Computation in Focused Intuitionistic Logic

Taus Brock-Nannestad, Nicolas Guenot, Daniel Gustafsson

Computation in Focused Intuitionistic Logic

Taus Brock-Nannestad Nicolas Guenot Daniel Gustafsson

INRIA Saclay, Palaiseau
taus.brock-nannestad@inria.fr

IT University of Copenhagen
{ngue,dagu}@itu.dk

Abstract

We investigate the control of evaluation strategies in a variant of the λ-calculus derived through the Curry-Howard correspondence from ***LJF***, a sequent calculus for intuitionistic logic implementing the focusing technique. The proof theory of focused intuitionistic logic yields a single calculus in which a number of known λ-calculi appear as subsystems obtained by restricting types to a certain fragment of ***LJF***. In particular, standard λ-calculi as well as the call-by-push-value calculus are analysed using this framework, and we relate cut elimination for ***LJF*** to a new abstract machine subsuming well-known machines for these different strategies.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Proof theory, Lambda-calculus and related systems

Keywords Intuitionistic Logic, Curry-Howard, Lambda-calculus, Focusing, Polarities, Evaluation Strategies, Abstract Machines

1. Focusing and Computation

Understanding the notion of *computation* has been a question of great interest to many logicians since the inception of the so-called Curry-Howard correspondence, contributing to the development of the functional paradigm, using expressive type systems, or the introduction of logic programming. The logical perspective on computation is now centered around a couple of results in proof theory, the most important ones being *normalisation* for natural deduction and *focalisation* in the sequent calculus. Normalisation provides an operational semantics of languages based on the pure λ-calculus, just as proof search in a focused sequent calculus gives the semantics of a logic programming language [2].

Over the last decades, the *"proofs-as-programs"* approach has been extended from its original form in a natural deduction setting to sequent calculus systems [21]. This required a departure from the traditional syntax of the λ-calculus [5], and lead to insights on various forms of computation [17]. On another side, following the *"proof-search-as-computation"* idea, the advent of focusing [2] as a technique stemming from the analysis of the sequent calculus through linear logic [19] provided clean logical foundations for logic programming, for linear logic as well as for more standard logics [26]. Over time, the scope of focusing has broadened, the role of the associated *polarities* has been investigated [24], and the perception of this result has shifted from a proof search technique to an important normal form in the sequent calculus and in other logical formalisms [6, 8].

The crucial observation at this point is that the variants of the λ-calculus using the sequent calculus formalism to provide some insights on reduction strategies are focused calculi [14, 21] where a certain form of focusing constrains reduction. In such a calculus, only terms enforcing one particular strategy can be typed, and the corresponding proof system is a fragment of the focused system for intuitionistic logic. Our goal is to give a computational interpretation for (a variant of) the focused system ***LJF*** of Liang and Miller [26], and show how the resulting λ-calculus provides a means of fine-grained control of the reductions involved in the evaluation of λ-terms. In particular, we show that the $\lambda\kappa$-calculus, which we introduce as a language of proof-terms for ***LJF***, contains the standard *call-by-name* and *call-by-value* calculi, obtained from the fragments ***LJT*** and ***LJQ***. The methodology we follow here has its importance: we wish to interpret ***LJF*** as a system entirely justified using proof-theoretical arguments, rather than an *ad-hoc* system for particular strategies. This approach confirms the two natural representations of the most standard strategies — CBN and CBV — in the focused type system, but it also reveals the tight connection between ***LJF*** and *call-by-push-value* [25], in which the two standard strategies can be embedded. The relation between CBPV and focusing was conjectured but still unclear: we establish it in details and discuss their differences.

The ability to control the reduction strategy of a term through dedicated operators, reflected at the level of types by the presence of explicit *polarity shifts* — the key ingredient in the focalisation result — is a striking example of *double discovery*. Indeed, the focusing technique was a rather syntactic artifact of linear logic that rose to the status of *"éminence grise"* in proof theory, while the call-by-push-value language stems from the thorough analysis of the semantics of the two major functional paradigms. Their convergence is a sign of their significance.

Moreover, we study the extraction of an abstract machine from the cut elimination procedure of a given focused proof system, and show how the procedures for the ***LJT*** and ***LJQ*** fragments of ***LJF*** yield essentially the most natural machines implementing CBN and CBV: the `KAM` [23] and the `CEK` machine [18]. Then, we describe an abstract machine implementing cut elimination in the ***LJF*** system and show how it relates to the `CK`-style machine defined in the literature for CBPV [25].

The investigation starts with the definition of the $\lambda\kappa$-calculus, the computational interpretation of ***LJF*** proofs, a discussion of its properties and the relation between reduction rules and cut elimination, in Section 2. The expressivity of ***LJF*** is illustrated in Section 3, by considering the fragments ***LJT*** and ***LJQ***, and by showing that the general proof-theoretical approach based upon focusing yields the expected results. This is further demonstrated in Section 4 by introducing abstract machines, and we describe a general scheme meant to derive a machine from some given

$$\mathit{fl}\;\frac{\Gamma, x : \downarrow N, [N] \vDash k : M}{\Gamma, x : \downarrow N \vdash x\ k : M} \qquad \mathit{fr}\;\frac{\Gamma \vDash p : [P]}{\Gamma \vdash \lceil p \rceil : \uparrow P}$$

$$\mathit{bl}\;\frac{\Gamma, x : P \vdash t : M}{\Gamma, [\uparrow P] \vDash \kappa x.t : M} \qquad \mathit{br}\;\frac{\Gamma \vdash t : N}{\Gamma \vDash \lfloor t \rfloor : [\downarrow N]}$$

$$\mathit{ir}\;\frac{\Gamma, x : P \vdash t : N}{\Gamma \vdash \lambda x.t : P \supset N}$$

$$\mathit{il}\;\frac{\Gamma \vDash p : [P] \quad \Gamma, [N] \vDash k : M}{\Gamma, [P \supset N] \vDash p :: k : M}$$

$$\mathit{axl}\;\frac{N \in \{a^-, \uparrow P\}}{\Gamma, [N] \vDash \varepsilon : N} \qquad \mathit{axr}\;\frac{P \in \{a^+, \downarrow N\}}{\Gamma, x : P \vDash x : [P]}$$

$$\mathit{hcl}\;\frac{\Gamma \vdash t : N \quad \Gamma, [N] \vDash k : M}{\Gamma \vdash t\ k : M} \qquad \mathit{hcr}\;\frac{\Gamma \vDash p : [P] \quad \Gamma, x : P \vdash t : N}{\Gamma \vdash p \text{ to } x.t : N}$$

$$\mathit{fcl}\;\frac{\Gamma, [M] \vDash k : N \quad \Gamma, [N] \vDash m : L}{\Gamma, [M] \vDash k \,@\, m : L} \qquad \mathit{fcr}^+\;\frac{\Gamma \vDash p : [P] \quad \Gamma, x : P \vDash q : [Q]}{\Gamma \vDash p \text{ to } x.q : [Q]}$$

$$\mathit{fcr}^-\;\frac{\Gamma \vDash p : [P] \quad \Gamma, x : P, [N] \vDash k : M}{\Gamma, [N] \vDash p \text{ to } x.k : M}$$

Figure 1. *Rules for the sequent calculus **LJF** and associated terms*

focused cut elimination procedure. We consider the extracted CBV and CBN machines and their relation to other machines. Then, Section 5 is devoted to the comparison between the $\lambda\kappa$-calculus and CBPV, the key contribution being a bijection between the two calculi — this result crucially depends on the particular flavour of the ***LJF*** system presented in Section 2. We come there to the conclusion that CBPV corresponds to a specific form of ***LJF*** where maximal inversion is not enforced, called a *weakly focused* system. This correspondence relies on the *bidirectional* [9] variant of the CBPV system, and it involves the introduction of delays to account for the ability to type λ-terms that are not η-long.

Related work. The impact of focalisation on the strategy used to evaluate terms has been mostly studied in the setting of classical logic, after the introduction of the $\bar{\lambda}\mu\tilde{\mu}$-calculus by Curien and Herbelin [10] and later refined in [11]. The analysis of polarities in classical logic can be traced back to Girard [20], and the ***LKT*** and ***LKQ*** systems [12], precursors of the ***LJT*** and ***LJQ*** systems. Notice that studying directly the intuitionistic focused system ***LJF*** allows to clarify the picture, and demonstrates how types for values and computations in the λ-calculus differ in their polarities [30]. On the purely logical level, our particular presentation of ***LJF*** relates to the *structural* approach to focalisation [29], and the question of polarity assigments in ***LJF*** studied in [28].

At the level of the $\lambda\kappa$-calculus, the operations involved can be traced to interpretations of ***LJ*** [15] or CBPV [25]. A relation between sequent calculus systems and abstract machines has been investigated using the $\bar{\lambda}\mu\tilde{\mu}$ framework [3], and the connection leading from explicit substitutions — related to *"cuts"* in natural deduction — to abstract machines has been described from the untyped perspective in [1]. Defining an abstract machine is here presented as a process of extraction of information from a focused cut elimination procedure.

2. A Computational Interpretation of *LJF*

In this section we will describe a focused sequent calculus called ***LJF***, with a few differences between our presentation and the original [26]. The idea of focusing is that a careful analysis of the properties of inference rules allows to classify them into two categories, one for fully invertible rules, called *asynchronous*, and one for partially invertible rules, which are called *synchronous*. It has been observed that these rules can be organised in a particular way inside any given proof — the result originated from linear logic [2], but it has been generalised to many other logics. For more details about the intuitive interpretation of focusing, we refer the reader to one of the many descriptions available in the literature, see for example [29].

There are essentially three differences between the system we use here and the original version of ***LJF***:

1. *concerning polarities*: in the original presentation of ***LJF***, the syntax of formulas is *unpolarised*, although the idea of polarity is already present, and used for example in the rules initiating and finishing the synchronous phases — we choose to make it explicit using *polarity shifts*, which mediate between negative and positive formulas,
2. *concerning maximal inversion*: the original presentation of ***LJF*** enforces not only maximality of synchronous phases, it also has maximal asynchronous phases, using the explicit inversion context, while we drop this restriction to consider the *weakly focused* version of ***LJF***, although the strongly focused variant is discussed in Section 6 — the computational behaviour of maximal inversion is currently unclear,
3. *concerning initial rules*: it is customary to have atomic initial rules in a focused calculus — one for negative atoms and one for positive atoms — but we observe that this is not necessary, and in particular one can prove from atomic initial rules the admissibility of the initial rules on shifts.

Remark 2.1. *Beyond the question of the computational meaning of maximal inversion, enforcing such a structure in a natural deduction setting [6] is difficult. Directly relating the* "strongly" *focused* ***LJF*** *to the weakly focused CBPV would be problematic due to the impedance mismatch between these systems.*

As mentioned, the syntax of formulas in this variant of ***LJF*** is explicitly polarised, based on the following grammar:

$$P, Q \ ::= \ a^+ \mid \downarrow N \qquad M, N \ ::= \ a^- \mid P \supset N \mid \uparrow P$$

and its inference rules are shown above in Figure 1. In the calculus presented there, rules are annotated with terms and sequents take the form of typing judgements, where Γ and Δ denote sets of labelled formulas. There are three forms of judgements in this system, that we will now simply call ***LJF***:

$\Gamma \vdash t : N$	*asynchronous*
$\Gamma, [N] \vDash k : M$	*left-synchronous*
$\Gamma \vDash p : [P]$	*right-synchronous*

In the synchronous phase, the formula written inside brackets is said to be *"under focus"*. By inspection of the inference rules, it is clear that foci must always be the principal formulas of the conclusions of the rules they appear in. Furthermore, this focus is passed on to the subformulas of the focused formula. In this way, focusing on a given formula ensures that it and its subformulas are decomposed in a *maximal chain* of inference steps. Conversely, in the asynchronous phase, formulas can be decomposed in any order. Choosing which formulas may be focused and when the synchronous phase ends is made explicit in the rules introducing the polarity shifts $\downarrow$ and $\uparrow$.

Since we only consider a weakly focused variant of ***LJF*** in this paper, the number of cut rules is smaller than in [26]. Additionally, we leave out entirely unfocused cuts (the Cut^+ and Cut^- of [26]), as they are in fact not needed for the cut elimination proof, and can easily be recovered as corollaries after the fact. Every cut rule in our system thus has the property that the cut formula appears under focus in one of the two premises.

Names of inference rules follow a uniform scheme. For the core cut-free calculus, the first letter indicates whether the principal formula is related to ***f****ocusing*, ***b****lurring*, or ***i****mplication*. The second letter indicates whether the principal formula is introduced on the ***l****eft* or the ***r****ight*. For the cut rules, we distinguish between the ***h****ead* cuts, where the principal reductions take place, and the ***f****ocused* cuts which handle all the commutative cases. Note that these can be distinguished by whether the conclusion contains a focus or not. Finally, for each of the cut rules, it is the case that the cut formula is focused in exactly one of the two premises. The ***l*** or ***r*** annotation indicates the premise that is *not* focused, and thus the premise which will be decomposed during cut reduction.

Judgments are annotated with terms of the $\lambda\kappa$-calculus, an extension of the λ-calculus based on the following grammar:

$$\begin{array}{rcl} t,u & ::= & \lambda x.t \mid x\,k \mid \lceil p \rceil \mid t\,k \mid p \;\mathsf{to}\; x.t \\ k,m & ::= & \varepsilon \mid p :: k \mid \kappa x.t \mid k \mathbin{@} m \mid p \;\mathsf{to}\; x.k \\ p,q & ::= & x \mid \lfloor t \rfloor \mid p \;\mathsf{to}\; x.q \end{array} \tag{1}$$

Note that there is a very tight correspondence between the proof terms and the derivations of ***LJF***. In a sense, the syntax of proof terms was *extracted* from the calculus by a recipe we will sketch now. The basic observation is the following: the focusing discipline in many cases obviates the need for variable binding in the proof terms. The prototypical example of this would be the ***il*** rule from the unfocused ***LJ***:

$$\textit{\textbf{il}}\;\dfrac{\Gamma, f : P \supset N \vdash t : P \qquad \Gamma, f : P \supset N, x : N \vdash u : M}{\Gamma, f : P \supset N \vdash \mathsf{let}\; x = f\; t \;\mathsf{in}\; u : M}$$

In the focused setting, the second premise must decompose the hypothesis x immediately, and it is therefore not necessary to introduce an explicit binder for x. Consequently, the proof term assignment for this rule in ***LJF*** is simply:

$$\textit{\textbf{il}}\;\dfrac{\Gamma \vDash p : [P] \qquad \Gamma, [N] \vDash k : M}{\Gamma, [P \supset N] \vDash p :: k : M}$$

According to this methodology, a binder should only occur when decomposition is not immediately forced. This explains why only ***bl*** and ***ir*** involve binders, while ***hcl*** and ***fcl*** do not.

The intuitive meaning of the $\lambda\kappa$-calculus will be discussed later in this section, but we first concentrate on the basic properties of the ***LJF*** system shown in Figure 1. First, when comparing it to the usual, unfocused system ***LJ*** for intuitionistic logic, we need to define $\mathsf{E}(-)$ to be the following *polarity erasure* function, mapping a polarised formula to its unpolarised counterpart:

$$\begin{array}{rclcrclc} \mathsf{E}(a^-) & = & a & \quad & \mathsf{E}(a^+) & = & a & \\ \mathsf{E}(\uparrow P) & = & \mathsf{E}(P) & & \mathsf{E}(\downarrow N) & = & \mathsf{E}(N) & \quad \mathsf{E}(P \supset N) = \mathsf{E}(P) \supset \mathsf{E}(N) \end{array}$$

Note that an unpolarised formula may be polarised in many different ways, by choosing the polarity of atoms, or by inserting redundant polarity shifts. In particular, we have $\mathsf{E}(\uparrow\downarrow N) = \mathsf{E}(N)$ and $\mathsf{E}(\downarrow\uparrow P) = \mathsf{E}(P)$, and this enables a more fine-grained control of the order in which formulas must be decomposed, since phases are controlled by polarity shifts.

The most important property of focusing is that it is sound and complete, regardless of the aforementioned choice of one particular polarisation. Following [29], we will state this result in terms of erasure.

Theorem 2.2. *Given any negative formula N, the sequent $\vdash \mathsf{E}(N)$ is provable in* ***LJ*** *if and only if $\vdash N$ is provable in* ***LJF****.*

Proof. The first direction follows immediately from the fact that the rules of ***LJF*** are just more restricted versions of the rules of ***LJ***. To show completeness of ***LJF***, one may show that the all usual unfocused inference rules of ***LJ*** are admissible in ***LJF***. This is straightforward once cut rules have been proven admissible in the focused system [29]. Alternatively, the result may be established by the "grand tour" strategy, as seen for example in [26]. □

We now turn to the most important result concerning the ***LJF*** focused system in this paper, describing the dynamics of proofs.

