Formulae-as-Types for an Involutive Negation

Guillaume Munch-Maccagnoni

Formulae-as-Types for an Involutive Negation *

Guillaume Munch-Maccagnoni
Université Paris 13, Sorbonne Paris Cité, LIPN, F-93430, Villetaneuse, France
Munch-Maccagnoni@lipn.univ-paris13.fr

Abstract

Negation is not involutive in the λC calculus because it does not distinguish captured stacks from continuations. We show that there is a formulae-as-types correspondence between the involutive negation in proof theory, and a notion of high-level access to the stacks studied by Felleisen and Clements.

We introduce polarised, untyped, calculi compatible with extensionality, for both of classical sequent calculus and classical natural deduction, with connectives for an involutive negation. The involution is due to the ℓ delimited control operator that we introduce, which allows us to implement the idea that captured stacks, unlike continuations, can be inspected. Delimiting control also gives a constructive interpretation to falsity. We describe the isomorphism there is between A and $\neg\neg A$, and thus between $\neg\forall$ and $\exists\neg$.

Categories and Subject Descriptors F.3.3 [*Logics and meanings of programs*]: Studies of Program Constructs; F.4.1 [*Mathematical logic and formal languages*]: Mathematical Logic—Lambda calculus and related systems, Proof theory

Keywords Classical logic, Formulae-as-types, Delimited control operators, Continuations, Polarization, Focalization.

1. Introduction

Constructiveness in classical logic is based on different assumptions than in intuitionistic logic. In order to give constructive contents to classical axioms such as excluded middle ($\forall A, A \vee \neg A$), we assume that we restrict the disjunction property, the property of existence… to formulae that are purely positive (Girard [14]). For instance, in arithmetic, purely positive formulae correspond to Σ^0_1 formulae. As a consequence, proofs of formulae $\forall\vec{x}(P(\vec{x}) \to Q(\vec{x}))$ where P, Q are purely positive—that is to say Π^0_2 formulae in arithmetic—correspond to algorithms (Murthy [35]).

One way to provide constructive contents to classical proofs is by considering variants of the Gödel-Gentzen double-negation translations, such as Friedman's [12]. We can translate classical proofs of $\vdash P$ into intuitionistic proofs of $\vdash (P' \to R) \to R$, where P' is the translation of P and where R is chosen arbitrarily. When P is purely positive, then P' does not depend on R and, furthermore, we have $P' = P$. Therefore, in this special case we can take $R = P$ and deduce an intuitionistic proof of $\vdash (P \to P) \to P$ and, in turn, one of $\vdash P$. In the other cases, the translation describes in fact how the behaviour of classical proofs depends on the context in which classical axioms are invoked. Thus the interpretation does not contradict intuitionism: the latter assumes that the behaviour of proofs is referentially transparent, while here we do not make such an assumption. This is why proofs of $\forall A, A \vee \neg A$ do not provide a decision procedure for A in general.

In this article, we are interested in the constructive interpretation of reasoning by contrapositive, or, in other words, of an involutive negation. In terms of double-negation translations, we must consider two techniques:

- Girard's polarisation [14], which distinguishes *negative* formulae N (which are translated differently, into $\vdash N^* \to R$) in addition to the positive P, and which was introduced precisely for the purpose of interpreting a negation satisfying $\neg\neg A = A$;
- The interpretation of falsity from Herbelin, Ariola and Ghilezan [1, 18, 4], closely related to Friedman's trick [12].

$$t, u ::= x \mid \lambda x.t \mid t\,u \mid C$$

(a) Quasi-proof terms

$$\frac{}{\Gamma, x : N \vdash x : N} \qquad \frac{}{\Gamma \vdash C : \neg\neg N \to N}$$

$$\frac{\Gamma, x : N \vdash t : M}{\Gamma \vdash \lambda x.t : N \to M} \qquad \frac{\Gamma \vdash t : N \to M \quad \Gamma \vdash u : N}{\Gamma \vdash t\,u : M}$$

$$\frac{\Gamma \vdash t : N}{\Gamma \vdash t : \forall \mathrm{x}\, N}\ {}^* \qquad \frac{\Gamma \vdash t : \forall \mathrm{x}\, N}{\Gamma \vdash t : N[\mathrm{u/x}]}$$

(b) First-order predicate calculus (*: x not free in Γ)

$$c ::= \langle t \,\|\, \pi \rangle \qquad t, u ::= \ldots \mid \mathsf{k}_\pi \qquad \pi ::= \mathsf{stop} \mid t \cdot \pi$$

(c) Machines, terms and stacks

$$\begin{aligned} \langle t\,u \,\|\, \pi \rangle &\succ_n \langle t \,\|\, u \cdot \pi \rangle \\ \langle \lambda x.t \,\|\, u \cdot \pi \rangle &\succ_n \langle t[u/x] \,\|\, \pi \rangle \\ \langle C \,\|\, t \cdot \pi \rangle &\succ_n \langle t \,\|\, \mathsf{k}_\pi \cdot \mathsf{stop} \rangle \\ \langle \mathsf{k}_\pi \,\|\, t \cdot \pi' \rangle &\succ_n \langle t \,\|\, \pi \rangle \end{aligned}$$

(d) Reductions

Figure 1. The λC calculus and its call-by-name abstract machine

1.1 Formulae-as-types

The previous notion of constructiveness is best understood through the interpretation of *formulae as types* in a programming language. Griffin [16] showed that double negation elimination can be implemented with Felleisen's variant $\mathcal{C}$ [11] of the *call/cc* operator of Scheme (Figure 1 recalls the λC calculus). Also, from this point of view, double-negation translations such as Girard's correspond to continuation-passing style compilation schemes for calculi with such

* This paper is a shortened version of the fourth chapter of the author's PhD thesis [32].

CSL-LICS '14, July 14–18, 2014, Vienna, Austria.

ACM 978-1-4503-2886-9/14/07…$15.00.
http://dx.doi.org/10.1145/2603088.2603156

Calculus	Fig.	Technique	Style	Red.	Equiv.	Strategy	$\neg\neg A \cong A$	Ref.
λC	1	Quasi-proof terms and abstract machine	Natural deduction	$\succ_n$	$\approx_n$	Call-by-name	No	[41]
$\bar{\lambda}\mu\tilde{\mu}_T$	2	L calculus	Sequent calculus	$\rhd_{R_n}$	$\simeq_{RE_n}$	Call-by-name	No	[7]
$L_{pol,\hat{\mathfrak{P}}}$	3	L calculus	Sequent calculus	$\rhd_{R_p}$	$\simeq_{RE_p}$	Polarised	Yes	
$\lambda\mathscr{C}$	4	Quasi-proof terms and abstract machine	Natural deduction	$\succ_p$	$\approx_p$	Polarised	Yes	

Table 1. A summary of the calculi presented in this article

operators (Murthy [34, 36]). Now, as is well known, in the presence of side-effecting operations (control operators, state, input/output, etc.), programs of certain types (functions, thunks, etc.) are opaque at runtime. In fact, programming languages usually guarantee to respond with absolute values only for simple enough types (integers, etc.).

One may think that double-negation translations have reduced classical proof theory to intuitionistic proof theory. But, given the wealth of double-negation translations, understanding the translations is at least as important as understanding the intuitionistic target. A *direct* interpretation of classical logic, by means of dedicated term calculi, amounts to studying both at the same time. It brings the combined advantages of succinctness and flexibility.

The simplest way to describe the reduction of calculi with control operators is with abstract machines. Let us recall Streicher and Reus's [41] variant of the Krivine (call-by-name) abstract machine in Figure 1. One insight is to enrich the set of terms with an operation k_π that appears during reduction, but which does not make sense for natural deduction.

The original terms, without k_π, are *quasi-proofs*. This is because we consider that logical systems only approximate constructive behaviours, just like type systems approximate correct programs—a point of view advocated by Krivine [23, 21] and Girard [15]. Thus, quasi-proofs are algorithmically more diverse than proofs, because they need not be typable. But they are somehow compatible with natural deduction, unlike k_π, in the sense that we could use them to extend the logic with new axioms. In this article, stating the results for untyped calculi guarantees more generality.

1.2 Involutive negation

Assuming $\neg A \stackrel{\text{def}}{=} A \to \bot$, then the C control operator participates in an equivalence $\neg\neg A \leftrightarrow A$, but not in an isomorphism of types $\neg\neg A \cong A$. The reason for asking more than a mere equivalence between A and $\neg\neg A$ is that there are too many choices for the contrapositive of a proposition such as the following:

$$\forall \mathrm{x}, \mathrm{y} \in A, (P(\mathrm{x}) \vee Q(\mathrm{y})) \to (\forall \mathrm{x} \in A, P(\mathrm{x})) \vee (\forall \mathrm{y} \in A, Q(\mathrm{y}))$$

for instance:

$$\begin{aligned}
\neg\big((\forall \mathrm{x}{\in}A, P(\mathrm{x})) \vee (\forall \mathrm{y} \in A, Q(\mathrm{y}))\big) &\to \neg\forall \mathrm{x}, \mathrm{y} \in A, (P(\mathrm{x}) \vee Q(\mathrm{y})) \\
(\neg\forall \mathrm{x}{\in}A, P(\mathrm{x})) \wedge (\neg\forall \mathrm{y}{\in}A, Q(\mathrm{y})) &\to \exists \mathrm{x}, \mathrm{y} \in A, \neg(P(\mathrm{x}) \vee Q(\mathrm{y})) \\
(\exists \mathrm{x}{\in}A, \neg P(\mathrm{x})) \wedge (\exists \mathrm{y}{\in}A, \neg Q(\mathrm{y})) &\to \exists \mathrm{x}, \mathrm{y} \in A, (\neg P(\mathrm{x}) \wedge \neg Q(\mathrm{y}))
\end{aligned}$$

We are no longer overwhelmed with choices once De Morgan laws are type isomorphisms: if there are too many proofs, then we must be able to choose a canonical one, one whose meaning is preserved.

In the λC calculus, however, a proposition as simple as the following (assuming that $\exists$ is obtained through the second-order encoding):

$$\neg\forall \mathrm{x} \in \mathbb{N}, A \to \exists \mathrm{y} \in \mathbb{N}, \neg A \tag{1}$$

has a proof with the following skeleton:

$$\lambda xy.(C\ \lambda k.(x\ \lambda e.(C\ \lambda l.(k\ (y\ e\ l)))))$$

Thus, in λC, such an elementary classical tautology has a non-obvious computational role, due to the presence of two control operators. More generally, reasoning by contrapositive is not immediate in the λC calculus due to the absence of isomorphisms of types such as $\neg\forall \mathrm{x}\ N \cong \exists \mathrm{x}\ \neg N$, following from the absence of an involutive negation. But, works such as the one of Krivine [23] show from a technical standpoint the importance of reasoning by contrapositive; for instance, the axiom of countable choice is only realised through its contrapositive.

1.3 Captured contexts are not continuations

As we will see, the computational contents of the De Morgan law (1) is as simple to understand as: *1)* capturing the stack of the form $n{\cdot}\pi$ that appears during the head reduction of the argument; *2)* decomposing it into head and tail, and return a pair (n, k_π) where k_π is the captured form of the tail. This contrasts with the proof in the λC calculus, which is convoluted because captured contexts are identified with *continuations*. By continuation we mean the functional abstraction of the remainder of a computation. Thus, the contents of the captured context, which is represented by a functional value, cannot be accessed in an immediate way.

