of formulas. Moreover, we show how to encode $\lambda$-terms with explicit substitutions inside formulas of JLSd and prove that there is a correspondence between proof search in JLSd and reduction in a $\lambda$-calculus with explicit substitutions. Finally, we present a restriction JLSn of JLSd which allows to establish the same correspondence with the standard $\lambda$-calculus equipped with $\beta$-reduction, and show that we can prove results on the reductions of our $\lambda$-calculus with explicit substitutions, as well as the correspondence between the standard $\beta$-reduction and explicit substitutions, by proving results on derivations in the JLSd and JLSn systems.

## 1. INTRODUCTION

This work is oriented towards the connection between structural *proof theory*, a field where the properties of proofs as objects of deductive systems as well as the design of logical rules and transformations applied on proofs are studied, and the design of programming languages, in particular in the setting of the two well-known paradigms of *functional programming* and *logic programming*. Both of these language paradigms have a central connection to logical systems and have greatly benefited from advances made in proof theory over the last decades. The goal of this paper is to show how some recent developments in proof theory can be applied to introduce new ways of reasoning about, and designing, programming languages.

**Deep inference**. This proof-theoretical methodology [GS01] is an extension of the traditional view of deductive systems such as *natural deduction* and the *sequent calculus*, where inference rules are allowed to be applied deep inside some formula. It has been implemented in several formalisms, and in particular in the *calculus of structures* [Brü03], where no meta-level structure such as a sequent is used, but formulas are directly rewritten by inference rules. As an example, the SKS system for classical logic is an elegant proof system which exhibits a nice top-down symmetry and where all inference rules are atomic, in the sense that they never modify, nor copy, nor erase non-atomic formulas [Brü06]. In this system, inference is not necessarily *shallow*, as illustrated in the following proof:

$$\dfrac{\dfrac{\dfrac{\top}{a \vee \neg a}}{(a \wedge (\neg b \vee b)) \vee \neg a}}{\neg b \vee ((a \wedge b) \vee \neg a)}$$

Moreover, the SKS proof system enjoys a surprisingly simple syntactic cut elimination procedure, which is quite different from the standard proof of the sequent calculus although the cut is defined as an inference rule within the system. The calculus of structures is a versatile tool, and it has also been used to study *non-commutativity* in the setting of linear logic [GS01]. Here we define a proof system for intuitionistic logic, which has not yet been studied thoroughly in this setting.

**Functional programming**. The $\lambda$-calculus is often considered as the theoretical core of the functional programming paradigm, since it provides clean foundations for its main abstraction, the notion of function, as well as the central ideas of bindings, scope and variables. Although it needs to be enriched with various features to be used as a reasonable programming language, it is the home ground for the mathematical study of the evaluation of functional programs, and in particular the study of evaluation strategies [Plo75]. There is a very important line of research on the connection between the proofs of intuitionistic logic and functional programs seen as $\lambda$-terms, often referred to as the *Curry-Howard tradition*. Its cornerstone is the correspondence between normalisation of proofs in natural deduction and the $\beta$ rule expressing the operational behaviour of the $\lambda$-calculus.

The Curry-Howard correspondence has been extended later to proofs of the sequent calculus [Her94] and to a decomposition of the $\lambda$-calculus through the reification inside the language of the notion of substitution, leading to an extensive literature on explicit substitutions [Kes07]. In this work we will rely heavily on this decomposition of the $\beta$-reduction process, since the first correspondence we establish between a system for intuitionistic logic and functional programs is based on a variant of standard $\lambda$-calculi that can be found in the literature. However, this nice

correspondence can be defined for the usual $\lambda$-calculus as well, by using the standard way of recovering plain $\beta$-reduction from finer reduction systems. It is important to notice our work does not belong to this tradition, since we are not using any kind of cut, and computation will not be modeled as normalisation of proofs, or as some kind of transformation.

**Logic programming**. Among modern programming languages, another common way of establishing theoretical foundations for programs consists in using the logical formulas as a language and the process of proving a formula, seen as a program, as the computation associated to this *logic* program. This methodology is often referred to as the *proof-search-as-computation* paradigm, and one can use deductive systems such as the sequent calculus to provide a foundation for such logic programming languages [MNPS91], that can be extended without losing logical purity to advanced features such as modules [Mil89]. This work belongs to this tradition in the sense that we are interpreting formulas of intuitionistic logic as functional programs, and then defining computation as the process of building a proof, even if this is an incomplete proof, in some deductive system.

**Contributions**. This paper introduces an analysis of functional computation, described by means of a $\lambda$-calculus with explicit substitutions, called $\lambda$s and based on the following grammar:

$$t, u ::= x \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u]$$

in a setting of logic programming, where proof search models computation. In Section 2 we present a new proof system for intuitionistic logic in the calculus of structures, called JS, and prove that it is sound and complete with respect to the standard sequent calculus system. The goal is to encode all $\lambda$s-terms into formulas using the following scheme:

$$\begin{aligned} [\![x]\!] &= x \\ [\![\lambda x.t]\!] &= x \to [\![t]\!] \\ [\![t\ u]\!] &= ([\![u]\!] \to \top) \to [\![t]\!] \\ [\![t[x \leftarrow u]]\!] &= ([\![u]\!] \to x) \to [\![t]\!] \end{aligned}$$

and to fit the reduction rules of the calculus with respect to this encoding, we define a restriction of JS, called JLSd, which is sound and comes with a partial completeness result. Then, in Section 3 we establish a precise correspondence between proof search in JLSd and reduction in the rewrite system defining the operational behaviour of our calculus. Properties of $\lambda$s and the relation with properties of JLSd are discussed. Finally, Section 4 introduces another restriction based on annotations in the style of *focusing* [And92], called JLSn, shows that this system yields a correspondence with the standard $\lambda$-calculus with $\beta$-reduction, and proves properties of $\lambda$s with respect to the $\lambda$-calculus using this correspondence. The relationships between different proof systems and calculi is shown in Figure 1, where arrows indicate which systems simulate other systems — we will also use JLSb,

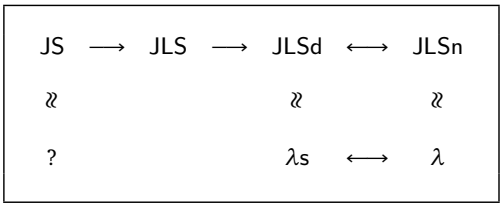


**Figure 1: Proof systems and calculi used in this paper**

a subset of the JLSd system which is not mentioned here. In the conclusion, we also discuss the possibility of extending this correspondence to the full JS system, which would require to move from the $\lambda$-calculus to a calculus with pattern-matching, such as the *pure pattern calculus* [JK09]. It is thus further work to fill the hole left in Figure 1 and interpret full JS.

**Related work**. There is not much work on intuitionistic logic in the setting of deep inference, and most of it is purely on the side of proof theory [Tiu06]. The only computational interpretation of an intuitionistic system in the calculus of structures [BM08] is based on the proofs-as-programs paradigm and cut-elimination. It also uses a different proof system relying on conjunction, and with no switch rule. We have a completely different approach, since we use the proof-search-as-computation paradigm. There exists however some work on the encoding of reduction in the $\lambda$-calculus with explicit substitutions into proof search [Rov11], also in the setting of deep inference, but we are using here plain intuitionistic logic rather than a variant of linear logic extended with non-commutativity and a renaming operator, and the result here is not restricted to the *linear* fragment of the $\lambda$-calculus. The methodology adopted to represent computation is also quite different in the two settings.

The point of this work is to establish a bridge between functional programming and the methodology of logic programming, which allows for a fine analysis of computation in terms of logic only. A goal would be to get new insights on the evaluation mechanisms of the $\lambda$-calculus through the use of proof-theoretical tools, and we can also hope for cross-fertilisation of the two fields if we can transpose results on one side to the other side.

# 2. INTUITIONISTIC LOGIC IN THE CALCULUS OF STRUCTURES

The proof systems we present here are all based on a common system JS, which can be seen as an intuitionistic variant of the KSg system [Brü03] for classical logic at the propositional level, using only the implication $\to$ as a connective. In this setting, any inference rule can be applied deep inside a formula, and a derivation is not a tree, but a sequence of rule instances.