Theorem 2.3 (Cut elimination)**.** *The cut rules* ***hcl****,* ***hcr****,* ***fcl****,* ***fcr***$^+$ *and* ***fcr***$^-$ *are admissible in* ***LJF****.*

Proof. We proceed by lexicographical induction over the structure of the cut formula and of the premises of the given cut. We assume the two given input derivations to be cut-free, corresponding to a strategy reducing the topmost cuts in the given derivation. Note that because we are working with the focused versions of the cut rule, every cut may be reduced by only considering cases on one of the premises, specifically the premise where the cut formula is *not* under focus. We show here only a few of the cases of the cut elimination argument — the full proof may be found in the appendix:

- Case ***hcl***: if the first premise ends in ***ir*** then this is the principal case for implication, and the reduction is:

$$\textit{\textbf{hcl}}\;\dfrac{\textit{\textbf{ir}}\;\dfrac{\overset{\mathcal{D}}{\Gamma, x : P \vdash t : N}}{\Gamma \vdash \lambda x.t : P \supset N} \qquad \textit{\textbf{il}}\;\dfrac{\overset{\mathcal{E}}{\Gamma \vDash p : [P]} \qquad \overset{\mathcal{F}}{\Gamma, [N] \vDash k : M}}{\Gamma, [P \supset N] \vDash p :: k : M}}{\Gamma \vdash (\lambda x.t)\,(p :: k) : M}$$

$$\begin{array}{lr} \mathcal{D}' :: \Gamma \vdash p \;\mathsf{to}\; x.t : N & \text{by } \textit{\textbf{hcr}} \text{ on } (P, \mathcal{E}, \mathcal{D}), \\ \quad\;\; \Gamma \vdash (p \;\mathsf{to}\; x.t)\; k : M & \text{by } \textit{\textbf{hcl}} \text{ on } (N, \mathcal{D}', \mathcal{F}). \end{array}$$

- Case ***hcl***: if the first premise ends in ***fr*** then this is the principal case for negative shift, and the reduction is:

$$\textit{\textbf{hcl}}\;\dfrac{\textit{\textbf{fr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]}}{\Gamma \vdash \lceil p \rceil : \uparrow P} \qquad \textit{\textbf{bl}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : P \vdash t : M}}{\Gamma, [\uparrow P] \vDash \kappa x.t : M}}{\Gamma \vdash \lceil p \rceil\,(\kappa x.t) : M}$$

$$\begin{array}{lr} \Gamma \vdash p \;\mathsf{to}\; x.t : M \qquad\qquad & \text{by } \textit{\textbf{hcr}} \text{ on } (P, \mathcal{D}, \mathcal{E}). \end{array}$$

- Case ***hcr***: if the second premise ends in ***fl*** then it is the principal case for positive shift, and the reduction is:

$$\textit{\textbf{hcr}}\;\dfrac{\textit{\textbf{fr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vdash u : M}}{\mathcal{D}' :: \Gamma \vDash \lfloor u \rfloor : [\downarrow M]} \qquad \textit{\textbf{bl}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : \downarrow M, [M] \vDash k : N}}{\Gamma, x : \downarrow M \vdash x\; k : N}}{\Gamma \vdash \lfloor u \rfloor \;\mathsf{to}\; x.(x\; k) : N}$$

$$\begin{array}{lr} \mathcal{E}' \;::\; \Gamma, [M] \vDash \lfloor u \rfloor \;\mathsf{to}\; x.k : N & \text{by } \textit{\textbf{fcr}}^- \text{ on } (\downarrow M, \mathcal{D}', \mathcal{E}), \\ \qquad\;\; \Gamma \vdash u\; (\lfloor u \rfloor \;\mathsf{to}\; x.k) : N & \text{by } \textit{\textbf{hcl}} \text{ on } (M, \mathcal{D}, \mathcal{E}'). \end{array}$$

and all other cases can be treated in a similar way. □

From this cut elimination theorem, we now extract a system of reduction rules on $\lambda\kappa$-terms, by considering each case as one rewrite on proof objects:

$$
\begin{array}{rcl}
(\lambda x.t)\,(q :: k) & \to & (q \,\mathtt{to}\, x.t)\, k \\
\lceil q \rceil\,(\kappa x.t) & \to & q \,\mathtt{to}\, x.t \\
(x\ k)\, m & \to & x\ (k \,@\, m) \\[4pt]
\varepsilon \,@\, m & \to & m \\
(q :: k) \,@\, m & \to & q :: (k \,@\, m) \\
(\kappa x.t) \,@\, m & \to & \kappa x.(t\ m) \\[4pt]
x \,\mathtt{to}\, y.t & \to & t\{x/y\} \\
p \,\mathtt{to}\, x.(\lambda y.t) & \to & \lambda y.(p \,\mathtt{to}\, x.t) \\
p \,\mathtt{to}\, x.\lceil q \rceil & \to & \lceil p \,\mathtt{to}\, x.q \rceil \\
p \,\mathtt{to}\, x.(y\ k) & \to & y\ (p \,\mathtt{to}\, x.k) \\
\lfloor u \rfloor \,\mathtt{to}\, x.(x\ k) & \to & u\ (\lfloor u \rfloor \,\mathtt{to}\, x.k) \\[4pt]
x \,\mathtt{to}\, y.q & \to & q\{x/y\} \\
p \,\mathtt{to}\, x.x & \to & p \\
p \,\mathtt{to}\, x.y & \to & y \\
p \,\mathtt{to}\, x.\lfloor t \rfloor & \to & \lfloor p \,\mathtt{to}\, x.t \rfloor \\[4pt]
x \,\mathtt{to}\, y.k & \to & k\{x/y\} \\
p \,\mathtt{to}\, x.\varepsilon & \to & \varepsilon \\
p \,\mathtt{to}\, x.(q :: k) & \to & (p \,\mathtt{to}\, x.q) :: (p \,\mathtt{to}\, x.k) \\
p \,\mathtt{to}\, x.(\kappa y.t) & \to & \kappa y.(p \,\mathtt{to}\, x.t)
\end{array} \tag{2}
$$

where, in the reductions where p appears we have a *proviso* that states p should *not* be a variable. The reason for this is that we wish to avoid the *aliasing* of names, created when bindings are chained so that a value is pointed to by several names.

Computational interpretation. Such a rewrite system for $\lambda\kappa$, extracted from the cut elimination proof, specifies the intuitive meaning to be attributed to each construct of the calculus. The symbols λ and κ as well as the to construction are binding *variable names*. The calculus obeys the laws of α-conversion, and binders are thus essentially treated as in the pure λ-calculus. An important distinction to be made in $\lambda\kappa$ is that t denotes a *term*, having the meaning of a computation, while p denotes a *value*, which is the result of a computation. Finally, k represents a *continuation*.

Some of the constructs in $\lambda\kappa$ are standard operations from the λ-calculus. Others are variations, such as the application $(x\ k)$, that can be thought of as a *continuation* k — or *context* — waiting for a function to be plugged in the place of x. The term k can be seen in most cases as a list of arguments given to the function plugged for x, and after plugging we obtain a proper application $t\ k$. Moreover, $k \,@\, m$ represents here the concatenation of two list of arguments, or the composition of two continuations. We can observe how reduction defines the use of a list of arguments, in the rule:

$$(\lambda x.t)\,(q :: k) \quad \to \quad q \,\mathtt{to}\, x.(t\ k)$$

Moreover, values can be bound to names, using the binder to which acts as an explicit substitution of a value p for a variable x. In the minimal system we are considering, values can be variables and are otherwise formed by placing a term in a *thunk*, that should be seen as delayed computations. The thunks can be *forced* by applying them to a list of arguments:

$$\lfloor u \rfloor \,\mathtt{to}\, x.(x\ k) \quad \to \quad u\ (\lfloor u \rfloor \,\mathtt{to}\, x.k)$$

where a copy of $\lfloor u \rfloor$ is passed on to the list, which may contain occurrences of x. The two remaining constructs are perhaps the most surprising ones, since they directly affect the *control flow* of the term. A term $\lceil p \rceil$ is indeed a computation that has finished, and *returns* a value, so that it releases control of the computation. The $\kappa x.t$ construction is the continuation that can then take control, binding the result p to the name x and starting computation t, as expressed in the rule:

$$\lceil q \rceil\,(\kappa x.t) \quad \to \quad q \,\mathtt{to}\, x.t$$

which already appeared in [15], although its typing there is not controlled by shifts and it requires no return construct to interact with. Note that substitutions are here eagerly decomposed, and propagated. We now turn to the properties of well-typed $\lambda\kappa$-terms, the usual ones being simple corollaries of cut elimination.

Corollary 2.4 (Subject Reduction)**.** *If $\Gamma \vdash t : N$ and $t \to u$ then we also have $\Gamma \vdash u : N$.*

Proof. Each reduction corresponds to one of the cases of the cut elimination argument, so the result is immediate. □

Moreover, cut elimination implies that reduction on well-typed terms is well-behaved, and leads to *normal forms*, which are terms typeable without the cut rules.

Corollary 2.5 (Normalisation)**.** *For a $\lambda\kappa$-term t, if $\Gamma \vdash t : N$ there exists a normal term u such that $\Gamma \vdash u : N$ and $t \to^* u$.*

As mentioned before, the ways in which a cut might be reduced are limited by the fact that the cut formula must be under focus in either the first or the second premise. This restriction yields cut elimination steps that are highly reminiscent of the steps used in the *tq*-protocol in [13]. A slightly more permissive variant of this protocol [16] may be summarised as follows:

1. if the cut formula is not principal in the second premise of the cut we wish to reduce, we permute the cut into this premise,
2. if the cut formula is not principal in the first premise of the cut, then we permute it into the first premise, and
3. if the cut formula is principal in both premises, then we reduce it as a principal cut.

The main difference between these cut elimination procedures is that when the first two cases are reduced in the *tq*-protocol, the cut must be permuted as far up as possible in one step. In contrast, we describe reductions as small steps, and more importantly we do not consider cases where a cut permutes above some other cut. As we will see in Section 4, it is necessary to introduce such permutations between cuts to construct an abstract machine for ***LJF***, but the proof of normalisation is more challenging.

Note that $\lambda\kappa$ cannot simulate a unique β step separately from other steps, since it is based on the syntax of the sequent calculus. However, we will see how it can implement reduction in other ways, and we observe that its reduction system is confluent, since it is left-linear, and it has no critical pair — under the *proviso* on variables applicable to the rules shown in (2). A particular interest of $\lambda\kappa$ is that it has two remarkably well-behaved fragments that correspond to the standard strategies used to reduce λ-terms.

3. Call-by-name and Call-by-value in *LJF*

In order to illustrate the expressivity of $\lambda\kappa$ and of the type system obtained from ***LJF***, we consider the two evaluation strategies in the λ-calculus from which the two major *lazy* and *strict* paradigms of functional programming are derived. As expected, ***LJF*** introduces a uniform treatment of these systems that matches the standard theories developed for these languages.

Call-by-name. The simplest strategy in the λ-calculus is to use β repeatedly until a normal form is reached, disregarding the shape of subterms. This is normally done following an outermost order, and disallows reduction in the argument of an application. This reduction strategy can be enforced by considering the type system shown in Figure 2, based on the ***LJT*** system. The resulting $\overline{\lambda}$-calculus allows the application of a term to a list rather than a single argument, based on the grammar:

$$
\begin{array}{rcl}
t,u & ::= & x\ k \mid \lambda x.t \mid t\ k \mid t[u/x] \\
k,m & ::= & \varepsilon \mid u :: k \mid k \,@\, m \mid k[u/x]
\end{array}
$$

$$
\textbf{ax}\,\frac{}{\Gamma,[a]\vDash \varepsilon : a}
\qquad
\textbf{il}\,\frac{\Gamma\vdash u : A \quad \Gamma,[B]\vDash k : C}{\Gamma,[A\supset B]\vDash u :: k : C}
\qquad\Bigg|\qquad
\textbf{hc}\,\frac{\Gamma\vdash t : A \quad \Gamma,[A]\vDash k : B}{\Gamma\vdash t\;k : B}
\qquad
\textbf{fhc}\,\frac{\Gamma,[B]\vDash k : A \quad \Gamma,[A]\vDash m : C}{\Gamma,[B]\vDash k\,@\,m : C}
$$

$$
\textbf{foc}\,\frac{\Gamma,x:A,[A]\vDash k : B}{\Gamma,x:A\vdash x\;k : B}
\qquad
\textbf{ir}\,\frac{\Gamma,x:A\vdash t : B}{\Gamma\vdash \lambda x.t : A\supset B}
\qquad\Bigg|\qquad
\textbf{mc}\,\frac{\Gamma\vdash u : A \quad \Gamma,x:A\vdash t : B}{\Gamma\vdash t[u/x] : B}
\qquad
\textbf{fmc}\,\frac{\Gamma\vdash u : A \quad \Gamma,x:A,[B]\vDash k : C}{\Gamma,[B]\vDash k[u/x] : C}
$$

Figure 2. *Rules for the sequent calculus **LJT** and associated terms*

This type system was introduced in [21] as the first λ-calculus isomorphic to a sequent calculus. In ***LJT***, the ***hc*** cut represents a β-redex, while reductions of ***fhc*** are simply administrative steps. Remaining cuts embody the substitution process. Cut elimination induces the following reductions, known to be *call-by-name*:

$$
\begin{array}{rcl}
(\lambda x.t)\,(u :: k) & \to & t[u/x]\;k \\
(x\;k)\;m & \to & x\;(k\,@\,m) \\[4pt]
\varepsilon\,@\,m & \to & m \\
(u :: k)\,@\,m & \to & u :: (k\,@\,m) \\[4pt]
(\lambda z.t)[u/x] & \to & \lambda z.t[u/x] \\
(x\;k)[u/x] & \to & u\;k[u/x] \\
(y\;k)[u/x] & \to & y\;k[u/x] \\[4pt]
\varepsilon[u/x] & \to & \varepsilon \\
(t :: k)[u/x] & \to & t[u/x] :: k[u/x]
\end{array}
\tag{3}
$$

The ***LJT*** proof system can be seen as a fragment of ***LJF*** through a typeability-preserving translation of terms into $\lambda\kappa$. The key idea is that the argument of a function needs to be placed into a thunk, to prevent reduction. This is consistent with the fact that $\lfloor u\rfloor\,\mathsf{to}\,x.t$ essentially performs in $\lambda\kappa$ the substitution of u for x into t. The translation is based on most ***LJT*** reductions appearing in ***LJF***.

Definition 3.1. *The translations $\mathsf{T}_t(\cdot)$ and $\mathsf{T}_k(\cdot)$ of $\overline{\lambda}$-terms into $\lambda\kappa$-terms are given by the following equations:*

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{T}_t(x\;k) & = & x\;\mathsf{T}_k(k) & \mathsf{T}_k(k[u/x]) & = & \lfloor \mathsf{T}_t(u)\rfloor\,\mathsf{to}\,x.\mathsf{T}_k(k) \\
\mathsf{T}_t(\lambda x.t) & = & \lambda x.\mathsf{T}_t(t) & \mathsf{T}_k(k\,@\,m) & = & \mathsf{T}_k(k)\,@\,\mathsf{T}_k(m) \\
\mathsf{T}_t(t\;k) & = & \mathsf{T}_t(t)\;\mathsf{T}_k(k) & \mathsf{T}_k(u :: k) & = & \lfloor \mathsf{T}_t(u)\rfloor :: \mathsf{T}_k(k) \\
\mathsf{T}_t(t[u/x]) & = & \lfloor \mathsf{T}_t(u)\rfloor\,\mathsf{to}\,x.\mathsf{T}_t(t) & \mathsf{T}_k(\varepsilon) & = & \varepsilon
\end{array}
$$

This translation is mostly homomorphic, and it reflects the simplicity of the translation necessary at the type level:

$$
\mathsf{T}[\![\,a\,]\!] = a^- \qquad \mathsf{T}[\![\,A\supset B\,]\!] = {\downarrow}\mathsf{T}[\![\,A\,]\!]\supset \mathsf{T}[\![\,B\,]\!]
$$

where all formulas are made negative by introduction of a positive shift on the left of implications to account for the positive formula expected in ***LJF***. We can now extend the translation to contexts in the obvious way, translating each formula in a given Γ separately, and prove that the translation preserves typeability.

Theorem 3.2. *The translations $\mathsf{T}_t(\cdot)$ and $\mathsf{T}_k(\cdot)$ are type-correct:*

(i) if $\Gamma\vdash t : A$ then ${\downarrow}\mathsf{T}[\![\,\Gamma\,]\!]\vdash \mathsf{T}_t(t) : \mathsf{T}[\![\,A\,]\!]$
(ii) if $\Gamma,[B]\vDash k : A$ then ${\downarrow}\mathsf{T}[\![\,\Gamma\,]\!],[\mathsf{T}[\![\,B\,]\!]]\vDash \mathsf{T}_k(k) : \mathsf{T}[\![\,A\,]\!]$

Proof. By mutual induction on the derivation. We only show the case for the ***hcl*** cut $t\;k$, others are treated similarly:

$$
\begin{array}{ll@{\qquad}r}
\mathcal{E} & :: \mathsf{T}[\![\,\Gamma\,]\!]\vdash \mathsf{T}_t(t) : \mathsf{T}[\![\,N\,]\!] & \text{by induction,} \\
\mathcal{F} & :: \mathsf{T}[\![\,\Gamma\,]\!],[\mathsf{T}[\![\,N\,]\!]]\vDash \mathsf{T}_k(k) : \mathsf{T}[\![\,M\,]\!] & \text{by induction,} \\
 & \mathsf{T}[\![\,\Gamma\,]\!]\vdash \mathsf{T}_t(t)\;\mathsf{T}_k(k) : \mathsf{T}[\![\,M\,]\!] & \text{by } \textbf{\textit{hcl}} \text{ on } \mathcal{E},\mathcal{F}.\ \square
\end{array}
$$

Observe that inside of $\lambda\kappa$, it is difficult to separate the ***LJT*** fragment by looking only at the types of terms. Indeed, a term with a type in the image of our translation might contain a cut on a formula outside of this fragment. We now consider the dynamics of $\overline{\lambda}$ and prove that reduction in this calculus is simulated through reduction in $\lambda\kappa$, after applying the translation.