That captured contexts are more primitive than continuations is obvious in programming languages, with the examples of the operation *getcontext* in the language C or the operation *thisContext* in the language Smalltalk, where the contents of captured contexts can be accessed. Felleisen sketched a control operator that would theorise this distinction in a note at the end of Ariola and Herbelin [2]. Then, Clements showed advantages of enabling a high-level access to the components of the contexts, such as obtaining portable, high-level implementations for debuggers [5].

In this paper, we establish the link between the idea theorised by Felleisen and Clements and the remarks below by Girard and by Krivine by which negation should be treated differently from the connective $\cdot \to \bot$.

1) Girard gave with his sequent calculus **LC** [14] an interpretation for an involutive negation. Girard's approach distinguishes positive $(P, Q \ldots)$ and negative $(N, M \ldots)$ formulae. Negation is given as an involutive mapping that inverts the polarity of the formula, thus it is strictly involutive $(\neg\neg A = A)$. It is therefore not given as a connective, which led some authors to qualify the computational contents of **LC**'s negation as "*not clear*".[1] However, according to Girard [14, p. 9], the identification of $\neg\neg A$ with A "*is not essential to* [his] *approach*".

In fact, the main insight of **LC** is, to us, the idea that the introduction rules in sequent calculus of negation, taken as a connective, hide cuts. In other words, the following *focalisation* steps have to be performed during cut-elimination in classical sequent calculus, when π is not simple enough (i.e. linear):

$$\dfrac{\dfrac{\Gamma, N \vdash \Delta}{\Gamma \vdash \neg N, \Delta}^{\pi} \qquad \Gamma', \neg N \vdash \Delta'^{\pi'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \rhd \dfrac{\dfrac{\dfrac{}{N \vdash N}}{\vdash \neg N, N} \qquad \Gamma', \neg N \vdash \Delta'^{\pi'}}{\Gamma' \vdash N, \Delta'} \qquad \Gamma, N \vdash \Delta^{\pi}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

$$\dfrac{\Gamma' \vdash \neg P, \Delta'^{\pi'} \qquad \dfrac{\Gamma \vdash P, \Delta}{\Gamma, \neg P \vdash \Delta}^{\pi}}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \rhd \dfrac{\Gamma \vdash P, \Delta^{\pi} \qquad \dfrac{\Gamma' \vdash \neg P, \Delta'^{\pi'} \qquad \dfrac{}{P \vdash P}}{P, \neg P \vdash}}{\Gamma, P \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

[1] See M.-M. [32, Section I.10.3] for a discussion.

These cut-elimination steps invert the order of π and π' in the tree. As we will see, the first cut-elimination step above is incompatible with the functional interpretation of negation ($N \to \bot$), and requires that $\neg N$ is given a positive polarity. The second cut-elimination step above is compatible with interpreting $\neg P$ as the negative type $P \to \bot$, but will correspond to evaluating the argument π by value.

2) A technique of Krivine alleviates the complexity of reasoning in the λC calculus, by allowing certain pseudo-types of a positive tinge to the left-hand side of implications [23, 22]. An essential pseudo-type in Krivine's work is $\mathcal{X}^-$, defined as the set $\{\mathsf{k}_\pi \mid \pi \in \mathcal{X}\}$. This also amounts to distinguishing a type of captured stacks from the type of continuations ($\mathcal{X} \to \bot$). The difference is, we will do so in a direct manner, making such types first class, in the sense that we define their meaning also when they are on the right-hand side of implications.

1.4 Formulae-as-types for an involutive negation

Our goal is to describe a natural deduction for classical logic with an involutive negation inspired by Girard, which realises a formulae-as-types correspondence with the idea of exposing a high-level interface to captured stacks. Following the above, we introduce a positive type $\sim A$ of *inspectable stacks*, together with constants ($D_{\dots}$) that provide access to the components of these stacks. The type $\sim A$ is therefore distinct from the negative type $A \to \bot$ of continuations. The involutive negation is then defined in function of the polarity:

$$\neg P \stackrel{\text{def}}{=} P \to \bot \qquad\qquad \neg N \stackrel{\text{def}}{=} \sim N$$

Defining the negation in function of the polarity of the sub-formula is reminiscent of Danos, Joinet and Schellinx [8].

Our setting admits extensionality (η-like) rules, which are used to establish the isomorphism $\neg\neg A$ and A. It is also untyped (Curry-style), so that there is generality in the involution result and it is not restricted to, say, predicate calculus.

Delimited control as an interpretation of falsity. In the setting that we introduce, we use control delimiters to provide a constructive interpretation for falsity ($\bot$). Control delimiters model the fact that the context has a finite extent, and are absent from the λC calculus. In terms of continuation-passing-style translations, the presence of control delimiters means that continuations can return and be composed. In this aspect, we follow the proof theoretic interpretation of delimited control by Herbelin and others [18, 4, 17, 20].

We found necessary to interpret falsity using delimited control because of our choice of being untyped. In this context, giving an interpretation to falsity compatible with extensionality, as we will see, is not immediate.[2] Thus our result shows how although delimited control does not prove new formulae compared to non-delimited control in the context of predicate calculus, it already gives better proofs from a constructive standpoint.

Computational interpretation of polarisation. Our approach, like Girard's, is only sound when both polarities are taken into account. One important contribution of this article is that we investigate a direct computational interpretation of polarities.

We build on our previous work [30] where negation was strictly involutive. There, we introduced the idea that positive and negative formulae are the types of strict and lazy terms (respectively) for the notion of polarity taken from Girard's **LC**. Strict involution meant that terms were identified with contexts. In this article, we restore the distinction between terms and contexts, and we also investigate how polarisation can be formulated in natural deduction, through a λ calculus with control operators (the $\lambda\ell$ calculus we introduce). We found that the latter task was not immediate, and we are not aware of any other model of an involutive negation in natural deduction which does not identify all proofs.

There is a technical reason for investigating a natural deduction variant of polarised classical logic. It is needed for having a notion of quasi-proofs. Yet quasi-proofs are essential in Krivine's programme for recovering models, in the sense of model theory, from classical realisability models [24]. Like the λC calculus, the $\lambda\ell$ calculus determines such a language of quasi-proof terms, in a sense that we will make precise.

This article is a companion to M.-M. [33] in which we give a direct characterisation of polarisation through a categorical structure where the associativity of composition is relaxed. The $\lambda\ell$ calculus fails to satisfy the associativity of composition, which is justified by the latter result.

1.5 Contributions

In Section 2, we review the relationship between sequent calculus and abstract machines and we introduce the untyped calculus $L_{pol,\hat{\mathsf{tp}}}$ and the corresponding sequent calculus. Not all proofs are identified, even in the presence of η-like rules.

In Section 3, we introduce the untyped calculus $\lambda\ell$ which provides the notion of programs, or quasi-proof terms, and we introduce the corresponding natural deduction. The method is to decompose it through the calculus $L_{pol,\hat{\mathsf{tp}}}$, from which it inherits an extensional equivalence on terms and therefore the type isomorphisms. Finally, in Section 4 we show that negation as defined above is involutive.

References. This article can be seen as a successor to Murthy's computational analysis of Girard's **LC** [36], enriched with the subsequent advances in the proof theory of classical sequent calculus (in particular Danos, Joinet and Schellinx [8] and Laurent [26]), in the representation and computational interpretation of sequent calculus (Curien and Herbelin [7]) and in the theory of (delimited) control operators (in particular Herbelin *et al.* [3, 2, 4]).

Acknowledgements. I wish to thank Pierre-Louis Curien, Pierre-Évariste Dagand, Olivier Danvy and Thomas Streicher for their careful reading, criticisms and suggestions about the thesis chapter. I wish to thank Jean-Yves Girard, Hugo Herbelin, Paul-André Melliès and Alexandre Miquel for the related discussions. Thanks to the anonymous reviewers for criticisms and suggestions. This work has been partially supported by the French National Research Agency[3].

2. Involutive negation in sequent calculus

2.1 Notations

If $\rhd$ is a rewriting relation, then $\rhd^*$ denotes the reflexive and transitive closure of $\rhd$; the *compatible* closure of $\rhd$ is denoted by $\to$ and the compatible equivalence relation $(\leftarrow \cup \to)^*$ is denoted by $\simeq$. Reductions are denoted with $\rhd_{\mathrm{R}}$ and expansions with $\rhd_{\mathrm{E}}$. In this context we define $\rhd_{\mathrm{RE}} \stackrel{\text{def}}{=} \rhd_{\mathrm{R}} \cup \rhd_{\mathrm{E}}$.

2.2 The constructive interpretation of sequent calculus

The λ calculus is universal because it represents combinators abstractly by their reduction rules. For instance, it suffices to write down the rule $\mathsf{S}xyz \succ xz(yz)$ to not only infer $\mathsf{S} \simeq_{\beta\eta} \lambda xyz.xz(yz)$, but also to propose $\mathsf{S} = \lambda xyz.xz(yz)$ as a definition, as if the reduction rule was an equation in S and the λ calculus a convenient mean to build solutions.

[2] We could also circumvent the issue of the connective $\bot$ by replacing the definition $\neg P \stackrel{\text{def}}{=} P \to \bot$ by an *ad hoc* dual to the connective $\sim$, but: *1)* that would leave open the question of its interaction with implication, and *2)* it is not clear that such a connective has a convincing formula-as-type interpretation.

[3] ANR-07-BLAN-0324 Choco, ANR-12-JS02-006-01 Coquas, ANR-10-BLAN-0213 Logoi and ANR-11-BS02-0010 Recre

$$t = t_{\ominus} ::= x \mid \mu\alpha.c \mid \lambda x.t$$
$$e = e_{\ominus} ::= \pi \mid \tilde{\mu}x.c$$
$$e_{\ominus} \supseteq \pi ::= \alpha \mid \mathsf{stop} \mid t\cdot\pi$$
$$c ::= \langle t \,\|\, e\rangle$$

(a) Terms, contexts, stacks and commands

$$\langle t \,\|\, \tilde{\mu}x.c\rangle \rhd_{\mathrm{R}_n} c[t/x]$$
$$\langle \mu\alpha.c \,\|\, \pi\rangle \rhd_{\mathrm{R}_n} c[\pi/\alpha]$$
$$\langle \lambda x.t \,\|\, u\cdot\pi\rangle \rhd_{\mathrm{R}_n} \langle t[u/x] \,\|\, \pi\rangle$$

(b) Reduction rules

$$e \rhd_{\mathrm{E}_n} \tilde{\mu}x.\langle x \,\|\, e\rangle$$
$$t \rhd_{\mathrm{E}_n} \mu\alpha.\langle t \,\|\, \alpha\rangle$$
$$t \rhd_{\mathrm{E}_n} \lambda x.\mu\alpha.\langle t \,\|\, x\cdot\alpha\rangle$$

(c) Expansion rules

$$t\,u \stackrel{\text{def}}{=} \mu\alpha.\langle t \,\|\, u\cdot\alpha\rangle$$
$$\mathsf{k}_e \stackrel{\text{def}}{=} \lambda x.\mu\alpha.\langle x \,\|\, e\rangle$$
$$\mathcal{C} \stackrel{\text{def}}{=} \lambda x.\mu\alpha.\langle x \,\|\, \mathsf{k}_\alpha\cdot\mathsf{stop}\rangle$$

(d) Embedding the $\lambda\mathcal{C}$ calculus

$$\Gamma = \vec{x_i} : \vec{N_i} \quad , \quad \Delta = \vec{\alpha_j} : \vec{M_j}$$
$$\Gamma \vdash t : N \mid \Delta \quad , \quad \Gamma \mid e : N \vdash \Delta \quad , \quad c : (\Gamma \vdash \Delta)$$

(e) Judgements

$$\frac{c : (\Gamma \vdash \Delta)}{c : (\Gamma, x : A \vdash \Delta)}\ (\mathrm{w}\vdash) \qquad \frac{c : (\Gamma, x : A, y : A \vdash \Delta)}{c[x/y] : (\Gamma, x : A \vdash \Delta)}\ (\mathrm{c}\vdash)$$
$$\frac{c : (\Gamma \vdash \Delta)}{c : (\Gamma \vdash \alpha : A, \Delta)}\ (\vdash \mathrm{w}) \qquad \frac{c : (\Gamma \vdash \alpha : A, \beta : A, \Delta)}{c[\alpha/\beta] : (\Gamma \vdash \alpha : A, \Delta)}\ (\vdash \mathrm{c})$$

and 8 similar rules with t and e replacing c.

