Polarised Intermediate Representation of Lambda Calculus with Sums

Guillaume Munch-Maccagnoni
University of Cambridge

Gabriel Scherer
Inria Paris-Rocquencourt

Abstract—Manipulating $\lambda$-terms with extensional sums is harder than with only pairs and functions. Following recent and less recent developments in proof theory, we propose an untyped representation—an intermediate calculus—for the $\lambda$-calculus with sums, based on the following principles: 1) Computation is described as the reduction of pairs of an expression and a context; the context is represented inside-out, 2) Operations are represented abstractly by their transition rule, 3) Positive and negative expressions are respectively eager and lazy; this polarity is an approximation of the type. We explain the approach from the ground up, and we review the benefits.

A structure of alternating phases naturally emerges through the study of normal forms, offering a reconstruction of focusing. But, on the side of proof theory, sums become as simple as products once cast into Gentzen’s sequent calculus [20], perhaps because sequent calculus has an inherent symmetry that reflects the categorical duality between sums and products. For instance, commutations (Figure 2) are not needed (see e.g. Girard [25]).

I. INTRODUCTION

The rewriting theory of $\lambda$-terms in the presence of extensional sums is sensibly more complex than with pairs and functions alone. Witness the diversity of approaches to decide the $\beta\eta$-equivalence in the simply-typed $\lambda$-calculus with sums, proposed since 1995: based either on a non-immediate rewriting theory (Ghani [21]; Lindley [38]) or on normalisation by evaluation (Altenkirch, Dybjer, Hofmann and Scott [3]; Balat, di Cosmo and Fiore [6]). (In Figure 1 we recall the simply-typed $\lambda$-calculus with sums in equational form, see e.g. [19, 38].)

But, on the side of proof theory, sums become as simple as products once cast into Gentzen’s sequent calculus [20], perhaps because sequent calculus has an inherent symmetry that reflects the categorical duality between sums and products. For instance, commutations (Figure 2) are not needed (see e.g. Girard [25]).

A. Abstract-machine-like calculi

Term syntaxes can benefit from the symmetry of sequent calculus at the condition of following certain principles, among which treating seriously the duality between expressions and contexts, as discovered with Curien and Herbelin’s abstract-machine-like calculi [11]. These calculi evidence a correspondence between sequent calculus (Gentzen [20]), abstract machines (Landin [32, 33]) and continuation-passing style (Van Wijngaarden [53]). In particular they illustrate the links that continuation-passing style has with categorical duality

April 2015.
We can assume that such polarities are involved in the validity of classical sequent calculus, connectives can be distinguished upon whether the hypothesis that composition is associative, in a sense back polarisation in the sense of Girard. For our current matters, polarisation corresponds to the intuitionistic polarized lambda-calculus with sums. Indeed, according to Girard for classical logic [22], a well-studied mean to circumvent a similar degeneracy that strikes Cartesian-closed categories in the presence of an isomorphism $A \cong R^A$ natural in $A$—by not requiring all the axioms of a category in the first place. In fact, polarisation relaxes the hypothesis that composition is associative, in a meaningful way that admits a positive characterisation with duploids [41, 43].

For our current matters, polarisation corresponds to the hypothesis that:

\[
(\lambda x.t) u > t[u/x] \quad \eta(\lambda t.(1:t,1)>t,1) > t_1 \\
\delta(t_1(t), x_1.u_1, x_2.u_2) > u_2[h/x]
\]

(a) Main reductions

\[
\delta(t, x_1.u_1, x_2.u_2)v > \delta(t, x_1.(u_1v), x_2.(u_2v)) \\
\eta(\delta(t, x_1.u_1, x_2.u_2)) > \delta(t, x_1.\eta(u_1), x_2.\eta(u_2)) \\
\delta(\delta(t, x_1.u_1, x_2.u_2)) > \delta(t, x_1.\delta(u_1), x_2.\delta(u_2))
\]

where $\delta(t) = \delta(t, y_1v_1, y_2v_2)$.

(b) Commutations

Figure 2: Reduction relation for the $\lambda$-calculus with sums

they are all retrieved in a methodical way, as we will see. Therefore we advocate that abstract-machine-like calculi should be seen as intermediate calculi that reveal the hidden structure of the terms—an analogy with intermediate representations used by compilers, which enable program transformation and analysis. In fact, our principles synthesise (and inherit from) continuation-passing style, defunctionalisation and direct style, as we will see.

B. Intuitionistic polarisation

Recall that in the presence of fixed points and extensional sums, any two $\lambda$-terms of the same type are equivalent, by a diagonal argument involving the fixed point of the function $\lambda x.\delta(x, y.t_2(y), y.t_1(y))$ [34, 31, 16]. As a consequence, the calculus from Figure 2 cannot be considered untyped.

Polarisation, introduced by Girard for classical logic [22], is a well-studied mean to circumvent a similar degeneracy that strikes Cartesian-closed categories in the presence of an isomorphism $A \cong R^A$ natural in $A$—by not requiring all the axioms of a category in the first place. In fact, polarisation relaxes the hypothesis that composition is associative, in a meaningful way that admits a positive characterisation with duploids [41, 43].

For our current matters, polarisation corresponds to the hypothesis that:

an expression is either positive or negative depending on the type; this polarity determines whether it reduces strictly or lazily.

We can assume that such polarities are involved in the validity of $\eta$-expansions in the $\lambda$-calculus with sums. Indeed, according to Danos, Joinet and Schellinx [13] (although in the context of classical sequent calculus), connectives can be distinguished upon whether the $\eta$-expansion seems to force or to delay the reduction. For instance, expanding $u[t/y]$ into $\delta(t, x_1.u[t_1(x_1)], x_2.u[t_2(x_2)/y])$ forces the evaluation of $t$. They show (with $\mathbf{L^C_{\eta}}$) how by making the reduction match, for each connective, the behaviour thus dictated by possible $\eta$-expansions, one essentially finds back polarisation in the sense of Girard. This is why declaring sums positive (strict) and functions negative (lazy) ensures that $\eta$-expansions do not modify the evaluation order.

Polarities only matter when call by value and call by name differ. In the context of the pure $\lambda$-calculus, only non-termination can discriminate call by value from call by name. Polarisation therefore suggests a novel approach to typed $\lambda$-calculi where associativity of composition is not seen as an axiom but as an external property, similar in status to normalisation.

The polarised calculus $\mathbf{L^C_{\eta}}$ is introduced in Section III, and it is a polarised abstract-machine-like calculus consistent with Girard’s [23, 26, Section 12.B] and Liang and Miller’s [37] suggestions for a polarised intuitionistic logic. $\mathbf{L^C_{\eta}}$ also determines an intuitionistic variant of Danos, Joinet and Schellinx’s $\mathbf{L^C_{\eta}}$ [13]. It is also meant to be a direct language for (variants of) Levy’s Call-by-push-value models [35, 36], though a precise correspondence will be the subject of another work.

In the article, we follow Curry’s style: the terms are not annotated by their types. The polarity is therefore also the least amount of information that has to be added to the calculus so that the evaluation order is uniquely determined, before any reference is made to the types. (We follow the same technique as appeared previously in M.-M. [39, 43, 42].)

C. Focusing and untyped normal forms

Focalisation is formulated as the phenomenon by which certain introduction rules hide cuts, as a consequence of polarisation. Thus we formulate focalisation as a restriction on the normalisation of proofs rather than on the structure of the proofs. From our point of view, focused proofs are retrieved through the study of normal forms.

Section IV shows that normal forms have a syntactic structure of alternating phases of constructors and abstractors. This indeed corresponds to focused proof disciplines (Andreoli [4]): in contrast with the lambda-calculus, $\mathbf{L^C_{\eta}}$ would systematically use invertible rules for each connective’s abstractor, with non-invertible rules only on constructors. We build an $\eta$-equivalence algorithm which is not type-directed, but inspects the syntactic structure of normal forms, adapting existing techniques (Abel and Coquand [10, 1]).

Variable and co-variable scope fully determines independence of neighbouring phases: if reordering two fragments of a normal form does not break any variable binding, it should be semantically correct. This is immediate for abstractor phases (which corresponds to the easy permutations of invertible steps), yet requires explicit purity assumptions for constructor phases, related to the associativity of composition.

Rewriting phases according to their dependencies corresponds to the idea of maximal multi-focusing [9, 8], a canonical representation of focused derivations. We give a fairly uniform, syntactic definition of canonical forms as normal forms of phase-permuting and phase-merging rewriting relations. In particular, $\beta\eta$-equivalence of typed $\lambda$-terms with sums can be decided by comparing the untyped normal forms.

D. Notations

In the grammars that we define, a dot indicates that variables before it are bound in what comes after. (For instance with $\mu(x.\star) c, \star$ are variables bound in $c$.)

If $\triangleright$ is a rewriting relation, then the compatible closure of $\triangleright$ is denoted by $\rightarrow$ and the compatible equivalence relation $\leftrightarrow (\rightarrow \leftrightarrow \leftrightarrow)$ is denoted with $\sim$. Reductions are denoted with $\triangleright R$ and expansions with $\triangleright E$. In this context we define $\triangleright RE \equiv \triangleright R \cup \triangleright E$. 
E. Acknowledgements

The sections II and III are adapted from the first author’s PhD thesis [41, Chapter I]—many thanks to Pierre-Évariste Dagand and Olivier Danvy for their criticisms and suggestions. The authors would like to thank the anonymous reviewers for their comments. The first author gratefully acknowledges partial support from the ERC project Ecsym and from the French National Research Agency (ANR-12-JS02-006-01, ANR-10-BLAN-0213, ANR-11-BS02-0010).

II. A Principled Introduction to Abstract-Machine-Like Calculi

In this section we explain the principles behind abstract-machine-like calculi and review their advantages in relationship to the study of \( \lambda \)-calculus with sums. We leave expansions aside until Section III. This exposition complements Waldier’s introduction to term assignments for Gentzen’s classical sequent calculi LK [54], Curien and the first author’s reconstruction of LK from abstract machines [12], and Spiwak’s motivation of abstract-machine-like calculi for the programming language theory [50].