Theorem 3.3. *For any t and u in $\overline{\lambda}$, if $t\to u$ then $\mathsf{T}_t(t)\to^+\mathsf{T}_t(u)$.*

Proof. By inspection of the reduction rules of $\overline{\lambda}$ and of the result of the translation $\mathsf{T}_t(\cdot)$ — all cases appear in the appendix. $\square$

Call-by-value. The second evaluation strategy commonly used in functional programming states that the β rule should only be applied when the argument is a *value* — a piece of code that has been fully reduced. In practice, this is restricted to *weak* reduction, where no reduction is performed under an abstraction, so that the term $\lambda x.t$ is considered a value. This implies some form of outermost reduction, but there is a choice between a *left-to-right* and a *right-to-left* order of evaluation when a term is applied to several arguments.

This strategy has been implemented as a λ-calculus typed with the sequent calculus ***LJQ***, as shown in Figure 3. The key construct here is the *generalised application*, $t[z = x\;u]$ similar to the `let` binding used by Moggi [27] and it leads to the representation of CBV in [10]. This form of application has already been investigated in the context of computational interpretations of ***LJ*** [22]. The grammar of this λq-calculus is:

$$
\begin{array}{rcl}
t,u & ::= & \lceil p\rceil \mid p\,\mathsf{to}\,x.t \mid t[z = x\;p] \mid t[u/x] \\
p,q & ::= & x \mid p\,\mathsf{to}\,x.q \mid \lambda x.t
\end{array}
$$

where $p\,\mathsf{to}\,x$ is an operation allowing the binding of a value, while the substitution of any non-value term is performed by $[u/x]$, representing the strict substitution never pushed until u is turned into some value and a binding is created. In ***LJQ***, the cuts follow the same scheme of *head-cuts* and *mid-cuts* as in ***LJT***, but there is no focused mid-cut. Rewritings resulting from cut elimination in ***LJQ*** correspond to a fragment of the reduction system of ***LJF*** — if we follow the scheme described in Section 2, although the system of [14] is quite different, in part due to modifications meant to prove a result of strong normalisation for the calculus. Our rules for λq are:

$$
\begin{array}{rcl}
z\,\mathsf{to}\,x.t & \to & t\{z/x\} \\
a\,\mathsf{to}\,x.\lceil p\rceil & \to & \lceil a\,\mathsf{to}\,x.p\rceil \\
a\,\mathsf{to}\,x.t[z = w\;p] & \to & (a\,\mathsf{to}\,x.t)[z = w\;(a\,\mathsf{to}\,x.p)] \\
a\,\mathsf{to}\,x.t[z = x\;p] & \to & (a\,\mathsf{to}\,x.t)[(a\,\mathsf{to}\,x.p)\,\mathsf{to}\,y.u/z] \\[4pt]
z\,\mathsf{to}\,x.p & \to & p\{z/x\} \\
a\,\mathsf{to}\,x.x & \to & a \\
a\,\mathsf{to}\,x.z & \to & z \\
a\,\mathsf{to}\,x.(\lambda z.t) & \to & \lambda z.(a\,\mathsf{to}\,x.t) \\[4pt]
t[\lceil p\rceil/x] & \to & p\,\mathsf{to}\,x.t \\
t[v[z = w\;p]/x] & \to & t[v/x][z = w\;p]
\end{array}
$$

where a always denotes the abstraction $\lambda y.u$, a *proviso* reflecting the fact that in ***LJQ***, abstraction is made into a value by placing it inside a thunk. In general, this system is obtained by translation, originated in the embedding of the CBV implication in ***LJF*** using polarity shifts.

Remark 3.4. *We are considering a calculus for CBV that is slightly different from the one found in [14] because that calculus would be overly complicated for the task at hand. Indeed, its rewrite system is specifically tailored to obtain a strong normalisation result, while we are looking for a CBV calculus that fits the general scheme.*

$$\textbf{\textit{ax}}\ \dfrac{}{\Gamma, x : A \vDash x : [A]} \qquad \textbf{\textit{ir}}\ \dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vDash \lambda x.t : [A \supset B]} \qquad \textbf{\textit{hc}}\ \dfrac{\Gamma \vDash p : [A] \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash p \,\mathtt{to}\, x.t : B} \qquad \textbf{\textit{fhc}}\ \dfrac{\Gamma \vDash p : [A] \quad \Gamma, x : A \vDash q : [B]}{\Gamma \vDash p \,\mathtt{to}\, x.q : [B]}$$

$$\textbf{\textit{foc}}\ \dfrac{\Gamma \vDash p : [A]}{\Gamma \vdash \lceil p \rceil : A} \qquad \textbf{\textit{il}}\ \dfrac{\Delta \vDash p : [A] \quad \Delta, z : B \vdash t : C}{\Delta = \Gamma, x : A \supset B \vdash t[z = x\ p] : C} \qquad \textbf{\textit{mc}}\ \dfrac{\Gamma \vdash u : A \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[u/x] : B} \qquad \textit{(no focused mid-cut)}$$

Figure 3. *Rules for the sequent calculus **LJQ** and associated terms*

Key ideas are the use of a continuation binder κ to translate a term under an application, and to place functions in thunks.

Definition 3.5. *The translations $\mathtt{Q}_t(\cdot)$ and $\mathtt{Q}_v(\cdot)$ from $\lambda\mathsf{q}$ to $\lambda\kappa$ are given by the equations:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{Q}_t(\lceil p \rceil) & = & \lceil \mathtt{Q}_v(p) \rceil & \mathtt{Q}_v(p \,\mathtt{to}\, x.q) & = & \mathtt{Q}_v(p) \,\mathtt{to}\, x.\mathtt{Q}_v(q) \\
\mathtt{Q}_t(p \,\mathtt{to}\, x.t) & = & \mathtt{Q}_v(p) \,\mathtt{to}\, x.\mathtt{Q}_t(t) & \mathtt{Q}_v(\lambda x.t) & = & \lfloor \lambda x.\mathtt{Q}_t(t) \rfloor \\
\mathtt{Q}_t(t[u/x]) & = & \mathtt{Q}_t(u)(\kappa x.\mathtt{Q}_t(t)) & \mathtt{Q}_v(x) & = & x \\
\mathtt{Q}_t(t[z = x\ p]) & = & x\ (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t)) & & &
\end{array}$$

This translation is slightly more complex than the one for ***LJT***, as it involves the introduction of shifts in two positions when translating an implication at the level of types:

$$\mathtt{Q}[\![\, a \,]\!] \;=\; a^+ \qquad \mathtt{Q}[\![\, A \supset B \,]\!] \;=\; {\downarrow}(\mathtt{Q}[\![\, A \,]\!] \supset {\uparrow}\mathtt{Q}[\![\, B \,]\!])$$

in order to turn subformulas of the original formula into positive formulas. In the case of a sequence of implications, for example in $a \supset (b \supset c)$, this will introduce a *delay* on the right of implications, as seen in ${\downarrow}(a \supset {\uparrow}{\downarrow}(b \supset {\uparrow}c))$, and this corresponds to the loss of focus in the premise of the right implication rule, reflecting the forceful integration of $\lambda x.t$ in the grammar of values. We now prove that the translation preserves typeability, as in ***LJT***.

Theorem 3.6. *The translations $\mathtt{Q}_t(\cdot)$ and $\mathtt{Q}_v(\cdot)$ are type-correct:*

(i) if $\Gamma \vdash t : A$ then $\mathtt{Q}[\![\, \Gamma \,]\!] \vdash \mathtt{Q}_t(t) : \mathtt{Q}[\![\, A \,]\!]$
(ii) if $\Gamma \vDash p : [A]$ then $\mathtt{Q}[\![\, \Gamma \,]\!] \vDash \mathtt{Q}_v(p) : [\mathtt{Q}[\![\, A \,]\!]]$

Proof. By mutual induction on the derivation. We show the case for ***il***, that is $t[z = x\ p]$ where $k = \mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t)$ so that we have $\mathtt{Q}[\![\, \Delta \,]\!] = \mathtt{Q}[\![\, \Gamma \,]\!], x : {\downarrow}(\mathtt{Q}[\![\, A \,]\!] \supset {\uparrow}\mathtt{Q}[\![\, B \,]\!])$, and:

$$\begin{array}{ll@{\qquad}r}
\mathcal{E} & :: \mathtt{Q}[\![\, \Delta \,]\!] \vDash \mathtt{Q}_v(p) : [\mathtt{Q}[\![\, A \,]\!]] & \text{by induction,} \\
\mathcal{F} & :: \mathtt{Q}[\![\, \Delta \,]\!], z : \mathtt{T}[\![\, B \,]\!] \vdash \mathtt{Q}_t(t) : \mathtt{Q}[\![\, C \,]\!] & \text{by induction,} \\
\mathcal{F}' & :: \mathtt{Q}[\![\, \Delta \,]\!][{\downarrow}\mathtt{Q}[\![\, B \,]\!]] \vDash \kappa z.t : \mathtt{Q}[\![\, C \,]\!] & \text{by } \textbf{\textit{bl}} \text{ on } \mathcal{F}, \\
\mathcal{F}'' & :: \mathtt{Q}[\![\, \Delta \,]\!][\mathtt{Q}[\![\, A \,]\!] \supset {\downarrow}\mathtt{Q}[\![\, B \,]\!]] \vDash k : \mathtt{Q}[\![\, C \,]\!] & \text{by } \textbf{\textit{il}} \text{ on } \mathcal{E}, \mathcal{F}', \\
 & \mathtt{Q}[\![\, \Delta \,]\!] \vdash x\ k : \mathtt{Q}[\![\, C \,]\!] & \text{by } \textbf{\textit{fl}} \text{ on } \mathcal{F}''. \ \square
\end{array}$$

The ***mc*** cut in $\lambda\mathsf{q}$ is a β-redex, and therefore studying which reduction equations it satisfies, we can learn that this system is *call-by-value*. The main rule is $t[\lceil p \rceil/x] \to p \,\mathtt{to}\, x.t$ which will only apply if the first premise is a value. Reading ***hc*** cuts as explicit substitutions, this is the rule that we expect for *CBV*. Because the first premise of the ***hc*** cut is a value, this guarantees that we will only substitute values in a term, and not other terms. Once again, we can prove that $\lambda\kappa$ simulates reduction in this calculus.

Theorem 3.7. *For any t and u in $\lambda\mathsf{q}$, if $t \to u$ then $\mathtt{Q}_t(t) \to^+ \mathtt{Q}_t(u)$.*

Proof. By inspection of the reduction rules of $\lambda\mathsf{q}$ and of the result of the translation $\mathtt{Q}_t(\cdot)$ — all cases appear in the appendix. $\square$

4. Evaluation Strategies and Abstract Machines

In practice, the functional programming languages using lazy and strict evaluation — based on CBN and CBV respectively — behave differently, and use implementations specific to their evaluation strategies. A detailed account of the operational semantics of a language is specified by its abstract machine: it is the theoretical representation of the code implementing an interpreter. At this level, strategies appear clearly and the transitions of two given machines can be compared intuitively — one could say that the meaning of a strategy is revealed by its abstract machine.

The assumption of this section is that since ***LJF*** can shed light on reduction strategies as a type system, it should already contain all the information needed to define an abstract machine for the strategies it can encode. We present here a *recipe* for building an abstract machine from the cut elimination procedure defined for a focused system.

Focused cuts and evaluation order. Given a focused sequent calculus, there is a simple methodology to follow when designing a cut elimination procedure based on rewrite rules: the reduction of a cut should be driven by the bottom rule in the premise where the cut formula appears unfocused. This uniquely identifies the *locus* of reduction, as opposed to the situation in an unfocused calculus. This is a requirement for the definition of an abstract machine, which is essentially a fully deterministic procedure for weak normalisation.

Following this idea, we have to consider which cut should be picked first for reduction. A machine takes a term as input and proceeds by decomposition of its structure: it is therefore natural to opt for an *outermost* reduction, where the bottom cut in a given proof is decomposed — more precisely, an abstract machine will typically implement a weak form of reduction, and thus the last rule in the proof should be a cut for any reduction to happen. In this situation, the order of reduction is deterministically specified by the aforementioned principle. But what should happen if the rule above the targeted premise of this cut is itself a cut? Nowhere in our cut elimination proof have we allowed two cuts to permute, and this would not even be a solution since this could never yield a terminating procedure. Logically, we are forced to consider the two cuts as a *block*, and by induction we observe that in general, the bottom of a proof during reduction will form a *trunk* of various cut instances stacked one above the other, and the reductions will happen at one of the leaves of this subtree.

Cut spines. Fortunately, a trunk of cuts need not be described in an abstract machine in an unspecified form: most cuts permute with other cuts and the task of designing the machine thus boils down to the specification of a certain normal form of cut trunks that will be interpreted as a *machine architecture*. We consider to this end *cut spines*, sequences of cuts stacked together.

There are two dual notions of cut spines: in a *left spine*, another smaller left spine appears in the left premise of a cut, whereas in a *right spine* the smaller spine appears in the right premise. We can describe these as lists of proofs, where the element at a node is either the right or the left premise. The topmost cut in a spine has one premise *"outside of the list"*, and this proof can be viewed as an annotated empty list — we call it the *target* of a spine. We can then write $\mathcal{F}[\mathcal{E}_n, \cdots, \mathcal{E}_1]$ and $[\mathcal{E}_1, \cdots, \mathcal{E}_n]\mathcal{F}$ for left and right spines respectively. For example, we can see the proof:

$$\dfrac{\dfrac{\mathcal{G} :: \Gamma \vdash L \quad \mathcal{F} :: \Gamma, [L] \vDash M}{\Gamma \vdash M} \qquad \mathcal{E} :: \Gamma, [M] \vDash N}{\Gamma \vdash N}$$

`TJAM` *(call-by-name machine)*

$$\begin{array}{rcl}
\langle e \mid \lambda x.t \mid u[g]\cdot c\rangle & \to & \langle e, x : u[g] \mid t \mid c\rangle \\
\langle e, x : t[g] \mid x\,k \mid c\rangle & \to & \langle g \mid t \parallel \circ \mid k[e, x : t[g]] + c\rangle \\
\langle e \mid t\ k \mid c\rangle & \to & \langle e \mid t \parallel \circ \mid k[e] + c\rangle \\
\langle e \mid t \parallel b \mid \varepsilon[g] + c\rangle & \to & \langle e \mid t \mid b\{c\}\rangle \\
\langle e \mid t \parallel b \mid (u :: k)[g] + c\rangle & \to & \langle e \mid t \parallel b\cdot u[g] \mid k[g] + c\rangle
\end{array}$$

terminal configuration: $\langle e \mid \lambda x.t \mid \circ\rangle$

`QJAM` *(call-by-value machine)*

$$\begin{array}{rcll}
\langle e \mid \lceil p\rceil \mid x.t[g]\cdot c\rangle & \to & \langle e \mid p \star x.t[g] \mid c\rangle & \\
\langle e \mid t[z = x\ p] \mid c\rangle & \to & \langle e \mid p \star y.u[f] \mid z.t[e]\cdot c\rangle & (x : (\lambda y.u)[f] \in e) \\
\langle e \mid p \,\mathsf{to}\, x.t \mid c\rangle & \to & \langle e \mid p \star x.t[e] \mid c\rangle & \\
\langle e \mid x \star z.t[g] \mid c\rangle & \to & \langle f \mid \lambda y.u \star z.t[g] \mid c\rangle & (x : (\lambda y.u)[f] \in e) \\
\langle e \mid \lambda x.u \star z.t[g] \mid c\rangle & \to & \langle g, z : (\lambda x.u)[e] \mid t \mid c\rangle &
\end{array}$$

terminal configuration: $\langle e \mid \lceil p\rceil \mid \circ\rangle$

Figure 4. *Transitions for the* `TJAM` *and* `QJAM` *machines extracted from **LJT** and **LJQ***

as the left spine $\mathcal{G}[\mathcal{F},\mathcal{E}]$ while the following proof:

$$\dfrac{\mathcal{E} :: \Gamma \vDash [P] \qquad \dfrac{\mathcal{F} :: \Gamma, x : P \vDash [Q] \quad \mathcal{G} :: \Gamma, x : P, y : Q \vdash N}{\Gamma, x : P \vdash N}}{\Gamma \vdash N}$$

is the right spine $[\mathcal{E},\mathcal{F}]\mathcal{G}$. Although one can use other groupings of cuts to organise the cut trunk at the bottom of a proof, these patterns will be the basic bricks for building our machines.

Stacks and environments. We can make one step further in the definition of the machines by observing different properties of left and right spines. The left premise of intuitionistic cuts has the cut formula in the right-hand side, and therefore they can never permute above any other cut in this premise. As a consequence, the elements of a left spine cannot be exchanged: their order is fixed and they really form a list. Conversely, cuts can permute along their right premise provided there is no actual dependency of a premise on the cut formula, and hence the elements of a right spine can be exchanged under that condition.