(g) Structure

$$\frac{}{x : N \vdash x : N \mid}\ (\vdash \mathrm{ax}) \qquad \frac{}{\mid \alpha : N \vdash \alpha : N}\ (\mathrm{ax}\vdash)$$
$$\frac{c : (\Gamma, x : N \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : N \vdash \Delta}\ (\tilde{\mu}\vdash) \qquad \frac{c : (\Gamma \vdash \alpha : N, \Delta)}{\Gamma \vdash \mu\alpha.c : N \mid \Delta}\ (\vdash \mu)$$
$$\frac{\Gamma \vdash t : N \mid \Delta \qquad \Gamma' \mid e : N \vdash \Delta'}{\langle t \| e\rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')}\ (\mathrm{cut})$$

(f) Identity

$$\frac{\Gamma, x : N \vdash t : M \mid \Delta}{\Gamma \vdash \lambda x.t : N \to M \mid \Delta}\ (\vdash \to) \qquad \frac{\Gamma \vdash t : N \mid \Delta \qquad \Gamma \mid \pi : M \vdash \Delta}{\Gamma \mid t\cdot\pi : N \to M \vdash \Delta}\ (\to \vdash_f)$$
$$\frac{\Gamma \vdash t : N \mid \Delta}{\Gamma \vdash t : \forall \mathrm{x}\, N \mid \Delta}\ (\vdash \forall^1)^* \qquad \frac{\Gamma \mid e : N[\mathrm{t/x}] \vdash \Delta}{\Gamma \mid e : \forall \mathrm{x}\, N \vdash \Delta}\ (\forall^1 \vdash)$$
$$\frac{}{\Gamma \mid \mathsf{stop} : \bot \vdash \Delta}\ (\bot \vdash)$$

(h) Logic

Figure 2. The $\bar{\lambda}\mu\tilde{\mu}_T$ calculus (top) and its typing rules in sequent calculus (bottom)

The sequent calculus ("L") approach, pictured by Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}_T$ calculus (Figure 2, top), is universal because it extends the previous point of view to the transitions of abstract machines. The binder μ, which binds stacks to co-variables $\alpha, \beta \ldots$, is introduced to abstractly represent terms by their transitions in a machine. For instance, the operations of the $\lambda\mathcal{C}$ calculus are characterised by their action on stacks (below, on the left), which define equations that we solve in the $\bar{\lambda}\mu\tilde{\mu}_T$ calculus (on the right):

$$t\,u : \quad \pi \mapsto \langle t \,\|\, u\cdot\pi\rangle \qquad t\,u \stackrel{\text{def}}{=} \mu\alpha.\langle t \,\|\, u\cdot\alpha\rangle$$
$$\mathsf{k}_\pi : \quad t\cdot\pi' \mapsto \langle t \,\|\, \pi\rangle \qquad \mathsf{k}_e \stackrel{\text{def}}{=} \lambda x.\mu\alpha.\langle x \,\|\, e\rangle$$
$$\mathcal{C} : \quad u\cdot\pi \mapsto \langle u \,\|\, \mathsf{k}_\pi\cdot\mathsf{stop}\rangle \qquad \mathcal{C} \stackrel{\text{def}}{=} \lambda x.\mu\alpha.\langle x \,\|\, \mathsf{k}_\alpha\cdot\mathsf{stop}\rangle$$

These definitions induce a translation from the $\lambda\mathcal{C}$ calculus into the calculus $\bar{\lambda}\mu\tilde{\mu}_T$ that simulates reduction:

Proposition 1 (Simulation)**.** *(Identifying machines of the $\lambda\mathcal{C}$ calculus with commands of the calculus $\bar{\lambda}\mu\tilde{\mu}_T$,) If $c \succ_n c'$ then $c \rhd^+_{\mathrm{R}_n} c'$.*

The abstract notation of reduction rules brings along three important simplifications [7, 42, 2]: *1)* thanks to the addition of explicit contexts, the equational theory is simpler to describe than in the $\lambda\mathcal{C}$ calculus or its variants; *2)* the possible choices regarding the order of evaluation appear clearer, as it displays a symmetry between call by value and call by name; *3)* typing rules are in correspondence with the ones of the sequent calculus.

2.2.1 A correspondence with sequent calculus

The $\bar{\lambda}\mu\tilde{\mu}_T$ calculus gives a constructive interpretation in call-by-name to the $\forall, \to, \bot$ fragment of Gentzen's classical sequent calculus **LK** [13] (Figure 2, bottom). To Gentzen's cut is associated the following typing rule:

$$\frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma' \mid \pi : A \vdash \Delta'}{\langle t \| \pi\rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')}$$

Γ becomes an environment of term variables $(x_1 : A_1, \ldots, x_n : A_n)$ and Δ becomes an environment of co-variables, $(\alpha_1 : B_1, \ldots, \alpha_m : B_m)$.

$$\boxed{\Gamma \vdash t : A \mid \Delta \quad , \quad \Gamma \mid \pi : A \vdash \Delta \quad , \quad \langle t \| \pi\rangle : (\Gamma \vdash \Delta)}$$

In the above frame, the first judgement is familiar: the type of a term is given by a conclusion of a sequent. The bar delineates which particular formula it is; this distinction corresponds in sequent calculus to the notion of *principal* or *active formula*. The second judgement describes the type of a stack by an hypothesis of the sequent, and reads "*π is a refutation of A in the context Γ, Δ*". The third judgement gives types to variables of a machine $\langle t \| \pi\rangle$, which has no type on its own. Notice that the cut is formulated in *multiplicative* rather than additive style (the contexts are split).

The slogan is that the reduction of commands corresponds to cut elimination (with the convention that we don't necessarily consider $\langle x \| e\rangle$ or $\langle t \| \alpha\rangle$ a cut, as suggested by Wadler [42]).

2.2.2 Adjoints

Elimination rules are obtained as the adjoints of left introduction rules, through the definition:

$$\boxed{\tau^*(t) \stackrel{\text{def}}{=} \mu\alpha.\langle t \| \tau(\alpha)\rangle}$$

The adjoint defines an operation that satisfies:

$$\langle \tau^*(t) \| \pi\rangle \rhd_{\mathrm{R}} \langle t \| \tau(\pi)\rangle$$

hence the name.

For instance, the derivation of the elimination rule of $\to$ in **LK** indeed gives $t\,u = (u\cdot -)^*(t)$, since it corresponds to the following derivation:

$$\dfrac{\dfrac{\Gamma \vdash t : A \to B \mid \Delta \qquad \dfrac{\Gamma' \vdash u : A \mid \Delta' \qquad \overline{\mid \alpha : B \vdash \alpha : B}\ (\mathrm{ax}\vdash)}{\Gamma' \mid u\cdot\alpha : A \to B \vdash \alpha : B, \Delta'}\ (\to\vdash)}{\langle t \| u\cdot\alpha\rangle : (\Gamma, \Gamma' \vdash \alpha : B, \Delta, \Delta')}\ (\mathrm{cut})}{\Gamma, \Gamma' \vdash (u\cdot -)^*(t) : B \mid \Delta, \Delta'}\ (\vdash \mu)$$

The typing rules of the $\lambda\mathcal{C}$ calculus can be derived (in multiplicative style[4]) from the rules of Figure 2. The abstract notation and the definition using adjoints solve the problem of commutative cuts in natural deduction (see M.-M. [32]).

The name adjoint is taken from Girard [15].

[4] The rules in additive style from Figure 1 can be recovered if Γ and Γ' are allowed to overlap. We prefer the multiplicative style, because conversely in additive-style **LK**, $\to$-elimination is not derivable but only admissible.

2.2.3 Non-linear contexts

Stacks (notation π) are linear contexts, in the sense of linear logic (see Laurent [26]). Curien and Herbelin introduced the $\tilde{\mu}$ binder that builds non-linear contexts (notation e). They represent contexts defined in terms of their interaction with a term.

The abstract machine for the $\lambda\mathcal{C}$ calculus only uses stacks: this is because stacks are sufficient if the goal is to describe the head reduction of a term. Adding $\tilde{\mu}$ is necessary to recover the correspondence between the syntax and the sequent calculus, as we will see below. It also makes the choice of evaluation order explicit, because it forces us to determine which of the binder μ or $\tilde{\mu}$ has the priority over the other in the reduction.

The context $\tilde{\mu}x.\langle t \,\|\, e\rangle$ is introduced by the following rule:

$$\frac{\langle t \,\|\, e\rangle : (\Gamma, x : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.\langle t \,\|\, e\rangle : A \vdash \Delta}\ (\tilde{\mu}\vdash)$$

Negative polarity Terms of the $\bar{\lambda}\mu\tilde{\mu}_T$ calculus are negative. Here this means that the reduction of μ is restricted to contexts that are stacks, and that the priority is given to the $\tilde{\mu}$ binder:

$$\langle t_\ominus \,\|\, \tilde{\mu}x^\ominus.c\rangle \rhd_{\mathrm{R}_n} c[t_\ominus/x^\ominus] \quad , \quad \langle \mu\alpha^\ominus.c \,\|\, \pi_\ominus\rangle \rhd_{\mathrm{R}_n} c[\pi_\ominus/\alpha^\ominus]$$

Thus, the evaluation of a negative term is delayed until it comes in head position, in the terminology of the λ calculus. In the calculus $\bar{\lambda}\mu\tilde{\mu}_T$, head position is determined by the fact that the context is a stack.