A. Abstract machines

Abstract machines are defined by a grammar of expressions \( t \), a grammar of contexts \( e \), and rewriting rules on pairs \( c = \langle t \parallel e \rangle \) (commands). A command \( \langle t \parallel e \rangle \) represents the computation of \( t \) in the context \( e \). Intuitively, contexts correspond to expressions with a hole \( \Box \) that appears exactly once. For instance, the following extension of the \( \lambda \)-calculus with sums. We leave expansions aside until Section III. This exposition complements Waldier’s introduction to term assignments for Gentzen’s classical sequent calculi LK [54], Curien and the first author’s reconstruction of LK from abstract machines [12], and Spiwak’s motivation of abstract-machine-like calculi for the programming language theory [50].

A. Abstract machines

Abstract machines are defined by a grammar of expressions \( t \), a grammar of contexts \( e \), and rewriting rules on pairs \( c = \langle t \parallel e \rangle \) (commands). A command \( \langle t \parallel e \rangle \) represents the computation of \( t \) in the context \( e \). Intuitively, contexts correspond to expressions with a hole \( \Box \) that appears exactly once. For instance, the following extension of the Krivine abstract machine decomposes a term such as \( \delta((\lambda x.t_1(t)) u, y, v, z, w) \) into the context \( \Box, y, v, z, w \) and the redex \( (\lambda x.t_1(t))u \). The abstract machine therefore determines an evaluation strategy—here, call by name—for reductions in Figure 1c.

The contexts are for now stacks \( S \) of the following form:

\[
(e \Rightarrow) S \equiv \star \mid u \cdot S \mid \tau_1 \cdot S \mid \tau_1 \cdot S \mid [x.u] [y.v] [z.w] \cdot S
\]

The symbol \( \star \) represents the empty context. The reduction relation \( \triangleright_R \) on commands is defined by two sets of reduction rules:

- **Main reductions**: Variables are substituted, pairs are projected and branches are selected:
  \[
  (\lambda x.t \parallel u \cdot S) \triangleright_R \langle \delta[u/x] \parallel S \rangle
  \]
  \[
  (t_1 \cdot t_2 \parallel \tau_1 \cdot S) \triangleright_R \langle t_1 \parallel S \rangle
  \]
  \[
  (t \parallel [x_1.u_1] [x_2.u_2] \cdot S) \triangleright_R \langle \delta[u_1/x_1] \parallel S \rangle
  \]

- **Adjoint reductions**: Function applications, projections and branching build up the context:
  \[
  (\delta(t, x, u, v) \parallel S) \triangleright_R \langle t \parallel [x.u] [y.v] \cdot S \rangle
  \]

Notice that these reductions define \( \triangleright_R \) as a deterministic relation: if \( c \triangleright_R c' \) and \( c \triangleright_R c'' \), then \( c' = c'' \).

The reductions that we call *adjoint* are all of the form:

\[
(\delta(f(t)) \parallel S) \triangleright_R \langle t \parallel f(S) \rangle
\]

In other words, adjoint reductions state that the destructive operations of \( \lambda \)-calculus are, by analogy with linear algebra, adjoint to the constructions on contexts (Girard [24]). As a consequence, they build the context inside-out, as in our example:

\[
(\delta((\lambda x.t_1(t)) u, y, v, z, w) \parallel \star) \triangleright_R \langle \delta((\lambda x.t_1(t)) u \parallel [y.v] [z.w] \parallel \star) \rangle
\]

\[
(\delta((\lambda x.t_1(t)) u \parallel [y.v] [z.w] \parallel \star) \rangle \triangleright_R \langle t_1(u[x]) \parallel [y.v] [z.w] \parallel \star \rangle
\]

\[
(\delta(t_1(u[x]) \parallel [y.v] [z.w] \parallel \star) \rangle \triangleright_R \langle t_1(u[x]) \parallel [y.v] [z.w] \parallel \star \rangle
\]

In this example, the context \( u \cdot [y.v] [z.w] \cdot \star \) corresponds to the expression with a hole \( \delta(\Box, y, v, z, w) \) read from the inside to the outside.

B. Solving equations for expressions

We would like to relate the evaluation of terms to their normalisation. Notice that the reductions from Figure 1c are not enough to normalize a term such as \( \delta(x, y_1, \lambda z.t, y_2, y_2, u) z \). Commutation rules (Figure 2), coming from natural deduction in logic, are necessary to obtain the simpler form \( \delta(x, y_1, t, y_2, u) \). A distinct solution, as we are going to see, is to represent the various constructs of the abstract machine abstractly—as the solutions to the equations given by their transition rules. Let us explain this latter idea.

Let us rephrase reductions (1) and (2) as mappings from stacks to commands:

\[
t u : S \mapsto \langle t \parallel u \cdot S \rangle
\]

As explained by Curien and the first author [12], one can “read” definitions off these mappings. A binder \( \mu \) is introduced for the purpose:

\[
t u \equiv \mu \star (\langle t \parallel u \parallel \star \rangle \mapsto \delta(t, x, u, y, v) \parallel \star \mapsto \mu \star (\langle t \parallel [x.u] [y.v] \parallel \star \rangle
\]

In words, destructors are represented abstractly by their transition rule in the machine. The expression \( \mu \star .c \) maps stacks to commands thanks to the following reduction rule:

\[
(\mu \star .c) \parallel S \triangleright_R \langle c \parallel S/\star \rangle
\]

In fact, any equation of the form (3), assuming that \( f \) is substitutive, can be solved in this way:

\[
f^*(t) \equiv \mu \star (\langle t \parallel f(\star) \rangle
\]

Let us pause on the idea of solving equations. This improved wording has two purposes:

- **Emphasise the conscious step taken**, to underline that abstract-machine-like calculi do not serve as replacements for \( \lambda \)-calculi—one still has to determine which equations are interesting to consider.

- **Making us comfortable with the fact** that, later in Section III, **there may be two solutions, depending on the polarity**. The empty context \( \star \) is now a context variable (or *covariable*) bound in \( \mu \) and is the only one that can be bound by \( \mu \). (Originally, the notation \( \mu \) comes from Parigot’s \( \lambda \mu \)-calculus [45], where different co-variables name the different conclusions of a classical sequent. The idea of restricting to a single co-variable for modelling intuitionistic sequents with one conclusion is due to Herbelin [30].)
Thus we extend hold for all the abstract-machine-like calculi of the article. technique (see for instance Nipkow [64]) has no critical pairs solving the corresponding transition rule: commands, we have a reduction (for instance Nipkow [64], Curien and M.-M. [12], for introductions.)

To relate this notion of compatible closure to the one of compatible closure implies the clause (for instance Nipkow [64]) has no critical pairs solving the corresponding transition rule: commands, we have a reduction (for instance Nipkow [64], Curien and M.-M. [12], for introductions.)

We introduce the type system for the calculus considered so far in Figure 3c. The correspondence with sequent calculus is not obtained yet:

1) Left-introduction rules cannot be performed on arbitrary formulae; they stack.
2) The rule for \([x.u \mid y.v] \cdot S\) is still the one from natural deduction.

E. Solving equations for contexts

The calculus in Figure 3 still fails to normalize terms without using coequalizations, for instance in the following:

\[\langle \delta(t_{x_1}, \lambda z.t_{y_2}, \lambda z.u) \rangle \sim \langle \delta(t_{x_1}, \lambda z.t_{y_2}, \lambda z.u) \rangle \]

The commutation on expressions is only rephrased as a similar commutation on contexts.

Once again, let us describe the context \([x.u \mid y.v] \cdot S\) as a pair of mappings:

\[\pi_1(x) \mapsto \langle u \rangle S, \pi_2(y) \mapsto \langle v \rangle S\]

This suggests that the branching context can be written abstractly, by introducing a new binder \(\mu\):

\[\mu x.\langle u \rangle S \mapsto \mu x.\langle u \rangle S\]

The stack \(\mu[x_1, c_1][x_2, c_2] \cdot S\) is symmetric in shape to the constructor for pairs, and associates to any injection \(\iota_i\) the appropriate command \(c_i\) or \(c_2\):

\[\langle \iota_i(t) \rangle \mapsto \mu[x_1, c_1][x_2, c_2] \sim \mu[x_1, c_1][x_2, c_2] \]

In fact, we only consider the contexts \([x.u \mid y.v] \cdot \star\) in the definition of branching:

\[\delta(t, u, v, y, t) \equiv \mu x.\langle u \rangle S \cdot \langle y, v \rangle S\]

While \(S\) was duplicated in (6), this definition is local.
we now make the contexts

\[ \langle \mu x.c \rangle \]

In particular, \( \mu \) against which the
difference between \( \mu x.c \) and \( \mu[x_1.c_1 \mid x_2.c_2] \), while we motivated
both in the same way.

Notice that \( \mu[x_1.c_1 \mid y.c''] \) belongs to the syntactic category \( S \)
of stacks. (It no longer describes \( S \), but any context
against which the \( \mu \) reduces.) We may criticise the call-by-
name bias for the first time, for introducing this arbitrary dif-
ference between \( \mu x.c \) and \( \mu[x_1.c_1 \mid x_2.c_2] \), while we motivated
both in the same way.

The calculus \( \mathcal{L}^\oplus \) (Figure 4) summarises our development so
far. Since \( \triangleright_R \) is still without critical pairs, \( \rightarrow_R \) is still con fluent.

\textbf{F. Commutation rules are redundant}

The term \( \delta(x, y_1, \lambda z.t, y_2, \lambda z.u) z \) now reduces as follows:

\[ \langle \delta(x, y_1, \lambda z.t, y_2, \lambda z.u) z \rangle \mapsto_R \langle x \mid \mu[y_1, \langle \lambda z.u \mid z \rangle ; y_2, \langle \lambda z.u \mid z \rangle ; S] \rangle ] \]

\[ \mapsto_R \langle x \mid \mu[y_1, \langle \lambda z.u \mid z \rangle ; y_2, \lambda z.u \mid S] \rangle ] \]

In particular, \( \langle \delta(x, y_1, \lambda z.t, y_2, \lambda z.u) z \rangle \mapsto_R \langle x \mid \mu[y_1, \langle \lambda z.u \mid z \rangle ; y_2, \lambda z.u \mid S] \rangle ] \) despite what commutation rules would prescribe. However, they have the following common reduction:

\[ \langle x \mid \mu[y_1, \langle \lambda z.u \mid z \rangle ; y_2, \lambda z.u \mid S] \rangle ] \]

\[ \mapsto_R \langle x \mid \mu[y_1, \langle \lambda z.u \mid z \rangle ; y_2, \lambda z.u \mid S] \rangle ] \]

The same happens with the other commutation rules. In other words, commutation rules are redundant in \( \mathcal{L}^\oplus \) and at the same time \( \mathcal{L}^\oplus \) offers a novel reduction theory compared to the \( \lambda \)-calculus with sums and commutations. In particular, normalisation in the proof theoretic sense is obtained with the compatible closure (\( \rightarrow_R \)) of evaluation (\( \triangleright_R \)).