We start with a set of *formulas*, denoted by capital latin letters such as $A$, $B$, $C$, and generated from a countable set of *atoms*, denoted by $a$, $b$, $c$ and so on, and the truth unit $\top$ by using the binary connective $\to$ for implication. Formally, we have:

$$L ::= a \mid \top \qquad A, B ::= L \mid A \to B$$

and we denote *literals* with letters such as $L$. Moreover, formulas are usually considered in the calculus of structures through a set of equations, so that *structures* are defined as equivalence classes of formulas generated by the following congruence rules:

$$\top \to A \equiv_u A \quad \text{and} \quad A \to (B \to C) \equiv_a B \to (A \to C)$$

Then, the set of inference rules for the JS system is shown in Figure 2. Being in the setting of deep inference means having the ability to apply inference rules inside some *context*, which is a structure with a hole $\{\ \}$, meant to be filled by a structure. To keep notations simple, we consider only *positive* contexts here, those where the hole is located on the left of an even number of implications, as follows:

$$\xi ::= \{\ \} \mid (\xi \to A) \to B$$

$$\mathsf{i}\,\frac{B}{(B\to A)\to A} \qquad \mathsf{s}\,\frac{((A\to B)\to C)\to D}{A\to((B\to C)\to D)}$$

$$\mathsf{w}\,\frac{B}{A\to B} \qquad \mathsf{c}\,\frac{A\to(A\to B)}{A\to B}$$

**Figure 2: Inference rules for system** JS

Contexts will be denoted as $\xi\{\ \}$ or $\zeta\{\ \}$, and for example $\xi\{A\}$ is the context $\xi$ where the hole is replaced by a structure or a context $A$. The inference rules in Figure 2 can only be applied in positive contexts, and any rule $\mathsf{r}$ can be instantiated in a context $\xi$ by placing its premise and conclusion inside:

$$\textit{(rule)} \qquad \mathsf{r}\,\frac{A}{B} \quad\longrightarrow\quad \mathsf{r}\,\frac{\xi\{A\}}{\xi\{B\}} \qquad \textit{(rule instance)}$$

We use standard notations for a *derivation* $\mathscr{D}$ from $A$ to $B$, and a *proof* $\mathscr{D}'$ of $B$, which is a derivation with $\top$ as premise:

$$\begin{array}{c} A \\ \mathscr{D}\,\Big\| \\ B \end{array} \qquad \text{and} \qquad \begin{array}{c} \\ \mathscr{D}'\,\Big\| \\ B \end{array}$$

To show that our system is suitable for intuitionistic logic, we prove soundness and completeness with respect to the sequent calculus system shown in Figure 3, that we call $\mathsf{LJ}\cup\{\mathsf{cut}\}$. This requires to translate sequents into structures:

$$[\![\vdash A]\!]_{\mathsf{S}} = A \quad\text{and}\quad [\![A,\Delta\vdash B]\!]_{\mathsf{S}} = A\to[\![\Delta\vdash B]\!]_{\mathsf{S}}$$

We use the short notation $\Gamma\to A$ to denote the translation of the sequent $\Gamma\vdash A$, which is sensible because the equation $\equiv_a$ allows to treat $A\to(B\to C)$ as the formula $(A\wedge B)\to C$.

Theorem 1 (Soundness of JS). *If a structure $A$ is provable in the system* JS*, then the sequent $\vdash A$ is provable in* $\mathsf{LJ}\cup\{\mathsf{cut}\}$.

Proof. The proof is standard, see for example [Brü03]. We proceed by induction on the given proof in JS. For each rule instance with premise $\xi\{A\}$ and conclusion $\xi\{B\}$ we can show by induction on the context that the sequent $\vdash\xi\{A\}\to\xi\{B\}$ is provable in $\mathsf{LJ}\cup\{\mathsf{cut}\}$. Then, we use a cut at each step. □

There is no equivalent of the cut rule in JS, and we thus prove completeness using the cut-free system LJ as a reference.

Theorem 2 (Completeness of JS). *If some sequent $\Gamma\vdash A$ is provable in* LJ*, then the structure $[\![\Gamma\vdash A]\!]_{\mathsf{S}}$ is provable in* JS.

Proof. By induction on a proof of $\Gamma\vdash A$ in LJ. Almost all cases are straightforward, the only interesting case is actually the left implication rule, shown below:

$$\to_{\mathsf{L}}\,\frac{\overset{\Pi_1}{\Gamma\vdash A} \qquad \overset{\Pi_2}{\Delta,B\vdash C}}{\Gamma,\Delta,A\to B\vdash C} \quad\longrightarrow\quad \equiv^{*}_{\mathsf{a}}\,\frac{\mathsf{s}^{*}\,\dfrac{\equiv_{\mathsf{u}}\,\dfrac{\begin{array}{c}\mathscr{D}_2\|\\ \Delta\to B\to C\end{array}}{\Delta\to(\top\to B)\to C}}{\Delta\to\Gamma\to(A\to B)\to C}}{\Gamma\to\Delta\to(A\to B)\to C}$$

where the proof $\mathscr{D}_2$ is obtained from $\Pi_2$ by induction hypothesis, and the derivation $\mathscr{D}_1$ by plugging the translation of $\Pi_1$ into the context $\Delta\to(\{\ \}\to B)\to C$, which is possible because of the deep inference methodology used here. □

$$\mathsf{ax}\,\frac{}{A\vdash A} \qquad \mathsf{cut}\,\frac{\Gamma\vdash A \quad \Delta,A\vdash B}{\Gamma,\Delta\vdash B} \qquad \top_{\mathsf{R}}\,\frac{}{\vdash\top}$$

$$\mathsf{weak}\,\frac{\Gamma\vdash B}{\Gamma,A\vdash B} \qquad \mathsf{cont}\,\frac{\Gamma,A,A\vdash B}{\Gamma,A\vdash B} \qquad \top_{\mathsf{L}}\,\frac{\Gamma\vdash A}{\Gamma,\top\vdash A}$$

$$\to_{\mathsf{R}}\,\frac{\Gamma,A\vdash B}{\Gamma\vdash A\to B} \qquad \to_{\mathsf{L}}\,\frac{\Gamma\vdash A \quad \Delta,B\vdash C}{\Gamma,\Delta,A\to B\vdash C}$$

**Figure 3: Inference rules for system** $\mathsf{LJ}\cup\{\mathsf{cut}\}$

Although it is a simple system for a fragment of intuitionistic logic, JS is already too general to represent the $\lambda$-calculus with precision in a proof search style. Indeed, no term corresponds to a formula with a compound formula in negative position, such as $(A\to(B\to C))\to D$, and even if we restrict formulas to images of the translation mentioned in the introduction — and formally defined in Section 3 — we still have problems with inference rules and equations that do not correspond to valid operations of our $\lambda$-calculus, as for example:

$$\begin{array}{rcl} \lambda x.t & \longrightarrow & \lambda x.\lambda x.t \\ \lambda x.\lambda y.t & \equiv_a & \lambda y.\lambda x.t \end{array}$$

Thus, we define a restriction of this system, called JLS, where formulas are limited to those where the subformulas in negative position, located on the left of an odd number of implications, can only be atoms, plus some restrictions on the use of the $\top$ unit, as shown in the following grammar:

$$\begin{array}{c} B ::= a \mid a\to B \mid \omega\to B \mid \delta\to B \\ \delta ::= B\to a \qquad \omega ::= B\to\top \end{array}$$

where $B$ defines the class of *restricted formulas*, that we denote as normal formulas, and $\delta$ defines the *blocks*, denoted by greek letters such as $\delta$, $\kappa$, $\mu$, which represent explicit substitutions in our encoding. Subformulas defined by $\omega$ are called *matchers* and denoted by $\omega$ or $\tau$, they represent the argument term in an application. We use contexts to restrict the congruence as well, keeping only the equation $\equiv_a$ on specific positive subformulas:

$$\xi\{\delta\to(\kappa\to A)\}\ \equiv_b\ \xi\{\kappa\to(\delta\to A)\}$$

Finally, the inference rules of the restricted system JLS are given in Figure 4. In this system, the switch rule and the rules of weakening and contraction can only be used to move, erase or duplicate blocks. Moreover, another part of the equation $\equiv_a$ is expressed in the rule $\mathsf{xr}$, which also can only exchange blocks. The identity rule is also restricted, through the condition that the premise cannot be the $\top$ unit, and it only applies on atoms. The new rule $\mathsf{isu}$ is a compound rule that embodies the use of an identity on $\top$, and corresponds to the following JS derivation:

$$\equiv_{\mathsf{u}}\,\frac{\mathsf{s}\,\dfrac{\mathsf{i}\,\dfrac{(B\to a)\to C}{((((B\to\top)\to\top)\to a)\to C)}}{(B\to\top)\to((\top\to a)\to C)}}{(B\to\top)\to(a\to C)}$$

This new inference rule corresponds, through the encoding of $\lambda\mathsf{s}$-terms, to the standard $\mathsf{B}$ rule which triggers a $\beta$-redex to turn an application into an explicit substitution, to be carried out:

$$(\lambda x.t)\,u \longrightarrow_{\mathsf{B}} t[x\leftarrow u]$$

$$\mathsf{ir}\ \frac{B}{(B \to a) \to a} \qquad \mathsf{isu}\ \frac{(B \to a) \to C}{(B \to \top) \to (a \to C)}$$

$$\mathsf{wr}\ \frac{A}{\delta \to A} \qquad \mathsf{cr}\ \frac{\delta \to (\delta \to A)}{\delta \to A}$$

$$\mathsf{xr}\ \frac{A \to (\delta \to B)}{\delta \to (A \to B)} \qquad \mathsf{sr}\ \frac{((\delta \to A) \to L) \to B}{\delta \to ((A \to L) \to B)}$$

**Figure 4: Inference rules for system** JLS

This system is clearly not going to be complete, since restrictions are so strong that for example, there is no way of writing a proof of $a \to (a \to \top) \to a$. However, this system can easily be shown sound with respect to its general version JS.