This leads us to the definition of two data structures commonly used in abstract machines: stacks and environments. A left spine can be used as a stack, while right spines in which all elements are *independent* proofs can be seen as an environment. In practice, it is convenient to consider all elements of an environment closed, which corresponds to proofs of sequents with the left-hand side empty. Note that in a right spine, the target proof has access to all cut formulas, each of them being bound to a name if not focused, and present in the left-hand side of the topmost right premise. A term reduced against some stack will thus be the target of a left spine, and the target of a right spine if reduced in an environment — we can easily extend the notion of target so that a term can be viewed on top of both a stack and an environment.

Program and machine syntax. When defining a machine for a λ-calculus with a focused type system, the language of programs is given, it is that of terms in this calculus. However, not all cuts should be included in the language, but rather only head cuts that are used to type a computation. Any other cut has a bureaucratic meaning, appearing as byproduct of elimination of head cuts: *the extra cuts are the logical counterpart of the machine architecture*, and they translate into particular machine configurations. In ***LJF***, for example, only ***hcl*** and ***hcr*** are unfocused head cuts, and in ***LJT*** and ***LJQ*** only ***hc*** cuts are. Note that head cuts belong to the syntax of the language but also translate to machine configurations, as they get integrated into the cut trunk during reduction:

$$\textit{machine syntax}\ \left\{\begin{array}{l} \left.\begin{array}{l}\textit{basic rules}\\ \textit{head cuts}\end{array}\right\}\ \textit{program syntax} \\ \textit{extra cuts}\end{array}\right.$$

whereas extra cuts appear by reduction inside the cut trunk and are therefore always part of the representation of the machine architecture — their reduction represents implementation details of the abstract machine.

The protocol to follow in the creation of an abstract machine for a given focused sequent calculus is in the end quite mechanical, as it consists essentially in the identification of a normal form for trunks of cuts, equivalent to the definition of an architecture for the machine, usually based on standard data structures such as environments and stacks. The rest is constrained by the logical system and its cut elimination procedure. We can now turn to the definition of a machine for $\lambda\kappa$ and its fragments, following these principles to design it on logical grounds.

Data structures for *LJF*. In the particular situation of this system, environments are built from value bindings and we will consider all elements of an environments to be closed. This can be achieved by associating to each element a local environment to close its free variables. Therefore, the environments we use are a recursive data structure. Moreover, we need lists containing values and continuations that are also closed with local environments. In precise terms, the grammar of lists and environments is:

$$\begin{array}{rcl}
c, d & ::= & \circ \mid p[e]\cdot c \mid (x.t[e])\cdot c \\
e, f, g & ::= & \circ \mid e, x : p[g]
\end{array}$$

This language is used to describe the architecture of a machine, but all of these constructs have a representation in terms of the $\lambda\kappa$-calculus. The translation is the following:

$$\begin{array}{rclrcl}
[\![r[\circ]]\!]_e & = & r & [\![r[e, x : p[g]]]\!]_e & = & [\![p[g]]\!]_e \,\mathsf{to}\, x.[\![r[e]]\!]_e \\
[\![\circ]\!]_c & = & \varepsilon & [\![p[e]\cdot c]\!]_c & = & [\![p[e]]\!]_e :: [\![c]\!]_c \\
 & & & [\![(x.t[e])\cdot c]\!]_c & = & \kappa x.([\![t[e]]\!]_e [\![c]\!]_c)
\end{array}$$

where r represents an object that is either t, q or k. Note that in the construction $(x.t[e])\cdot c$, the x is not bound in c, so that this translation into $\lambda\kappa$ does not exploit the full scope of κx. We need to add the following reductions to the rewrite system:

$$\begin{array}{rcl}
(t\ m)\ k & \to & t\ (m \mathbin{@} k) \\
p \,\mathsf{to}\, x.(t\ k) & \to & (p \,\mathsf{to}\, x.t)\ (p \,\mathsf{to}\, x.k) \\
p \,\mathsf{to}\, x.(q \,\mathsf{to}\, y.r) & \to & (p \,\mathsf{to}\, x.q) \,\mathsf{to}\, y.(p \,\mathsf{to}\, x.r)
\end{array} \qquad (4)$$

where $x \in \mathrm{fv}(q)$ and r can be have the shape of either t, p or k, so that some cuts can be permuted without looping. This is necessary to implement on $\lambda\kappa$-terms the behaviour of the sequences and environments. These reductions participate in the maintenance of the encoding in $\lambda\kappa$ of a machine configuration.

Building CBN and CBV machines. Based on ideas described above, we define abstract machines, starting with ***LJT***. The `TJAM` shown in Figure 4 is created by adding to the $\overline{\lambda}$ system the first two rules of (4) and defining machine configurations that support the kind of cuts appearing in ***LJT***. For this, we need to handle a part of context as a *difference list*, which is just a list denoted by b where an element x is added to the tail — we write $b \cdot x$ for this operation. Moreover, we denote by $b\{c\}$ the operation producing a list by plugging a list c as the tail of b. We can now describe the two kinds of configurations defining the `TJAM` architecture.

Definition 4.1 (TJAM). *The configurations of the* TJAM *are:*

$$\begin{aligned}\langle e \mid t \mid c \rangle &\triangleq (\llbracket t[e] \rrbracket_e)\, \llbracket c \rrbracket_c \\ \langle e \mid t \parallel b \mid k[g] + c \rangle &\triangleq (\llbracket t[e] \rrbracket_e)\, \llbracket b\{\llbracket k[g] \rrbracket_e \mathbin{@} \llbracket c \rrbracket_c\} \rrbracket_c\end{aligned}$$

The first kind of configuration is used to reduce terms, and the second is only necessary to normalise the structure holding lists of arguments. From the transitions in Figure 4, we see that the TJAM is essentially the KAM [23], extended to handle lists of arguments and with some explicit mechanism to normalise stacks — it is not needed in the original KAM, since it never pushes more than one argument at a time on the stack.

Remark 4.2. *The abstract machines we consider here are extremely detailed, and one might wonder if this level of precision is useful. In particular, the bureaucratic reductions maintaining the encoding of configurations could be factored out and made silent. However, it is always possible to make a machine more implicit if it represents all the details, whereas in other direction one would have to recover the precise behaviour of cut elimination that lead to the design of a particular machine. For example, stack normalisation in the* TJAM *can be made implicit, so that only a single kind of configuration is needed. If we consider the terms encoding pure λ-terms in $\overline{\lambda}$, we observe that the machine transitions are exactly the ones defining the* KAM*:*

$$\begin{aligned}\langle e \mid \lambda x.t \mid u[g] \cdot c \rangle &\to \langle e, x : u[g] \mid t \mid c \rangle \\ \langle e, x : t[g] \mid x\ \varepsilon \mid c \rangle &\to \langle g \mid t \mid c \rangle \\ \langle e \mid t\ (u :: \varepsilon) \mid c \rangle &\to \langle e \mid t \mid u[e] \cdot c \rangle\end{aligned}$$

Figure 4 also presents the QJAM machine, extracted from the ***LJQ*** cut elimination and using two configurations as well.

Definition 4.3 (QJAM). *The configurations of the* QJAM *are:*

$$\begin{aligned}\langle e \mid t \mid c \rangle &\triangleq \llbracket c\{\llbracket t[e] \rrbracket_e\} \rrbracket_c \\ \langle e \mid p \star x.t[g] \mid c \rangle &\triangleq \llbracket c\{\llbracket p[e] \rrbracket_e \ \mathtt{to}\ x.\llbracket t[g] \rrbracket_e\} \rrbracket_c\end{aligned}$$

This machine is quite different from the TJAM, as it is based on a left spine of ***mc*** cuts, representing the stack c of a configuration. It is built as opposite of the TJAM, and although it is slightly more complex, one can observe the similarity to the CEK machine [18]. Also, the reduced cut is not at the bottom of the proof anymore, but at the other end of the spine. The principle of its mechanism is that only computed values can be pushed in the environment, leading to a CBV reduction.

Multi-strategy machine. Following the same methodology, we now derive a machine for the full $\lambda\kappa$-calculus, based on the cut elimination proof for ***LJF*** from Section 2. This machine is slightly more complex that the previous ones, with three configurations.

Definition 4.4 (FJAM). *The configurations of the* FJAM *are:*

$$\begin{aligned}\langle e \mid t \mid c \rangle &\triangleq (\llbracket t[e] \rrbracket_e)\, \llbracket c \rrbracket_c \\ \langle e \mid t \parallel b \mid k[g] + c \rangle &\triangleq (\llbracket t[e] \rrbracket_e)\, \llbracket b\{\llbracket k[g] \rrbracket_e \mathbin{@} \llbracket c \rrbracket_c\} \rrbracket_c \\ \langle e \mid p \star x.t[g] \mid c \rangle &\triangleq (\llbracket p[e] \rrbracket_e \ \mathtt{to}\ x.\llbracket t[g] \rrbracket_e)\, \llbracket c \rrbracket_c\end{aligned}$$

The implementation of this machine is quite similar to that of the TJAM and the QJAM, but it unifies them by having all of their modes of evaluation. The transitions for this machine are shown in Figure 5: we can see that most of the transitions are reminiscent of either one of the previous machines. It is also interesting to see how a transition is represented in the term implementation. Consider for example these reductions:

$$\begin{aligned}(\lambda x.t)[e]\,(p[g] :: k) &\to (p[g]\ \mathtt{to}\ x.t[e])\, k \\ (\lfloor u \rfloor[e]\ \mathtt{to}\ z.t[g])\, k &\to t[g, z : \lfloor u \rfloor[e]]\, k\end{aligned}$$

expressed in terms of environments, and corresponding to the second and last transitions respectively. The basic idea of the FJAM is to let the control flow be exchanged by the operators related to a polarity shift. This illustrates how focusing provides explicit mechanisms for influencing the reduction strategy.

FJAM *(multi-strategy machine)*

$$\begin{aligned}\langle e \mid \lceil p \rceil \mid x.t[g] \cdot c \rangle &\to \langle e \mid p \star x.t[g] \mid c \rangle \\ \langle e \mid \lambda x.t \mid p[g] \cdot c \rangle &\to \langle g \mid p \star x.t[e] \mid c \rangle \\ \langle e, x : \lfloor t \rfloor[g] \mid x\ k \mid c \rangle &\to \langle g \mid t \parallel \circ \mid k[e, x : \lfloor t \rfloor[g]] + c \rangle \\ \langle e \mid t\ k \mid c \rangle &\to \langle e \mid t \parallel \circ \mid k[e] + c \rangle \\ \langle e \mid p\ \mathtt{to}\ x.t \mid c \rangle &\to \langle e \mid p \star x.t[e] \mid c \rangle\end{aligned}$$

$$\begin{aligned}\langle e \mid t \parallel b \mid \varepsilon[g] + c \rangle &\to \langle e \mid t \mid b\{c\} \rangle \\ \langle e \mid t \parallel b \mid (p :: k)[g] + c \rangle &\to \langle e \mid t \parallel c \cdot p[g] \mid k[g] + c \rangle \\ \langle e \mid t \parallel b \mid (\kappa x.t)[g] + c \rangle &\to \langle e \mid t \mid (b \cdot (x.t[g]))\{c\} \rangle\end{aligned}$$

$$\begin{aligned}\langle e, x : p[f] \mid x \star z.t[g] \mid c \rangle &\to \langle f \mid p \star z.t[g] \mid c \rangle \\ \langle e \mid \lfloor u \rfloor \star z.t[g] \mid c \rangle &\to \langle g, z : \lfloor u \rfloor[e] \mid t \mid c \rangle\end{aligned}$$

. . . .

terminal configurations: $\langle e \mid \lceil p \rceil \mid \circ \rangle \quad \langle e \mid \lambda x.t \mid \circ \rangle$

Figure 5. *Transitions for the* FJAM *machine extracted from **LJF***

The TJAM machine can be emulated inside of the FJAM machine, this will be proved by translating TJAM machine configurations into FJAM configurations, in such a way that reduction is preserved by the translation. This works by mapping terms of ***LJT*** to terms of ***LJF*** and preserving the invariant that all the values in both the context and environment are thunks. The individual components of a TJAM machine will be translated into a FJAM machine component in the following way:

$$\begin{aligned}\mathsf{T}_e\llbracket \circ \rrbracket &= \circ \\ \mathsf{T}_e\llbracket e, x : t[g] \rrbracket &= \mathsf{T}_e\llbracket e \rrbracket, x : \lfloor \mathsf{T}_t(t) \rfloor[\mathsf{T}_e\llbracket g \rrbracket] \\ \mathsf{T}_c\llbracket \circ \rrbracket &= \circ \\ \mathsf{T}_c\llbracket u[g] \cdot c \rrbracket &= \lfloor \mathsf{T}_t(u) \rfloor[\mathsf{T}_e\llbracket g \rrbracket] \cdot \mathsf{T}_c\llbracket c \rrbracket \\ \mathsf{T}_c\llbracket k[e] + c \rrbracket &= \mathsf{T}_k(k)[\mathsf{T}_e\llbracket e \rrbracket] + \mathsf{T}_c\llbracket c \rrbracket\end{aligned}$$

Then, a TJAM configuration translates to a configuration of the FJAM by applying the translation to each component, pointwise:

$$\begin{aligned}\mathsf{FT}\llbracket \langle e \mid t \mid c \rangle \rrbracket &= \langle \mathsf{T}_e\llbracket e \rrbracket \mid \mathsf{T}_t(t) \mid \mathsf{T}_c\llbracket c \rrbracket \rangle \\ \mathsf{FT}\llbracket \langle e \mid t \parallel c \mid d \rangle \rrbracket &= \langle \mathsf{T}_e\llbracket e \rrbracket \mid \mathsf{T}_t(t) \parallel \mathsf{T}_c\llbracket c \rrbracket \mid \mathsf{T}_c\llbracket d \rrbracket \rangle\end{aligned}$$

Theorem 4.5 (TJAM in FJAM). *Given any two* TJAM *configurations U and V such that $U \to V$, we have* $\mathsf{FT}\llbracket U \rrbracket \to^+ \mathsf{FT}\llbracket V \rrbracket$.

Proof. By case analysis on the TJAM configurations and on possible reductions — the full proof can be found in the appendix. □

Moreover, note that translation preserves terminal states. In a similar way, we embed the QJAM inside the FJAM. This translation is also done pointwise, although environments consist of thunks that should be considered as values. The components of the QJAM machine are translated as follows:

$$\begin{aligned}\mathsf{Q}_e\llbracket \circ \rrbracket &= \circ \\ \mathsf{Q}_e\llbracket e, x : p[g] \rrbracket &= \mathsf{Q}_e\llbracket e \rrbracket, x : \mathsf{Q}_v(p)[\mathsf{Q}_e\llbracket g \rrbracket] \\ \mathsf{Q}_s\llbracket \circ \rrbracket &= \circ \\ \mathsf{Q}_c\llbracket x.t[g] \rrbracket &= x.\mathsf{Q}_t(t)[\mathsf{Q}_e\llbracket g \rrbracket] \\ \mathsf{Q}_s\llbracket c \cdot s \rrbracket &= \mathsf{Q}_c\llbracket c \rrbracket \cdot \mathsf{Q}_s\llbracket s \rrbracket\end{aligned}$$

By applying the translations pointwise, we translate from a QJAM configuration to a FJAM configuration as follows:

$$\begin{aligned}\mathsf{FQ}\llbracket \langle e \mid t \mid s \rangle \rrbracket &= \langle \mathsf{Q}_e\llbracket e \rrbracket \mid \mathsf{Q}_t(t) \mid \mathsf{Q}_s\llbracket s \rrbracket \rangle \\ \mathsf{FQ}\llbracket \langle e \mid p \star c \mid s \rangle \rrbracket &= \langle \mathsf{Q}_e\llbracket e \rrbracket \mid \mathsf{Q}_v(p) \star \mathsf{Q}_c\llbracket c \rrbracket \mid \mathsf{Q}_s\llbracket s \rrbracket \rangle\end{aligned}$$

Theorem 4.6 (QJAM in FJAM). *Given any two* QJAM *configurations U and V such that $U \to V$, we have* $\mathsf{FQ}\llbracket U \rrbracket \to^+ \mathsf{FQ}\llbracket V \rrbracket$.