\textbf{G. Focalisation}

Any proof in the propositional intuitionistic sequent calculus can be retrieved as an \( \mathcal{L}^\oplus \) derivation. Indeed, the typing rule for \( \mu \) (Figure 4d) allows left-introduction rules to be performed

on arbitrary formulae—in particular, contexts \( u.e \) and \( \pi_i.e \) corresponding to unrestricted left-introduction rules (Figure 4e) can be defined, using \( \tilde{\mu} \), as the solutions to the following equations:

\[ \langle t \parallel u.e \rangle \triangleright_R \langle \mu \star.t \parallel u.e \rangle \parallel e \]

\[ \langle t \parallel \pi_i.e \rangle \triangleright_R \langle \mu \star.t \parallel \pi_i.e \rangle \parallel e \]

The rules from Figure 4e are derivable modulo weakening of the
premise \( e \), which is admissible in the standard way. (For simplicity we did not formulate \( \mathcal{L}^\oplus \) in the multiplicative style with explicit weakening, which would let us state directly that the unrestricted rules are derivable.)

In words, unrestricted left-introduction rules in \( \mathcal{L}^\oplus \) hide cuts. We call focalisation this phenomenon by which certain introduction rules hide cuts. It is indeed responsible for the shape of synchronous phases in focusing.

\textbf{H. Inside-out contexts are primitive}

Every context \( e \) corresponds to a term with a hole \( E[e] \).
Thus, every command \( \langle t \parallel e \rangle \) (or equivalently every expression \( \mu \star.t \parallel e \rangle \) corresponds to a \( \lambda \)-term \( E[e] \). We map commands to expressions as follows:

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]

\[ E_\star \parallel \square \]

\[ E_{\pi_i \star} \parallel \square \]

\[ E_{\mu \star.t} \parallel \square \]

\[ E_{\nu \star.t} \parallel \square \]
(It might matter for rewriting purposes that the reduction is actually a single parallel reduction step.)

In this sense, contexts are equivalent to expressions with a hole. But we argue that inside-out contexts are more primitive.

- When the reduction is directly defined on expressions $E[x]$, an external property—the so-called unique context lemma—replaces the adjoint reductions in the role of reaching a main reduction.

- Above, it is more difficult to express inside-out contexts as expressions with a hole than the contrary, because it requires an external definition by induction.

Inside-out contexts reveal intermediate steps in the reduction, which, once they are expressed as part of the formalism, render such external properties and definitions unnecessary.

I. A parte: Defunctionalised CPS in direct style

The contexts $\mu x. c$ are functional abstractions of the remainder of the computation, that is, per definition, continuations. In fact, abstract-machine-like calculi could alternatively be introduced as the continuity of continuation-passing-style semantics. Indeed, although inside-out contexts are considered in term calculi since Herbelin’s study of the intuitionistic sequent calculus [29], such an inverted representation appeared with continuations as early as Wand [55].

Moreover, Danvy and Nielsen [15] recognised in Wand’s technique an instance of Reynold’s defunctionalisation [46]. They also mention the “isomorphism” between expressions with a hole and inside-out contexts. In addition, Ager, Bier- nakci, Danvy, and Midtgard [2] show how to obtain abstract machines from evaluators via CPS transformation, closure conversion, and defunctionalisation.

Thus, in terms of continuation-passing style, abstract-machine-like calculi describe defunctionalised CPS, in direct style (in the sense that the encodings of $\lambda$-calculi are local, like in Figure 3c). But it is novel to affirm that the representation of contexts inside-out can and should be taken as primitive, since it is made possible by the idea of representing destructors abstractly, as the adjoints of context constructors.

III. Polarisation and Extensional Sums

A. Conversions on expressions vs. on commands

In this section, we consider extensionality principles formulated as expansions $\triangleright E$ defined, unlike $\triangleright R$, on expressions and on contexts. The following expansions are standard:

$$ t \triangleright E \mu x. (t \parallel x \parallel *) \quad e \triangleright E \mu x. (x \parallel e \parallel x \notin \text{IV}(e)) $$

They enunciate that one has $t \approx_{RE} u$ if and only if for all $e$, $(t \parallel e) \approx_{RE} (u \parallel e)$ (symmetrically for contexts): there is only one notion of equivalence which coincides between expressions, contexts and commands.

B. Extensionality for sums with a regular shape

Converting, as described in Section II-H, contexts $e$ into expressions with a hole $E[\ ]$ (and conversely) shows that the following:

$$ e \approx_{RE} \mu[x. (t_1(x) \parallel e \parallel y. (t_2(y) \parallel e \parallel e)] $$

is equivalent to the standard $\eta$-expansion of sums:

$$ E[t] \approx_{RE} \delta(t, x. E[t_1(x)], y. E[t_2(y)]) \quad (8) $$

In fact, the traditional $\eta$-expansions of sequent calculus suggests extensionality axioms that all have a regular shape:

$$ t \triangleright E \mu(x \parallel *). (t \parallel x \parallel *) \quad t \triangleright E \mu x. (t \parallel \pi_1 \parallel \pi_2 \parallel *) \quad e \triangleright E \mu[x. (t_1(x) \parallel e) \parallel y. (t_2(y) \parallel e)] \quad (9) $$

However, the degeneracy of $\lambda$-calculus with extensional sums and fixed points mentioned in Section 1-B applies: in untyped $L_\text{int}$ one has $t \approx_{RE} u$ for any two expressions $t$ and $u$ by an immediate adaptation of Dougherty [16].

C. Positive sums

Positive sums, unlike the sums of $L_\text{int}^\ominus$, are called by value. A difference is therefore introduced, among expressions, between values $V$ (such as $t_1(x)$) and non-values (such as $tu$), as follows:

- The reduction of $\mu$ is restricted to values:

$$ (V \parallel \mu x. c) \triangleright_R e[V/x] $$

- The reduction of a non-value $\mu x. c$ (including $tu$) takes precedence over the one defined by the context; in other words every positive context is a stack.

As a consequence, the equation (9) no longer converts an arbitrary context into a stack. In addition, it becomes equivalent to the following $\eta$-expansion for sums restricted to values:

$$ E[V] \approx_{RE} \delta(V, x. E[t_1(x)], y. E[t_2(y)]) $$

Thus, when sums are positive, (as observed for the classical sequent calculus [13]), the extensionality axiom does not interfere with the order of evaluation.

D. The polarised calculus $L_\text{int}$

$L_\text{int}$ is a polarised variant of $L_\text{int}^\ominus$ introduced in Figure 5. In addition to the negative connectives $\rightarrow, \times$ it proposes positive pairs and sums ($\forall, \oplus$).

a) Polarisation as a locally-determined strategy: Polarisation, unlike the call-by-value evaluation strategy, assigns a strict evaluation to sums without changing the evaluation order of the other connectives of $L_\text{int}^\ominus$. The evaluation order is determined at the level of each command by a polarity $\varepsilon \in \{\forall, \oplus\}$ assigned to every expression and every context, impacting the reduction rules as follows:

<table>
<thead>
<tr>
<th>Negative polarity</th>
<th>Every expression is a value (CBN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive polarity</td>
<td>Every context is a stack (CBV)</td>
</tr>
</tbody>
</table>

Polarities in this sense are similar to Danos, Joinet and Schellinx’s $\forall t/q$ annotations on the classical sequent calculus—$L_\text{int}$ in fact determines an intuitionistic variant of their polarised classical logic $L_\odot$.
With more complex type systems than the simple types, such as part of the grammar: that is variables (\(x\)) and expressions (\(\langle t \rangle\)). Now, calculating on terms together with their typing derivations is tedious—it requires intimate knowledge of the properties of the type system (for instance subject reduction). With more complex type systems than the simple types, such properties can even be uncertain.

What we show with \(L\text{int}\) is that all the information relevant for evaluation can be captured in a grammar, requiring no complete inspection of the terms, thanks to appropriate polarity annotations. We only needed to make sure that the conversion rules are grammatically well defined despite the presence of annotations, but this is easy to verify.

On the other hand, polarities are no types:

- like in the pure \(\lambda\) calculus, it is possible to define arbitrary fixed-points, and in particular \(L\text{int}\) is Turing-complete.
- it is possible to obtain ill-formed commands \((t_\oplus \parallel e_\oplus)\) by reducing well-polarised (but ill-typed) commands such as \(\mu(x^{\oplus} \cdot \star)(x^{\oplus} \parallel \star)\). (Such mixed commands do not reduce, and the reader should not expect to see them appear in the rest of the article.)

\(L\text{int}\) has a rich structure—more complex types (in \(\mu\) and \(\nu\)) are never a value, and \(\mu\) and \(\nu\) are never a stack. With this observation in mind, it is easy to verify that:

\(R\rightarrow\): The subscript/superscript convention: The polarity of an expression (\(t_\oplus\) or \(t_\odot\)) or a context (\(e_\oplus\) or \(e_\odot\)) is defined in the Figures 5c and 5d. Whenever necessary, the polarity annotation is part of the grammar: that is variables (\(X^{\oplus}\), \(X^{\odot}\)), binders (\(\mu^{\star \cdot \cdot \cdot \cdot} c\), \(\mu^{\star \cdot \cdot \cdot \cdot} c\)), and type variables (\(\Gamma\)). These notations conform to the following convention for polarity annotations:

- a polarity in superscript (\(x^\oplus\)) indicates an annotation which is part of the grammar,
- a polarity in subscript (\(t_\oplus\)) is not part of the grammar and only asserts that the term has this polarity.

Polarisation as an approximation of types: A system of simple types, which corresponds to an intuitionistic sequent calculus, is provided on top of the Curry-style \(L\text{int}\). It is easy to see that:

- if one has \(t : A\), then \(t\) has polarity \(\varepsilon\),
- if one has \(e \vdash A\), then \(e\) has polarity \(\varepsilon\).