The distinction between stacks and negative contexts also appears in the rule $(\to \vdash_f)$: only stacks are allowed as a premiss. This is a focalisation constraint: it essentially means that the generic rule, which can be derived in terms of the restricted rule, hides a cut:

$$t{\cdot}e \stackrel{\text{def}}{=} \tilde{\mu}x.\langle x\, t \,\|\, e\rangle = \tilde{\mu}x.\left\langle \mu\alpha.\langle x \,\|\, t{\cdot}\alpha\rangle \,\middle\|\, e\right\rangle$$
$$\frac{\Gamma \vdash t : N \mid \Delta \qquad \Gamma' \mid e : M \vdash \Delta'}{\Gamma, \Gamma' \mid t{\cdot}e : N \to M \vdash \Delta, \Delta'}\ (\to\vdash)$$

In other words, in the $\bar{\lambda}\mu\tilde{\mu}_T$ calculus, priority is given to the evaluation of the context *inductively*:

$$\langle t \,\|\, u{\cdot}e\rangle \to^*_{\mathrm{R}_n} \left\langle \mu\alpha.\langle t \,\|\, u{\cdot}\alpha\rangle \,\middle\|\, e\right\rangle \text{ when } e \text{ is not a stack}$$

As noticed by Danos, Joinet and Schellinx [8], focalisation is notably meant to ensure the compatibility of reductions with expansions. Indeed, if we were to treat $t{\cdot}e$ as a stack, *i.e.* if we considered the following rules:

$$\langle \mu\alpha.c \,\|\, t{\cdot}e\rangle \rhd_{\mathrm{R}'_n} c[t{\cdot}e/\alpha] \quad , \quad \langle \lambda x.u \,\|\, t{\cdot}e\rangle \rhd_{\mathrm{R}'_n} \langle u[t/x] \,\|\, e\rangle$$

then unsolvable critical pairs would arise in the presence of expansions:

$$\begin{aligned} c[t{\cdot}\tilde{\mu}y.c'/\alpha] &\lhd_{\mathrm{R}'_n} \langle \mu\alpha.c \,\|\, t{\cdot}\tilde{\mu}y.c'\rangle \\ &\to_{\mathrm{E}_n} \left\langle \lambda x.\mu\beta.\langle \mu\alpha.c \,\|\, x{\cdot}\beta\rangle \,\middle\|\, t{\cdot}\tilde{\mu}y.c'\right\rangle \\ &\rhd_{\mathrm{R}'_n} \left\langle \lambda x.\mu\beta.\langle \mu\alpha.c \,\|\, t{\cdot}\beta\rangle \,\middle\|\, \tilde{\mu}y.c'\right\rangle \\ &\rhd_{\mathrm{R}_n} c'[\lambda x.\mu\beta.\langle \mu\alpha.c \,\|\, t{\cdot}\beta\rangle/y] \end{aligned}$$

Positive polarity In the $L_{pol,\hat{\mathsf{tp}}}$ calculus introduced in the next section, we add a positive polarity. The positive $\tilde{\mu}$ defines a stack (in our terminology), and when a cut is between positive term and context, the reduction of $\tilde{\mu}$ is restricted to terms that are values (V, to be introduced), which gives the priority to the μ binder:

$$\langle V_+ \,\|\, \tilde{\mu}x^+.c\rangle \rhd_{\mathrm{R}_p} c[V_+/x^+] \quad , \quad \langle \mu\alpha^+.c \,\|\, \pi_+\rangle \rhd_{\mathrm{R}_p} c[\pi_+/\alpha^+]$$

This corresponds to a call-by-value evaluation of positive terms, defined with their transition rules using the μ binder.

For positive terms, focalisation appears for terms. For instance, in the calculus introduced next section, pairs hide cuts in general because they compute down to values by inductively computing their components via "ς" rules (reusing Wadler's [42] terminology):

$$\langle (t_+, u) \,\|\, e_+\rangle \rhd_{\mathrm{R}_p} \left\langle t_+ \,\middle\|\, \tilde{\mu}x.\langle (x, u) \,\|\, e_+\rangle\right\rangle \qquad \text{when } t_+ \text{ is not a value}$$
$$\langle (V, t_+) \,\|\, e_+\rangle \rhd_{\mathrm{R}_p} \left\langle t_+ \,\middle\|\, \tilde{\mu}y.\langle (V, y) \,\|\, e_+\rangle\right\rangle \qquad \text{when } t_+ \text{ is not a value}$$

2.3 The calculus $L_{pol,\hat{\mathsf{tp}}}$

In this section we introduce the calculus $L_{pol,\hat{\mathsf{tp}}}$. We present the calculus $L_{pol,\hat{\mathsf{tp}}}$ in Figure 3. It enriches the $\bar{\lambda}\mu\tilde{\mu}_T$ calculus with:

- The positive conjunction $\otimes$, with the corresponding focusing rules;
- The positive negation $\sim$, and the corresponding focusing rule;
- Alternate rules for $\bot$ for which we introduce the operators $\hat{\mathsf{tp}}$ and $\mu\hat{\mathsf{tp}}$.

Let-bindings The binder $\tilde{\mu}$ allows us to derive below a *let*-binding:

$$\boxed{\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, u_\varepsilon \stackrel{\text{def}}{=} \mu\alpha^\varepsilon.\left\langle t \,\middle\|\, \tilde{\mu}x.\langle u_\varepsilon \,\|\, \alpha^\varepsilon\rangle\right\rangle}$$

The term $\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, u$ has the same polarity as u. Similarly, the binder $\tilde{\mu}$ is used to decompose pairs of values:

$$\boxed{\mathsf{let}\, (x, y)\, \mathsf{be}\, t_+\, \mathsf{in}\, u_\varepsilon \stackrel{\text{def}}{=} \mu\alpha^\varepsilon.\left\langle t_+ \,\middle\|\, \tilde{\mu}(x, y).\langle u_\varepsilon \,\|\, \alpha^\varepsilon\rangle\right\rangle}$$

The operator $\mu\hat{\mathsf{tp}}$ The goal with $\hat{\mathsf{tp}}$ is to reconcile the rule $(\bot\vdash)$ with extensionality equations. The difficulty comes from the fact that at type $\bot$, the natural equation for the $\bar{\lambda}\mu\tilde{\mu}_T$ calculus ($\forall\pi,\ \mathsf{stop} \simeq \pi$) implies that all the terms are identified. In $L_{pol,\hat{\mathsf{tp}}}$, stop is replaced by $\hat{\mathsf{tp}}$, whose essential difference is to invalidate the following equation:

$$\langle \mu\alpha^\ominus.c \,\|\, \hat{\mathsf{tp}}\rangle \not\rhd c[\hat{\mathsf{tp}}/\alpha^\ominus]$$

The reduction of $\mu\hat{\mathsf{tp}}.c$ is possible only when c is of the form $\langle t_\ominus \| \hat{\mathsf{tp}}\rangle$:

$$\mu\hat{\mathsf{tp}}.\langle t_\ominus \,\|\, \hat{\mathsf{tp}}\rangle \rhd_{\mathrm{R}_p} t_\ominus$$

The $\hat{\mathsf{tp}}$ variable is bound dynamically: the above rule holds even if $\hat{\mathsf{tp}}$ appears in $t_\ominus$. Thus $\hat{\mathsf{tp}}$ is not subject to the usual scoping and α-conversion rules.

In fact, the operator $\mu\hat{\mathsf{tp}}$ implements a list of stacks, defined inductively as follows:

$$\boxed{\begin{aligned} c\{\,\} &\stackrel{\text{def}}{=} c \\ c\{\pi^1_\ominus, \ldots, \pi^n_\ominus\} &\stackrel{\text{def}}{=} \langle \mu\hat{\mathsf{tp}}.c \,\|\, \pi^1_\ominus\rangle\{\pi^2_\ominus, \ldots, \pi^n_\ominus\} \end{aligned}}$$

And the context $\hat{\mathsf{tp}}$ corresponds to an operation that extracts the head of the list:

$$\langle t_\ominus \| \hat{\mathsf{tp}}\rangle\{\pi^1_\ominus, \pi^2_\ominus, \ldots, \pi^n_\ominus\} \to_{\mathrm{R}_p} \langle t_\ominus \,\|\, \pi^1_\ominus\rangle\{\pi^2_\ominus, \ldots, \pi^n_\ominus\}$$

Per the above definition, $\mu\hat{\mathsf{tp}}.c$ corresponds to the operation that adds a stack on top of the list. In typed settings we can expect these stacks to be of type $\bot$. Using $\hat{\mathsf{tp}}$ to interpret the elimination of falsity is inspired from Herbelin *et al.* [3, 18].

Accessing stacks In order to interpret the negation of a negative formula N, we introduce the positive type $\sim N$ of *inspectable stacks*. An inspectable stack is a value that denotes a captured stack and that exports accessors to its components. An inspectable stack is denoted with $V_+ = [\pi]$.

The calculus $L_{pol,\hat{\mathsf{tp}}}$ introduces the binder $\tilde{\mu}[\alpha].c$ which is responsible for accessing the stacks. We extend it to a λ abstraction as follows:

$$\lambda[\alpha].t \stackrel{\text{def}}{=} \lambda x.\mu\beta.\left\langle x \,\middle\|\, \tilde{\mu}[\alpha].\langle t \,\|\, \beta\rangle\right\rangle$$

For instance, we can access the head of a captured stack and return it paired with the tail of the stack with the following function:

$$D_\to = \lambda[x{\cdot}\gamma].(x, [\gamma]) \stackrel{\text{def}}{=} \lambda[\alpha^\ominus].\mu\beta^+.\left\langle \lambda x.\mu\gamma.\langle (x, [\gamma]) \,\|\, \beta^+\rangle \,\middle\|\, \alpha^\ominus\right\rangle$$

[…] : Main additions to Figure 2.

$$t\begin{cases} t_\ominus ::= x^\ominus \mid \lambda x.t \mid \mu\alpha^\ominus.c \mid \boxed{\mu\hat{\mathsf{tp}}.c} \\ t_+ ::= \boxed{x^+ \mid (t,t) \mid [e] \mid \mu\alpha^+.c} \end{cases} \quad V \subsetneq t \begin{cases} t_\ominus \\ V_+ ::= \boxed{x^+ \mid (V,V) \mid [\pi]} \end{cases} \quad e\begin{cases} e_\ominus ::= \alpha^\ominus \mid t\cdot e \mid \tilde{\mu}x^\ominus.c \mid \boxed{\hat{\mathsf{tp}}} \\ e_+ ::= \boxed{\alpha^+ \mid \tilde{\mu}x^+.c \mid \tilde{\mu}(x,y).c \mid \tilde{\mu}[\alpha].c} \end{cases} \quad \pi \subsetneq e \begin{cases} \pi_\ominus ::= \alpha^\ominus \mid V\cdot\pi \\ e_+ \end{cases}$$

$$c ::= \boxed{\langle t_+ \,\|\, e_+ \rangle} \mid \langle t_\ominus \,\|\, e_\ominus \rangle$$

(a) Terms, values, contexts, stacks, and commands

$$(R_{\tilde{\mu}}) \quad \langle V \,\|\, \tilde{\mu}x.c \rangle \quad \triangleright_{\mathrm{R}_p} c[V/x]$$
$$(R_{\mu}) \quad \langle \mu\alpha.c \,\|\, \pi \rangle \quad \triangleright_{\mathrm{R}_p} c[\pi/\alpha]$$
$$(R_{\to}) \quad \langle \lambda x.t \,\|\, V\cdot\pi \rangle^\dagger \quad \triangleright_{\mathrm{R}_p} \langle t[V/x] \,\|\, \pi \rangle$$
$$\boxed{(R_{\otimes})} \quad \langle (V,W) \,\|\, \tilde{\mu}(x,y).c \rangle^\dagger \quad \triangleright_{\mathrm{R}_p} c[V/x, W/y]$$
$$\boxed{(R_{\sim})} \quad \langle [\pi_\varepsilon] \,\|\, \tilde{\mu}[\alpha^\varepsilon].c \rangle \quad \triangleright_{\mathrm{R}_p} c[\pi_\varepsilon/\alpha^\varepsilon]$$
$$\boxed{(R_{\hat{\mathsf{tp}}})} \quad \mu\hat{\mathsf{tp}}.\langle t_\ominus \,\|\, \hat{\mathsf{tp}} \rangle^\ddagger \quad \triangleright_{\mathrm{R}_p} t_\ominus$$

†When polarities match pairwise. ‡Even if $\hat{\mathsf{tp}}$ occurs in $t_\ominus$.