THEOREM 3 (SOUNDNESS OF JLS). *If there is a derivation from $A$ to $B$ in* JLS*, then there is a derivation from $A$ to $B$ in* JS*.*

PROOF. Any derivation from $A$ to $B$ in JLS can be immediately converted into a derivation in JS. Indeed, the rules ir, wr, cr and sr are restrictions of the rules of JS, the equation $\equiv_b$ is a restriction of $\equiv_a$ and the xr rule also corresponds to another use of this equation. Finally, any instance of the isu rule can be replaced with a derivation of JS, as it was shown above. □

It is interesting to notice that the identity on the unit $\top$ we use in the derivation corresponding to isu is not even needed in JS, since it will be shown admissible. To preserve the upper bound on the length of proofs during the process, it is useful to consider the system JS', a variant of JS where the usual switch is replaced with a compound rule called *super-switch* [Str03]:

$$\mathsf{ss}\ \frac{\xi\{\delta\} \to B}{\delta \to (\xi\{\top\} \to B)}$$

which is equivalent to a derivation of several switches, so that there are obvious translations between JS and JS'. We can use this alternative system to count a sequence of switches as only one rule instance in the length of a proof.

PROPOSITION 4. *If there is a proof of a structure $\xi\{A\}$ in* JS*, then there is a proof of $\xi\{(A \to \top) \to \top\}$ as well in* JS*, not using an identity on these $\top$ occurrences.*

PROOF. By induction on the length of the given proof $\mathscr{D}$ of $A$ in JS, translated into the JS' system. If $\mathscr{D}$ is of length 0, we replace $\top$ with $(\top \to \top) \to \top$ using $\equiv_u$. Then, in the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$. We can always rewrite the conclusion and use the induction hypothesis to rewrite the premise, but in the case of a switch moving some structure $E$ inside $A$, where $A$ is $(B \to C) \to D$, we must use an additional switch:

$$\mathsf{s}\ \frac{\xi\{((E \to B) \to C) \to D\}}{\xi\{E \to ((B \to C) \to D)\}}$$

$$\longrightarrow \quad \mathsf{s}\ \frac{\xi\{((((E \to B) \to C) \to D) \to \top) \to \top\}}{\xi\{((E \to ((B \to C) \to D)) \to \top) \to \top\}}\ \mathsf{s}\ \frac{}{\xi\{E \to ((((B \to C) \to D) \to \top) \to \top)\}}$$

| | |
|---|---|
| *apply* wr *on* $(B \to a) \to C$ | : *only if* $\lvert C\rvert_a = 0$ |
| *apply* crn *on* $(B \to a) \to C$ | : *only if* $\lvert C\rvert_a \geq 2$ |
| *apply* xr *on* $(B \to a) \to (C \to D)$ | : *only if* $\lvert C\rvert_a = 0$ |
| *apply* sr *on* $(B \to a) \to ((C \to L) \to D)$ | : *only if* $\lvert D\rvert_a = 0$ |

$\xi\{(C \to a) \to ((D \to b) \to E)\} \equiv_b \xi\{(D \to b) \to ((C \to a) \to E)\}$
*this equation holds only if we have* $\lvert D\rvert_a = 0$ *and* $\lvert C\rvert_b = 0$

**Figure 5: Restrictions used to define** JLSd **from** JLS

which can be rewritten into a super-switch ss instance. Note that in the case of the contraction c, we need to use the induction hypothesis twice, and this is possible because the transformation is length-preserving, thanks to the super-switch. □

Now, we will show that the JLS proof system is actually not so far from being complete with respect to the restricted fragment of intuitionistic logic. In order to do this, we consider a smaller fragment, by imposing more restrictions on structures: only one negative occurrence of each atom is allowed, and all its positive occurrences must appear *in the scope* of this negative occurrence. We use the notation $a \in B$ to denote that some atom $a$ appears in the structure $B$, and $a \in \xi$ if $a$ appears in the context $\xi\{\ \}$.

DEFINITION 5. *The* (positive) multiplicity *of an atom $a$ in some structure $B$, denoted by $\lvert B\rvert_a^+$, is the number of occurrences of $a$ in positive position in $B$. Its* negative multiplicity*, denoted by $\lvert B\rvert_a^-$, is its number of occurrences in negative position in $B$.*

The formulas of the more restricted class defined here are called *functional structures* because they will be the ones corresponding to $\lambda$s-terms in the rest of the paper.

DEFINITION 6. *A restricted structure $B$ is said to be* functional *if for any $a \in B$, there is a context $\xi\{\ \}$ and structures $C$ and $D$ such that we have either $B \equiv_b \xi\{a \to C\}$ or $B \equiv_b \xi\{(D \to a) \to C\}$, with $a \notin \xi$, $a \notin D$, $\lvert C\rvert_a^+ \geq 0$ and $\lvert C\rvert_a^- = 0$.*

Then, we can observe that functional structures are not stable under application of inference rules of JLS, in particular under contraction. Therefore we need to tweak our inference rules.

To be able to use contraction on such structures, we consider the following variation of the cr rule:

$$\mathsf{crn}\ \frac{(B \to d) \to ((B \to a) \to C_{[d/a]})}{(B \to a) \to C}$$

where $C_{[d/a]}$ denotes the same $C$ with exactly one occurrence of the atom $a$ replaced by an occurrence of a fresh atom $d$. This rule can be shown sound by a induction on proofs. Finally, we define the proof system JLSd as $\{\mathsf{ir}, \mathsf{isu}, \mathsf{wr}, \mathsf{crn}, \mathsf{xr}, \mathsf{sr}\}$, with extra conditions on the conclusion of rules and on the congruence, summarised in Figure 5. These conditions ensure that functional structures are stable under application of rules of JLSd, as well as its congruence.

These new restrictions correspond, on the side of our $\lambda$-calculus, to the idea that we want to manipulate terms up to renaming of variables, so that no variable name is bound twice, and we can use implicitly $\alpha$-conversion to change names.

The JLSd system is sound with respect to intuitionistic logic, since we only added restrictions on inference rules of JLS and we observed that the variant rule crn was sound too, and we will now study its completeness. As we already noticed, this system is not complete — the unit $\top$ cannot appear as the premise of a derivation, so that there is no proof *per se* in this system — but we can establish a partial completeness result. We do that in three steps: first we use a subset of the JLSd system, then we deal with some particular weakenings forbidden in JLSd, and finally we build the proof premise $\top$ from a simple formula, by dealing again with weakenings. The goal is to show that if some $A$ is provable in JS, we can build a proof of the shape:

$$\begin{array}{r} \top \\ \{\mathsf{wm},\mathsf{iaw}\}\Big\| \\ B \\ \mathsf{JLSb}\Big\| \\ A \end{array} \tag{1}$$

**A terminating subset of JLSd.** We consider the system JLSb, which is defined as JLSd without the isu rule, for which we show that proof search is terminating. To do that, we need a measure on functional structures that will decrease during proof search, and the first part of this measure can be defined in a simple way.

Definition 7. *Given any functional structure $A$, we can define as follows its* block-complexity*, denoted by $\mathscr{C}(A)$ and its* net size*, denoted by $\mathscr{N}(A)$, using the following induction:*

$$\begin{array}{rcl} \mathscr{C}(a) & = & 0 \\ \mathscr{C}(a \to B) & = & \mathscr{C}(B) \\ \mathscr{C}((C \to \top) \to B) & = & \mathscr{C}(C) + \mathscr{C}(B) \\ \mathscr{C}((C \to a) \to B) & = & \mathscr{C}(C) + \mathscr{C}(B) + \mathscr{N}(B) \end{array}$$

$$\begin{array}{rcl} \mathscr{N}(a) & = & 1 \\ \mathscr{N}(a \to B) & = & 1 + \mathscr{N}(B) \\ \mathscr{N}((C \to \top) \to B) & = & \mathscr{N}(C) + \mathscr{N}(B) \\ \mathscr{N}((C \to a) \to B) & = & \mathscr{N}(B) \end{array}$$

Remark 8. *The block-complexity is invariant under congruence, namely the equation $\equiv_b$, because the blocks that can be exchanged this way are exactly the substructures that are not counted in the size of a given structure.*

This complexity measure is simply the sum, for each block $\delta$, of the size of the structure in the scope of $\delta$. We can use this to show that the process of building a derivation by applications of inference rules of JLSb terminates — we call this *proof search* although we are building derivations and not proofs.

Lemma 9. *Proof search in* JLSb *is terminating.*

Proof. Given some functional structure $A$, if we apply any inference rule of JLSb on $A$ other than the contraction crn rule, we obtain a functional structure $B$ such that $\mathscr{C}(B) < \mathscr{C}(A)$, as can be checked for the rules ir, wr, xr and sr.
In the case of crn, we can observe that one positive occurrence of an atom has been replaced with an occurrence of some fresh atom. We define for any functional structure $F$ a measure $M(F)$ as the multiset of $|F|_d^+$ for all $d \in F$, under multiset ordering, and with crn we have $M(B) < M(A)$ since $|B|_e^+ < |A|_e^+$ for some $e \in A$, and the introduced atom has a multiplicity of 1 in $B$.
Finally, we can use an induction on the pair $(M(A), \mathscr{C}(A))$, under lexicographic order, and we reach the case of a structure $G$ such that $(M(A), \mathscr{C}(A)) = (0,0)$, which means that there is no block $\delta$ in $G$. This implies that no rule of JLSb can be applied on $G$. □

Moreover, we can prove that the rules JLSb are invertible in JS, so that proof search preserves provability in JS. This means we can use JLSb on a structure $A$ to produce by proof search a $B$ such that $A$ is provable in JS if and only if $B$ is provable in JS.