Therefore, for typeable terms, the type determines the order of evaluation.

Now, calculating on terms together with their typing derivations is tedious—it requires intimate knowledge of the properties of the type system (for instance subject reduction). With more complex type systems than the simple types, such properties can even be uncertain.
Let us denote with $L_{\text{int}}^{\perp}$ and $\Lambda^{\perp}$ the simply-typed $L_{\text{int}}$ (without $\otimes$) and $\lambda$-calculus with sums, that is to say restricted by the typing judgements from Figures 5 (minus the positive pair $\otimes$) and 1 respectively. For simplicity we will omit the typing judgements and may choose to write explicit type annotations (à la Church). We give these systems the same grammar of types by assuming that the set of type variables $X^{\perp}$ coincides with the set of positive and negative type variables $X^{+}, X^{\otimes}$ of $L_{\text{int}}^{\perp}$. In Figure 6 we define two translations:

$$N^\perp[.] : L_{\text{int}}^{\perp} \rightarrow \Lambda^{\perp}, \quad S^\perp[.] : \Lambda^{\perp} \rightarrow L_{\text{int}}^{\perp}$$

These translations are well defined: if $\Gamma \vdash_{L_{\text{int}}} t : A$ then $\Gamma \vdash_{\Lambda} N[\Gamma] : A$ and if $\Gamma \vdash_{\Lambda} t : A$ then $\Gamma \vdash_{L_{\text{int}}} S[\Gamma] : A$ in addition:

**Proposition 1.**

1) If $\Gamma \vdash_{L_{\text{int}}} t : A$ then $t \approx_{RE} S[N[\Gamma]]$.

2) If $\Gamma \vdash_{\Lambda} t : A$ then $t \approx_{RE} S[\Gamma]$.

3) If $t \approx_{RE} u$ in $L_{\text{int}}^{\perp}$ then $N[\Gamma] \approx_{RE} N[u]$.

Of course, it cannot be the case that $t \approx u$ in $\Lambda^{\perp}$ implies $S[\Gamma] \approx_{RE} S[u]$ in $L_{\text{int}}^{\perp}$: unless $L_{\text{int}}$ identifies all expressions. Indeed, one would have $(\lambda x^\perp. b \otimes \lambda x^\perp. y^\perp)(y^\perp) \approx_{RE} b \otimes \lambda x^\perp. (y^\perp)$ in $L_{\text{int}}$. For $y = \lambda z. t_+ $ and $b_\otimes = \lambda z. \mu z^\otimes. c([z_\otimes a])^f / x^\perp$ when $t_+$ and $c$ are arbitrary, this implies $(t_+ \mu x^\perp. c) \approx_{RE} c[t_+/x^\perp]$—where the focalising substitution $[u^\perp/x^\perp]$ is defined using focalisation in the sense of Section II-G:

**Definition 2.** Unrestricted expressions and contexts are defined as follows (in addition to $t_+$ defined in Figure 6a):

$$t_+ \cdot e \equiv \mu x^\perp. (t_+ \parallel \mu y^\perp. (x^\perp \parallel y^\perp \cdot e))$$

$$V \cdot e_\otimes \equiv \mu x^\otimes. (\mu z^\otimes. (x^\otimes \parallel V \cdot e_\otimes))$$

($\vdash$: When $t_+$ is not a value—$:; When $e_\otimes$ is not a stack.)

Now let us call ($\beta$) the following equation:

$$(t_+ \parallel \mu x^\perp. c) \beta \rightarrow_{R} c[t_+/x^\perp]$$

which is equivalent to the associativity of the composition in $L_{\text{int}}$ (See M.-M. [43].)

In $L_{\text{int}}$ one can already derive modulo $\approx_{RE}$ all the conversions of $\Lambda^{\perp}$ with the following restriction: the arguments of $\lambda$ and $\delta$ are restricted to values. Using ($\beta$) one can substitute positive variables, which are values, with arbitrary positive expressions. Therefore:

**Proposition 3.** If $t \approx u$ in $\Lambda^{\perp}$ then $S[\Gamma] \approx_{RE} S[u]$ in $L_{\text{int}}^{\perp}$.

In fact, in (untyped) $L_{\text{int}}$ extended with ($\beta$), one can apply the same diagonal argument as in Section III-B, and therefore ($\beta$) is incoherent in $L_{\text{int}}$ as we claimed. But, ($\beta$) is compatible with $L_{\text{int}}$ in the following sense:

**Lemma 4.** Let $t_+$ be an expression such that $\vdash_{L_{\text{int}}} t_+ : B_+$. There exists a closed value $V_+$ such that $(t_+ \parallel \star) \beta \rightarrow_{R} V_+ \parallel \star$, and therefore $t_+ \approx_{RE} V_+$.

**Proof:** This is a corollary of the adequacy lemma form M.-M. [39], adapted to intuitionistic logic (as in [40]). Adapting the polarised orthogonality-based realizability from [39] is straightforward: any intuitionistic proof can be seen as a classical proof. Now, the linearity restriction on $\star$ implies $\star \notin \mathbf{fv}(V_+)$ as we already noticed for $L_{\text{int}}^{\perp}$ in Section II-C, therefore we can conclude $t_+ \approx_{RE} V_+$ in words, intuitionistic computation is referentially transparent.

**Proposition 5.** $L_{\text{int}}^{\perp}$ satisfies ($\beta$) whenever $t_+$ is closed.

In a separate note, we prove that ($\beta$) actually belongs to the observational closure of $\approx_{R}$. The technique, based on
polared realizability, generalises beyond simple types—for instance, extensions to second- and higher-order are immediate. Polariation therefore offers an approach to typed λ-calculi where the associativity of composition is an external property similar to normalisation. If we adopt this point of view, then it is not surprising that the rewriting theory of the λ-calculus with sums is complex, and that it is even computationally incoherent in the presence of fixed-points.

IV. The Structure of Normal Forms and Their Equivalence

This section studies the \(\rightarrow_R\)-normal forms of \(L_{\text{int}}\). Their equivalence can be defined and decided by equivalence relations that are more uniform than the rewriting-based approaches targeting \(\lambda\)-calculus.

A. The phase structure of normal forms

Figure 7a describes the syntax of valid R-normal forms, where a constructor is never opposed to an abstractor for another connective; for example \(\langle t_1(t) \parallel \mu(x, y).c \rangle\) would be an invalid normal form. A normal command of the form \(\langle V_x \parallel \star \rangle\) or \(\langle x^\oplus \parallel S_{\oplus} \rangle\) starts with a constructor, head computation is finished. A command of the form \((x^\oplus \parallel e_\ast \setminus \mu x^\chi.c)\) (any \(e_\ast\) that is not a \(\mu x^\chi.c\)) or \((\langle t_\alpha \parallel \mu^\oplus \chi.c \parallel \star \rangle\) starts with an abstractor whose computation is blocked by the lack of information on the opposite side. We thus decompose the structure of normal commands as a succession of phases, with constructor phases \(c^C\) and abstractor phases \(c^D\). Remark that functions have their constructor and abstractors inversed with respect to their constructor and destructor in \(\lambda\) calculi: function introduction \(\nu(x \cdot \star).c\) is an abstractor expression, while function application \(u \cdot e\) is a context constructor.

Figure 7b describes the structure of phase-separated normal forms. We call them focused forms. A focused form \(\langle t \parallel \star \rangle\) is a focused command \(c^\oplus\) for some phase variable \(x\) which is either \(C\) or \(D\)—we will write \(\phi\) for the phase opposite to \(\phi\). A focused command \(c^\oplus\) is either a focused value \(V^\phi\) against \(\star\), or a focused stack \(S^\phi\) against some variable \(x\). Constructor values \(V^C\) start with a head expression constructor, followed by other head constructors until \(\mu^\oplus \chi.f\) or a variable is reached, ending the phase. Constructor stacks \(S^C\) are non-empty compositions of stack constructors ended by \(\mu x^\chi.f\). Abstractor values or stacks are either a variable or \(\star\), or a \(\mu\)-form that immediately ends the phase. An informal way to think of focused commands, which correspond to our intuition of normal sequent proofs, is to see the focused command \(\langle x \parallel S^\phi \rangle\) as a "play on \(x\)" and, correspondingly \(\langle V^\phi \parallel \star \rangle\) as a "play on \(\star\)".

This syntactic phase separation resembles focused proofs [4]. We work on an untyped calculus, so we cannot enforce that phases be maximally long; but otherwise constructor phases correspond to non-invisible, or synchronous rules, while abstractor phases correspond to invisible, or asynchronous inference rules.

Not all \(\rightarrow_q\)-normal forms are focused forms, as they may not respect the constraint that phase change be explicitly marked by a \(\mu\)-form. But a normal form can be easily rewritten into a focused form: inserting a phase change is a valid expansion. For example, \(\langle t_1(x^\oplus) \parallel \star \rangle\) \((E\mu\)-expands into the phase-separated command \(\langle t_1(\mu^\oplus \chi.(x^\oplus \parallel \star)) \parallel \star \rangle\).

**Lemma 6.** Any valid \((\rightarrow_R)\)-normal form is \((\rightarrow^*_E)\)-expansible to a unique focused form \(f\).

B. Algorithmic equality for expansion rules

We define the algorithmic equality relation \(\equiv_{\Lambda}\) as a system of inference rules on pairs of commands, which captures extensional equivalence: \((\equiv_{\Lambda})\) decides \((\cong_{\text{RE}})\) on \(\rightarrow_R\)-normal terms. If we worked on typed normal forms, we could perform maximal type-directed \(\eta\)-expansion; for the sake of generality and its intrinsic interest, we work with untyped normal forms. The purely syntactic approach is to perform, in both commands to be compared, the \(\eta\)-expansions for the abstractors that appear in either commands. This untyped equivalence is inspired by previous works [10, 11] that compute \(\eta\)-equivalence of \(\lambda\)-terms by looking at the terms only, doing an \(\eta\)-expansion when either term starts with a \(\lambda\).

In Figure 8a we define \((\equiv_{\Lambda})\) by mutual recursion with a sub-relation \((\equiv_{\Lambda})\) that only relates \(R\)-normal forms.