(b) Reduction rules

$$\boxed{(\varsigma_{\to_1})} \quad \langle u_\ominus \,\|\, t_+\cdot e \rangle^\dagger \quad \triangleright_{\mathrm{R}_p} \left\langle t_+ \,\middle\|\, \tilde{\mu}x.\langle u_\ominus \,\|\, x\cdot e \rangle \right\rangle$$
$$\boxed{(\varsigma_{\to_2})} \quad \langle u_\ominus \,\|\, V\cdot e_\ominus \rangle^\ddagger \quad \triangleright_{\mathrm{R}_p} \left\langle \mu\alpha.\langle u_\ominus \,\|\, V\cdot\alpha \rangle \,\middle\|\, e_\ominus \right\rangle$$
$$\boxed{(\varsigma_{\otimes_1})} \quad \langle (t_+,u) \,\|\, e_+ \rangle^\dagger \quad \triangleright_{\mathrm{R}_p} \left\langle t_+ \,\middle\|\, \tilde{\mu}x.\langle (x,u) \,\|\, e_+ \rangle \right\rangle$$
$$\boxed{(\varsigma_{\otimes_2})} \quad \langle (V,t_+) \,\|\, e_+ \rangle^\dagger \quad \triangleright_{\mathrm{R}_p} \left\langle t_+ \,\middle\|\, \tilde{\mu}y.\langle (V,y) \,\|\, e_+ \rangle \right\rangle$$
$$\boxed{(\varsigma_{\sim})} \quad \langle [e_\ominus] \,\|\, e_+ \rangle^\ddagger \quad \triangleright_{\mathrm{R}_p} \left\langle \mu\beta^\ominus.\langle [\beta^\ominus] \,\|\, e_+ \rangle \,\middle\|\, e_\ominus \right\rangle$$

†When t_+ is not a value. ‡When $e_\ominus$ is not a stack.

(c) Expansion rules

$$(E_{\tilde{\mu}}) \quad e \quad \triangleright_{\mathrm{E}_p} \tilde{\mu}x.\langle x \,\|\, e \rangle$$
$$(E_{\mu}) \quad t \quad \triangleright_{\mathrm{E}_p} \mu\alpha.\langle t \,\|\, \alpha \rangle$$
$$(E_{\to}) \quad t_\ominus \quad \triangleright_{\mathrm{E}_p} \lambda x.\mu\alpha.\langle t_\ominus \,\|\, x\cdot\alpha \rangle$$
$$\boxed{(E_{\otimes})} \quad \pi_\ominus \quad \triangleright_{\mathrm{E}_p} \tilde{\mu}(x,y).\langle (x,y) \,\|\, \pi_\ominus \rangle$$
$$\boxed{(E_{\sim})} \quad e_+ \quad \triangleright_{\mathrm{E}_p} \tilde{\mu}[\alpha].\langle [\alpha] \,\|\, e_+ \rangle$$
$$\boxed{(E_{\hat{\mathsf{tp}}})} \quad c \quad \triangleright_{\mathrm{E}_p} \langle \mu\hat{\mathsf{tp}}.c \,\|\, \hat{\mathsf{tp}} \rangle$$

(d) Formulae

$$\neg A \stackrel{\text{def}}{=} \begin{cases} \sim N & \text{if } A = N \\ P \to \bot & \text{if } A = P \end{cases}$$

(e) Negation

$$\boxed{f{:}A ::= f_+{:}P \mid f_\ominus{:}N} \text{ for } f \in \{x, \alpha, t, e\}$$
$$\Gamma = \vec{x_i} : \vec{A_i} \quad , \quad \Delta = \vec{\alpha_j} : \vec{B_j}$$
$$\Gamma \vdash t : A \mid \Delta \quad , \quad \Gamma \mid e : A \vdash \Delta \quad , \quad c : (\Gamma \vdash \Delta)$$

(f) Judgements

$$\frac{}{x : A \vdash x : A \mid} \text{(⊢ ax)} \qquad \frac{}{\mid \alpha : A \vdash \alpha : A} \text{(ax ⊢)}$$
$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma, \tilde{\mu}x.c : A \vdash \Delta} \text{($\tilde{\mu}$ ⊢)} \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta} \text{(⊢ μ)}$$
$$\frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma' \mid e : A \vdash \Delta'}{\langle t \,\|\, e \rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')} \text{(cut)}$$

(g) Identity

$$\frac{c : (\Gamma \vdash \Delta)}{c : (\Gamma, x : A \vdash \Delta)} \text{(w ⊢)} \qquad \frac{c : (\Gamma, x : A, y : A \vdash \Delta)}{c[x/y] : (\Gamma, x : A \vdash \Delta)} \text{(c ⊢)}$$
$$\frac{c : (\Gamma \vdash \Delta)}{c : (\Gamma \vdash \alpha : A, \Delta)} \text{(⊢ w)} \qquad \frac{c : (\Gamma \vdash \alpha : A, \beta : A, \Delta)}{c[\alpha/\beta] : (\Gamma \vdash \alpha : A, \Delta)} \text{(⊢ c)}$$

and 8 similar rules with t and e replacing c.

(h) Structure

$$\frac{\Gamma, x : A \vdash t : B \mid \Delta}{\Gamma \vdash \lambda x.t : A \to B \mid \Delta} \text{(⊢ →)} \qquad \frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma' \mid e : B \vdash \Delta'}{\Gamma, \Gamma' \mid t\cdot e : A \to B \vdash \Delta, \Delta'} \text{(→ ⊢)}$$
$$\frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma' \vdash u : B \mid \Delta'}{\Gamma, \Gamma' \vdash (t,u) : A \otimes B \mid \Delta, \Delta'} \text{(⊢ ⊗)} \qquad \frac{c : (\Gamma, x : A, y : B \vdash \Delta)}{\Gamma \mid \tilde{\mu}(x,y).c : A \otimes B \vdash \Delta} \text{(⊗ ⊢)}$$
$$\frac{\Gamma \mid e : A \vdash \Delta}{\Gamma \vdash [e] : \sim A \mid \Delta} \text{(⊢ ∼)} \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \mid \mu[\alpha].c : \sim A \vdash \Delta} \text{(∼ ⊢)}$$
$$\frac{\Gamma \vdash t_\ominus : N \mid \Delta}{\Gamma \vdash t_\ominus : \forall x\, N \mid \Delta} \text{(⊢ $\forall^1$)*} \qquad \frac{\Gamma \mid e_\ominus : N[t/x] \vdash \Delta}{\Gamma \mid e_\ominus : \forall x\, N \vdash \Delta} \text{($\forall^1$ ⊢)}$$
$$\frac{\Gamma \vdash t_+ : P[t/x] \mid \Delta}{\Gamma \vdash t_+ : \exists x\, P \mid \Delta} \text{(⊢ ∃)} \qquad \frac{\Gamma \mid e_+ : P \vdash \Delta}{\Gamma \mid e_+ : \exists x\, P \vdash \Delta} \text{(∃ ⊢)*}$$
$$\frac{c : (\Gamma \vdash \Delta)}{\Gamma \vdash \mu\hat{\mathsf{tp}}.c : \bot \mid \Delta} \text{(⊢ ⊥)} \qquad \frac{}{\Gamma \mid \hat{\mathsf{tp}} : \bot \vdash \Delta} \text{(⊥ ⊢)}$$

(i) Logic (*: $x \notin \mathbf{fv}(\Gamma, \Delta)$)

Figure 3. The $L_{pol,\hat{\mathsf{tp}}}$ calculus (top) and its typing rules in sequent calculus (bottom)

Properties

Proposition 2 (Confluence). *The reduction $\to_{\mathrm{R}_p}$ is confluent.*

Proof. Because the reduction $\triangleright_{\mathrm{R}_p}$ is left-linear and has no critical pairs (Nipkow [37]). □

In M.-M. [32, Section IV.4.2], we give a continuation-passing-style translation of $L_{pol,\hat{\mathsf{tp}}}$ into a λ calculus with surjective pairs. We show that the translation preserves equivalence and simulates reduction. Thus:

Proposition 3 (Coherence). *If x and y are two distinct polarised variables, then $x \not\simeq_{\mathrm{RE}_p} y$.*

In other words the calculus $L_{pol,\hat{\mathsf{tp}}}$ does not identify all proofs.

Proposition 4 (Strong normalisation). *Typable commands of $L_{pol,\hat{\mathsf{tp}}}$ are strongly $\to_{\mathrm{R}}$-normalising.*

Of course the normalisation result is of limited interest given that our type system is inexpressive. Also, the orthogonality technique (adapted to the polarised case [31, 6]) probably applies and gives more elegant proofs.

Call-by-name delimited continuations / Saurin's $\Lambda\mu$ The calculus $L_{pol,\hat{\mathsf{tp}}}$ restricted to fully negative terms corresponds to Herbelin and Ghilezan's $\lambda\mu\hat{\mathsf{tp}}_n$ calculus [18]. Herbelin and Ghilezan show that the calculus $\lambda\mu\hat{\mathsf{tp}}_n$ is in correspondence with the $\Lambda\mu$ calculus of De Groote and Saurin [10, 39]. (The $\Lambda\mu$ calculus is interesting because Saurin showed that it satisfies a Böhm theorem.)

Shift$_0$/Reset$_0$ We can show that the calculus implements (a variant of) the operators Shift$_0$/Reset$_0$ [9, 40, 28]. (See M.-M. [32, Section IV.4.3].)

3. The involutive negation in natural deduction

In this section we introduce the $\lambda\ell$ calculus. The role of the $\lambda\ell$ calculus is to show how the λC calculus can be extended so as to correspond to a natural deduction with an involutive negation. This is why two constraints guided the design of the calculus:

1. Negation must be there as a connective;
2. There must be a clear distinction between quasi-proof terms and terms that appear during the evaluation in a machine.

Quasi-proof terms of the $\lambda\ell$ calculus are defined in Figure 4 (top), together with a polarised predicate calculus in natural deduction. This calculus extends the λC calculus with a control operator ℓ that refines the C operator. Both terms and stacks have *polarities* determined by the function ϖ; let us write t_+, $t_\ominus$, π_+ or $\pi_\ominus$ to refer to a term or a stack of a given polarity.

At the bottom of Figure 4 we define the evaluation of terms with a machine that extends Krivine's. Initial stacks are infinitely many constants $\alpha, \beta \ldots$ that have a fixed polarity.

Constraints 1. and 2. above prevail at times over simplicity. For instance, to ensure that we can statically determine a polarity for each term, application is annotated with the expected polarity of the result:

$$(t\,u)^+ \text{ or } (t\,u)^\ominus$$

Note that an annotation t^ε is therefore a part of the grammar and defines the term to be of polarity ε. By contrast, the notation t_ε only asserts that t has this polarity and is not part of the grammar. We omit the annotation when it can be deduced from the context.

3.1 Negating a positive: polarised arrows

The negation of a positive formula P is given with $P \to \bot$. Let us first explain the polarised arrow. The arrow is in call by value, by which we mean that in $t\,u$, the argument u is evaluated first, when it is not already a value:

$$\boxed{\langle t_\ominus\, u_+ \,\|\, \pi\rangle \succ_p \left\langle u_+ \,\middle\|\, \tilde{\mu}x^+.\langle t_\ominus \,\|\, x^+\cdot\pi\rangle \right\rangle} \text{ if } u_+ \text{ is not a value}$$
$$\langle t_\ominus\, V \,\|\, \pi\rangle \succ_p \langle t_\ominus \,\|\, V\cdot\pi\rangle$$

In particular, contexts of the type $A \to \bot$, when captured by control operators, are guaranteed to be stacks of the form $V\cdot\pi$ where V is a value.