Lemma 10. *If there is a proof in* JS *of a functional structure $\xi\{(B \to a) \to a\}$, then there is a proof in* JS *for $\xi\{B\}$.*

Proof. By induction on the length of a given proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to a\}$, we build a proof of at most the same length for $\xi\{B\}$. If $\mathscr{D}$ has length 2, it uses identities on $((b \to b) \to a) \to a$, and we use the identity on $b \to b$ only. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $(B \to a) \to a$, we can rewrite the conclusion into $\xi\{B\}$, and use the induction hypothesis to rewrite the premise accordingly.
2. If r only affects a structure inside this occurrence of $B$, we rewrite the conclusion into $\xi\{B\}$, use the same rule inside a smaller context, and then the induction hypothesis.
3. If r is a contraction c on this occurrence of $B \to a$, we use the induction hypothesis on only one copy of $B \to a$.
4. If r is a switch s moving a structure $D$ on the left of $B \to a$, we can remove it and go on by induction hypothesis:
$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to a\}}{\xi\{E \to ((B \to a) \to a)\}} \quad \longrightarrow \quad \xi\{E \to B\}$$

Notice that there can be no weakening on this $B \to a$ since there would be no $a$ left in negative position to complete the proof. □

Lemma 11. *If there is a proof in* JS *of some functional structure $\xi\{(B \to a) \to C\}$, and $|C|_a = 0$, then there is a proof in* JS *for the structure $\xi\{C\}$.*

Proof. By induction on the length of a given proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to C\}$, we build a proof of at most the same length for $\xi\{C\}$. If $\mathscr{D}$ has length 2, it uses a weakening and an identity on $(B \to a) \to (c \to c)$, and we use the identity on $c \to c$ only. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to a$, we rewrite the conclusion into $\xi\{C\}$ and use the induction hypothesis to rewrite the premise accordingly.
2. If r only affects a structure inside this occurrence of $B$, we rewrite the conclusion into $\xi\{C\}$, use the same rule inside a smaller context, and then the induction hypothesis.
3. If r is a weakening w on this occurrence of $B \to a$, we remove it in the conclusion and the result is immediate.
4. If r is a contraction c on this occurrence of $B \to a$, we can rewrite the conclusion into $\xi\{C\}$ and then use twice the induction hypothesis, which is possible since the first proof obtained has at most the same length as the original.
5. If r is a switch s moving a structure $E$ on the left of $B \to a$, we replace it by a weakening, as shown below:
$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to C\}}{\xi\{E \to ((B \to a) \to C)\}} \quad \longrightarrow \quad \mathsf{w}\,\frac{\xi\{C\}}{\xi\{E \to C\}}$$
and we can go on by induction hypothesis. □

LEMMA 12. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to a) \to ((C \to L) \to D)\}$*, and* $|D|_a = 0$*, then there is a proof in* JS *for the structure* $\xi\{((((B \to a) \to C) \to L) \to D)\}$.

PROOF. By induction on the length of a given proof $\mathscr{D}$ in JS of $\xi\{(B \to a) \to ((C \to L) \to D)\}$ translated to JS', we build a proof of at most the same length for $\xi\{((((B \to a) \to C) \to L) \to D)\}$. If $\mathscr{D}$ has length 3, it uses on $((c \to c) \to a) \to ((a \to b) \to b)$ three identities and we do the same. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to a$, we rewrite the conclusion into $\xi\{(((((B \to a) \to C) \to L) \to D)\}$ and use the induction hypothesis to rewrite the premise.
2. If r only affects a structure inside this occurrence of $B$, we rewrite the conclusion again, use the same rule inside a smaller context, and then use the induction hypothesis.
3. If r is a weakening w on this occurrence of $B \to a$, we rewrite the conclusion again and use a weakening.
4. If r is a contraction c on this occurrence of $B \to a$, we rewrite the conclusion again and use twice the induction hypothesis, which is possible since the proof obtained the first time has at most the same length as the original.
5. If r is a switch s moving a structure $E$ on the left of $B \to a$, we replace it by a super-switch, as shown below:

$$\mathsf{s}\,\frac{\xi\{((E \to B) \to a) \to ((C \to L) \to D)\}}{\xi\{E \to ((B \to a) \to ((C \to L) \to D))\}} \quad\longrightarrow\quad \mathsf{ss}\,\frac{\xi\{(((((E \to B) \to a) \to C) \to L) \to D\}}{\xi\{E \to ((((B \to a) \to C) \to L) \to D)\}}$$

   and then we go on by induction hypothesis. The case of a super-switch is treated exactly the same way.

In the end, we have a proof in JS' that we can turn into a proof of the same structure in JS by expanding super-switches. □

LEMMA 13. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to a) \to C\}$*, and* $|C|_a \geq 2$*, then there is a proof in* JS *for the structure* $\xi\{(B \to a) \to ((B \to a) \to C)\}$.

PROOF. Such a proof can be obtained as follows:

$$\mathsf{w}\,\frac{\overset{\mathscr{D}}{\|}\;\;\xi\{(B \to a) \to C\}}{\xi\{(B \to a) \to ((B \to a) \to C)\}}$$

where $\mathscr{D}$ is the given proof of $\xi\{(B \to a) \to C\}$ in JS. □

LEMMA 14. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to a) \to (C \to D)\}$*, and* $|C|_a = 0$*, then there is a proof in* JS *for the structure* $\xi\{C \to ((B \to a) \to D)\}$.

PROOF. This corresponds to the use of the congruence in the JS system, namely the equation $\equiv_a$, so that this is immediate. □

We can use all these lemmas to prove that proof search in JLSb has the interesting property of preserving provability, which will be a crucial argument in our partial completeness result.

LEMMA 15. *For any structures A and B, if there is a derivation from A to B in the* JLSb *system and B is provable in* JS*, then A is provable in* JS.

PROOF. By induction on the length of the given derivation $\mathscr{D}$ from $A$ to $B$ in JLSb. If $\mathscr{D}$ has length 0, the result is immediate since $A$ is $B$. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$, to prove that if there is a proof of its conclusion $A$ in JS, there is a proof of its premise, by either Lemma 10, Lemma 11, Lemma 13, Lemma 14 or Lemma 12. □

**Weakening matchers.** We consider the following rule, which corresponds to a case of weakening forbidden in JLS:

$$\mathsf{wm}\,\frac{A}{(B \to \top) \to A}$$

Then, we show that this inference rule is complete, in the sense that using it during a proof search cannot turn some provable structure into an unprovable one — it is an *invertible* rule.

LEMMA 16. *If there is a proof in* JS *of some functional structure* $\xi\{(B \to \top) \to A\}$*, then there is a proof of* $\xi\{A\}$ *in* JS.

PROOF. By induction on the given proof $\mathscr{D}$ in JS, we build a proof of $\xi\{A\}$, preserving the upper bound on the length of the proof. If $\mathscr{D}$ has length 1, it uses a weakening on $(B \to \top) \to \top$, and the result is immediate. In the general case, we use a case analysis on the bottommost rule instance r in $\mathscr{D}$:

1. If r does not affect this occurrence of $B \to \top$, we remove it from the conclusion and use the induction hypothesis to rewrite the premise accordingly.
2. If r only affects a structure inside this occurrence of $B$, we remove this instance and use the induction hypothesis.
3. If r is a weakening w on this occurrence of $B \to \top$, the result is immediate.
4. If r is a contraction c on this occurrence of $B \to \top$, we can rewrite the conclusion and we use twice the induction hypothesis, which is possible since the proof obtained the first time has at most the same length as the original.
5. If r is a switch s moving a structure $C$ on the left of $B$, we replace it by a weakening, as shown below, and go on by induction hypothesis:

$$\mathsf{s}\,\frac{\xi\{((C \to B) \to \top) \to A\}}{\xi\{C \to ((B \to \top) \to A)\}} \quad\longrightarrow\quad \mathsf{w}\,\frac{\xi\{A\}}{\xi\{C \to A\}}$$

In this analysis, there is no need to consider the case where r is an instance of the identity i used on $(B \to \top) \to \top$, since we can always remove it from any proof in JS by using Proposition 4. □

**Closing the proof.** We consider a functional structure where no block and no matcher appears, and observe that it is of the shape $b_1 \to \cdots \to b_n \to a$. Such a structure is provable if and only if there is an $i$ such that $b_i$ is $a$, and we can use the following inference rule, not to be applied inside a context:

$$\mathsf{iaw}\,\frac{\top}{b_1 \to \cdots \to a \to \cdots \to b_n \to a}$$

which can be applied on a structure of this shape if and only if it is provable, since it is equivalent to many weakenings and one identity instance. Now, we can glue the pieces together to produce a proof of weak completeness, which states that given a functional structure $A$ provable in JS, although there is no proof of $A$ in JLSb nor in JLSd, there is a derivation from a structure $B$ to $A$ in JLSb, such that $B$ can easily be mechanically checked.