We have omitted the symmetric counterparts of the abstractor rules. In the two last cases, we consider that \((\equiv_{\Lambda})\) is defined on expressions and contexts as the congruence closure of \((\equiv_{\Lambda})\) on sub-commands. For example, \(t_1(\mu x.c) \equiv_{\Lambda} t'\) if and only if \(t'\) is of the form \(t_1(\mu x'c')\) for some \(c'\) such that \(c \equiv_{\Lambda} c'\). We use the term algorithmic because this equivalence is almost syntax-directed. If both sides of the equivalence start with an abstractor application, there is a non-syntax-directed choice of which side to inspect first: for \(t, u\) fixed there may be several distinct derivations of \(t \equiv_{\Lambda} u\).

Notice that, whenever \(c_1\) is a \(\rightarrow_R\)-normal form, then the substitutions involved in the definition of algorithm equivalence, such as \(c_1[\{x_1/x_2\}^\chi]\), are strongly normalizing as well. Indeed, while the substitution may create new reduction opportunities, for example in \((\chi^\oplus \parallel \mu(\gamma_1, \gamma_2).c')\), those reductions would in turn only create substitutions of variables \(\{x_1/\gamma_1\}\) and \(\{x_2/\gamma_2\}\) in the example above), which do not produce any new redexes.

**Lemma 7.** Soundness of algorithmic equivalence: if \(t \equiv_{\Lambda} u\) then \(t \equiv_{\text{RE}} u\).

C. Completeness of algorithmic equality on normal forms

Abel and Coquand [1] discuss the problem of transitivity of their syntax-directed algorithmic equality: while it morally captures our intuition of "good" \(\eta\)-equivalences, it fails to account for some seemingly-absurd equivalences that can be derived on untyped terms, such as \(\lambda x.(t x) \equiv_{\eta} \langle t_1; t_2 \rangle\), the transitive combination of \(\lambda x.(t x) \equiv_{\eta} t\) and \(t \equiv_{\eta} \langle t_1; t_2 \rangle\). To accept those absurd equivalences, Abel and Coquand had to extend their relation with type-incorrect rules such as:

\[
\frac{t \equiv_{\Lambda} n \chi \pi \equiv_{\Lambda} r \pi' \equiv_{\Lambda} s}{\lambda x.(t x) \equiv_{\Lambda} \langle r; s \rangle}
\]

This change suffices to regain transitivity and capture untyped \(\eta\)-conversion. Unfortunately, this technique requires a number of additional rules quadratic in the number of different
constructors of the language. Our richer syntax allows the definition in Figure 8b of an extended algorithmic equivalence (\(\equiv_{A+}\)) without introducing any new rule, but by extending each constructor rule \(A\)-pair, \(A\)-pair etc. in a simple (but potentially type-incorrect) way.

The notation \(c[t/V]\) substitutes syntactic occurrences of the term \(V\) by \(t\) in an unique context decomposition of \(c\) into \(c'[V]\) with \(V \not\subseteq c\), and then \(c[t/V]\) is \(c'[t]\). On valid terms, this extended equivalence coincides with the previous algorithmic equivalence. Indeed, in the case of a normal command \(\langle V_c , \mu(x_1,x_2), c \rangle\) for example, a valid \(V_c\) can only be a variable \(x^\star\) in which case the rule \(A\)-pair and \(A\)-pair are identical. The extension only applies when \(V_c\) has a head constructor that is not the pair, in which case the expression is in normal form but invalid. Figure 8c shows that this extension subsumes the rules of Abel and Coquand: we can derive their instances.

Notice how the left-hand-side of the left premise of the topmost rule is obtained by the apparently-nonsensical substitution \(\langle x_1, x_2 \rangle \Rightarrow \mu(x_1,x_2)\langle x_1, x_2 \rangle\) produced by the extended sum rule on an invalid command. We consider this baroque extension only a curiosity of the meta-theory.

**Lemma 8** (Soundness). \(t \equiv_{A+} u\) implies \(t \equiv_{RE} u\).

**Lemma 9** (Reflexivity). If \(t\) is \(\not\equiv_{\mathbb{R}}\)-normalizing then \(t \equiv_{A+} t\).

**Lemma 10** (Transitivity). If \(t_1 \equiv_{A+} t_2\) and \(t_2 \equiv_{A+} t_3\) then \(t_1 \equiv_{A+} t_3\).

**Lemma 11** (Completeness). If \(f \equiv_{RE} f'\) then \(f \equiv_{A+} f'\).

**Theorem 12.** If \(c_1\) and \(c_2\) are \(\not\equiv_{\mathbb{R}}\)-normalizable commands, then \(c_1 \equiv_{RE} c_2\) if and only if \(c_1 \equiv_{A+} c_2\).

**D. Phase equivalence**

Algorithmic equivalence demonstrates that the focused forms are a suitable structure to compute \(\equiv_{RE}\)-equivalence. However, this \(\equiv_{RE}\)-equivalence is more restrictive than the usual \(\not\equiv_{\mathbb{R}}\)-equivalences considered for \(\lambda\)-terms. For example, let \(y^\phi b \equiv x V\langle t_0 \rangle\) is not \(\equiv_{RE}\)-equivalent to \(t_1\) (let \(y^\phi b \equiv x V\langle t_0 \rangle\)—intuitively, the applications of constructors may perform arbitrary effects and thus are not reorderable.)
We can strengthen the equational theory of the untyped language, under the additional assumption that the terms are pure and strongly normalising setting, by allowing identical phases to be merged, and unused phases to be dropped. This is the purpose of Section IV-E, which recovers a theory equivalent to $\beta\eta$ for $\lambda$-calculus. As a first step, we study in this section the reordering of independent phases. This is no stronger than ($\approx_{\text{RE}}$), but captures the global aspect of equirmoring: merging and weakening can then be defined locally.

We write $c^\phi_1$, $V^\phi_1$ and $S^\phi_1$ for focused command contexts (respectively value contexts, stack contexts) with one or several holes, all in a position where a subcommand is expected. We are interested in reordering $c_1[c_2][\ldots]$ into $c_2[c_1][\ldots]$. Two command contexts $c_1[\ldots]$ and $c_2[\ldots]$ are independent, written $c_1 \neq c_2$, if reordering them respects scope: the variables (and $\ast$) of $c_1$ bound by $c_2$ are not free in $c_2$, and conversely.

**Definition 13.** We define the commutative contexts for focused commands, values and stacks by the grammar of Figure 9.

For a given phase $\phi$, we define the phase equivalence ($\approx_\phi$) with the reordering:

$c_2[c_1][\ldots] \triangleleft_\phi c_1[c_2][\ldots]$

when $c_1 \neq c_2$.

Note that some abstractor commutative contexts, e.g. $(x^\ast)[\tilde{\mu}[x_1][x_2][\ldots]]$, may have several holes—phase reordering may thus result in (de-)duplication. Constructor commutative contexts have exactly one hole; for example, $(V^C c_1[\ldots])$ and $(l[\ldots], V^C c_2)$ are both valid shapes for a commutative value context, but we do not define $(l[\ldots])$ as a commutative context. One can relate this difference to the additive or multiplicative status of connectives in linear logic, or to the syntactic idea that control may flow into either branches of the sum elimination construction. In particular, $\tilde{\mu}[x_1][x_2][\ldots]$ is not a commutative context, since $c[\tilde{\mu}[y_1][y_2][\ldots]]$ is different from $\tilde{\mu}[y_1][y_2][\ldots]$, but $\tilde{\mu}[y_1][y_2][\ldots]$ is the same as $(x^\ast)[\tilde{\mu}[x_1][x_2][\ldots]]$ in general.

**Lemma 14.** ($\approx_p$) is derivable: if $c_1 \approx_p c_2$ then $c_1 \approx_{\text{RE}} c_2$.

While ($\approx_p$) is defined entirely on the structure of focused normal forms, the intermediary ($\approx_{\text{RE}}$)-reasoning step used to justify commutativity are free to disrespect this focused structure. These exchanges between the normalized and the non-normalized world are interesting and justify, we think, studying first the dynamics of the system (or, said otherwise, proof/type systems with a cut), and looking at their normal forms only later, instead of starting with cut-free systems directly.

**Definition 15.** We call phase equivalence the relation ($\approx_p$) defined as the (closure of the) union of ($\approx_{p_c}$) and ($\approx_{p_d}$).

Appendix A gives a rewriting relation to decide ($\approx_p$)-equivalence, similarly to the preemptive rewriting of maximal multi-focusing [9].

**E. Recovering $\beta\eta$-equivalence**

If $\rho$ is a substitution from variables to variables, and $V^\phi_1$ (resp. $S^\phi_1$) is a focused context, we write $V^\phi_1[\rho[\ldots]]$ for the focused context where each of the variables bound by $V^\phi_1$ that scope over its hole are renamed according to $\rho$, and only those are in the domain of $\rho$. In particular, there exists a $\rho$ such that $V^\phi_1[\rho[\ldots]] = V^\phi_2[\ldots]$ when $V^\phi_1$ and $V^\phi_2$ modulo renaming of outgoing variables—for example $\mu(x_1,x_2)[\ldots]$ and $\mu(x_1',x_2')[\ldots]$.

**Definition 16.** We define the merging simplification ($\triangleright_{\text{M}}$) as the smallest relation such that, for each pair of independent commutative contexts $c_1 \neq c_2$ and substitution $\rho$ such that $c_1[\ldots] = c_2[\ldots]$ and $c_2[\ldots]$, we have $c_1\triangleright_{\text{M}} c_2[\ldots].$

$\triangleright_{\text{M}}$ is normalizing, since it decreases strictly the size of the command.

**Definition 17.** We define the weakening simplification ($\triangleright_{\text{W}}$) as the smallest relation such that, whenever none of the variable bound by the commutative context $c[\ldots]$ over its hole are used in $f$, we have $c[f] \triangleright_{\text{W}} f$.

Note that commutative contexts of the form $\langle V^\phi[\ldots] \rangle$ are neither mergeable nor weakenable, since all possible holes of $V^\phi[\ldots]$ shadow the variable $\ast$, breaking the independence requirement, and all pluggable commands $\rho$ use the variable $\ast$, breaking the weakening requirement. Semantically it would make no sense, of course, to claim something to the effect of $t_1(t_1(\mu[\ast].c)) \triangleright_{\text{M}} t_1(\mu[\ast].c)$ or $t_1(\mu[\ast].f) \triangleright_{\text{W}} f$, but we directly recover this from $\triangleright_{\text{M}}$ without having to break the uniformity of our definitions.