Remark 5. In this context, polarisation means that given three terms:

$$\Gamma \vdash t_+ : P \quad , \quad \Gamma, x^+ : P \vdash u_\ominus : N \quad , \quad \Gamma, y^\ominus : N \vdash v : A$$

there are two ways of composing them:

$$(\lambda y^\ominus.v)(\lambda x^+.u_\ominus\, t_+)^\ominus \qquad \text{and} \qquad (\lambda x^+.(\lambda y^\ominus.v)u_\ominus)\,t_+$$

which correspond to the following distinct behaviours:

$$\langle (\lambda y^\ominus.v)(\lambda x^+.u_\ominus\, t_+)^\ominus \,\|\, \pi\rangle \succ_p^* \langle v[(\lambda x^+.u_\ominus\, t_+)^\ominus/y^\ominus] \,\|\, \pi\rangle$$
$$\langle (\lambda x^+.(\lambda y^\ominus.v)u_\ominus)\,t_+ \,\|\, \pi\rangle \succ_p^* \left\langle t_+ \,\middle\|\, \tilde{\mu}x^+.\langle (\lambda y^\ominus.v)u_\ominus \,\|\, \pi\rangle \right\rangle$$

Thus, for lack of associativity of composition, the $\lambda\ell$ calculus escapes from the following argument of category theory, which historically opposed the existence of non-boolean categorical models of classical logic: as it is well-known, a cartesian-closed category never has a dualising object $\bot$, that is to say satisfying the natural isomorphism $\bot^{\bot^A} \cong A$, unless it is a boolean algebra. This follows more generally from the difficulty of interpreting negation in intuitionistic logic already: in a bi-cartesian-closed category, there is at most one morphism from any object to the initial object [25, p.67].

But, lack of associativity is only characteristic of polarisation, as we showed in M.-M. [33].

3.2 Falsity: delimited control

We assume that execution happens in a machine of the following form:

$$\boxed{\langle t \,\|\, \pi\rangle\{\pi^1_\ominus, \ldots, \pi^n_\ominus\}}$$

where $\sigma = \pi^1_\ominus, \ldots, \pi^n_\ominus$ is a list of negative stacks. (As a consequence, the notation $c \succ_p c'$ is an abbreviation which denotes $\forall\sigma,\, c\{\sigma\} \succ_p c'\{\sigma\}$.)

The list σ interacts with control operators: we shall see that the send operator lets it grow whereas the operator ℓ lets it shrink. Compared to the C operator, the ℓ operator prefers to grab the nearest stack in the list σ to install it as the new context, instead of installing the stack stop from the λC calculus. We can think of σ as a list of exception handlers, with terms of type $\bot$ being handled by raising an exception.

3.3 Inspectable stacks

The constants $D_\to$, $D_\forall$[5] and $D_\bot$ let us access the components of an inspectable stack:

$$\boxed{\begin{array}{l} D_\to : {\sim}(A \to B) \to A \otimes {\sim}B \\ D_\forall : {\sim}(\forall x\, A) \to \exists x\, {\sim}A \\ D_\bot : {\sim}\bot \to A \to A \end{array}}$$
$$\langle D_\to \,\|\, [V\cdot\pi_1]\cdot\pi_2\rangle \succ_p \langle (V, [\pi_1]) \,\|\, \pi_2\rangle$$
$$\langle D_\forall \,\|\, [\pi_1]\cdot\pi_2\rangle \succ_p \langle [\pi_1] \,\|\, \pi_2\rangle$$
$$\langle D_\bot \,\|\, [\pi_\ominus]\cdot t\cdot\pi'\rangle \succ_p \langle t \,\|\, \pi'\rangle\{\pi_\ominus\}$$

Also, a captured stack can be positive, which gives a value of the positive type ${\sim}P$. However it has no accessor.

Example 6. We can combine $D_\forall$ and $D_\to$ to get a proof of:

$$\neg\forall x(A \to N) \to \exists x\,(A \otimes \neg N)$$

Take $B = N$ in the following:

$$D_{\forall\to} \stackrel{\text{def}}{=} \lambda x^+.\mathsf{let}\, y^+ \,\mathsf{be}\, D_\forall\, x^+ \,\mathsf{in}\, D_\to\, y^+$$
$$D_{\forall\to} : {\sim}\forall x(A \to B) \to \exists x\,(A \otimes {\sim}B)$$
$$\langle D_{\forall\to} \,\|\, [V\cdot\pi]\cdot\pi_+\rangle\{\sigma\} \succ_p^* \langle (V, [\pi]) \,\|\, \pi_+\rangle\{\sigma\}$$

The immediateness of the operation is in sharp contrast with its homologue of the λC calculus given in the introduction. The essential difference is that $D_{\forall\to}$ accesses the components of a stack which is already captured, whereas the term of the λC calculus contains two control operators.

Example 7. We can combine $D_\to$ and $D_\bot$ to obtain a proof in particular of $\neg\neg P \to P$ (take $A = P$):

$$D_\neg \stackrel{\text{def}}{=} \lambda x^+.\mathsf{let}\,(y^\varepsilon, z^+)\,\mathsf{be}\, D_\to\, x^+ \,\mathsf{in}\,(D_\bot\, z^+\, y^\varepsilon)^\varepsilon$$
$$D_\neg : {\sim}(A \to \bot) \to A$$
$$\langle D_\neg \,\|\, [V_\varepsilon\cdot\pi_\ominus]\cdot\pi'_\varepsilon\rangle\{\sigma\} \succ_p^* \langle V_\varepsilon \,\|\, \pi'_\varepsilon\rangle\{\pi_\ominus, \sigma\}$$

The term $D_\neg$ keeps $\pi_\ominus$ at the head of the list σ. A variant erases $\pi_\ominus$:

$$D'_\neg \stackrel{\text{def}}{=} \lambda x^+.\mathsf{let}\,(y^\varepsilon, z^+)\,\mathsf{be}\, D_\to\, x^+ \,\mathsf{in}\, y^\varepsilon$$
$$D'_\neg : {\sim}(A \to \bot) \to A$$
$$\langle D'_\neg \,\|\, [V_\varepsilon\cdot\pi_\ominus]\cdot\pi'_\varepsilon\rangle\{\sigma\} \succ_p^* \langle V_\varepsilon \,\|\, \pi'_\varepsilon\rangle\{\sigma\}$$

But $D_\neg$, and not $D'_\neg$, will take part in the isomorphism $\neg\neg P \cong P$.

[5] The behaviour of $D_\forall$ seems trivial because it corresponds to a sub-typing relation ${\sim}(\forall x\, A) <: \exists x\, {\sim}A$. Thus the examples that follow would be even simpler if we took the trouble of introducing a notion of sub-typing.

… : Main additions to Figure 1.

$\varepsilon ::= + \mid \ominus$

$t, u ::= x^{\varepsilon} \mid \lambda x.t \mid (t_{\ominus}\, u)^{\varepsilon} \mid$ let x^{+} be t_{+} in u $\mid$

$(t,u) \mid$ let (x,y) be t_{+} in $u \mid$ send $\mid \ell \mid D_{\to} \mid D_{\bot} \mid D_{\forall}$

t_{ε} : term t such that $\varpi(t) = \varepsilon \in \{+, \ominus\}$

(a) Quasi-proof terms (C is removed)

t	$\varpi(t)$
let … in u	$\varpi(u)$
x^{ε} , $(t\,u)^{\varepsilon}$	ε
$\lambda x.t$, send , ℓ , $D_{\to}$, $D_{\bot}$, $D_{\forall}$	$\ominus$
(t,u)	$+$

(b) Polarity $\varpi(t) \in \{+, \ominus\}$ of quasi-proof terms

Formulae:
as in Figures 3d and 3e.

$x : A ::= x^{+} : P \mid x^{\ominus} : N$

$\Gamma = \vec{x_i} : \vec{A_i}$

$\Gamma \vdash t_{+} : P \quad , \quad \Gamma \vdash t_{\ominus} : N$

(c) Judgements

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\mathrm{ax}) \qquad \frac{\Gamma \vdash t_{+} : P \quad \Gamma, x^{+} : P \vdash u : B}{\Gamma \vdash \text{let } x^{+} \text{be } t_{+} \text{ in } u : B}\ (\mathrm{let})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_i) \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t_{\ominus}\, u)^{\varepsilon} : B}\ (\to_e, \varepsilon = \varpi(B))$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash (t,u) : A \otimes B}\ (\otimes_i) \qquad \frac{\Gamma \vdash t_{+} : A \otimes B \quad \Gamma, x : A, y : B \vdash u : C}{\Gamma \vdash \text{let } (x,y) \text{ be } t_{+} \text{ in } u : C}\ (\otimes_e)$$

$$\frac{\Gamma \vdash t_{\ominus} : N}{\Gamma \vdash t_{\ominus} : \forall \mathrm{x}\, N}\ (\forall_i)^{*} \qquad \frac{\Gamma \vdash t_{\ominus} : \forall \mathrm{x}\, N}{\Gamma \vdash t_{\ominus} : N[\mathrm{t/x}]}\ (\forall_e)$$

$$\frac{\Gamma \vdash t_{+} : P[\mathrm{t/x}]}{\Gamma \vdash t_{+} : \exists \mathrm{x}\, P}\ (\exists_i) \qquad \frac{\Gamma \vdash t_{+} : \exists \mathrm{x}\, P \quad \Gamma, x^{+} : P \vdash u : A}{\Gamma \vdash \text{let } x^{+} \text{be } t_{+} \text{ in } u : A}\ (\exists_e)^{*}$$

$\ell : (A \to \bot) \to \sim A \quad , \quad \text{send} : \sim A \to (A \to \bot) \quad ,$

$D_{\bot} : \sim\bot \to A \to A \quad , \quad D_{\to} : \sim(A \to B) \to A \otimes \sim B \quad , \quad D_{\forall} : \sim\forall x\, N \to \exists x \sim N$

*: $\mathrm{x} \notin \mathbf{fv}(\Gamma, A)$.