THEOREM 17 (WEAK COMPLETENESS OF JLSb). *For any given functional structure $A$, if there is a proof of $A$ in* JS*, then there is a structure $B$ such that there is a derivation from $B$ to $A$ in* JLSb *and a proof of $B$ in* {wm, iaw}.

PROOF. As a first step, we apply Lemma 9 to produce by proof search a derivation $\mathscr{D}_1$ from some structure $A_1$ to $A$ in the JLSb system. Notice that there can be no block — that is, structures of the shape $(B \to c)$ in negative position — in the structure $A_1$ since that would imply that at least one inference rule of JLSb could be applied. Moreover, by Lemma 15 we know that if $A$ is provable in JS, then $A_1$ is provable in JS too. Then, we apply as much as possible the wm rule on $A_1$ to produce a derivation $\mathscr{D}_2$ from a structure $A_2$ with no matchers — that is, structures of the shape $(B \to \top)$ in negative position — to $A_1$. By Lemma 16 we know that if $A_1$ is provable in JS, then $A_2$ is provable in JS too. Finally, if $A_2$ is provable in JS and does not contain matchers, then we can use one instance of iaw to build a proof of the shape described in (1), where the expected derivation from $B$ to $A$ in JLSb is $\mathscr{D}_1$, since this $A_1$ is provable in {wm, iaw}. □

COROLLARY 18 (WEAK COMPLETENESS OF JLSd). *For any given functional structure $A$, if there is a proof of $A$ in* JS*, then there is a structure $B$ such that there is a derivation from $B$ to $A$ in* JLSd *and a proof of $B$ in* {wm, iaw}.

This result tells us that JLSb is indeed a sensible system to deal with functional structures, and therefore JLSd can also be used. The problem of proving functional structures in the JLSd system is more subtle that in JLSb, since it allows the use of the isu rule which is not invertible, but we can avoid using the wm rule to get a complete proof in some cases. This is not always possible, as shown by the following example proof:

$$\mathsf{wm}\,\frac{\mathsf{iaw}\,\dfrac{\top}{a \to a}}{(B \to \top) \to (a \to a)}$$

In this case, we cannot use the isu rule if the structure $B$ is not provable in JS, while the complete structure is provable in JS. The JLSd system allows to use the minimal amount of instances of wm to get a proof in JS, but there is no way in general to build a proof from JLSd and the iaw rule only. The class of structures that can be proved using JLSd and iaw only is more behaved than general functional structures, since the structure $B$ in a structure of the shape $(B \to \top) \to C$ is not logically *relevant*.

# 3. PROOF SEARCH AS REDUCTION WITH EXPLICIT SUBSTITUTIONS

We now consider a $\lambda$-calculus with explicit substitutions [Kes07], that we will call $\lambda$s, and show how its terms can be encoded into logical structures, so that the process of building a derivation in JLSd will simulate the process of applying reduction rules in the $\lambda$s-calculus. We say *proof search*, although we are not building complete proofs but rather open derivations, to emphasize the relation of this work with the usual *proof-search-as-computation* paradigm [MNPS91].

It is a direct approach, based on incomplete derivations of the proof system, in the sense that if two given structures $A$ and $B$ represent programs such that $A$ can be reduced to $B$ through the operational semantics of the language, there is a derivation from $B$ to $A$ rather than a proof of $A \to B$, as we will see.

$$
\begin{array}{rcll}
(\lambda x.t)\,u & \longrightarrow_{\mathtt{B}} & t[x \leftarrow u] & \\
\\
x[x \leftarrow u] & \longrightarrow_{\mathtt{var}} & u & \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{rm}} & t & \text{if } |t|_x = 0 \\
t[x \leftarrow u] & \longrightarrow_{\mathtt{dup}} & t_{[y/x]}[x \leftarrow u][y \leftarrow u] & \text{if } |t|_x \geq 2 \\
\\
(\lambda y.t)[x \leftarrow u] & \longrightarrow_{\mathtt{lam}} & \lambda y.t[x \leftarrow u] & \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apl}} & t[x \leftarrow u]\ v & \text{if } |v|_x = 0 \\
(t\ v)[x \leftarrow u] & \longrightarrow_{\mathtt{apr}} & t\ v[x \leftarrow u] & \text{if } |t|_x = 0 \\
\\
t[y \leftarrow v][x \leftarrow u] & \longrightarrow_{\mathtt{cmp}} & t[y \leftarrow v[x \leftarrow u]] & \text{if } |t|_x = 0 \\
\hline
t[y \leftarrow v][x \leftarrow u] \equiv t[x \leftarrow u][y \leftarrow v] & & & \text{if } |v|_x = |u|_y = 0
\end{array}
$$

**Figure 6: Reduction rules and equation for the $\lambda$s-calculus**

Our $\lambda$-calculus is very similar to many other calculi with explicit substitutions in the literature, and in particular it borrows its handling of duplication using a linear renaming operation from the *structural $\lambda$-calculus* [AK10]. The syntax of $\lambda$s-terms can be defined by the following grammar:

$$t, u ::= x \mid \lambda x.t \mid t\ u \mid t[x \leftarrow u]$$

where the object $[x \leftarrow u]$, which is called an *explicit substitution*, is a binder for the variable $x$, so that $x$ is bound in $t[x \leftarrow u]$. Moreover, terms are considered *modulo* $\alpha$-conversion, so that a variable is always bound at most once in any $\lambda$s-term. We will need to count the use of variables in a term, using the following definition.

DEFINITION 19. *The* multiplicity *of a variable $x$ in a term $t$, denoted by $|t|_x$, is the number of occurrences of the variable $x$ in the term $t$, not including the use of the name $x$ in a binder.*

The reduction rules defining the operational behaviour of the $\lambda$s-calculus are shown in Figure 6, where the construction $t_{[y/x]}$ denotes the term $t$ where exactly one occurrence of $x$ has been replaced with $y$. Notice that in the dup rule, the new variable $y$ must of course be fresh to avoid capture by some binder. This system of reduction rules is standard, and is similar to the one of the $\lambda$es-calculus [Kes07], except in the handling of duplications of substitutions.

We can now define an encoding of $\lambda$s-terms into structures. For that, we need to consider a bijection between logical atoms and variables in the calculus, so that to any variable $x$ corresponds an atom also denoted by $x$.

DEFINITION 20 (ENCODING OF $\lambda$s). *The encoding $[\![\cdot]\!]_\lambda$ from $\lambda$s-terms into structures of the* JLSd *system is defined as follows:*

$$
\begin{array}{rcl}
[\![x]\!]_\lambda & = & x \\
[\![\lambda x.t]\!]_\lambda & = & x \to [\![t]\!]_\lambda \\
[\![t\ u]\!]_\lambda & = & ([\![u]\!]_\lambda \to \top) \to [\![t]\!]_\lambda \\
[\![t[x \leftarrow u]]\!]_\lambda & = & ([\![u]\!]_\lambda \to x) \to [\![t]\!]_\lambda
\end{array}
$$

Notice that through this encoding, $\lambda$s-terms correspond only to functional structures as they were defined for use with the JLSd system — this is of course the reason why we restricted the rules of JS this way. Moreover, the equation on $\lambda$s-terms allowing to exchange unrelated explicit substitutions exactly corresponds to the equation $\equiv_b$ used on functional structures.

REMARK 21. *It is easy to see that the encoding $[\![\cdot]\!]_\lambda$ defines a bijection between λs-terms and functional structures. Indeed, each shape of structure defined in the grammar for restricted structures corresponds to exactly one construction in the λ-terms syntax, and the extra restrictions for functional structures correspond to the writing of a λs-term with a correct scope structure α-converted to avoid repetition of bindings on the same name.*

We can now state the theorem establishing the correspondence, at the computational level, between the operational behaviour of the λs-calculus and the behaviour of proof search in the JLSd system, where the $\longrightarrow_{\mathtt{R}}$ relation is the reduction defined by the rules of Figure 6, $\longrightarrow_{\mathtt{R}}^{*}$ is its reflexive and transitive closure, and $\longrightarrow_{\mathtt{S}}$ is the same as the first, where the B rule is not used.