We will write ($\approx_{\text{REPMW}}$) the closure of the unions of ($\approx_{\text{RE}}$), ($\approx_p$), ($\approx_{\text{M}}$) and ($\approx_{\text{W}}$). It is decidable on $\rightarrow_{\text{R}}$—normalizable terms by normalizing by ($\rightarrow_q$), then taking the ($\approx_{\text{p}}$)-normal form, the $\rightarrow_{\text{WM}}$—normal form, and finally ($\approx_{\text{A}}$) to decide equivalences. We now prove that is sound and complete with respect to ($\approx_{\text{RE}}$), which corresponds to the $\beta\eta$-equality of $\lambda$-calculus as demonstrated in Section III-E.

**Proposition 18.** If $t \approx_{\text{WM}} u$ then $t \approx_{\text{RE}} u$.

**Theorem 19.** If $t \approx_{\text{RE}} u$, then $t \approx_{\text{REPMW}} u$.

This concludes our study of the normal forms. We have demonstrated that the equivalence ($\approx_{\text{RE}}$), which corresponds on typed terms to the natural equality of well-typed $\lambda$-terms, can be re-defined (and decided) in a syntactically uniform way based on sequences of constructor and abstractor phases.

**References**


Master's thesis, DIKU, Computer Science Department, University of


Deconstructive Logic: Linear Logic computation under focus


proofs via multi-focusing

putation (2014).

normalisation and type-directed partial evaluation for typed lambda calculus with sums, POPL, 2004, pp. 64–76.

Alois Brunel, Quantitative classical realizability, Information and Computation (2014).


Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin, Canonical sequent proofs via multi-focusing, Fifth Itp International Conference On Theo-


Marcelo Fiore, Roberto Di Cosmo, and Vincent Balat, Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums, POPL, 2004, pp. 64–76.

Alois Brunel, Quantitative classical realizability, Information and Computation (2014).


Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin, Canonical sequent proofs via multi-focusing, Fifth Itp International Conference On Theo-


Gerhard Gentzen, Untersuchungen über das logische Schließen, Mathe-


Le Point Aveugle, Cours de logique, Tome I: Vers la Perfection, Vision des Sciences, Hermann, 2006, published subsequently in English [27].

Le Point Aveugle, Cours de logique, Tome II: Vers l'imperfection, Visions des Sciences, Hermann, 2007, published subsequently in English [27].


**Contents**

I Introduction ......................................................................................................................... 1
   I-A Abstract-machine-like calculi ................................................................. .......................... 1
   I-B Intuitionistic polarisation ......................................................................................... 2
   I-C Focusing and untyped normal forms ................................................................. 2
   I-D Notations .................................................................................................................... 2
   I-E Acknowledgements ................................................................................................. 3

II A Principled Introduction to Abstract-Machine-Like Calculi ........................................... 3
   II-A Abstract machines ................................................................................................. 3
   II-B Solving equations for expressions ......................................................................... 4
   II-C Compatible reduction and its confluence ............................................................. 4
   II-D Typing of machines ............................................................................................... 4
   II-E Solving equations for contexts .............................................................................. 4
   II-F Commutation rules are redundant ........................................................................ 5
   II-G Focalisation ............................................................................................................ 5
   II-H Inside-out contexts are primitive ....................................................................... 5
   II-I A parte: Defunctionalised CPS in direct style ...................................................... 5

III Polarisation and Extensional Sums ................................................................................ 6
   III-A Conversions on expressions vs. on commands .................................................... 6
   III-B Extensionality for sums with a regular shape ...................................................... 6
   III-C Positive sums ....................................................................................................... 6
   III-D The polarised calculus $L_{\text{int}}$ ......................................................................... 6
   III-E Relating $L_{\text{int}}$ to the $\lambda$-calculus with sums ..................................................... 8

IV The Structure of Normal Forms and Their Equivalence ................................................ 8
   IV-A The phase structure of normal forms ............................................................... 8
   IV-B Algorithmic equality for expansion rules .......................................................... 9
   IV-C Completeness of algorithmic equality on normal forms ..................................... 9
   IV-D Phase equivalence ............................................................................................... 10
   IV-E Recovering $\beta\eta$-equivalence ............................................................................ 11

References .................................................................................................................................. 11

Appendix A: Phase Normal Forms ....................................................................................... 13

Appendix B: Proofs from Section IV .................................................................................... 14

**Appendix A**

**Phase Normal Forms**

This section proves that phase equivalence is decidable.

As $(\approx_{\text{pc}})$ is not derivable, phase equivalence relates terms that are not in $(\approx_{\text{re}})$. For example, we have

$$
\begin{align*}
\langle t_1(\mu\ast\cdot, (x^+\triangleleft V\cdot \tilde{y}^+\cdot c)) \triangleright \star \rangle & \approx_{\text{pc}} \langle x^+ \triangleleft V\cdot \tilde{y}^+\cdot (t_1(\mu\ast\cdot, c)) \triangleright \star \rangle.
\end{align*}
$$

We wish to prove, in the remainder of this section, that combining $(\approx_p)$ and the algorithmic equivalence $(\approx_{A_{+}})$ of the previous section (which decides $\eta$-equivalences) captures the natural equivalence on “effect-free” commands, and in particular coincides with the $(\approx_{\beta\eta})$ equivalence of the lambda-calculus. This implies that the only difference between $(\approx_{\text{re}})$ (which we deconstructed as $(\not\approx_{\text{pc}})$-normalization followed by algorithmic equivalence) and $(\approx_{\beta\eta})$ is the addition $(\approx_{\beta\eta})$, reordering of independent constructors.

Inspired by the work on multi-focusing [9], Figure 10 defines two subrelations of $(\approx_{\beta\eta})$: a local phase equivalence $(\approx_{\text{lp}})$, permuting focused values and stacks inside a single phase, and a global phase reordering $(\approx_{\text{gp}})$. The global rewrite lets a focused commutative context of phase $\phi$ (which may be only a fragment of a larger context) jump across another context of opposite phase $\varphi$. Note that it must jump into another context of the same phase, and thus cannot split the $\phi$-context in two—we say that a focused context inside a focused command is maximal if its the surrounding contexts are of the opposite phase. This restriction is necessary for the rewrite to be terminating: with the orientation of the reduction and this restriction, the outermost maximal contexts grows bigger at each rewrite step.

**Lemma 20.** $(\triangleright_{\text{gp}})$ is terminating.

Note that the reordered $\tilde{c}^\phi_1$ may be duplicated if $c_2$ has several holes. The termination order is thus the tree of sizes of maximal contexts; the tree with the biggest element coming first is the smaller. This is well-ordered if the total size of trees arising from the rewrite of a given element is bounded. The possible rewrites of a term of size $N$ are bounded in size by $2^N$ (impossible worst case where all elements are duplicating and get rewritten in a such a way that they do duplicate the rest of the term).

The relations $(\approx_{\text{lp}})$ and $(\triangleright_{\text{gp}})$ are not just subrelations: combined, they subsume $(\approx_{\beta\eta})$.

**Lemma 21.** $f_1 \approx_{\text{lp}} f_2$ if and only if $f_1 \triangleright_{\text{gp}}^* f'_1$ and $f_2 \triangleright_{\text{gp}}^* f'_2$ with $f'_1 \approx_{\text{lp}} f'_2$.

The equivalence $(\approx_{\text{lp}})$ is decidable and local, so it is natural to manipulate terms up to this equivalence. One way to do this is to introduce an explicit multi-focusing syntax that performs a series of independent plays simultaneously on different variables and optionally on $\ast$; that corresponds to a multi-let syntax (let $x$ be $\epsilon$ in $t$) in some calculi, or to the multi-case pattern-matching construct of Altenkirch et al. [3]. Another equivalent choice, which we adopt for simplicity, is to pick an arbitrary order to order local phases.

**Definition 22.** We define $[f]_{\text{LP}}$ as an arbitrary choice of representative in the $(\approx_{\text{lp}})$-equivalence class of the focused command $f$. We will also write $[f]_{\text{GP}}$ for the unique $(\not\approx_{\text{gp}})$-normal form of $f$, and $[f]_P$ for $[[f]]_{\text{GP}}$. Finally, if $c$
is a ($\rightarrow_R$)-normalizable command, whose normal form has a (unique) focused form $f$, we will write $[c]_p$ for $[f]_p$.

**Appendix B**

**Proofs from Section IV**

**Proof of Lemma 6:**

We deterministically rewrite any normal command $c$ into a focused command $[c]$ as follows:

- $[[V_1 || V_2]] = \langle (\langle V_1 \rangle, \langle V_2 \rangle) \rangle$
- $[[x^t || e_\sigma]] = \langle \langle x^t \rangle, \langle e_\sigma \rangle, \langle \mu x^t e_\sigma \rangle \rangle$
- $[[S_1 \parallel S_2]] = \langle \langle S_1 \rangle, \langle S_2 \rangle \rangle$
- $[[\sigma \parallel e_\sigma]] = \langle \langle \sigma \rangle, \langle e_\sigma \rangle \rangle, \langle \mu \sigma e_\sigma \rangle \rangle$

We just need to extend this result to prove that $c \equiv_{RE} \langle V_1 || \mu(x_1, x_2).c[(x_1, x_2)\parallel V_2] \rangle$.

**Proof of Lemma 9:**

By induction. To justify the induction we need an induction measure that is strictly decreasing on, for example,

$c[(x_1, x_2)\parallel V_2] \equiv_A \langle (x_1, x_2) \parallel \mu(x_1, x_2).c[(x_1, x_2)\parallel V_2] \rangle$

As in the article of Abel and Coquand and in Goguen they reference, we simply need an induction measure that favors abstractors over constructors, eg:

$\langle V || q\langle \parallel c \rangle \rangle \Rightarrow \langle (q \parallel c) \parallel V \rangle$

$\langle \langle \parallel q \parallel c \rangle \parallel S \rangle \Rightarrow 2.\langle c \parallel S \rangle$

Remark then that the sum of the measures of the premises is strictly lower than the sum of the measures of its conclusion, justifying building such derivations by induction.