(d) Rules

$D_{\forall\to} \stackrel{\text{def}}{=} \lambda x^{+}.\text{let } y^{+} \text{be } D_{\forall}\, x^{+} \text{in } D_{\to}\, y^{+}$
$\quad : \sim\forall \mathrm{x}(A \to B) \to \exists \mathrm{x}\,(A \otimes \sim B)$

$D_{\neg} \stackrel{\text{def}}{=} \lambda x^{+}.\text{let } (y^{\varepsilon}, z^{+}) \text{be } D_{\to}\, x^{+} \text{in } (D_{\bot}\, z^{+}\, y^{\varepsilon})^{\varepsilon}$
$\quad : \sim(A \to \bot) \to A$

$\mathcal{A} \stackrel{\text{def}}{=} \lambda x^{\ominus}.\text{let } (y^{\varepsilon}, z^{+}) \text{be } D_{\to}\, \ell\_.x^{\ominus} \text{in } y^{\varepsilon}$
$\quad : \bot \to A$

$E \stackrel{\text{def}}{=} \lambda x.\ell\, y^{\ominus}.(y^{\ominus}\, x)^{\ominus}$
$\quad : A \to \sim(A \to \bot)$

$\mathcal{T} \stackrel{\text{def}}{=} \lambda x^{\ominus}.(D_{\neg}\, \ell\, y^{\ominus}.(x^{\ominus}\, (\ell\, y^{\ominus})^{+}))$
$\quad : (\sim A \to \bot) \to A$

$C \stackrel{\text{def}}{=} \lambda x^{\ominus}.\text{let } (y, z^{+}) \text{be } D_{\to}\, (\ell\, x^{\ominus})^{+} \text{in } y$
$\quad : ((A \to \bot) \to \bot) \to A$

(e) Derived terms (notation $\ell x.t_{\ominus} \stackrel{\text{def}}{=} (\ell\, \lambda x.t_{\ominus})^{+}$)

$t ::= \ldots \mid [\pi] \mid \mathsf{j}_{\pi_{+}}$

$t \supseteq V, W ::= t_{\ominus} \mid x^{+} \mid (V, W) \mid [\pi]$

$\pi ::= \alpha^{\varepsilon} \mid V \cdot \pi \mid \tilde{\mu} x^{+}.c \mid \tilde{\mu}(x,y).c$

(f) Terms, values and stacks

$c ::= \langle t_{\ominus} \| \pi_{\ominus} \rangle \mid \langle t_{+} \| \pi_{+} \rangle$

$m ::= c\{\sigma\}$

$\sigma ::= \cdot \mid \pi_{\ominus}, \sigma$

(g) Commands and machines

t	$\varpi(t)$
$\mathsf{j}_{\pi_{+}}$	$\ominus$
$[\pi]$	$+$

π	$\varpi(\pi)$
$\alpha^{\ominus}$, $V \cdot \pi$	$\ominus$
α^{+} , $\tilde{\mu} x^{+}.c$, $\tilde{\mu}(x,y).c$	$+$

(h) Polarity $\varpi(t), \varpi(\pi) \in \{+, \ominus\}$ of terms and stacks

$$\langle (t_{\ominus}\, u)^{\varepsilon} \| \pi_{\varepsilon} \rangle\{\sigma\} \succ_p \begin{cases} \langle t_{\ominus} \| V \cdot \pi_{\varepsilon} \rangle\{\sigma\} & \text{if } u = V \\ \langle u \| \tilde{\mu} x^{+}.\langle t_{\ominus} \| x^{+} \cdot \pi_{\varepsilon} \rangle \rangle\{\sigma\} & \text{otherwise} \end{cases}$$

$$\begin{array}{rcl}
\langle \lambda x.t \| V \cdot \pi \rangle\{\sigma\} & \succ_p & \langle t[V/x] \| \pi \rangle\{\sigma\} \\
\langle \text{let } q \text{ be } t_{+} \text{ in } u \| \pi \rangle\{\sigma\} & \succ_p & \langle t_{+} \| \tilde{\mu} q.\langle u \| \pi \rangle \rangle\{\sigma\} \\
\langle V_{+} \| \tilde{\mu} x^{+}.c \rangle\{\sigma\} & \succ_p & c[V_{+}/x^{+}]\{\sigma\} \\
\langle (t_{+}, u) \| \pi \rangle\{\sigma\} & \succ_p & \langle t_{+} \| \tilde{\mu} x^{+}.\langle (x^{+}, u) \| \pi \rangle \rangle\{\sigma\} \text{ if } t_{+} \neq V \\
\langle (V, t_{+}) \| \pi \rangle\{\sigma\} & \succ_p & \langle t_{+} \| \tilde{\mu} x^{+}.\langle (V, x^{+}) \| \pi \rangle \rangle\{\sigma\} \text{ if } t_{+} \neq V \\
\langle (V, W) \| \tilde{\mu}(x,y).c \rangle\{\sigma\} & \succ_p & c[V/x, W/y]\{\sigma\}
\end{array}$$

$$\begin{array}{rcl}
\langle \ell \| t_{\ominus} \cdot \pi_{+} \rangle\{\pi_{\ominus}, \sigma\} & \succ_p & \langle t_{\ominus} \| \mathsf{j}_{\pi_{+}} \cdot \pi_{\ominus} \rangle\{\sigma\} \\
\langle \mathsf{j}_{\pi_{+}} \| \pi \rangle\{\sigma\} & \succ_p & \langle [\pi] \| \pi_{+} \rangle\{\sigma\} \\
\langle \text{send} \| [\pi_{\varepsilon}] \cdot t_{\varepsilon} \cdot \pi'_{\ominus} \rangle\{\sigma\} & \succ_p & \langle t_{\varepsilon} \| \pi_{\varepsilon} \rangle\{\pi'_{\ominus}, \sigma\} \\
\langle D_{\to} \| [V \cdot \pi] \cdot \pi_{+} \rangle\{\sigma\} & \succ_p & \langle (V, [\pi]) \| \pi_{+} \rangle\{\sigma\} \\
\langle D_{\bot} \| [\pi_{\ominus}] \cdot t_{\varepsilon} \cdot \pi'_{\varepsilon} \rangle\{\sigma\} & \succ_p & \langle t_{\varepsilon} \| \pi'_{\varepsilon} \rangle\{\pi_{\ominus}, \sigma\} \\
\langle D_{\forall} \| V \cdot \pi' \rangle\{\sigma\} & \succ_p & \langle V \| \pi' \rangle\{\sigma\}
\end{array}$$

(i) Reductions

Figure 4. The $\lambda\ell$ calculus and its typing rules in natural deduction (top); its abstract machine (bottom)

3.4 Capturing and installing stacks

Given a captured stack $[\pi]$, the stack is re-installed as the context of another term t by the constant send:

$$\langle \text{send} \| [\pi] \cdot t \cdot \pi'_{\ominus} \rangle\{\sigma\} \succ_p \langle t \| \pi \rangle\{\pi'_{\ominus}, \sigma\}$$

In other words, the constant send converts a captured stack into a continuation :

send : $\sim A \to A \to \bot$

The stack $\pi'_{\ominus}$, supposedly of type $\bot$ according to the type of send, is added on top of σ.

The operator responsible for the apparition of inspectable stacks is ℓ :

$\ell : (A \to \bot) \to \sim A$

The notation $\ell x.t_{\ominus}$ stands for $(\ell\, \lambda x.t_{\ominus})^{+}$. It evaluates $t_{\ominus}$ until x comes in head position, that is to say in front of a stack π. When this happens, the operator ℓ captures π and supplies the inspectable stack $[\pi]$ to the context where ℓ was applied. Last, the operator ℓ falls back to the head of the list σ in case $t_{\ominus}$ returns without using x.[6]

This operation is formally described by introducing the j operator. The operator ℓ saves in j the context π_{+} in which ℓ is applied, and

[6] This explains why Murthy's computational interpretation of **LC** must evaluate "*under the λ-abstraction*" [36]. Murthy was in fact describing the behaviour of a ℓ-like operator, rather than a λ-abstracttion.

$$
\begin{array}{ll}
(t_\ominus\, u)^\epsilon & \stackrel{\text{def}}{=} \mu\alpha^\epsilon.\langle t_\ominus \,\|\, u\cdot\alpha^\epsilon\rangle \\
\mathsf{let}\, x^+ \,\mathsf{be}\, t_+ \,\mathsf{in}\, u_\epsilon & \stackrel{\text{def}}{=} \mu\alpha^\epsilon.\left\langle t_+ \,\middle\|\, \tilde{\mu}x^+.\langle u_\epsilon \,\|\, \alpha^\epsilon\rangle\right\rangle \\
\mathsf{let}\,(x,y)\,\mathsf{be}\, t_+ \,\mathsf{in}\, u_\epsilon & \stackrel{\text{def}}{=} \mu\alpha^\epsilon.\left\langle t_+ \,\middle\|\, \tilde{\mu}(x,y).\langle u_\epsilon \,\|\, \alpha^\epsilon\rangle\right\rangle \\
\mathsf{send} & \stackrel{\text{def}}{=} \lambda[\alpha^\epsilon].\lambda x^\epsilon.\mu\hat{\mathsf{tp}}.\langle x^\epsilon \,\|\, \alpha^\epsilon\rangle \\
\ell & \stackrel{\text{def}}{=} \lambda x^\ominus.\mu\alpha^+.\langle x^\ominus \,\|\, \mathsf{j}_{\alpha^+}\cdot\hat{\mathsf{tp}}\rangle \\
D_\to & \stackrel{\text{def}}{=} \lambda[\alpha^\ominus].\mu\beta^+.\left\langle \lambda x.\mu\gamma.\langle (x,[\gamma]) \,\|\, \beta^+\rangle \,\middle\|\, \alpha^\ominus\right\rangle \\
D_\bot & \stackrel{\text{def}}{=} \lambda[\alpha^\ominus].\lambda x.\mu\beta.\left\langle \mu\hat{\mathsf{tp}}.\langle x \,\|\, \beta\rangle \,\middle\|\, \alpha^\ominus\right\rangle \\
D_\forall & \stackrel{\text{def}}{=} \lambda[\alpha].[\alpha] \\
\multicolumn{2}{l}{\text{where } \lambda[\alpha].t \stackrel{\text{def}}{=} \lambda x.\mu\beta.\left\langle x \,\middle\|\, \tilde{\mu}[\alpha].\langle t \,\|\, \beta\rangle\right\rangle}
\end{array}
$$

(a) Quasi-proof terms

$$
\begin{array}{ll}
\mathsf{j}_{e_+} & \stackrel{\text{def}}{=} \mu\alpha.\langle [\alpha] \,\|\, e_+\rangle \\
\alpha^\epsilon & \stackrel{\text{def}}{=} \alpha^\epsilon \text{ (free)} \\
c\{\,\} & \stackrel{\text{def}}{=} c \\
c\{\pi^1_\ominus, \ldots, \pi^n_\ominus\} & \stackrel{\text{def}}{=} \langle \mu\hat{\mathsf{tp}}.c \,\|\, \pi^1_\ominus\rangle\{\pi^2_\ominus, \ldots, \pi^n_\ominus\}
\end{array}
$$

(b) Terms, stack constants, machines

Figure 5. Definition of the constructs of $\lambda\ell$ in $L_{pol,\hat{\mathsf{tp}}}$

installs the head $\pi_\ominus$ as the new context:

$$\langle \ell \,\|\, t_\ominus\cdot\pi_+\rangle\{\pi_\ominus, \sigma\} \succ_p \langle t_\ominus \,\|\, \mathsf{j}_{\pi_+}\cdot\pi_\ominus\rangle\{\sigma\}$$

Once the operator j_{π_+} comes in head position, it captures the stack and restores the context π_+:

$$\langle \mathsf{j}_{\pi_+} \,\|\, \pi\rangle\{\sigma\} \succ_p \langle [\pi] \,\|\, \pi_+\rangle\{\sigma\}$$

Example 8. Using ℓ, we derive an operator $\mathcal{A}$ (*abort*) which interprets the rule *Ex Falso Quodlibet:*

$$\mathcal{A} \stackrel{\text{def}}{=} \lambda x^\ominus.\mathsf{let}\,(y^\epsilon, z^+)\,\mathsf{be}\, D_\to\, \ell\_.x^\ominus\,\mathsf{in}\, y^\epsilon$$
$$\mathcal{A} : \bot \to A$$
$$\langle \mathcal{A} \,\|\, t_\ominus\cdot\pi\rangle\{\pi'_\ominus, \sigma\} \succ^*_p \langle t_\ominus \,\|\, \pi'_\ominus\rangle\{\sigma\}$$

Example 9. By combining $\lambda xy.yx : A \to ((A \to \bot) \to \bot)$ with ℓ we obtain a proof of $P \to \neg\neg P$ (take $A = P$):

$$E : A \to \sim(A \to \bot)$$
$$E \stackrel{\text{def}}{=} \lambda x.\ell y^\ominus.(y^\ominus x)^\ominus$$
$$\langle E \,\|\, V\cdot\pi_+\rangle\{\pi_\ominus, \sigma\} \succ^*_p \langle [V\cdot\pi_\ominus] \,\|\, \pi_+\rangle\{\sigma\}$$

Example 10. We obtain the elimination of double negation $\neg\neg N \to N$ as follows (take $A = N$):

$$\mathcal{T} : (\sim A \to \bot) \to A$$
$$\mathcal{T} \stackrel{\text{def}}{=} \lambda x^\ominus.(D_\neg\, \ell\, y^\ominus.(x^\ominus\,(\ell\, y^\ominus)^+))$$
$$\langle \mathcal{T} \,\|\, t_\ominus\cdot\pi\rangle\{\pi'_\ominus, \pi''_\ominus, \sigma\} \succ^*_p \langle t_\ominus \,\|\, [\pi]\cdot\pi'_\ominus\rangle\{\pi''_\ominus, \sigma\}$$

The $\mathcal{T}$ operator is a variant of the $\mathcal{C}$ operator that supplies the context under the form of an inspectable stack, rather than a continuation.