Proof. By case analysis on the QJAM configurations and on possible reductions — the full proof can be found in the appendix. □

$$\textit{ax}\ \frac{}{\Gamma, x : P \vDash x \Rightarrow P} \qquad \textit{ni}\ \frac{\Gamma \vDash p \Leftarrow P}{\Gamma \vdash \lceil p \rceil \Leftarrow \uparrow P} \qquad \textit{pi}\ \frac{\Gamma \vdash t \Leftarrow N}{\Gamma \vDash \lfloor t \rfloor \Leftarrow \downarrow N}$$

$$\textit{ie}\ \frac{\Gamma \vdash t \Rightarrow P \supset N \quad \Gamma \vDash p \Leftarrow P}{\Gamma \vdash t\ p \Rightarrow N} \qquad \textit{ii}\ \frac{\Gamma, x : P \vdash t \Leftarrow N}{\Gamma \vdash \lambda x.t \Leftarrow P \supset N}$$

$$\textit{ne}\ \frac{\Gamma \vdash u \Rightarrow \uparrow P \quad \Gamma, x : P \vdash t \Leftarrow M}{\Gamma \vdash u \texttt{ to } x.t \Leftarrow M} \qquad \textit{pe}\ \frac{\Gamma \vDash p \Rightarrow \downarrow N}{\Gamma \vdash \mathsf{F}\ p \Rightarrow N}$$

....

$$\textit{mt}\ \frac{\Gamma \vdash t \Rightarrow N \quad N \in \{a^-, \uparrow P\}}{\Gamma \vdash t \Leftarrow N} \qquad \textit{ct}\ \frac{\Gamma \vdash t \Leftarrow N}{\Gamma \vdash t \Rightarrow N}$$

$$\textit{mp}\ \frac{\Gamma \vDash p \Rightarrow P \quad P \in \{a^+, \downarrow N\}}{\Gamma \vDash p \Leftarrow P} \qquad \textit{cp}\ \frac{\Gamma \vDash p \Leftarrow P}{\Gamma \vDash p \Rightarrow P}$$

Figure 6. *Rules for bidirectional **NJPV** with associated terms*

Remark 4.7. *The* FJAM *supports the use of multiple strategies in the sense that a term can be reduced partly under the CBN order, and partly using CBV. This coexistence of opposite strategies is made possible by the extra syntax in λκ-terms, reflected by polarity shifts at the level of types. This is also the case of the call-by-push-value machine [25], as we will see in the next section.*

5. Call-by-push-value and *LJF*

As we have seen in the previous sections, the ***LJF*** system offers a versatile framework for typing λ-terms extended by advanced constructs, providing at least a partial control over the reduction strategies. Of course, the introduction of shifts at the level of types and the encodings given for CBN and CBV are reminiscent of the *call-by-push-value* language [25] in which the markers U and F establish the distinction between *value types* and *computation types*. It appears clear that there is a connection here, but this raises the question: *are **LJF** and CBPV describing precisely the same language?* This section will show that they are almost the same, but not exactly.

The type system defining CBPV is recalled in Figure 6, although we use a *bidirectional* presentation that allows to isolate normal forms as proofs not using the *coercion* rules ***ct*** and ***cp***. We name this system ***NJPV*** to emphasise that it is a natural deduction calculus. For this reason, we remove the `let` construct found in CBPV, as it can be implemented as follows:

$$\frac{\Gamma \vDash p \Leftarrow [P] \quad \Gamma, x : P \vdash t \Leftarrow N}{\Gamma \vdash \texttt{let } p \texttt{ be } x.t \Leftarrow N} \equiv \frac{\dfrac{\dfrac{\Gamma \vDash p \Leftarrow P}{\Gamma \vdash \lceil p \rceil \Leftarrow \uparrow P}}{\Gamma \vdash \lceil p \rceil \Rightarrow \uparrow P} \quad \Gamma, x : P \vdash t \Leftarrow N}{\Gamma \vdash \lceil p \rceil \texttt{ to } x.t \Leftarrow N}$$

or equivalently by a *detour* on the implication $P \supset N$. This rule corresponds to a cut in a sequent calculus, leading to a reduction in CBPV reflected as a reduction on the translation.

We have adapted the CBPV syntax to fit our general framework, but one can easily see that U is $\downarrow$ and F is $\uparrow$, making $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ stand for *thunk* and *return* respectively, while F denotes the *forcing* of a value and the CBPV application p‘t is translated into $t\ p$. Formally, we use the following grammar:

$$\begin{array}{rcl} t, u & ::= & \lambda x.t \mid t\ p \mid \mathsf{F}\,p \mid \lceil p \rceil \mid u \texttt{ to } x.t \\ p, q & ::= & x \mid \lfloor t \rfloor \end{array}$$

and the terms $t\ p$ and $\mathsf{F}\,p$ are called *synthesised* terms, while the others are *checked* terms, in reference to the form of the typing rules. The operational semantics of this language is specified by an abstract machine [25] derived from the CK machine. It has configurations containing just a term and a stack, and transitions involving the full substitution of a term for a variable in one step, as opposed to the machines we used, that performed only local operations. The transitions of the machine are:

$$\begin{array}{rcl} \langle\, u \texttt{ to } x.t \mid s\,\rangle & \to & \langle\, u \mid (\texttt{to } x.t)\cdot s\,\rangle \\ \langle\, \lceil p \rceil \mid (\texttt{to } x.t)\cdot s\,\rangle & \to & \langle\, t\{p/x\} \mid s\,\rangle \\ \langle\, \mathsf{F}\lfloor t \rfloor \mid s\,\rangle & \to & \langle\, t \mid s\,\rangle \\ \langle\, t\ p \mid s\,\rangle & \to & \langle\, t \mid p\cdot s\,\rangle \\ \langle\, \lambda x.t \mid p\cdot s\,\rangle & \to & \langle\, t\{p/x\} \mid s\,\rangle \end{array}$$

in which the stack s contains either values or term continuations, as shown in the typing rules below, found in [25], and presented in the style of a focused sequent calculus, as these objects cannot be typed in natural deduction style:

$$\textit{axl}\ \frac{}{\Gamma, [a] \vDash \circ : a} \qquad \textit{il}\ \frac{\Gamma \vDash p : [P] \quad \Gamma, [N] \vDash k : M}{\Gamma, [P \supset N] \vDash p \cdot k : M}$$

$$\textit{bc}\ \frac{\Gamma, x : P \vdash t : N \quad \Gamma, [N] \vDash k : M}{\Gamma, [\uparrow P] \vDash (x.t)\cdot k : M}$$

Since the operational semantics of CBPV is given through an abstract machine, it is natural to compare it to the FJAM we defined for ***LJF***. However, the CK machine is not an abstract machine the same sense as the ones we presented here, as it performs actual substitutions on terms instead of maintaining an environment.

In order to carry out the comparison, we introduce a variant of this CK machine where substitutions are implemented using an environment. Moreover, in order to limit inspection of the term to its surface construct, we use a value-forcing marker F. These changes require us to distinguish between a mode for evaluating terms and another mode to treat values, as in the FJAM machine. The grammar of contexts in this PVAM machine is:

$$c \ ::= \ \circ \mid p[e]\cdot c \mid (x.t[e])\cdot c \mid \mathsf{F}\cdot c$$

while environments are value bindings and programs terms of CBPV. Transitions for this machine are shown in Figure 7, and one can observe in particular that the reduction of $\mathsf{F}\lfloor t \rfloor$, performed in one step in the CK machine, requires two steps, going through *value mode*, in the PVAM machine. The syntax of the machine is consistent with our previous framework, so that transitions may be read intuitively as in the FJAM.

Natural deduction and sequent calculi. The machine we just introduced is remarkably similar to the FJAM machine we have previously extracted from ***LJF***. However, they evaluate different kinds of terms, as $\lambda\kappa$ is based on the sequent syntax of ***LJF*** and CBPV on a natural deduction presentation. This is more generally the difference separating the TJAM and QJAM machines from the more standard KAM and CEK machines.

In order to have a clear technical overview of the relation of the FJAM to the CBPV machine, we have defined a bidirectional system that is easier to relate to the sequent calculus. In particular, the coercion rules ***ct*** and ***cp*** closely correspond to cuts, and the *meet* rules ***mt*** and ***mp*** to the identity axiom.

From the perspective of our comparison, we see that CBPV and the $\lambda\kappa$-calculus, typed by ***LJF***, are not exactly the same: any term typeable in $\lambda\kappa$ must be η-long — up to atoms but also shifts, as for example $\downarrow N$ can be the type of some variable x — while in ***NJPV*** this restriction is not enforced. Of course, one could use the η-expansion result of CBPV, valid at non-atomic types, but it would hinder the correspondence between CBPV and $\lambda\kappa$. Indeed, if t has type N in CBPV and gets translated to a u of type N in ***LJF***, the η-expansion of t has type N in CBPV but the expansion of u is not always typeable in ***LJF***. To be more precise, the bijection is

PVAM *(call-by-push-value machine)*

$$\begin{array}{rcl}
\langle e \mid u \,\mathsf{to}\, x.t \mid c \rangle & \to & \langle e \mid u \mid (x.t[e]) \cdot c \rangle \\
\langle e \mid \lceil p \rceil \mid (x.t[e]) \cdot c \rangle & \to & \langle e \mid p \star (x.t[e]) \mid c \rangle \\
\langle e \mid \mathsf{F}\, p \mid c \rangle & \to & \langle e \mid p \star \mathsf{F} \cdot c \rangle \\
\langle e \mid t\; p \mid c \rangle & \to & \langle e \mid t \mid p[e] \cdot c \rangle \\
\langle e \mid \lambda x.t \mid p[g] \cdot c \rangle & \to & \langle g \mid p \star (x.t[e]) \mid c \rangle \\
\langle e, x : p[f] \mid x \star z.u[g] \mid c \rangle & \to & \langle f \mid p \star z.u[g] \mid c \rangle \\
\langle e \mid \lfloor t \rfloor \star z.u[g] \mid c \rangle & \to & \langle g, z : \lfloor t \rfloor[e] \mid u \mid c \rangle \\
\langle e \mid \lfloor t \rfloor \star \mathsf{F} \mid c \rangle & \to & \langle e \mid t \mid c \rangle
\end{array}$$

....

terminal configurations: $\langle e \mid \lceil p \rceil \mid \circ \rangle \quad \langle e \mid \lambda x.t \mid \circ \rangle$

Figure 7. *Transitions for the* PVAM *machine for CBPV*

between well-typed $\lambda\kappa$ terms and CBPV terms where all subterms of functional type are η-long. This justifies from a computational viewpoint the extension of the identity axiom to $\downarrow N$ and $\uparrow P$: these are the only non-atomic types in ***LJF*** that admit η-expansion.

Comparing machines. The translation between CBPV and $\lambda\kappa$ proceeds in two steps: first we encode the implication $- \supset -$ of CBPV as $\uparrow\downarrow(- \supset -)$ to account for η-long normal forms found in $\lambda\kappa$, and this is reflected on terms, as for example:

$$\begin{array}{rcl}
[\![t\; p]\!]_{\uparrow\downarrow} & = & [\![t]\!]_{\uparrow\downarrow} \,\mathsf{to}\, x.(\mathsf{F}\, x)\, [\![p]\!]_{\uparrow\downarrow} \\
[\![\lambda x.t]\!]_{\uparrow\downarrow} & = & \lceil \lfloor \lambda x. [\![t]\!]_{\uparrow\downarrow} \rfloor \rceil
\end{array}$$

Next, we translate the resulting terms as well as the machine configurations and observe the simulation of the transitions of these machines. In the following, we assume that CBPV terms are in η-long form.

Following the observations made about $\overline{\lambda}$ [21] we notice that eliminations in natural deduction correspond to lists in a sequent calculus, involved in an application. Similarly, to relate ***NJPV*** to ***LJF*** we will translate synthesised terms by building a list and using it as the argument in either a left focus rule or an ***hcl*** cut. We can now give the full translation:

$$\begin{array}{rcll}
\mathsf{P}_c(\lambda x.t) & = & \lambda x.\mathsf{P}_c(t) & \\
\mathsf{P}_c(\lceil p \rceil) & = & \lceil \mathsf{P}_v(p) \rceil & \\
\mathsf{P}_c(u \,\mathsf{to}\, x.t) & = & \mathsf{P}_c(u \mid \kappa x.\mathsf{P}_c(t)) & \\
\mathsf{P}_c(t) & = & \mathsf{P}_c(t \mid \varepsilon) & \textit{otherwise} \\
\mathsf{P}_c(t\; p \mid k) & = & \mathsf{P}_c(t \mid \mathsf{P}_v(p) :: k) & \\
\mathsf{P}_c(\mathsf{F}\, p \mid k) & = & \mathsf{P}_v(p \mid k) & \\
\mathsf{P}_c(t \mid k) & = & \mathsf{P}_c(t)\, k & \textit{otherwise} \\
\mathsf{P}_v(\lfloor t \rfloor) & = & \lfloor \mathsf{P}_c(t) \rfloor & \\
\mathsf{P}_v(x) & = & x & \\
\mathsf{P}_v(x \mid k) & = & x\, k & \\
\mathsf{P}_v(p \mid k) & = & \mathsf{P}_v(p) \,\mathsf{to}\, x.x\, k & \textit{otherwise}
\end{array}$$

in which the last case, for $\mathsf{P}_v(p \mid k)$, is rather uninteresting in the basic setting, where this p can only be $\lfloor t \rfloor$. This would involve more cases in the system extended with positive connectives, such as disjunction. It is then easy to prove that this translation, on pure terms, respects the typing derivations in ***NJPV*** and ***LJF***.

Theorem 5.1. *The translation from CBPV to $\lambda\kappa$ is type-correct:*

(*i*) *if* $\Gamma \vdash t \Leftarrow N$ *then* $\Gamma \vdash \mathsf{P}_c(t) : N$
(*ii*) *if* $\Gamma \vdash u \Rightarrow N$ *and* $\Gamma, [N] \vDash k : M$ *then* $\Gamma \vdash \mathsf{P}_c(u \mid k) : M$
(*iii*) *if* $\Gamma \vDash p \Leftarrow P$ *then* $\Gamma \vDash \mathsf{P}_v(p) : [P]$
(*iv*) *if* $\Gamma \vDash q \Rightarrow a^+$ *then* $\Gamma \vDash \mathsf{P}_v(q) : [a^+]$
(*v*) *if* $\Gamma \vDash q \Rightarrow \downarrow N$ *and* $\Gamma, [N] \vDash k : M$ *then* $\Gamma \vdash \mathsf{P}_v(q \mid k) : M$

Proof. The proof proceeds by mutual induction over derivations, following the structure of the translation. □

In order to proceed and translate machine configurations, we will need a notion of applicative context, which is essentially a list of application arguments with a hole for the function to be applied. Such a context will correspond in $\lambda\kappa$ to a list of values with a hole for the term applied, and it is defined as:

$$K \;::=\; \cdot \mid K\, p$$

In particular, any synthesised term u can be decomposed into an applicative context composed with either $\mathsf{F}\, p$ or some checked term t. The translation of machine states uses this fact to translate synthesised terms. The environments and stacks are translated pointwise using $\mathsf{F}_e[\![\cdot]\!]$ and $\mathsf{F}_c[\![\cdot]\!]$ respectively, and the terms are translated using $\mathsf{P}_c(\cdot)$. Note that inside a stack, the continuation construct $\mathsf{to}\; x.t[e]$ is translated to $x.\mathsf{P}_v(t)[\mathsf{F}_e[\![e]\!]]$, and a value p closed by an environment e is translated as $\mathsf{P}_v(p)[\mathsf{F}_e[\![e]\!]]$. The full translation of PVAM configurations to FJAM configurations is based on a particular recursive definition meant to handle the applicative contexts of CBPV. It is defined by three equations:

$$\begin{array}{rcl}
\mathsf{PF}[\![\langle e \mid K\{\mathsf{F}\, q\} \mid s \rangle]\!] & = & \mathsf{PF}[\![\langle e \mid q \star \mathsf{F} \mid K[e] \cdot s \rangle]\!] \\
\mathsf{PF}[\![\langle e, x : p[g] \mid x \star \mathsf{F} \mid s \rangle]\!] & = & \mathsf{PF}[\![\langle g \mid p \mid s \rangle]\!] \\
\mathsf{PF}[\![\langle e, x : p[g] \mid x \star z.u[f] \mid s \rangle]\!] & = & \mathsf{PF}[\![\langle g \mid p \star z.u[f] \mid s \rangle]\!]
\end{array}$$

where the notation $K[e] \cdot s$ denotes the integration of a context K into the stack s, specified using the equations $\cdot[e] \cdot s = s$ and $K\{p\}[e] \cdot s = K[e] \cdot (p[e] \cdot s)$. This is a meta-level operation, that helps compute the stack configuration representing a term using an applicative context. The rest of the translation, applicable after all applicative contexts have been handled, is:

$$\begin{array}{l}
\mathsf{PF}[\![\langle e \mid t \mid s \rangle]\!] \\
= \langle \mathsf{F}_e[\![e]\!] \mid \mathsf{P}_c(t) \mid \mathsf{F}_c[\![s]\!] \rangle \\
\mathsf{PF}[\![\langle e \mid \lfloor t \rfloor \star (\mathsf{to}\; x.u[g]) \mid s \rangle]\!] \\
= \langle \mathsf{F}_e[\![g, x : \lfloor t \rfloor[e]]\!] \mid \mathsf{P}_c(u) \mid \mathsf{F}_c[\![s]\!] \rangle \\
\mathsf{PF}[\![\langle e \mid \lfloor t \rfloor \star \mathsf{F} \mid s \rangle]\!] \\
= \langle \mathsf{F}_e[\![e]\!] \mid \mathsf{P}_c(t) \mid \mathsf{F}_c[\![s]\!] \rangle
\end{array}$$

We can define $\mathsf{P}_c(K\{u\} \mid m)$ to be $\mathsf{P}_c(u \mid K \,@\, m)$, where $(K \,@\, m)$ is a notation for the list specified by the equations $\cdot \,@\, m = m$ and $K\{p\} \,@\, m = K \,@\, (\mathsf{P}_v(p) :: m)$, reminiscent of the treatment given to applicative contexts in the translation of the PVAM machine. In addition, we can prove the following simple lemma, used in the final simulation result.