**Proof of Lemma 7:**

By induction on the derivation of algorithmic equivalence. For example in the $\mu(x, y)$ case, we need to prove that $\langle x^t \parallel \mu(x_1, x_2).c \rangle \equiv_{RE} c'$. Let us remark that we have, for any command $c''$:

$c'' \downarrow_{E_\theta} \langle x^t \parallel \mu x^t e_{c''} \rangle$

We just need to extend this result to prove that $c \equiv_{RE} \langle V_1 || \mu(x_1, x_2).c[(x_1, x_2)\parallel V_2] \rangle$.

**Proof of Lemma 8:**

The proof is as for the non-extended version. We used as an intermediate lemma the fact that, for any $c$, we have $c \equiv_{RE} \langle x^t \parallel \mu(x_1, x_2). c[(x_1, x_2)\parallel x^t] \rangle$. **Lemma 23.**

Any derivation of $[c_1]_{A+N} \equiv_{A+N} [c_2]_{A+N}$ can be turned into a derivation of $[c_1[q/S_1]]_{A+N} \equiv_{A+N} [c_2[q/S_1]]_{A+N}$ (resp. $[c_1[q/S_1]]_{A+N} \equiv_{A+N} [c_2[q/S_1]]_{A+N}$) of equal or smaller height when $q$ does not appear in $c_1, c_2$.

**Proof of Lemma 23:**

The proof goes by induction on the $(\equiv_{A+N})$-derivation, with a strengthened induction hypothesis: for any substitution $\sigma$, pattern $q$ and value $V_\sigma$, any derivation of $[c_1]_{\sigma} \equiv_{A+N} [c_2]_{\sigma}$ can be turned into a derivation of $[c_1[q/S_1]]_{\sigma} \equiv_{A+N} [c_2[q/S_1]]_{\sigma}$ of equal or smaller length, when $q$ does not appear in $c_1, c_2$.

Let us assume, for instance, that the last rules of our assumption are of the following form:

$$[c_1[q/S_1]]_{\sigma} \equiv_{A+N} [c_2[q/S_1]]_{\sigma}$$

We have to find a derivation for $[V_1 || \mu q_1 || c_1[q/S_1]]_{\sigma} \equiv_{A+N} [c_2[q/S_1]]_{\sigma}$ of equal or smaller length. There are two possible cases for the structure of $[V_1 || \mu q_1 || c_1[q/S_1]]_{\sigma}$ that is $\langle V_1 || \mu q_1 || c_1[q/S_1] \rangle_{\sigma}$:

- either the constructor $V_1$ of $[V_1 || \mu q_1 || c_1[q/S_1]]_{\sigma}$ does not match the abstractor $\mu q_1$, and the $\rightarrow_R$-normal form is just $\langle V_1 || q/S_1 \rangle_{\sigma}$
- or it does, and we have a head $\rightarrow_R$-reduction to perform.
In the first case, let us write $V'_1$ for $[V[q/V]]_R$. We can perform the following first steps of inference (with the top sequent still to be proved):

\[
\begin{align*}
\frac{\left[ [c_1[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \equiv_{\Lambda_N} \left[ c_2[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \right]}{\left( V'_1 \sigma \equiv \mu_{q_1} \cdot [c_1[q/V]\sigma]_R \right) \equiv_{\Lambda_N} \left[ c_2[q/V]\sigma]_R \right]}
\end{align*}
\]

Notice that $\left[ [c_1[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \right]$ is equal to $\left[ [c_1[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \right]$. The top sequent is then proved by induction hypothesis, using the larger substitution $\sigma[q_1/V'_1]\sigma]$.

In the second case, we assume that $V_1[q/V]$ starts with the same head constructor as the pattern $\sigma_1$. But we assumed that $(V_1 \parallel \mu_{q_1} c_1)$ is a normal form, so we know that $V_1 \sigma$ and $q_1$ do not match. If $V_1 \sigma$ does not match $q_1$ but $V_1[q/V]\sigma$ does, the matching part is either brought by $q$ (this means that the occurrence of $V$ in $V_1$ is at the top) or the substitution of $V[q/V]$ by $\sigma$. But the latter is impossible, given that we assumed that $\sigma$ does not appear in $q$. So we have that $V_1 = V$, and $V_1[q/V]\sigma = V[q/V]\sigma = q \rightarrow q = q$. In particular, we can always re-name $q_1$ to be equal to $q$, and then we can prove our goal

\[
\left( V_1 \parallel \mu_{q_1} c_1 \right) [q/V]\sigma]_R \equiv_{\Lambda_N} [c_2[q/V]\sigma]_R
\]

from our assumption

\[
\left[ [c_1[q/V]\sigma]_R \equiv_{\Lambda_N} [c_2[q/V]\sigma]_R \right]
\]

by the following equational reasoning:

\[
\begin{align*}
V_1 &= \equiv \left( V_1 \parallel [q/V]\mu_{q_1} c_1 \right) [q/V]\sigma]_R \equiv_{\Lambda_N} \left[ c_2[q/V]\sigma]_R \right] \\
q_1 &= \equiv \left( [q/V]\mu_{q_1} c_1 \right) [q/V]\sigma]_R \equiv_{\Lambda_N} \left[ c_2[q/V]\sigma]_R \right] \\
(q_1 = q) &\equiv \left( [V_1 [q/V]\sigma]_R \equiv_{\Lambda_N} [c_2[q/V]\sigma]_R \right) \\
(q_1 = q) &\equiv \left( [c_1[q/V]\sigma]_R \equiv_{\Lambda_N} [c_2[q/V]\sigma]_R \right)
\end{align*}
\]

Proof of lemma 10.: Combining $c_1 \equiv_{\Lambda} c_2$ and $c_2 \equiv_{\Lambda} c_3$ gives the following series of relations: $c_1 \rightarrow_{R} c_2' \equiv_{\Lambda_N} c_2' \leftarrow_{R} c_3' \rightarrow_{R} c_2' \equiv_{\Lambda} c_1' \leftarrow_{R} c_3$. By confluence of $(\rightarrow_R)$ we know that $c_2' = c_3'$ (they are both normal forms), so we have to prove transitivity with $c_1' \equiv_{\Lambda_N} c_2' \equiv_{\Lambda} c_3'$. We need to inspect both inference rules; the easy cases when $c_1$ starts with an abstractor which guides the equivalence on both sides. The more interesting cases arise when the abstractors guiding the equivalence are in $c_1'$ or $c_3'$. We detail one of those cases. Suppose we have:

\[
\begin{align*}
&\frac{c_1[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \equiv_{\Lambda_N} \left[ c_2[q/V]\sigma]_R[q_1/V'_1]\sigma]_R \right] \\
&\left( V_1 \parallel \mu_{q_1} c_1 \right) \equiv_{\Lambda_N} c_2 \\
&\frac{c_2[q/V]\sigma]_R[q_2/V]\sigma]_R \equiv_{\Lambda} \left( c_3[q/V]\sigma]_R[q_2/V]\sigma]_R \right) \\
&\frac{c_2[q/V]\sigma]_R[q_2/V]\sigma]_R \equiv_{\Lambda_N} \left( \mu q_3 c_3 \right) \equiv_{\Lambda} \left( S_3 \right)
\end{align*}
\]

Let us take $S_3' := S_3[q_1/V'_1]$. We build a derivation of the form:

\[
\begin{align*}
\left[ c_1[q_1/V'_1]\right]_R[q_3/S'_1]\sigma]_R \equiv_{\Lambda} \left[ c_2[q_1/V'_1]\right]_R[q_3/S'_1]\sigma]_R \\
\left[ c_1[q_1/V'_1]\right]_R[q_3/S'_1]\sigma]_R \equiv_{\Lambda_N} \left( \mu q_3 c_3[q_1/V'_1]\right) \equiv_{\Lambda} \left( S'_3 \right) \\
\left( V'_1 \parallel \mu q_1 \cdot c_1 \right) \equiv_{\Lambda_N} \left( \mu q_3 c_3[q_1/V'_1]\right) \equiv_{\Lambda} \left( S'_3 \right)
\end{align*}
\]

To conclude our proof, we need to prove the premise $c_1[q_1/V'_1]\parallel q_3/S'_1] \equiv_{\Lambda} c_1[q_1/V'_1]\parallel q_3/S'_1]$. (we omit the $\rightarrow_R$-normalization marks for readability of the notation). We build it by induction hypothesis, using transitivity on two $(\equiv_{\Lambda})$ derivations of smaller height. The first is $c_1[q_1/V'_1]\parallel q_3/S'_1] \equiv_{\Lambda} c_2[q_1/V'_1]\parallel q_3/S'_1]$, obtained from our hypothesis $c_1[q_1/V'_1]\parallel q_3/S'_1]$ by applying the previous lemma—it is thus strictly smaller than the derivation of $\left( V'_1 \parallel \mu q_1 \cdot c_1 \right) \equiv_{\Lambda_N} c_2$ we had as hypothesis. The second, $c_1[q_1/V'_1]\parallel q_3/S'_1] \equiv_{\Lambda} c_3[q_1/V'_1]\parallel q_3/S'_1]$ requires applying the technical lemma on our assumption $c_2 \equiv_{\Lambda_N} \left( \mu q_1 c_3 \right) \equiv_{\Lambda} \left( S_3 \right)$, to get a non-larger derivation of $c_1[q_1/V'_1]\parallel q_3/S'_1] \equiv_{\Lambda} \left( \mu q_3 c_3[q_1/V'_1] \parallel q_3/S'_1] \right)$; inspecting this derivation, which starts with the same structure as the one for $c_2 \equiv_{\Lambda_N} \left( \mu q_1 c_3 \right) \equiv_{\Lambda} \left( S_3 \right)$, gives a non-larger derivation of $c_2[q_1/V'_1]\parallel q_3/S'_1] \equiv_{\Lambda} c_2[q_1/V'_1]\parallel q_3/S'_1]$.