THEOREM 22 (COMPUTATIONAL ADEQUACY OF JLSd). *For any given λs-terms $t$ and $u$, there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in the JLSd system if and only if $t \longrightarrow_{\mathtt{R}}^{*} u$.*

PROOF. By induction on the reduction steps from $t$ to $u$. If the reduction path is empty, $t$ and $u$ are the same and the result is trivial because the encoding $[\![\cdot]\!]_\lambda$ is uniquely defined. In the general case, we consider the first step in the reduction, and use the induction hypothesis on the rest of the reduction. We thus simply have to check that there is an inference rule instance in JLSd with premise $[\![u]\!]_\lambda$ and conclusion $[\![t]\!]_\lambda$ if and only if we have $t \longrightarrow_{\mathtt{R}} u$. This is immediately done by observing that each reduction rule corresponds exactly to one case of application of an inference rule of JLSd:

$$\mathsf{isu}\,\frac{(B \to a) \to C}{(B \to \top) \to (a \to C)} \quad\longleftrightarrow\quad \mathtt{B}\,\frac{t[x \leftarrow u]}{(\lambda x.t)\,u}$$

$$\mathsf{ir}\,\frac{B}{(B \to a) \to a} \quad\longleftrightarrow\quad \mathtt{var}\,\frac{u}{x[x \leftarrow u]}$$

$$\mathsf{wr}\,\frac{C}{(B \to a) \to C} \quad\longleftrightarrow\quad \mathtt{rm}\,\frac{t}{t[x \leftarrow u]}$$

$$\mathsf{cr}\,\frac{(B \to d) \to ((B \to a) \to C_{[d/a]})}{(B \to a) \to C} \quad\longleftrightarrow\quad \mathtt{dup}\,\frac{t_{[y/x]}[x \leftarrow u][y \leftarrow u]}{t[x \leftarrow u]}$$

$$\mathsf{xr}\,\frac{c \to ((B \to a) \to D)}{(B \to a) \to (c \to D)} \quad\longleftrightarrow\quad \mathtt{lam}\,\frac{\lambda y.t[x \leftarrow u]}{(\lambda y.t)[x \leftarrow u]}$$

$$\mathsf{xr}\,\frac{(C \to \top) \to ((B \to a) \to D)}{(B \to a) \to ((C \to \top) \to D)} \quad\longleftrightarrow\quad \mathtt{apl}\,\frac{t[x \leftarrow u]\,v}{(t\,v)[x \leftarrow u]}$$

$$\mathsf{sr}\,\frac{(((B \to a) \to C) \to \top) \to D}{(B \to a) \to ((C \to \top) \to D)} \quad\longleftrightarrow\quad \mathtt{apr}\,\frac{t\,v[x \leftarrow u]}{(t\,v)[x \leftarrow u]}$$

$$\mathsf{sr}\,\frac{(((B \to a) \to C) \to e) \to D}{(B \to a) \to ((C \to e) \to D)} \quad\longleftrightarrow\quad \mathtt{cmp}\,\frac{t[y \leftarrow v[x \leftarrow u]]}{t[y \leftarrow v][x \leftarrow u]}$$

Notice that the conditions on the multiplicity of variables in the reduction rules for λs-terms exactly match the restrictions that were imposed on inference rules to define JLSd from JLS. □

This result establishes a tight connection between our restricted intuitionistic system and the λs-calculus. The interesting point is then that a theorem that we prove on derivations of the JLSd system on the logical side also holds in its *computational* form on reduction paths in the λs-calculus. As an example, we can prove that the subsystem $\longrightarrow_{\mathtt{S}}$ terminates.

THEOREM 23. *The reduction subsystem $\longrightarrow_{\mathtt{S}}$ terminates.*

PROOF. This is a direct corollary of Lemma 9, considering derivations of JLSb as reduction paths in the $\longrightarrow_{\mathtt{S}}$ subsystem through the correspondence defined by Theorem 22. □

The other way around, if we have some result on reduction in the λs-calculus, then we can directly transpose this result to the logical side. For example, the Church-Rosser property states that the $\longrightarrow_{\mathtt{R}}$ rewriting system is confluent, and we could prove it using the standard method of parallel reductions from Tait and Martin-Löf, see for example [Kes07] — to prove a similar result on derivations of the JLSd system. We would thus obtain a proof of the following proposition.

PROPOSITION 24. *For any structures A, B and C, if there are derivations from B to A and from C to A in JLSd, then there is a structure D such that there are derivations from D to B and from D to C in JLSd.*

Moreover, we observed that the class of structures which can be proven by proof search in JLSd without using wm is more interesting than plain functional structures, since this rule does not correspond to a valid rewriting on λs-terms. Also notice that the conclusion of the iaw rule, where no structure of the shape $B \to \top$ appears, is exactly a λs-term in normal form. From this we can derive a characterisation of *weakly normalising* terms.

THEOREM 25. *A λs-term $t$ is weakly normalising if and only if there is a proof of $[\![t]\!]_\lambda$ in the JLSd ∪ {iaw} system.*

PROOF. First, if the term $t$ is weakly normalising, then there is a $u$ in normal form such that $t \longrightarrow_{\mathtt{R}}^{*} u$, and by Theorem 22 we have a derivation $\mathscr{D}$ from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in JLSd. We can thus use the iaw rule on $[\![u]\!]_\lambda$ to produce a proof of $[\![t]\!]_\lambda$. Then, if there is a proof of $[\![t]\!]_\lambda$ in JLSd ∪ {iaw}, we have such a derivation $\mathscr{D}$ and we can use Theorem 22 the other way around to get a term $u$ in normal form such that $t \longrightarrow_{\mathtt{R}}^{*} u$. □

It would therefore be interesting to learn more about this class of structures, inside the logic, to get insights on weakly normalising terms in the λs-calculus. Furthermore, an important class is the one of *strongly normalising* terms, for which any reduction path reaches a normal form, and which are often characterised using type systems [LLD+04]. It is not clear whether this class can be characterised through a particular variant of the JLSd system. An interesting problem would then be in defining an efficient procedure to decide whether a given structure is provable in this system, to check if some λs-term is strongly normalising without computing its typing derivation. In particular, this means that we could ensure termination of a program without knowing its type: if this can be done efficiently using an algorithm such as a variant of resolution for a fragment of intuitionistic logic, this is interesting, but it does not ensure that the composition of two well-behaved programs is well-behaved. It is thus unclear what kind of mechanism could take the role of typing in this setting.

The tight correspondence established by Theorem 22 allows for direct reasoning on reduction paths in the λs-calculus, where each step can be handled separately. In particular, if we have a trace of computation corresponding to the reduction of a term $t$ to some term $u$ and another trace for the reduction from $u$ to some term $v$, composing these traces is immediate and provides a trace of computation from $t$ to $v$. Contexts can also be handled in a very natural way, as it is done in the calculus of structures.

$$\mathsf{ir}\ \frac{B}{(B \to a) \to \Downarrow a} \qquad \mathsf{isu}\ \frac{(B \to a) \to \Downarrow C}{(B \to \top) \to (a \to C)}$$

$$\mathsf{wr}\ \frac{A}{\delta \to \Downarrow A} \qquad \mathsf{cr}\ \frac{\delta \to \Downarrow (\delta \to \Downarrow A)}{\delta \to \Downarrow A}$$

$$\mathsf{xr}\ \frac{A \to (\delta \to \Downarrow B)}{\delta \to \Downarrow (A \to B)} \qquad \mathsf{sr}\ \frac{((\delta \to \Downarrow A) \to L) \to B}{\delta \to \Downarrow ((A \to L) \to B)}$$

**Figure 7: Inference rules for system** JLSn

Moreover, permutations of rule instances, and transformations on derivations in the JLSd system allow for a reorganisation of a reduction path between two given $\lambda$s-terms. There are strong decomposition results in the calculus of structures [GS10], and results of this kind can be interesting when studying reduction strategies of the $\lambda$s-calculus. In the following section, we will present a restriction of JLSd yielding such a reduction strategy.

# 4. FOCUSED PROOF SEARCH AND REDUCTION STRATEGIES

Following our methodology, there is a direct connection between the cases where an inference rule can be applied on a structure and the reduction strategy implemented by the reduction system chosen for $\lambda$s-terms. Indeed, we could use a variant of JLSd not using extra conditions on the multiplicity of atoms in the rules, but this would induce highly non-deterministic reduction rules for the $\lambda$s-calculus. We made the reduction system deterministic by imposing a strategy in the sense that one reduction rule can be applied in only one way, but there is still a non-deterministic dimension in the choice of which redex to pick and reduce, given a $\lambda$s-terms with several redexes. We now address the question of this choice, which is usually called choosing a *reduction strategy*.

The JLSd system can be restricted further by using annotations on structures in the spirit of focusing [And92], and this can be interpreted on the computational side as restricting the choice of the next redex to be rewritten. This restriction is similar to a standard formulation of *focusing* for intuitionistic logic [LM07], although the deep inference setting used here does not allow the exact same treatment. It should be noted that focusing in the calculus of structures cannot be immediately defined through a translation of usual focusing in the sequent calculus, although there is some work on the topic [Gue10]. For example, there is no separation between different branches, and we thus need to duplicate focusing annotations, with copies being moved to different parts of the structure, corresponding to branches.

The inference rules for the JLSn system are shown in Figure 7, where the syntax of structures is extended with annotations, so that $\xi\{\Downarrow A\}$ is a valid structure for any $\xi\{\ \}$ and $A$ — annotations are only allowed in positive position. Then, the *decision* action is embedded inside the isu rule, which picks a structure on the right of a block of the shape $B \to a$ and forces to move this block inside this structure until its contents, in $B$, are released by the ir rule. This sequence of rule applications guided by annotations is called a focused *phase*, and we are mainly interested in the structures at the borders of such a phase.