Lemma 5.2. *The* FJAM *can always perform the following reduction:*

$$\langle \mathsf{F}_e[\![e]\!] \mid \mathsf{P}_v(p) \star \mathsf{F}_c[\![c]\!] \mid \mathsf{F}_c[\![s]\!] \rangle \to^* \mathsf{PF}[\![\langle e \mid p \star c \mid s \rangle]\!]$$

Proof. By induction on the structure of the value p and using the definition of $\mathsf{P}_v(\cdot)$ and $\mathsf{PF}[\![\cdot]\!]$ in particular. □

We finally come to the simulation result stating that the FJAM can reduce terms as the PVAM would. Unlike the results for the TJAM and QJAM machines, the simulation does not guarantee that a step will yield at least one step inside the FJAM, because of the representation of applicative context — as can be observed in the statement of Lemma 5.2.

Theorem 5.3 (PVAM in FJAM)**.** *Given any two* PVAM *configurations* U *and* V *such that* $U \to V$*, we have* $\mathsf{PF}[\![U]\!] \to^* \mathsf{PF}[\![V]\!]$.

Proof. By case analysis on the PVAM configurations and on possible reductions. Reductions for the value mode hold immediately. Then, terms can reduce to value mode as shown by Lemma 5.2. In other cases, reduction is simulated with one step in FJAM for one step in the PVAM. □

The translation in the other direction is simpler, and it is worth pointing out that both the ***hcl*** cut and ***hcr*** cut are translated by using a coercion, exhibiting the tight relation between coercion in natural deduction and cut in the sequent calculus.

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{F}_t(\lambda x.t) & = & \lambda x.\mathsf{F}_t(t) & \mathsf{F}_k(t \mid \varepsilon) & = & t \\
\mathsf{F}_t(x\ k) & = & \mathsf{F}_k(\mathsf{F}\,x \mid k) & \mathsf{F}_k(t \mid p :: k) & = & \mathsf{F}_k(t\ \mathsf{F}_v(p) \mid k) \\
\mathsf{F}_t(\lceil p \rceil) & = & \lceil \mathsf{F}_v(p) \rceil & \mathsf{F}_k(u \mid \kappa x.t) & = & u\ \mathtt{to}\ x.\mathsf{F}_t(t) \\
\mathsf{F}_t(t\ k) & = & \mathsf{F}_k(\mathsf{F}_t(t) \mid k) & \mathsf{F}_v(x) & = & x \\
\mathsf{F}_t(p\ \mathtt{to}\ x.t) & = & (\lambda x.\mathsf{F}_t(t))\ \mathsf{F}_v(p) & \mathsf{F}_v(\lfloor t \rfloor) & = & \lfloor \mathsf{F}_t(t) \rfloor
\end{array}$$

This is once again a translation that preserves the typeability of terms from ***LJF*** to the ***NJPV*** system.

Theorem 5.4. *The translation from $\lambda\kappa$ to CBPV is type-correct:*

(i) *if* $\Gamma \vdash t : N$ *then* $\Gamma \vdash \mathsf{F}_t(t) \Leftarrow N$
(ii) *if* $\Gamma \vdash t : N$ *and* $\Gamma, [N] \vDash k : M$ *then* $\Gamma \vdash \mathsf{F}_k(t \mid k) \Leftarrow M$
(iii) *if* $\Gamma \vDash p : [P]$ *then* $\Gamma \vDash \mathsf{F}_v(p) \Leftarrow P$

Proof. By mutual induction on the given derivations, based on the translations given above and by inspection of the rules of ***NJPV*** and ***LJF***. □

A configuration of the FJAM machine will then relate to one in PVAM by a simple translation, that has in this direction no problem related to applicative contexts. Note that there are no list mode configurations in the PVAM: these are mapped the same way as the first configuration exiting the list mode. This is similar to what is done for the value mode in the other translation. The translation is defined as follows:

$$\begin{array}{l}
\mathsf{FP}[\![\, \langle e \mid t \mid c \rangle \,]\!] \\
= \langle \mathsf{P}_e[\![\, e \,]\!] \mid \mathsf{F}_t(t) \mid \mathsf{P}_c[\![\, c \,]\!] \rangle
\end{array}$$

$$\begin{array}{l}
\mathsf{FP}[\![\, \langle e \mid p \star c \ \mid\ s \rangle \,]\!] \\
= \langle \mathsf{P}_e[\![\, e \,]\!] \mid \mathsf{F}_v(p) \star \mathsf{P}_c[\![\, c \,]\!] \ \mid\ \mathsf{P}_c[\![\, s \,]\!] \rangle
\end{array}$$

$$\begin{array}{l}
\mathsf{FP}[\![\, \langle e \mid t \parallel c \mid k[g] + s \rangle \,]\!] \\
= \langle \mathsf{P}_e[\![\, e \,]\!] \mid \mathsf{F}_t(t) \mid \mathsf{P}_c[\![\, c \,]\!] + k[\mathsf{P}_e[\![\, g \,]\!]] \cdot s \rangle
\end{array}$$

where $k[e] \cdot s$ is a notation specified by the following equations:

$$\begin{array}{rcl}
\varepsilon[g] \cdot s & = & \mathsf{P}_c[\![\, s \,]\!] \\
(p :: k)[g] \cdot s & = & \mathsf{F}_v(p)[g] \cdot (k[g] \cdot s) \\
(\kappa x.t)[g] \cdot s & = & (x.\mathsf{F}_t(t)[g]) \cdot \mathsf{P}_c[\![\, s \,]\!]
\end{array}$$

and we can finally state the simulation of the FJAM machine inside its CBPV counterpart.

Theorem 5.5 (FJAM in PVAM)**.** *Given any two* FJAM *configurations U and V such that $U \to V$, we have* $\mathsf{FP}[\![\, U \,]\!] \to^* \mathsf{FP}[\![\, V \,]\!]$.

Proof. By case analysis on the FJAM configurations and on possible reductions. □

We can therefore conclude that $\lambda\kappa$ and CBPV are describing the same language, up to the question of η-expansion.

6. Interpreting Extensions of *LJF*

We now consider a few of the directions in which the preceding analysis may be extended. The first and most obvious extension would be to add more of the connectives of intuitionistic logic. As an example, the inference rules for disjunction in ***LJF*** are shown in Figure 8. In terms of the underlying proof theory, this requires a small addition to the cut elimination process, corresponding to the following reduction rules in the $\lambda\kappa$ calculus:

$$\begin{array}{rcl}
(\mathtt{inl}\,p)\ \mathtt{to}\ x.x[y.t \mid z.u] & \to & p\ \mathtt{to}\ y.(\mathtt{inl}\,p)\ \mathtt{to}\ x.t \\
(\mathtt{inr}\,p)\ \mathtt{to}\ x.x[y.t \mid z.u] & \to & p\ \mathtt{to}\ z.(\mathtt{inr}\,p)\ \mathtt{to}\ x.u \\
p\ \mathtt{to}\ x.y[z.t \mid w.u] & \to & y[z.p\ \mathtt{to}\ x.t \mid w.p\ \mathtt{to}\ x.u]
\end{array}$$

$$\textbf{\textit{dr}}_0\ \frac{\Gamma \vDash p : [P]}{\Gamma \vDash \mathtt{inl}\,p : [P \vee Q]} \qquad \textbf{\textit{dr}}_1\ \frac{\Gamma \vDash p : [Q]}{\Gamma \vDash \mathtt{inr}\,p : [P \vee Q]}$$

$$\textbf{\textit{dl}}\ \frac{\Delta, y : P \vdash t : N \quad \Delta, z : Q \vdash u : N}{\Delta = \Gamma, x : P \vee Q \vdash x[y.t \mid z.u] : N}$$

. . . .

$$\textbf{\textit{di}}_0\ \frac{\Gamma \vDash p \Leftarrow P}{\Gamma \vDash \mathtt{inl}\,p \Leftarrow P \vee Q} \qquad \textbf{\textit{di}}_1\ \frac{\Gamma \vDash p \Leftarrow Q}{\Gamma \vDash \mathtt{inr}\,p \Leftarrow P \vee Q}$$

$$\textbf{\textit{de}}\ \frac{\Gamma \vDash p \Rightarrow P \vee Q \quad \Gamma, y : P \vdash t \Leftarrow N \quad \Gamma, z : Q \vdash u \Leftarrow N}{\Gamma \vdash p[y.t \mid z.u] \Leftarrow N}$$

Figure 8. *Rules for disjunction in **LJF** above and **NJPV** below*

These changes also lead to a change in the abstract machine, in order to support this matching construction:

$$\begin{array}{rcl}
\langle e_L \mid x[y.t \mid z.u] \mid c \rangle & \to & \langle g \mid p \star y.t[e_L \mid c \rangle \\
\langle e_R \mid x[y.t \mid z.u] \mid c \rangle & \to & \langle g \mid p \star y.u[e_R \mid c \rangle \\
\langle e \mid \mathtt{inl}\,p \star y.t[g \mid c] \rangle & \to & \langle g, y : \mathtt{inl}\,p[e] \mid t \mid c \rangle \\
\langle e \mid \mathtt{inr}\,p \star y.t[g \mid c] \rangle & \to & \langle g, y : \mathtt{inr}\,p[e] \mid t \mid c \rangle
\end{array}$$

where $e_L = e, x : (\mathtt{inl}\,p)[g]$ and $e_R = e, x : (\mathtt{inr}\,p)[g]$. The same analysis as for the basic machine can be conducted here, so that the disjunction in CBPV, generalised into a sum, can be related to these additional operations in the FJAM. A number of other connectives could be added to the system, but for example conjunction being a negative would be simpler that disjunction.

Maximal Inversion. One feature of focusing that we explicitly left out in our presentation of ***LJF*** is the asynchronous maximality, which not the key feature of focused proofs — in terms of normal forms — but is often adopted, in particular for proof search. As mentioned previously, this is necessary in order to have a calculus that does not enforce η-long normal forms. With this in mind, it is informative to consider whether there is a reasonable interpretation of the added structure coming from the maximal asynchronous phases. Having atomic initial rules enforces the η-long structure, that is driven by the type of the succedent, and thus the context plays no role in determining whether a term is η-long or not.

In a similar way, we can see maximal inversion as an extension of the η-long form to a setting where we consider *open* terms. In this case, whether a term is η-long in this extended sense is driven by both the succedent and the antecedents. This would lead to another notion of expansion, slightly more complex than η since a term could be expanded *along a free variable* present in the typing context. This is of particular interest if one intends to reason about programs written in a system such as $\lambda\kappa$: normally this is done by considering η-long forms, but analysing the structure of a typed, open $\lambda\kappa$-term also requires dealing with free variables in the context. As a general observation, systems based on sequent calculi are more complicated but offer the possibility of handling open terms in a principled way, and focusing provides the necessary structure on sequent calculi.

7. Conclusion and Future Work

The fundamental idea of this paper is that focusing is not simply an important normal form in pure proof theory, but is also relevant to the computational interpretation of proof systems. We have seen how in this setting, polarities can be related to evaluation strategies in a precise way — this is sometimes viewed as folklore, and has been investigated in particular cases, but never described in a systematic fashion and in the general case.

The fact that ***LJT*** and ***LJQ*** can be identified as fragments of ***LJF*** was first noted in [26], albeit only at the level of *provability*, and not in terms of the dynamics of cut elimination. In part, this is because Liang and Miller do not specify a proof term language for the ***LJF*** calculus, and consequently it is difficult to state formally exactly how cut elimination in the three systems are related. As we have shown here, the same tight correspondence exists at the level of the operational semantics given by reduction rules.

We showed how to construct an abstract machine called FJAM for the $\lambda\kappa$-calculus by considering *"cut trunks"* as machine states, and applying the same methodology to ***LJT*** and ***LJQ*** lead to the TJAM and QJAM machines, variants of the KAM and CEK machines following the principles of sequent-based calculi. This provides a unifying framework in which the FJAM corresponds to the machine given for CBPV and integrates the mechanisms of both paradigms of functional programming, following a purely proof-theoretical methodology. The fact that CBPV subsumes other paradigms [25] is therefore explained by the fact that it harnesses the power of polarities, which is exactly what focusing does in proof theory.

An important consequence of building a calculus on a logical foundation is that extension to richer languages becomes an easy task, as shown by the history of the Curry-Howard approach. We used a bare minimum of logical connectives here: just implication, in order to be able to construct an appropriate λ-calculus. In part, this choice is forced by the comparison with ***LJT*** and ***LJQ***, since implication is the only connective shared by these systems. We note however that all the results investigated extend to the full range of connectives of ***LJF***, as hinted in Section 6.

The approach to constructing abstract machines outlined in this paper is based on viewing configurations of cuts as machine states, and as observed this results in machines that are somewhat *low-level*. Another recent approach to abstract machines is that of *distillation* in the *linear substitution calculus* [1]. It would be quite interesting to see if these approaches can be either combined or contrasted, and in particular if this can assign a motivation to the LSC methodology in terms of structural proof theory, especially focusing — we have reasons to believe so, given the roots of the LSC in proof-nets for linear logic.

The focused calculus ***LJF*** enjoys a special status among calculi for intuitionistic logic because of the added structure enforced therein by the focusing discipline. There are many other focused calculi that admit the same analysis, such as intuitionistic linear logic and its extensions with *subexponentials* [7] or *fixpoints* [4]. These logics enjoy well-behaved focused calculi, and we expect that an abstract machine can be extracted from cut elimination by following our methodology. Another interesting direction would be to apply this approach to the variants of the λ-calculus defined to interpreted classical logic with a stronger form of control.

Acknowledgments

This work is funded by grant 10-092309 of the Danish Council for Strategic Research to *Demtech* (Guenot and Gustafsson) and by the ERC Advanced Grant *ProofCert* (Brock-Nannestad).

References

[1] B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. In J. Jeuring and M. Chakravarty, editors, *ICFP'14*, pages 363–376, 2014.

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

[3] Z. Ariola, A. Bohannon, and A. Sabry. Sequent calculi and abstract machines. *ACM Transactions on Prog. Lang. and Syst.*, 31(4), 2009.

[4] D. Baelde and D. Miller. Least and greatest fixed points in linear logic. In N. Dershowitz and A. Voronkov, editors, *LPAR'07*, volume 4790 of *LNCS*, pages 92–106, 2007.

[5] H. Barendregt and S. Ghilezan. Lambda-terms for natural deduction, sequent calculus and cut elimination. *Journal of Functional Programming*, 10(1):121–134, 2000.

[6] T. Brock-Nannestad and C. Schürmann. Focused natural deduction. In C. Fermüller and A. Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS*, pages 157–171, 2010.

[7] K. Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In A. Dawar and H. Veith, editors, *CSL'10*, volume 6247 of *LNCS*, pages 185–199, 2010.

[8] K. Chaudhuri, N. Guenot, and L. Straßburger. The focused calculus of structures. In M. Bezem, editor, *CSL'11*, volume 12 of *LIPIcs*, pages 159–173, 2011.

[9] S. Cittadini and W. Sieg. Normal natural deduction proofs (in non-classical logics). In *Mechanizing Mathematical Reasoning*, 2005.

[10] P.-L. Curien and H. Herbelin. The duality of computation. In *ICFP'00*, pages 233–243, 2000.

[11] P.-L. Curien and G. Munch-Maccagnoni. The duality of computation under focus. In C. S. Calude and V. Sassone, editors, *IFIP TCS'10*, volume 323 of *AICT*, pages 165–181, 2010.

[12] 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*, number 222 in LMS Lecture Note Series, pages 211–224. 1995.

[13] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *The Journal of Symbolic Logic*, 62(03):755–807, 1997.

[14] R. Dyckhoff and S. Lengrand. LJQ: a strongly focused calculus for intuitionistic logic. In A. Beckmann and *et al.*, editors, *CiE'06*, volume 3988 of *LNCS*, pages 173–185. Springer, 2006.

[15] J. Espírito Santo. Completing Herbelin's programme. In S. Ronchi, editor, *TLCA'07*, volume 4583 of *LNCS*, pages 118–132, 2007.

[16] J. Espírito Santo. Delayed substitutions. In F. Baader, editor, *RTA'07*, volume 4533 of *LNCS*, pages 169–183, 2007.

[17] J. Espírito Santo. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems*, 45(4):963–994, 2009.

[18] M. Felleisen and D. P. Friedman. *Control operators, the SECD-machine and the λ-calculus*. Indiana University, 1986.

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

[20] J.-Y. Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[21] H. Herbelin. A λ-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.

[22] F. Joachimski and R. Matthes. Standardization and confluence for a lambda calculus with generalized applications. In L. Bachmair, editor, *RTA'00*, volume 1833 of *LNCS*, pages 141–155, 2000.

[23] J. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[24] O. Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, 2002.

[25] P. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4): 377–414, 2006.

[26] C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

[27] E. Moggi. Computational λ-calculus and monads. In *LICS'89*, pages 14–23, 1989.

[28] V. Nigam and E. Pimentel. Relating focused proofs with different polarity assignments. In *LFMTP'13*, 2013. Work in Progress.

[29] R. Simmons. Structural focalization. *ACM Transactions on Computational Logic*, 15(3):21, 2014.