Proof of lemma 11.: The proof goes by proving each equation of $(\equiv_{RE})$ admissible for $(\equiv_{\Lambda})$. This is immediate for $(\rightarrow_R)$-reductions thanks to the rule A-R. For the expansions, we have for example the case $(x^+ \parallel e_+ \rightarrow_{E_{\emptyset}} (x^+ \parallel \mu \langle x_1, x_2 \rangle \parallel e_+))$ which is proved by the following inference:

\[
\begin{align*}
\langle x_1, x_2 \rangle \parallel e_+ \rightarrow_{E_{\emptyset}} \left( x^+ \parallel \mu x_1.x_2 \parallel e_+ \right)
\end{align*}
\]

Note that we do not need to handle the case of the extensions $E_{\mu}$ and $E_{\tilde{\mu}}$, only those that inspect value or stack constructors. Indeed, putting a normal form into focused form, as described in Lemma 6, normalizes the $E_{\mu}, \tilde{\mu}$ structure; we have $f \vdash_{E_{\mu, \tilde{\mu}}} f'$ only if $f = f'$.

Proof of lemma 14.: Consider for example the case $c_1 = (x^+ \parallel \tilde{\mu} \langle z_1[1] \parallel z_2[1] \parallel z_3[1] \rangle)$. Given an arbitrary $c_2$, let us define:

\[
e d_1 \equiv \tilde{\mu} x^+, c_2[c_1] = (x^+ \parallel \mu \langle z_1[1] \parallel z_2[1] \parallel z_3[1] \rangle)
\]

Then we have:

\[
c_2[c_1[1]] \rightarrow_{E_{\emptyset}} (x^+ \parallel \mu \langle z_1[1] \parallel e_+ \rangle \parallel c_2[c_1]) \rightarrow_{E_{\emptyset}} (x^+ \parallel \mu \langle z_1[1] \parallel c_2[c_1[1] \parallel z_2[1] \parallel z_3[1] \parallel z_4[1] \rangle
\]

This concludes the proof in the case $c_1 = (x^+ \parallel \tilde{\mu} \langle z_1[1] \parallel z_2[1] \parallel z_3[1] \rangle)$. The other cases are rather similar. For example, if $c_1 = (x^+ \parallel \tilde{\mu} (y_1, y_2)[1])$, then again with
We have that $f$ such that for some fresh $\langle \cdot \rangle^*$,

$$
\rightarrow_{R_{\beta}}^{*} (\vec{x}^{\dagger} || \vec{y}(y_1, y_2))\rightarrow_{E_{\beta}}^{*} (\vec{x}^{\dagger} || \vec{y}(y_1, y_2))
$$

$\Rightarrow f[c_2]$

Proof of 18: We show that $(\simeq_M) \subseteq (\simeq_{R_{\beta}})$ and $(\simeq_W) \subseteq (\simeq_{R_{\beta}})$ separately.

For $(\simeq_W) \subseteq (\simeq_{R_{\beta}})$ we have to show that if $c[.]$ is a commutative context and $f$ uses no variable bound by $c[.]$ over its hole(s), then $c[f] \simeq_{R_{\beta}} f$. By case analysis on commutative contexts, we see that $c[.]$ is necessarily of the form $c'[\vec{\mu q}^{+}[\vec{\delta}]]$ (c cannot bind $\star$ as $f$ necessarily uses it). We thus have that $c[f] = c'[\vec{\mu q}^{+}[\vec{\delta}]].c_{\mu_q^{+}[\vec{\delta}]}$ for a fresh $x$. We now prove that, if $f$ does not use $q$, then $\vec{\mu q}^{+}[\vec{\delta}].f \simeq_{R_{\beta}} \vec{\mu q}^{+}[\vec{\delta}].f$ for a fresh $y$; this allows to conclude with $\langle \vec{\mu}^{+}.c'[\star]\rangle^{*} [\vec{\mu q}^{+}[\vec{\delta}], \vec{\mu q}^{+}[\vec{\delta}]] \simeq_{R\theta} \langle \vec{\mu}^{+}.c'[\star]\rangle^{*} [\vec{\mu q}^{+}[\vec{\delta}], \vec{\mu q}^{+}[\vec{\delta}]] \bowtie_{R_{\beta}} f$.

If $\vec{\mu q}^{+}[\vec{\delta}].f$ is of the form $\vec{\mu z}^{+}[\vec{\delta}].f$ for some variable $z$, the result is immediate by $a$-conversion.

If $\vec{\mu q}^{+}[\vec{\delta}].f$ is of the form $\vec{\mu z}^{+}[\vec{\delta}].f$ where $f$ does not use $z_1$ or $z_2$, then we have $\vec{\mu z}^{+}[\vec{\delta}].f \simeq_{R_{\beta}} \vec{\mu z}^{+}[\vec{\delta}].f$ for some fresh $x^\prime$. This is the $(E_{\otimes})$-expansion of $\vec{\mu q}^{+}[\vec{\delta}]$. If $\vec{\mu q}^{+}[\vec{\delta}].f$ is of the form $\vec{\mu z}^{+}[\vec{\delta}].f$ where $f$ does not use $z$, we have that $\vec{\mu z}^{+}[\vec{\delta}].f \simeq_{R_{\beta}} \vec{\mu z}^{+}[\vec{\delta}].f$ for some fresh $x^\prime$, and this is the $(E_{\otimes})$-expansion of $\vec{\mu q}^{+}[\vec{\delta}]$.

For $(\simeq_M) \subseteq (\simeq_{R_{\beta}})$, we have to prove that for any pair of independent commutative contexts $c_1^{\delta} 
parallel c_2^{\delta}$ and substitution $\rho$ such that $c_1^{\delta} = c_2^{\delta}[\rho]$, we have $c_1^{\delta}[c_2^{\delta}[f]] \simeq_{R_{\beta}} c_2^{\delta}[c_2^{\delta}[f]]$ for any focused command $f$. Again by case analysis on commutative contexts we see that $c_1, c_2$ cannot bind $\star$: a context that binds $\star$ over its hole must necessarily have a free occurrence of it, and thus cannot be independent with (a-renaming) of itself. Thus $c_1, c_2$ are of the form $c'[\vec{\mu q}^{+}[\vec{\delta}]]$, $c'[\vec{\mu q}^{+}[\vec{\delta}]]$ for $q_1$ and $q_2$ $a$-equivalent through the variable-variable substitution $\rho$. We prove that $c'[\vec{\mu q}^{+}[\vec{\delta}]].c'[\vec{\mu q}^{+}[\vec{\delta}]].f_{q_1,q_2}$ as follows for some fresh $q = q_1 = q_2$:

$$
\Rightarrow_{R_{\mu}} \langle \vec{\mu}^{+}.c'[\star]\rangle^{*} [\vec{\mu q}^{+}[\vec{\delta}], \vec{\mu q}^{+}[\vec{\delta}]]
$$

Proof of theorem 19: We have to show that $\langle x_{\dagger} \parallel \vec{\mu x}^{+}.c_0 \rangle \simeq_{R_{\beta}} c_0 [t]_d x_{\dagger}^{+}$. The term $t$ is either of

the form $t^* \cdot d_0$, or it is a value—in which case the result is immediate. Let us define $c_1$ as the focused-normal form of $c_0$, and $d_1$ as the focused normal form of $d_0$. We also define as $d[f]$ the command context, with holes expressing contexts, such that $d[f] = d_1$ and $\star \notin \mathbf{fv}(d)$, and as $c[.]$ the command context, with holes expressing expressions, such that $c[x^\dagger] = c_1$ and $x^\dagger \notin \mathbf{fv}(c)$. We need to prove that $\langle \vec{\mu}^{+}.d[\star] \rangle^{*} \simeq_{R_{\beta}} \langle \vec{\mu}^{+}.d[\star] \parallel \vec{\mu} x^{+}.c[x^\dagger] \rangle$, the latter being $(\simeq_{R_{\beta}})$-equivalent to $d[\mu x^{+}.c[x^\dagger]]$.

The main idea of the proof is that $c[\vec{\mu}^{+}[\star]]$ is covered by (compositions of) commutative contexts independent from $d[\vec{\mu} x^{+}[\star]]$: it can be described as a multi-hole context $c[\vec{\mu} x^{+}[\star]$, $\ldots$, $\vec{\mu} x^{+}[\star]]$ such that for each family of values $V_1, \ldots, V_n$ and each family of holes $i \in [1; n]$, $c[V_1] \cdots V_{i-1} || V_i || V_{i+1} \cdots V_n$ is a composition of commutative contexts so that $c[V_1] \cdots V_{i-1} || V_i || V_{i+1} \cdots V_n \simeq_{p} d[\vec{\mu} x^{+}.c[V_1] \cdots V_{i-1} || V_i || V_{i+1} \cdots V_n]$.Note that the commutation is possible because the contexts are independent. Indeed, the commutative contexts of $c[\vec{\mu}^{+}[\star]]$ neither use $x^\dagger$ (by construction of $d[\star]$) nor bind it over its hole (any expression binder could be $a$-renamed), and $d[\vec{\mu} x^{+}]$ does not use $\star$ by construction, and cannot bind it over its hole, for $\star$ would then be shadowed in $d_1 = d[\star]$.

In the special case $n = 0$, $c[.]$ is in fact a focused command $c$, and we have to show that $d[\vec{\mu} x^{+}].c \simeq_{R_{\beta}} c$. This is immediate by weakening simplification $(\bowtie_{W})$, as $x^\dagger$ is not used in $c$ by construction.

Otherwise ($n \geq 1$), by iteration on $[1; n]$ we can reorder $n$ copies of $d[\vec{\mu} x^{+}[\star]]$ in front of $c$, obtaining the term:

$$
d[\vec{\mu} x^{+}[\star] \cdots d[\vec{\mu} x^{+}.c[\vec{\mu}^{+}[\star] || \star] \cdots [\vec{\mu}^{+}[\star]], (\star || \star)]$$

those copies can be merged, so this is $(\simeq_{M})$-equivalent to $d[\mu x^{+}.c[x^\dagger]]$ as desired.

Note that there is no need to interleave merging and phase reordering, as merging does not create new opportunities for phase reordering. It is essential, however, to apply merging on the maximally-reordered command, as reordering command creates merging opportunities.

We remark that there is a fundamental asymmetry in the proof technique that follows the asymmetry of the intuitionistic restriction: we permute, duplicate and reorder the value context $d[\vec{\mu} x^{+}[\star]]$ rather than the control context $c[\vec{\mu}^{+}[\star]]$, relying on a notion of “purity” that would be invalid in the full classical setting.