Definition 26. *A functional structure $B$ is said to be* basic *if all the structures in negative position inside $B$ are either atoms or matchers — it contains no block, as $C \to a$ in negative position.*

The point of the JLSn system is then to handle basic structures by choosing a redex for the isu rule, to introduce a block $B \to a$ through this rule, and then maximally use the JLSb subsystem to produce a new basic structure, by removing all the remaining blocks. In particular, we add the restriction that the rule isu is not used on a structure that already contains a focus annotation. It is easy to see that any basic formula corresponds through the encoding $[\![\cdot]\!]_\lambda$ to some pure $\lambda$-term, which is a $\lambda$s-term without explicit substitutions. We can therefore consider the standard rule for $\beta$-reduction in the pure $\lambda$-calculus:

$$(\lambda x.t)\, u \longrightarrow_\beta t\{u/x\}$$

and show that there is a computational adequacy result between JLSn and this simple rewriting system, through the same $[\![\cdot]\!]_\lambda$ encoding as before. As a first step, we need a lemma explaining the effect of a focusing phase on a structure. In the following, we denote by $\xi\{A\}^+$ a context with several holes filled with the structure $A$, and by $\xi\{A\}^*$ a context with zero or more holes filled with $A$. Moreover, the notation $\xi\{a\}^*$ implicitly means there is no occurrence of $a$ in $\xi\{\ \}^*$ except in its holes.

Lemma 27. *Given any functional structure without annotations, of the shape $\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}$, there is a derivation from $\xi\{\zeta\{B\}^*\}$ to this structure in* JLSn.

Proof. Given a functional structure without annotations, of the shape $\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}$ we prove that there is a derivation $\mathscr{D}$ in JLSn such that we have the following situation:

$$\mathsf{isu}\ \frac{\begin{array}{c}\xi\{\zeta\{B\}^*\}\\ \mathscr{D}\,\|\\ \xi\{(B \to a) \to \Downarrow \zeta\{a\}^*\}\end{array}}{\xi\{(B \to \top) \to (a \to \zeta\{a\}^*)\}}$$

We proceed by induction on the size of the context $\zeta\{\ \}^*$ to prove that there is such a $\mathscr{D}$, using a case analysis on its toplevel shape:

1. If $\zeta\{\ \}^*$ is $\{\ \}$, then we can use an identity rule, as shown below, and we are done since we have our derivation $\mathscr{D}$.
$$\mathsf{ir}\ \frac{\xi\{B\}}{\xi\{(B \to a) \to \Downarrow a\}}$$
2. If $\zeta\{\ \}^*$ is $C$, where the atom $a$ does not appear in $C$, then we can use a weakening rule, as shown below, and we are also done with the induction:
$$\mathsf{wr}\ \frac{\xi\{C\}}{\xi\{(B \to a) \to \Downarrow C\}}$$
3. If $\zeta\{\ \}^*$ is $(C \to L) \to \theta\{\ \}^+$, then we can use an exchange rule, since the atom $a$ does not appear in $C$, and then go on by induction hypothesis:
$$\mathsf{xr}\ \frac{\xi\{(C \to L) \to ((B \to a) \to \Downarrow \theta\{a\}^+)\}}{\xi\{(B \to a) \to \Downarrow ((C \to L) \to \theta\{a\}^+)\}}$$
4. If $\zeta\{\ \}^*$ is $(\theta\{\ \}^+ \to L) \to C$, then we can use a switch rule, since the atom $a$ does not appear in $C$, and then go on by induction hypothesis:
$$\mathsf{sr}\ \frac{\xi\{(((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to C\}}{\xi\{(B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to C)\}}$$

5. If $\zeta\{\ \}^*$ is $(\theta\{\ \}^+ \to L) \to \theta'\{a\}^+$, then we have to use both the exchange and switch rules, after using a contraction rule to duplicate the block $B \to a$, and then go on by using twice the induction hypothesis:

$$\mathsf{cr}\ \dfrac{\mathsf{sr}\ \dfrac{\mathsf{xr}\ \dfrac{\xi\{(((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to ((B \to a) \to \Downarrow \theta'\{a\}^+)\}}{\xi\{(B \to a) \to \Downarrow (((B \to a) \to \Downarrow \theta\{a\}^+) \to L) \to \theta'\{a\}^+\}}}{\xi\{(B \to a) \to \Downarrow ((B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to \theta'\{a\}^+))\}}}{\xi\{(B \to a) \to \Downarrow ((\theta\{a\}^+ \to L) \to \theta'\{a\}^+)\}}$$

Notice that if the context $\zeta\{\ \}^*$ had no hole, the weakening rule is applied and no replacement is performed. □

THEOREM 28 (COMPUTATIONAL ADEQUACY OF JLSn). *For any two λ-terms $t$ and $u$, there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in the* JLSn *system if and only if $t \longrightarrow_\beta^* u$.*

PROOF. By induction on the reduction steps from $t$ to $u$. If the reduction path is empty, $t$ and $u$ are the same and the result is trivial because the encoding $[\![\cdot]\!]_\lambda$ is uniquely defined. In the general case, we consider the first step in the reduction, and use the induction hypothesis on the rest of the reduction. We thus simply have to check that there is an inference rule instance in JLSn with premise $[\![u]\!]_\lambda$ and conclusion $[\![t]\!]_\lambda$ if and only if we have $t \longrightarrow_\beta u$. To do it, we consider a particular redex, such that $t$ is the term $C[(\lambda x.v)\ w]$ for some context $C[\cdot]$, and show that $u$ is $C[v\{w/x\}]$ if and only if there is a derivation from $[\![u]\!]_\lambda$ to $[\![t]\!]_\lambda$ in JLSn. More precisely, this derivation exactly consists of one phase, where the bottommost rule instance is an instance of isu with premise $([\![w]\!]_\lambda \to x) \to \Downarrow [\![v]\!]_\lambda$, and the premise of the phase is a basic structure which must be $[\![u]\!]_\lambda$, as a direct consequence of Lemma 27. □

COROLLARY 29 (FULL COMPOSITION). *For any given λs-terms $t$ and $u$, we have $t[x \leftarrow u] \longrightarrow_{\mathsf{S}}^* t\{u/x\}$.*

PROOF. This is a corollary of Lemma 27 considered through the encoding $[\![\cdot]\!]_\lambda$ using Theorem 28. Indeed, the therm $t\{u/x\}$ using an implicit substitution corresponds through $[\![\cdot]\!]_\lambda$ to the structure $[\![t]\!]_\lambda$ where all occurrences of the atom $x$ are replaced with the structure $[\![u]\!]_\lambda$. □

This theorem establishes a precise correspondence between one $\beta$-reduction big step, performing an implicit substitution inside the term, and a focusing phase in JLSn. We can now express the relation between the big-step reduction in the $\lambda$-calculus and the small-step operational behaviour which is implemented in the $\lambda$s-calculus, and this corresponds to the study of soundness and completeness of JLSn with respect to JLSd.

THEOREM 30 (SOUNDNESS OF JLSn). *For two basic structures $A$ and $B$, if there is a derivation from $A$ to $B$ in* JLSn *then there is a derivation from $A$ to $B$ in* JLSd.

PROOF. Each inference rule in the JLSn system is actually an inference rule of JLSd with focus annotations. Therefore, we just need to remove annotations in the given derivation from $A$ to $B$ in JLSn to produce a derivation from $A$ to $B$ in JLSd. □

The meaning of this theorem, on the computational side, is that any reduction path from $t$ to $u$ in the restricted reduction system of $\beta$-reduction can be replaced with a reduction sequence from $t$ to $u$ in the more general $\longrightarrow_{\mathsf{R}}$ rewrite system, which is exactly stepwise simulation of $\beta$-reduction by explicit substitutions.

COROLLARY 31 (SIMULATION OF $\beta$). *For any λ-terms $t$ and $u$, if $t \longrightarrow_\beta^* u$ then $t \longrightarrow_{\mathsf{S}}^* u$.*

The other direction of the correspondence between JLSd and JLSn is more interesting, since it indicates that the simulation of $\beta$-reduction in the $\lambda$s-calculus does not rely on the use of a reduction system that would not be sensible.

LEMMA 32. *For any basic structure $A$, if there is a derivation from $A$ to a structure $\xi\{(B \to c) \to \zeta\{c\}^*\}$ in* JLSd*, there is a derivation of at most the same length from $A$ to $\xi\{\zeta\{B\}^*\}$ in* JLSd.

PROOF. By induction on the context $\zeta\{\ \}^*$. If $\zeta\{\ \}^*$ is $\{\ \}$, then we can use the ir rule and an induction on the given derivation $\mathscr{D}$ to remove the structure $(B \to c)$ and replace $c$ with $B$. This is similar to the result of Lemma 10, stating the invertibility of the rule ir. In the general case, we can use a case analysis on the shape of $\zeta\{\ \}^*$ and in each case use an induction on the given derivation, as for ir. This induction is a variant of invertibility for the rules of JLSb, which preserves the upper bound on the length of the derivation, and relies on the fact that the given derivation uses these rules because its premise $A$ is a basic structure — and $B \to c$ thus does not appear in $A$. □

THEOREM 33 (COMPLETENESS OF JLSn). *Given any two basic structures $A$ and $B$, if there is a derivation from $A$ to $B$ in the* JLSd *system, there is a derivation from $A$ to $B$ in the* JLSn *system.*

PROOF. By induction on a given derivation $\mathscr{D}$ from $A$ to $B$ in the JLSd system. If $\mathscr{D}$ is of length 0, then $A$ is $B$ and there is a trivial derivation from $A$ to $B$ in JLSn. In the general case, since $B$ is a basic structure, there is no structure of the shape $C \to d$ in negative position in $B$ and the bottommost rule instance must be an instance of the isu rule, and has a structure of the shape $\xi\{(C \to d) \to \zeta\{d\}^*\}$ as premise. Thus, by Lemma 32 there is a derivation $\mathscr{D}_1$ of smaller length than $\mathscr{D}$ from $A$ to $\xi\{\zeta\{C\}^*\}$ in JLSd, and by Lemma 27 there is a derivation $\mathscr{D}_2$ from $\xi\{\zeta\{C\}^*\}$ to $B$ in JLSn. We can then apply the induction hypothesis to $\mathscr{D}_1$ to produce a derivation $\mathscr{D}_3$ and the result is the composition of the two derivations $\mathscr{D}_2$ and $\mathscr{D}_3$. □

On the computational side, this theorem says that the $\lambda$s-calculus is sensible with respect to the standard $\lambda$-calculus. Indeed, a way of defining a reduction system that can simulate $\beta$-reduction is to add *too many* possible reductions, thus loosing all good properties. This is not the case here, since we have stated in this theorem the projection of reduction with explicit substitutions inside $\beta$-reduction. Notice that in the following corollary, it is important that the given terms $t$ and $u$ are plain $\lambda$-terms, and not terms with explicit substitutions.