[30] N. Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1-3):66–96, 2008.

A. Cut admissibility for *LJF*

The following theorem establishes cut admissibility for ***LJF*** as discussed in Section 2, with details of the cases.

Theorem A.1 (Cut admissibility)**.** *The rules **hcl**, **hcr**, **fcl**, **fcr**$^+$ and **fcr**$^-$ are admissible in **LJF**.*

Proof. The proof proceeds by lexicographical induction on the structure of the cut formula and the premises of the cut. Note that this is also a proof of weak normalisation for the reduction rules for ***LJF*** proof terms described previously. We assume the given input derivations are cut-free, corresponding to a reduction strategy that reduces the innermost redexes, corresponding to the topmost cuts in the given derivation.

Note that because we are working with focused versions of the cut rule, every cut may be reduced by only considering cases on one of the premises, specifically the premise where the cut formula is *not* under focus. In addition to this, we also show the cases where the first premise ends in an axiom rule. This is to justify the fact that we eagerly reduce cases that amount to simple variable substitution.

A.1 Cases for *hcl*

First premise ends in ***ir*** (principal case for implication):

$$\textit{\textbf{hcl}}\;\dfrac{\textit{\textbf{ir}}\;\dfrac{\overset{\mathcal{D}}{\Gamma, x : P \vdash t : N}}{\Gamma \vdash \lambda x.t : P \supset N} \qquad \textit{\textbf{il}}\;\dfrac{\overset{\mathcal{E}}{\Gamma \vDash p : [P]} \qquad \overset{\mathcal{F}}{\Gamma, [N] \vDash k : M}}{\Gamma, [P \supset N] \vDash p :: k : M}}{\Gamma \vdash (\lambda x.t)\,(p :: k) : M}$$

$\mathcal{D}' :: \Gamma \vdash p\ \mathsf{to}\ x.t : N$ by ***hcr*** on $(P, \mathcal{E}, \mathcal{D})$.
$\Gamma \vdash (p\ \mathsf{to}\ x.t)\ k : M$ by ***hcl*** on $(N, \mathcal{D}', \mathcal{F})$.

First premise ends in ***fr*** (principal case for negative shift):

$$\textit{\textbf{hcl}}\;\dfrac{\textit{\textbf{fr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]}}{\Gamma \vdash \lceil p \rceil : \uparrow P} \qquad \textit{\textbf{bl}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : P \vdash t : M}}{\Gamma, [\uparrow P] \vDash \kappa x.t : M}}{\Gamma \vdash \lceil p \rceil\,(\kappa x.t) : M}$$

$\Gamma \vdash p\ \mathsf{to}\ x.t : M$ by ***hcr*** on $(P, \mathcal{D}, \mathcal{E})$.

First premise ends in ***fl*** (left-commutative case):

$$\textit{\textbf{hcl}}\;\dfrac{\textit{\textbf{fl}}\;\dfrac{\overset{\mathcal{D}}{\Gamma, x : \downarrow M, [M] \vDash k : N}}{\Gamma, x : \downarrow M \vdash x\ k : N} \qquad \overset{\mathcal{E}}{\Gamma, x : \downarrow M, [N] \vdash m : L}}{\Gamma, x : \downarrow M \vdash (x\ k)\ m : L}$$

$\Gamma, x : \downarrow M, [M] \vDash k \mathbin{@} m : L$ by ***fcl*** on $(N, \mathcal{D}, \mathcal{E})$.
$\Gamma, x : \downarrow M \vdash x\ (k \mathbin{@} m) : L$ by ***fl***.

A.2 Cases for *hcr*:

First premise ends in ***axr***:

$$\textit{\textbf{hcr}}\;\dfrac{\textit{\textbf{axr}}\;\dfrac{}{\Gamma, x : P \vDash x : [P]} \qquad \overset{\mathcal{E}}{\Gamma, x : P, y : P \vdash t : N}}{\Gamma, x : P \vdash x\ \mathsf{to}\ y.t : N}$$

$\Gamma, x : P \vdash t\{x/y\} : N$ by renaming substitution.

Second premise ends in ***fl*** (principal case for positive shift):

$$\textit{\textbf{hcr}}\;\dfrac{\textit{\textbf{fr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vdash u : M}}{\mathcal{D}' :: \Gamma \vDash \lfloor u \rfloor : [\downarrow M]} \qquad \textit{\textbf{bl}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : \downarrow M, [M] \vDash k : N}}{\Gamma, x : \downarrow M \vdash x\ k : N}}{\Gamma \vdash \lfloor u \rfloor\ \mathsf{to}\ x.(x\ k) : N}$$

$\mathcal{E}' :: \Gamma, [M] \vDash \lfloor u \rfloor\ \mathsf{to}\ x.k : N$ by ***fcr***$^-$ on $(\downarrow M, \mathcal{D}', \mathcal{E})$.
$\Gamma \vdash u\ (\lfloor u \rfloor\ \mathsf{to}\ x.k) : N$ by ***hcl*** on $(M, \mathcal{D}, \mathcal{E}')$.

Second premise ends in ***fl*** (right-commutative case):

$$\textit{\textbf{hcr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]} \qquad \textit{\textbf{fl}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : P, y : \downarrow M, [M] \vDash k : N}}{\Gamma, x : P, y : \downarrow M \vdash y\ k : N}}{\Gamma, y : \downarrow M \vdash p\ \mathsf{to}\ x.(y\ k) : N}$$

$\Gamma, y : \downarrow M, [M] \vDash p\ \mathsf{to}\ x.k : N$ by ***fcr***$^-$ on $(P, \mathcal{D}, \mathcal{E})$.
$\Gamma, y : \downarrow M \vdash y\ (p\ \mathsf{to}\ x.k) : N$ by ***fl***.

Second premise ends in ***ir*** (right-commutative case):

$$\textit{\textbf{hcr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]} \qquad \textit{\textbf{ir}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : P, y : Q \vdash t : N}}{\Gamma, x : P \vdash \lambda y.t : Q \supset N}}{\Gamma \vdash p\ \mathsf{to}\ x.(\lambda y.t) : Q \supset N}$$

$\Gamma, y : Q \vdash p\ \mathsf{to}\ x.t : N$ by ***hcr*** on $(P, \mathcal{D}, \mathcal{E})$.
$\Gamma \vdash \lambda y.(p\ \mathsf{to}\ x.k) : Q \supset N$ by ***ir***.

Second premise ends in ***fr*** (right-commutative case):

$$\textit{\textbf{hcr}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]} \qquad \textit{\textbf{fr}}\;\dfrac{\overset{\mathcal{E}}{\Gamma, x : P \vDash q : [Q]}}{\Gamma, x : P \vdash \lceil q \rceil : \uparrow Q}}{\Gamma \vdash p\ \mathsf{to}\ x.\lceil q \rceil : \uparrow Q}$$

$\Gamma \vDash p\ \mathsf{to}\ x.q : [Q]$ by ***fcr***$^+$ on $(P, \mathcal{D}, \mathcal{E})$.
$\Gamma \vdash \lceil p\ \mathsf{to}\ x.q \rceil : \uparrow Q$ by ***fr***.

A.3 Cases for *fcl*:

First premise ends in ***axl***:

$$\textit{\textbf{fcl}}\;\dfrac{\textit{\textbf{axl}}\;\dfrac{}{\Gamma, [M] \vDash \varepsilon : M} \qquad \overset{\mathcal{E}}{\Gamma, [M] \vDash m : L}}{\Gamma, [M] \vDash \varepsilon \mathbin{@} m : L}$$

$\Gamma, [M] \vDash m : L$ by $\mathcal{E}$.

First premise ends in ***il*** (left-commutative case):

$$\textit{\textbf{fcl}}\;\dfrac{\textit{\textbf{il}}\;\dfrac{\overset{\mathcal{D}}{\Gamma \vDash p : [P]} \qquad \overset{\mathcal{E}}{\Gamma, [M] \vDash k : N}}{\Gamma, [P \supset M] \vDash p :: k : N} \qquad \overset{\mathcal{F}}{\Gamma, [N] \vDash m : L}}{\Gamma, [P \supset M] \vDash (p :: k) \mathbin{@} m : L}$$

$\mathcal{E}' :: \Gamma, [M] \vDash k \mathbin{@} m : L$ by ***fcl*** on $(N, \mathcal{E}, \mathcal{F})$.
$\Gamma, [P \supset M] \vDash p :: (k \mathbin{@} m) : L$ by ***il*** on $\mathcal{D}$ and $\mathcal{E}'$.

First premise ends in ***bl*** (left-commutative case):

$$\textit{\textbf{fcl}}\;\dfrac{\textit{\textbf{il}}\;\dfrac{\overset{\mathcal{D}}{\Gamma, x : P \vdash t : N}}{\Gamma, [\uparrow P] \vDash \kappa x.t : N} \qquad \overset{\mathcal{E}}{\Gamma, [N] \vDash m : L}}{\Gamma, [\uparrow P] \vDash (\kappa x.t) \mathbin{@} m : L}$$

$\Gamma, x : P \vdash t\ m : L$ by ***hcl*** on $(N, \mathcal{D}, \mathcal{E})$.
$\Gamma, [\uparrow P] \vDash \kappa x.(t\ m) : L$ by ***bl***.

A.4 Cases for *fcr*$^+$:

First premise ends in ***axr***:

$$\textit{fcr}^+\ \dfrac{\textit{axr}\ \dfrac{}{\Gamma, x : P \vDash x : [P]} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma, x : P, y : P \vDash p : [Q]\end{array}}{\Gamma, x : P \vDash x\ \mathsf{to}\ y.p : [Q]}$$

$\Gamma, x : P \vDash p\{x/y\} : [Q]$ by renaming substitution.

Second premise ends in ***axr*** (right-commutative case):

$$\textit{fcr}^+\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, y : P \vDash p : [P]\end{array} \qquad \textit{axr}\ \dfrac{}{\Gamma, x : P, y : P \vDash y : [Q]}}{\Gamma, y : P \vDash p\ \mathsf{to}\ x.y : [Q]}$$

$\Gamma, y : P \vDash y : [Q]$ by ***axr***.

Second premise ends in ***axr*** (principal case):

$$\textit{fcr}^+\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vDash p : [P]\end{array} \qquad \textit{axr}\ \dfrac{}{\Gamma, x : P \vDash x : [P]}}{\Gamma \vDash p\ \mathsf{to}\ x.x : [P]}$$

$\Gamma \vDash p : [P]$ by $\mathcal{D}$.

Second premise ends in ***br*** (right-commutative case):

$$\textit{fcr}^+\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vDash p : [P]\end{array} \qquad \textit{br}\ \dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma, x : P \vdash u : N\end{array}}{\Gamma, x : P \vDash \lfloor u \rfloor : [\downarrow N]}}{\Gamma \vDash p\ \mathsf{to}\ x.\lfloor u \rfloor : [\downarrow N]}$$

$\Gamma \vdash p\ \mathsf{to}\ x.u : N$ by ***hcr*** on $(P, \mathcal{D}, \mathcal{E})$.
$\Gamma \vDash \lfloor p\ \mathsf{to}\ x.u \rfloor : [\downarrow N]$ by ***br***.

A.5 Cases for *fcr*$^-$:

First premise ends in ***axr***:

$$\textit{fcr}^-\ \dfrac{\textit{axr}\ \dfrac{}{\Gamma, x : P \vDash x : [P]} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma, x : P, y : P, [N] \vDash k : M\end{array}}{\Gamma, x : P, [N] \vDash x\ \mathsf{to}\ y.k : M}$$

$\Gamma, x : P, [N] \vDash k\{x/y\} : M$ by renaming substitution.

Second premise ends in ***axl***:

$$\textit{fcr}^-\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vDash p : [P]\end{array} \qquad \textit{axl}\ \dfrac{}{\Gamma, x : P, [N] \vDash \varepsilon : N}}{\Gamma, [N] \vDash p\ \mathsf{to}\ x.\varepsilon : N}$$

$\Gamma, [N] \vDash \varepsilon : N$ by ***axl***.

Second premise ends in ***il***:

$$\textit{fcr}^-\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vDash p : [P]\end{array} \qquad \textit{il}\ \dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma, x : P \vDash q : [Q]\end{array} \qquad \begin{array}{c}\mathcal{F}\\ \Gamma, x : P, [N] \vDash k : M\end{array}}{\Gamma, x : P, [Q \supset N] \vDash q :: k : M}}{\Gamma, [Q \supset N] \vDash p\ \mathsf{to}\ x.(p :: k) : M}$$

$\mathcal{E}' :: \Gamma \vDash p\ \mathsf{to}\ x.q : [Q]$ by ***fcr***$^+$ on $(P, \mathcal{D}, \mathcal{E})$.
$\mathcal{F}' :: \Gamma, [N] \vDash p\ \mathsf{to}\ x.k : M$ by ***fcr***$^-$ on $(P, \mathcal{D}, \mathcal{F})$.
$\Gamma, [Q \supset N] \vDash (p\ \mathsf{to}\ x.q) :: (p\ \mathsf{to}\ x.k) : M$ by ***il*** on $\mathcal{E}'$ and $\mathcal{F}'$.

Second premise ends in ***bl***:

$$\textit{fcr}^-\ \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vDash p : [P]\end{array} \qquad \textit{bl}\ \dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma, x : P, y : Q \vdash t : M\end{array}}{\Gamma, x : P, [\uparrow Q] \vDash \kappa y.t : M}}{\Gamma, [\uparrow Q] \vDash p\ \mathsf{to}\ x.(\kappa y.t) : M}$$

$\Gamma, y : Q \vdash p\ \mathsf{to}\ x.t : M$ by ***hcr*** on $(P, \mathcal{D}, \mathcal{E})$.
$\Gamma, [\uparrow Q] \vDash \kappa y.(p\ \mathsf{to}\ x.t) : M$ by ***bl***.

This completes the proof □

B. $\overline{\lambda}$ and λq reductions simulated in $\lambda\kappa$

Reductions in $\overline{\lambda}$ can be simulated in $\lambda\kappa$ using the translations $\mathsf{T}_t(\cdot)$ and $\mathsf{T}_k(\cdot)$ defined in Section 3.

Theorem B.1. *For any t and u in $\overline{\lambda}$, if $t \to u$ then $\mathsf{T}_t(t) \to^+ \mathsf{T}_t(u)$.*

Proof. By inspection of the reduction rules of $\overline{\lambda}$ and of the result of the translation $\mathsf{T}_t(\cdot)$:

$$\begin{aligned}
\mathsf{T}_t((\lambda x.t)(u :: k)) &= (\lambda x.\mathsf{T}_t(t))(\lfloor \mathsf{T}_t(u) \rfloor :: \mathsf{T}_k(k))\\
&\to \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_t(t)\\
&= \mathsf{T}_t(t[u/x]\ k)\\
\mathsf{T}_t((x\ k)\ m) &= (x\ \mathsf{T}_k(k))\ \mathsf{T}_k(m)\\
&\to x\ (\mathsf{T}_k(k) \mathbin{@} \mathsf{T}_k(m))\\
&= \mathsf{T}_t(x\ (k \mathbin{@} m))\\
\mathsf{T}_t((\lambda z.t)[u/x]) &= \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\lambda z.\mathsf{T}_t(t)\\
&\to \lambda z.\lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_t(t)\\
&= \mathsf{T}_t(\lambda z.t[u/x])\\
\mathsf{T}_t((x\ k)[u/x]) &= \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.x\ \mathsf{T}_k(k)\\
&\to \mathsf{T}_t(u)\ (\lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_k(k))\\
&= \mathsf{T}_t(u\ k[u/x])\\
\mathsf{T}_t((y\ k)[u/x]) &= \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.y\ \mathsf{T}_k(k)\\
&\to y\ (\lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_k(k))\\
&= \mathsf{T}_t(y\ k[u/x])
\end{aligned}$$

□

Theorem B.2. *For any lists k and m in $\overline{\lambda}$, if $t \to u$ then $\mathsf{T}_k(t) \to^+ \mathsf{T}_k(u)$.*

Proof. By inspection of the reduction rules of $\overline{\lambda}$ and of the result of the translation $\mathsf{T}_k(\cdot)$:

$$\begin{aligned}
\mathsf{T}_k(\varepsilon \mathbin{@} m) &= \varepsilon \mathbin{@} \mathsf{T}_k(m)\\
&\to \mathsf{T}_k(m)\\
\mathsf{T}_k((u :: k) \mathbin{@} m) &= (\lfloor \mathsf{T}_t(u) \rfloor :: \mathsf{T}_k(k)) \mathbin{@} \mathsf{T}_k(m)\\
&\to \lfloor \mathsf{T}_t(u) \rfloor :: (\mathsf{T}_k(k) \mathbin{@} \mathsf{T}_k(m))\\
&= \mathsf{T}_k(u :: (k \mathbin{@} m))\\
\mathsf{T}_k(\varepsilon[u/x]) &= \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\varepsilon\\
&\to \varepsilon\\
&= \mathsf{T}_k(\varepsilon)\\
\mathsf{T}_k((t :: k)[u/x]) &= \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.(\lfloor \mathsf{T}_t(t) \rfloor :: \mathsf{T}_k(k)\\
&\to (\lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\lfloor \mathsf{T}_t(t) \rfloor) :: \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_k(k)\\
&\to \lfloor \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_t(t) \rfloor :: \lfloor \mathsf{T}_t(u) \rfloor\ \mathsf{to}\ x.\mathsf{T}_k(k)\\
&= \mathsf{T}_k(t[u/x] :: k[u/x])
\end{aligned}$$

□

Reductions in λq can be simulated in $\lambda\kappa$ using the translations $\mathsf{Q}_t(\cdot)$ and $\mathsf{Q}_v(\cdot)$ defined in Section 3.