COROLLARY 34 (PROJECTION IN $\beta$). *For any λ-terms $t$ and $u$, if $t \longrightarrow_{\mathsf{S}}^* u$ then $t \longrightarrow_\beta^* u$.*

We have now all the elements to compare the standard $\lambda$-calculus and the $\lambda$s-calculus with explicit substitutions only in terms of comparisons between the logical systems JLSn and JLSd that we have defined. Moreover, there are many possible restrictions of the JLSd system that correspond to other $\lambda$-calculi or various reduction strategies. For example, we could add restrictions on inference rules of JLSn to enforce a normal order evaluation, so that this variant would correspond exactly to the *call-by-name* weak reduction of $\lambda$-terms. Enforcing a *call-by-value* reduction strategy is more complicated, since the required restrictions on

inference rules would be less natural, because of the problem of detecting structures that represent values. Notice that using a shallow proof search strategy betrays the idea of using the deep inference methodology, the same way as using a weak reduction strategy *betrays the very spirit of the λ-calculus* [Asp98].

# 5. CONCLUSION

We have defined here a deductive system for a small fragment of intuitionistic logic and established a correspondence between the proof search process in this system and computation in the functional programming setting, by means of a λ-calculus with explicit substitutions. This methodology can provide interesting links between results on the logical side, such as the soundness and completeness results of a deductive system, and results in the λ-calculus, such as the simulation of a reduction system in another system.

The use of a deep inference system is essential in the sense that it induces a very elegant correspondence between inference rules and reduction rules, and allows to model a reduction sequence as a derivation from one structure to another in a logical system, so that computation traces can be easily composed. If we want to have this ability, we need to be able to apply inference rules deep inside formulas to handle reductions inside a λ-term, and not reduce the study to weak evaluation strategies only. There is much future work left in this project, and we give now some details on further research directions to be explored.

**Extension to full JS.** In this paper we have restricted our study to a small fragment of the JS proof system, based on restrictions that are quite artificial from the logical viewpoint, which were only motivated by the structure of our λ-calculus. Although we can still explore the relations between logical and computational notions in this setting, it would surely be interesting to design a correspondence for the full JS system. The question is then to find a meaning for non-atomic formulas in negative position, which could be done with a generic notion of *pattern matching*, as is defined in the pure pattern calculus [JK09]. Indeed, this calculus generalises the notion of λ-abstraction from $\lambda x.t$ into the *case* construct $u \to t$, where the $x$ in negative position is replaced with some arbitrary term $u$ is negative position. This extension is not trivial, since we have to handle explicit pattern matching, and syntactic distinctions between case, application and explicit matching. Moreover, we would still have to restrict JS since we cannot accept some term manipulations, such as the transformation from $u \to v \to t$ to $v \to u \to t$ in the general case. Such an extension would provide a logical background for the pure pattern calculus, allowing to explore the benefits of using the standard proof-theoretical techniques.

**Evaluation strategies.** Another important direction is the study of different evaluation strategies available in the λ-calculus with explicit substitutions, ranging from highly non-deterministic to highly constrained procedures. It was interesting to notice the connection between the focusing technique and call-by-name, and we can now look for more subtle restrictions which would correspond to known strategies, such as call-by-value. Normal forms for proofs can be used as a source of inspiration for new evaluation strategies, and because of the methodology used here, all the work on improving the efficiency of proof search can be used to improve efficiency in reduction strategies. For example, if we can prove that for a given set of formulas, some restricted system only allows *short proofs* — that is, bound in some way with respect to the size of the formula — it means that we have an evaluation strategy where reduction is also bounded. It might be interesting to translate the question of optimal reduction in this setting.

**Variant systems.** Many variants of our inference rules can be defined in the calculus of structures, and in particular local rules, which never modify, copy or erase non-atomic formula. Such a local variant of JLSd would induce through our correspondence a calculus where all reduction steps are local, which can be a nice feature in the implementation of a functional language. The problem is that designing a local presentation for intuitionistic logic might be quite difficult, or induce a highly complex system that would not fit our needs [Tiu06].

**Classes of λ-terms.** There is one aspect of the standard results on λ-terms that has been set aside in this study: the class of *simply-typed*, and weakly or strongly normalising terms, is quite difficult to understand in our proof-search setting. It is unclear if these subsets of the set of all programs make any sense here, since types are usually identified with logical formulas. It would be interesting to provide characterisations, in terms of restricted proof systems, of important sets of terms, such as those that terminate, or those that can be composed while preserving some good properties.

**Other logics and calculi.** The methodology introduced here is a general notion of correspondence, and could be applied in other settings. As an example, *linear logic* [Gir87] is a refinement of intuitionistic logic that is often presented as a logic of *resources*, as it offers a greater control over duplication and erasure inside its proof system. The interpretation of a system for intuitionistic linear logic could thus provide a basis for the study of λ-calculi with resource operators [KL05]. Moreover, our correspondence could be translated to a classical setting, its computational side taking the form of some *process algebra* rather than a sequential programming language, as suggested by existing work on the encoding of concurrent computation in proof search within the setting of deep inference [Bru02].

**Acknowledgements.** The paper has greatly benefited from the many interesting comments and useful suggestions made by the anonymous reviewers.

# 6. REFERENCES

[AK10] B. Accattoli and D. Kesner. The structural λ-calculus. In A. Dawar and H. Veith, editors, *CSL'10*, volume 6247 of *LNCS*, pages 381–395, 2010.

[And92] J-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[Asp98] A. Asperti. Optimal reduction of functional expressions. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *PLILP'98*, volume 1490 of *LNCS*, pages 427–428, 1998.

[BM08] K. Brünnler and R. McKinley. An algorithmic interpretation of a deep inference system. In I. Cervesato, H. Veith, and A. Voronkov, editors, *LPAR'08*, volume 5330 of *LNCS*, pages 482–496, 2008.

[Bru02] P. Bruscoli. A purely logical account of sequentiality in proof search. In P. J. Stuckey, editor, *ICLP'02*, volume 2401 of *LNCS*, pages 302–316, 2002.

[Brü03] K. Brünnler. *Deep Inference and Symmetry in Classical Proofs*. PhD thesis, Technische Universität Dresden, September 2003.

[Brü06] K. Brünnler. Locality for classical logic. *Notre Dame Journal of Formal Logic*, 47:557–580, 2006.

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

[GS01] A. Guglielmi and L. Straßburger. Non-commutativity and MELL in the calculus of structures. In L. Fribourg, editor, *CSL'01*, volume 2142 of *LNCS*, pages 54–68, 2001.

[GS10] A. Guglielmi and L. Straßburger. A system of interaction and structure IV: The exponentials and decomposition. To appear in *ACM Transactions on Computational Logic*, 2010.

[Gue10] N. Guenot. Focused proof search for linear logic in the calculus of structures. In M. Hermenegildo and T. Schaub, editors, *ICLP'10 (Technical Comm.)*, volume 7 of *LIPIcs*, pages 84–93, 2010.

[Her94] H. Herbelin. A $\lambda$-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *CSL'94*, volume 933 of *LNCS*, pages 61–75, 1994.

[JK09] B. Jay and D. Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

[Kes07] D. Kesner. The theory of calculi with explicit substitutions revisited. In J. Duparc and T. A. Henzinger, editors, *CSL'07*, volume 4646 of *LNCS*, pages 238–252, 2007.

[KL05] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In J. Giesl, editor, *RTA'05*, volume 3467 of *LNCS*, pages 407–422, 2005.

[LLD+04] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[LM07] C. Liang and D. Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL'07*, volume 4646 of *LNCS*, pages 451–465, 2007.

[Mil89] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6(1&2):79–108, 1989.

[MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Plo75] G. Plotkin. Call-by-name, call-by-value and the $\lambda$-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

[Rov11] L. Roversi. Linear $\lambda$-calculus with explicit substitutions as proof-search in deep inference. Accepted at *TLCA'11*, 2011.

[Str03] L. Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, Technische Universität Dresden, July 2003.

[Tiu06] A. Tiu. A local system for intuitionistic logic. In M. Hermann and A. Voronkov, editors, *LPAR'06*, volume 4246 of *LNCS*, pages 242–256, 2006.