Theorem B.3. *For any t and u in λq, if $t \to u$ then $\mathsf{Q}_t(t) \to^+ \mathsf{Q}_t(u)$.*

Proof. By inspection of the reduction rules of $\lambda\mathsf{q}$ and of the result of the translation $\mathtt{Q}_t(\cdot)$, notice that for all t' and variables x, y the translation $\mathtt{Q}_t(t'\{x/y\}) = \mathtt{Q}_t(t')\{x/y\}$, furthermore let $a = \lambda y.u'$, so $\mathtt{Q}_v(a) = \lfloor \lambda y.\mathtt{Q}_t(u') \rfloor$:

$$\begin{aligned}
&\mathtt{Q}_t(z \;\mathtt{to}\; x.t)\\
&= z \;\mathtt{to}\; x.\mathtt{Q}_t(t)\\
&\to \mathtt{Q}_t(t)\{z/x\}\\
&= \mathtt{Q}_t(t\{z/x\})
\end{aligned}$$

$$\begin{aligned}
&\mathtt{Q}_t(a \;\mathtt{to}\; x.\lceil p \rceil)\\
&= \mathtt{Q}_v(a) \;\mathtt{to}\; x.\lceil \mathtt{Q}_v(p) \rceil\\
&\to \lceil \mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p) \rceil\\
&= \mathtt{Q}_t(\lceil a \;\mathtt{to}\; x.p \rceil)
\end{aligned}$$

$$\begin{aligned}
&\mathtt{Q}_t(a \;\mathtt{to}\; x.t[z = w\, p])\\
&= \mathtt{Q}_v(a) \;\mathtt{to}\; x.w\, (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t))\\
&\to w\, (\mathtt{Q}_v(a) \;\mathtt{to}\; x.(\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t)))\\
&\to w\, ((\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p)) :: \mathtt{Q}_v(a) \;\mathtt{to}\; x.\kappa z.\mathtt{Q}_t(t))\\
&\to w\, ((\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p)) :: \kappa z.\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_t(t))\\
&= \mathtt{Q}_t((a \;\mathtt{to}\; x.t)[z = w\, (a \;\mathtt{to}\; x.p)])
\end{aligned}$$

$$\begin{aligned}
&\mathtt{Q}_t(a \;\mathtt{to}\; x.t[z = x\, p])\\
&= \mathtt{Q}_v(a) \;\mathtt{to}\; x.x\, (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t))\\
&\to (\lambda y.\mathtt{Q}_t(u'))\, (\mathtt{Q}_v(a) \;\mathtt{to}\; x.(\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t)))\\
&\to (\lambda y.\mathtt{Q}_t(u'))\, ((\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p)) :: \mathtt{Q}_v(a) \;\mathtt{to}\; x.\kappa z.\mathtt{Q}_t(t))\\
&\to (\lambda y.\mathtt{Q}_t(u'))\, ((\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p)) :: \kappa z.(\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_t(t)))\\
&\to ((\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_v(p)) \;\mathtt{to}\; y.\mathtt{Q}_t(u'))(\kappa z.(\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_t(t)))\\
&= \mathtt{Q}_t((a \;\mathtt{to}\; x.t)[(a \;\mathtt{to}\; x.p) \;\mathtt{to}\; y.u/z)
\end{aligned}$$

$$\begin{aligned}
&\mathtt{Q}_t(t[\lceil p \rceil/x])\\
&= \lceil \mathtt{Q}_v(p) \rceil\, (\kappa x.\mathtt{Q}_t(t))\\
&\to \mathtt{Q}_v(p) \;\mathtt{to}\; x.\mathtt{Q}_t(t)\\
&= \mathtt{Q}_t(p \;\mathtt{to}\; x.t)
\end{aligned}$$

$$\begin{aligned}
&\mathtt{Q}_t(t[v[z = w\, p]/x])\\
&= (y\, (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(v)))\, (\kappa x.\mathtt{Q}_t(t))\\
&\to y\, ((\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(v)) \mathbin{@} (\kappa x.\mathtt{Q}_t(t)))\\
&\to y\, (\mathtt{Q}_v(p) :: ((\kappa z.\mathtt{Q}_t(v)) \mathbin{@} (\kappa x.\mathtt{Q}_t(t))))\\
&\to y\, (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(v)\, (\kappa x.\mathtt{Q}_t(t)))\\
&= \mathtt{Q}_t(t[v/x][z = w\, p])
\end{aligned}$$

□

Theorem B.4. *For any value p and q in $\lambda\mathsf{q}$, if $p \to q$ then $\mathtt{Q}_v(p) \to^+ \mathtt{Q}_v(q)$.*

Proof. By inspection of the reduction rules of $\lambda\mathsf{q}$ and of the result of the translation $\mathtt{Q}_v(\cdot)$, notice that for all p' and variables x, y the translation $\mathtt{Q}_v(p'\{x/y\}) = \mathtt{Q}_v(p')\{x/y\}$, furthermore let $a = \lambda y.u'$, so $\mathtt{Q}_v(a) = \lfloor \lambda y.\mathtt{Q}_t(u') \rfloor$:

$$\begin{aligned}
\mathtt{Q}_v(z \;\mathtt{to}\; x.p) &= z \;\mathtt{to}\; x.\mathtt{Q}_v(p)\\
&\to \mathtt{Q}_v(p)\{z/x\}\\
&= \mathtt{Q}_v(p\{z/x\})\\
\mathtt{Q}_v(a \;\mathtt{to}\; x.x) &= \mathtt{Q}_v(a) \;\mathtt{to}\; x.x\\
&\to \mathtt{Q}_v(a)\\
\mathtt{Q}_v(a \;\mathtt{to}\; x.z) &= \mathtt{Q}_v(a) \;\mathtt{to}\; x.z\\
&\to z\\
&= \mathtt{Q}_v(z)\\
\mathtt{Q}_v(a \;\mathtt{to}\; x.(\lambda z.t)) &= \mathtt{Q}_v(a) \;\mathtt{to}\; x.\lfloor \lambda z.\mathtt{Q}_t(t) \rfloor\\
&\to \lfloor \mathtt{Q}_v(a) \;\mathtt{to}\; x.\lambda z.\mathtt{Q}_t(t) \rfloor\\
&\to \lfloor \lambda z.\mathtt{Q}_v(a) \;\mathtt{to}\; x.\mathtt{Q}_t(t) \rfloor\\
&= \mathtt{Q}_v(\lambda z.a \;\mathtt{to}\; x.t)
\end{aligned}$$

□

C. `TJAM` and `QJAM` simulated inside `FJAM`

The `TJAM` machine can be simulated inside `FJAM` as discussed in Section 4.

Theorem C.1 (`TJAM` in `FJAM`)**.** *Given any two* `TJAM` *configurations U and V such that $U \to V$, we have $\mathtt{FT}\llbracket\, U \,\rrbracket \longrightarrow^* \mathtt{FT}\llbracket\, V \,\rrbracket$.*

Proof. By case analysis on the `TJAM` configurations and on possible reductions, notice that for all b and d the translation $\mathtt{T}_c\llbracket\, b\{d\} \,\rrbracket = \mathtt{T}_c\llbracket\, b \,\rrbracket\{\mathtt{T}_c\llbracket\, d \,\rrbracket\}$:

$$\begin{aligned}
&\;\; \mathtt{FT}\llbracket\, \langle e \mid \lambda x.t \mid u[g] \cdot c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \lambda x.\mathtt{T}_t(t) \mid \lfloor \mathtt{T}_t(u) \rfloor[\mathtt{T}_e\llbracket\, g \,\rrbracket] \cdot \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, g \,\rrbracket \mid \lfloor \mathtt{T}_t(u) \rfloor \star x.\mathtt{T}_t(t)[\mathtt{T}_e\llbracket\, e \,\rrbracket] \mid \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, e \,\rrbracket, x : \lfloor \mathtt{T}_t(u) \rfloor[\mathtt{T}_e\llbracket\, g \,\rrbracket] \mid \mathtt{T}_t(t) \mid \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FT}\llbracket\, \langle e, x : u[g] \mid t \mid c \rangle \,\rrbracket
\end{aligned}$$

$$\begin{aligned}
&\;\; \mathtt{FT}\llbracket\, \langle e, x : t[g] \mid x\, k \mid c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{T}_e\llbracket\, e \,\rrbracket, x : \lfloor \mathtt{T}_t(t) \rfloor[\mathtt{T}_e\llbracket\, g \,\rrbracket] \mid x\, \mathtt{T}_k(k) \mid \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, g \,\rrbracket \mid \mathtt{T}_t(t) \parallel \circ \mid \mathtt{T}_k(k)[\mathtt{T}_e\llbracket\, e \,\rrbracket, x : \lfloor \mathtt{T}_t(t) \rfloor[\mathtt{T}_e\llbracket\, g \,\rrbracket]] + \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FT}\llbracket\, \langle g \mid t \parallel \circ \mid k[e, x : t[g]] + c \rangle \,\rrbracket
\end{aligned}$$

$$\begin{aligned}
&\;\; \mathtt{FT}\llbracket\, \langle e \mid t\; k \mid c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t)\; \mathtt{T}_k(k) \mid \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t) \parallel \circ \mid \mathtt{T}_k(k)[\mathtt{T}_e\llbracket\, e \,\rrbracket] + \mathtt{T}_c\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FT}\llbracket\, \langle e \mid t \parallel \circ \mid k[e] + c \rangle \,\rrbracket
\end{aligned}$$

$$\begin{aligned}
&\;\; \mathtt{FT}\llbracket\, \langle e \mid t \parallel b \mid \varepsilon[g] + d \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t) \parallel \mathtt{T}_c\llbracket\, b \,\rrbracket \mid \varepsilon[\mathtt{T}_e\llbracket\, g \,\rrbracket] + \mathtt{T}_c\llbracket\, d \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t) \mid \mathtt{T}_c\llbracket\, b \,\rrbracket\{\mathtt{T}_c\llbracket\, d \,\rrbracket\} \rangle\\
&= \;\; \mathtt{FT}\llbracket\, \langle e \mid t \mid b\{d\} \rangle \,\rrbracket
\end{aligned}$$

$$\begin{aligned}
&\;\; \mathtt{FT}\llbracket\, \langle e \mid t \parallel b \mid (u :: k)[g] + d \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t) \parallel \mathtt{T}_c\llbracket\, b \,\rrbracket \mid (\lfloor \mathtt{T}_t(u) \rfloor :: \mathtt{T}_k(k))[\mathtt{T}_e\llbracket\, g \,\rrbracket] + \mathtt{T}_c\llbracket\, d \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{T}_e\llbracket\, e \,\rrbracket \mid \mathtt{T}_t(t) \parallel \mathtt{T}_c\llbracket\, b \,\rrbracket \cdot \lfloor \mathtt{T}_t(u) \rfloor[\mathtt{T}_e\llbracket\, g \,\rrbracket] \mid \mathtt{T}_k(k)[\mathtt{T}_e\llbracket\, g \,\rrbracket] + \mathtt{T}_c\llbracket\, d \,\rrbracket \rangle\\
&= \;\; \mathtt{FT}\llbracket\, \langle e \mid t \parallel b \cdot u[g] \mid k[g] + d \rangle \,\rrbracket
\end{aligned}$$

□

The `QJAM` machine can be simulated inside `FJAM` as discussed in Section 4.

Theorem C.2 (`QJAM` in `FJAM`)**.** *Given any two* `QJAM` *configurations U and V such that $U \to V$, we have $\mathtt{FQ}\llbracket\, U \,\rrbracket \longrightarrow^+ \mathtt{FQ}\llbracket\, V \,\rrbracket$.*

Proof. By case analysis on the `QJAM` configurations and on possible reductions:

$$\begin{aligned}
&\;\; \mathtt{FQ}\llbracket\, \langle e \mid \lceil p \rceil \mid x.t[g] \cdot c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \lceil \mathtt{Q}_v(p) \rceil \mid x.\mathtt{Q}_t(t)[\mathtt{Q}_e\llbracket\, g \,\rrbracket] \cdot \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \mathtt{Q}_v(p) \star x.\mathtt{Q}_t(t)[\mathtt{Q}_e\llbracket\, g \,\rrbracket] \mid \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FQ}\llbracket\, \langle e \mid p \star x.t[g] \mid c \rangle \,\rrbracket
\end{aligned}$$

Let $x : (\lambda y.u)[f] \in e$ and $x : \lfloor \lambda y.\mathtt{Q}_t(u) \rfloor[\mathtt{Q}_e\llbracket\, f \,\rrbracket] \in \mathtt{Q}_e\llbracket\, e \,\rrbracket$

$$\begin{aligned}
&\;\; \mathtt{FQ}\llbracket\, \langle e \mid t[z = x\, p] \mid c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid x\, (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t)) \mid \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, f \,\rrbracket \mid \lambda y.\mathtt{Q}_t(u) \parallel \circ \mid (\mathtt{Q}_v(p) :: \kappa z.\mathtt{Q}_t(t))[\mathtt{Q}_e\llbracket\, e \,\rrbracket] + \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, f \,\rrbracket \mid \lambda y.\mathtt{Q}_t(u) \parallel \mathtt{Q}_v(p)[\mathtt{Q}_e\llbracket\, e \,\rrbracket] \mid (\kappa z.\mathtt{Q}_t(t))[\mathtt{Q}_e\llbracket\, e \,\rrbracket] + \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, f \,\rrbracket \mid \lambda y.\mathtt{Q}_t(u) \mid \mathtt{Q}_v(p)[\mathtt{Q}_e\llbracket\, e \,\rrbracket] \cdot z.\mathtt{Q}_t(t)[\mathtt{Q}_e\llbracket\, e \,\rrbracket] \cdot \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \mathtt{Q}_v(p) \star y.\mathtt{Q}_t(u)[\mathtt{Q}_e\llbracket\, f \,\rrbracket] \mid z.\mathtt{Q}_t(t)[\mathtt{Q}_e\llbracket\, e \,\rrbracket] \cdot \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FQ}\llbracket\, \langle e \mid p \star y.u[f] \mid z.t[e] \cdot c \rangle \,\rrbracket
\end{aligned}$$

$$\begin{aligned}
&\;\; \mathtt{FQ}\llbracket\, \langle e \mid p \;\mathtt{to}\; x.t \mid c \rangle \,\rrbracket\\
&= \;\; \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \mathtt{Q}_v(p) \;\mathtt{to}\; x.\mathtt{Q}_t(t) \mid \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&\longrightarrow \langle \mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \mathtt{Q}_v(p) \star x.\mathtt{Q}_t(t)[\mathtt{Q}_e\llbracket\, e \,\rrbracket \mid \mathtt{Q}_s\llbracket\, c \,\rrbracket \rangle\\
&= \;\; \mathtt{FQ}\llbracket\, \langle e \mid p \star x.t[e] \mid c \rangle \,\rrbracket
\end{aligned}$$

Let $x : (\lambda y.u)[f] \in e$ and $x : \lfloor \lambda y.\mathtt{Q}_t(u) \rfloor[\mathtt{Q}_e[\![f]\!]] \in \mathtt{Q}_e[\![e]\!]$

$$\begin{array}{ll}
 & \mathtt{FQ}[\![\langle e \mid x \star z.t[g] \mid c \rangle]\!] \\
= & \langle \mathtt{Q}_e[\![e]\!] \mid x \star z.\mathtt{Q}_t(t)[\mathtt{Q}_e[\![g]\!]] \mid \mathtt{Q}_s[\![c]\!] \rangle \\
\longrightarrow & \langle \mathtt{Q}_e[\![f]\!] \mid \lfloor \lambda y.\mathtt{Q}_t(u) \rfloor \star z.\mathtt{Q}_t(t)[\mathtt{Q}_e[\![g]\!]] \mid \mathtt{Q}_s[\![c]\!] \rangle \\
= & \mathtt{FQ}[\![\langle f \mid \lambda y.u \star z.t[g] \mid c \rangle]\!]
\end{array}$$

$$\begin{array}{ll}
 & \mathtt{FQ}[\![\langle e \mid \lambda x.u \star z.t[g] \mid c \rangle]\!] \\
= & \langle \mathtt{Q}_e[\![e]\!] \mid \lfloor \lambda x.\mathtt{Q}_t(u) \rfloor \star z.\mathtt{Q}_t(t)[\mathtt{Q}_e[\![g]\!]] \mid \mathtt{Q}_s[\![c]\!] \rangle \\
\longrightarrow & \langle \mathtt{Q}_e[\![g]\!], z : \lfloor \lambda x.\mathtt{Q}_t(u) \rfloor[\mathtt{Q}_e[\![e]\!]] \mid \mathtt{Q}_t(t) \mid \mathtt{Q}_s[\![c]\!] \rangle \\
= & \mathtt{FQ}[\![\langle g, z : (\lambda x.u)[e] \mid t \mid c \rangle]\!]
\end{array}$$

□