3.5 Translating $\lambda\ell$ in $L_{pol,\hat{\mathsf{tp}}}$

Like for the λC calculus, we can define the operations of the calculus in the calculus $L_{pol,\hat{\mathsf{tp}}}$ by *solving their reduction rules*. In addition to the definitions in Figure 5, stack constants of the $\lambda\ell$ calculus are interpreted by free co-variables of $L_{pol,\hat{\mathsf{tp}}}$.[7] These definitions induce a translation of $\lambda\ell$ into $L_{pol,\hat{\mathsf{tp}}}$, which is defined by induction. We identify elements of $\lambda\ell$ to their image in $L_{pol,\hat{\mathsf{tp}}}$.

[7] Interpreting such stack constants as open variables goes back to Hofmann and Streicher [19].

Remark 11. Like the quasi-proofs of $\lambda\mathcal{C}$ (with respect to $\bar{\lambda}\mu\tilde{\mu}_T$), quasi-proofs of $\lambda\ell$ form a compositional sub-language of terms of $L_{pol,\hat{\mathsf{tp}}}$ that have no free co-variable.

We easily show:

Proposition 12 (Simulation)**.** *If $m \succ_p m'$ for two machines m and m' of the calculus $\lambda\ell$, then $m \to^+_{\mathrm{R}_p} m'$ in the calculus $L_{pol,\hat{\mathsf{tp}}}$.*

Proposition 13. *If one has $\Gamma \vdash t : A$ in natural deduction, then one has $\Gamma \vdash t : A \mid$ in sequent calculus.*

Consequently:

Proposition 14 (Normalisation)**.** *If t is a typable term of $\lambda\ell$, then the machine $\langle t \,\|\, \alpha_0\rangle\{\alpha^\ominus_1, \ldots, \alpha^\ominus_n\}$ normalises (because it is a typable command of $L_{pol,\hat{\mathsf{tp}}}$).*

Definition 15. In $\lambda\ell$, we define the compatible equivalence relation $\approx_p$ between quasi-proof terms with:

$$t \approx_p u \overset{\text{def}}{\iff} t \simeq_{\mathrm{RE}_p} u$$

For instance, one can show:

$$\ell \approx_p \lambda x^\ominus.(\mathcal{T}\, \lambda y^+.(x^\ominus\,(\mathsf{send}\, y^+\,(\mathcal{T}\, \lambda z^+.z^+))))$$

In other words we can obtain ℓ from $\mathcal{T}$. Interestingly, however, we could not derive the type $\ell : (A \to \bot) \to \sim A$ from the one of $\mathcal{T}$. Thus we have to take ℓ (positive negation introduction) as a primitive rather than $\mathcal{T}$ (double negation elimination).

Finally, following from Proposition 3 we have:

Proposition 16 (Coherence)**.** *If x and y are two distinct polarised variables, then $x \not\approx_p y$.*

4. Negation is involutive in $\lambda\ell$ and $L_{pol,\hat{\mathsf{tp}}}$

In terms of the equivalences $\simeq_{\mathrm{RE}_p}$ on $L_{pol,\hat{\mathsf{tp}}}$ and $\approx_p$ on $\lambda\ell$, we can prove that there are isomorphisms of type $\neg\neg A \cong A$ for both A positive and negative. Due to lack of associativity of composition when A and B do not have the same polarity, we ask that $A \cong B$ is defined only when they have the same polarity.

Definition 17. Two types (or formulae) A and B are isomorphic if they have the same polarity and there exist two terms:

$$x : A \vdash \phi(x) : B \qquad \text{and} \qquad y : B \vdash \psi(y) : A$$

with:

$$\mathsf{let}\, y\,\mathsf{be}\, \phi(x)\,\mathsf{in}\, \psi(y) \simeq x \qquad \text{and} \qquad \mathsf{let}\, x\,\mathsf{be}\, \psi(y)\,\mathsf{in}\, \phi(x) \simeq y$$

In this case we write $A \cong_{\phi,\psi} B$.

In the calculus $L_{pol,\hat{\mathsf{tp}}}$, we take $\simeq\, = \simeq_{\mathrm{RE}_p}$ above. In the $\lambda\ell$ calculus, we take $\simeq\, = \approx_p$. In the latter case, notice that terms must be quasi-proofs since only quasi-proofs are derivable in natural deduction.

Proposition 18 ($P \cong \neg\neg P$)**.** *One takes, referring to the terms from Examples 7 and 9 (also defined in Figure 4e):*

$$\phi_+(x^+) \stackrel{\text{def}}{=} (E\, x^+)^+$$
$$\psi_+(y^+) \stackrel{\text{def}}{=} (D_\neg\, y^+)^+$$

In $L_{pol,\hat{\mathsf{tp}}}$ and $\lambda\ell$, one has $P \cong_{\phi_+,\psi_+} \sim(P \to \bot)$.

Proposition 19 ($N \cong \neg\neg N$)**.** *We define, referring to the term from Example 10 (also defined in Figure 4e):*

$$\phi_\ominus(x^\ominus) \stackrel{\text{def}}{=} \lambda y^+.(\mathsf{send}\, y^+\, x^\ominus)^\ominus$$
$$\psi_\ominus(y^\ominus) \stackrel{\text{def}}{=} (\mathcal{T}\, y^\ominus)^\ominus$$

In $L_{pol,\hat{\mathsf{tp}}}$ and $\lambda\ell$, one has $N \cong_{\phi_\ominus,\psi_\ominus} \sim N \to \bot$.

The complete proofs appear in the author's PhD thesis [32, Section IV.5].

5. Conclusion

The calculi $L_{pol,\hat{\mathfrak{tp}}}$ and $\lambda\ell$ are shaped by a special constraint, that of having two readings: as proof systems and as programming languages. But we do not believe that they should be regarded as mere proof of concept, because the various ideas that they synchretise can be taken separately. For instance, it should be possible to restrict the calculi $L_{pol,\hat{\mathfrak{tp}}}$ and $\lambda\ell$ into direct-style languages for the more general Call-by-Push-Value models [27].

Also, relaxing an equality $\neg\neg A = A$ into an isomorphism as we do appears to be closely related to Melliès's dialogue chiralities [29]. But, if the simpler-to-manage strict involution is desired in a proof theoretic context, then it is certainly possible to recover a notion of quasi-proofs from the one we presented. Notably, it remains to see how much classical realisability results can be made simpler by using a setting where polarities are first-class. But to my surprise, the distinction between continuations and inspectable contexts seems to also have some importance in the formulae-as-types interpretation of Gödel's Dialectica translation from the same proceedings (Pédrot [38], private communication).

The making of this article is the story of how to avoid useless complexity in the calculi and systems we introduce, which are already quite complex. It seems essential for this to use an L calculus as an intermediate language in which program constructs are defined abstractly by their transition rules. Thus, Gentzen's discovery that sequent calculus is easier to reason about than natural deduction [13] reflects in term calculi.

Finally, high-level access to the stacks is an interesting and natural feature of delimited control operators, but it does not seem to be very developed in the programming languages literature.

References

[1] Zena M. Ariola and Hugo Herbelin, *Minimal Classical Logic and Control Operators*, ICALP '03, LNCS, vol. 2719, Springer, 2003, pp. 871–885.

[2] _____, *Control Reduction Theories: the Benefit of Structural Substitution*, Journal of Functional Programming **18** (2008), no. 3, 373–419.

[3] Zena M. Ariola, Hugo Herbelin, and Amr Sabry, *A Type-Theoretic Foundation of Continuations and Prompts*, ICFP '04, ACM, 2004, pp. 40–53.

[4] _____, *A type-theoretic foundation of delimited continuations*, Higher-Order and Symbolic Computation **22** (2009), no. 3, 233–273.

[5] John Clements, *Portable and high-level access to the stack with continuation marks*, Ph.D. thesis, Northeastern University, 2006.

[6] Pierre-Louis Curien, *Strong normalisation for focalising system L*, Unpublished lectures notes for the Oregon Programming Languages Summer School, July 2012.

[7] Pierre-Louis Curien and Hugo Herbelin, *The duality of computation*, ACM SIGPLAN Notices **35** (2000), 233–243.

[8] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx, *A New Deconstructive Logic: Linear Logic*, Journal of Symbolic Logic **62 (3)** (1997), 755–807.

[9] Olivier Danvy and Andrzej Filinski, *Abstracting control*, LISP and Functional Programming, 1990, pp. 151–160.

[10] Philippe de Groote, *A CPS-Translation of the λμ-Calculus*, CAAP, 1994, pp. 85–99.

[11] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba, *A syntactic theory of sequential control*, Theor. Comput. Sci. **52** (1987), 205–237.

[12] Harvey Friedman, *Classically and intuitionistically provably recursive functions*, Higher Set Theory (D. S. Scott and G. H. Muller, eds.), Lecture Notes in Mathematics, vol. 699, Springer-Verlag, 1978, pp. 21–28.

[13] Gerhard Gentzen, *Untersuchungen über das logische Schließen*, Mathematische Zeitschrift **39** (1935), 176–210,405–431.

[14] Jean-Yves Girard, *A new constructive logic: Classical logic*, Math. Struct. Comp. Sci. **1** (1991), no. 3, 255–296.

[15] _____, *Locus Solum: From the Rules of Logic to the Logic of Rules*, Mathematical Structures in Computer Science **11** (2001), 301–506.

[16] Timothy G. Griffin, *A Formulae-as-Types Notion of Control*, Seventeenth Annual ACM Symposium on Principles of Programming Languages, ACM Press, 1990, pp. 47–58.

[17] Hugo Herbelin, *An Intuitionistic Logic that Proves Markov's Principle*, LICS, 2010, pp. 50–56.

[18] Hugo Herbelin and Silvia Ghilezan, *An Approach to Call-by-Name Delimited Continuations*, Proc. POPL '08, ACM, 2008.

[19] Martin Hofmann and Thomas Streicher, *Completeness of continuation models for λμ-calculus*, Inf. Comput. **179** (2002), no. 2, 332–355.

[20] Danko Ilik, *Delimited control operators prove double-negation shift*, Annals of Pure and Applied logic **163** (2010), no. 11.

[21] Jean-Louis Krivine, *Lambda-calculus, types and models*, Ellis Horwood, 1993.

[22] _____, *Structures de réalisabilité, RAM et ultrafiltre sur N*, To appear, 2008.

[23] _____, *Realizability in Classical Logic*, Panoramas et Synthèses **27** (2009), 197–229, Circulated since 2004.

[24] _____, *Realizability algebras II : new models of ZF + DC*, Logical Methods in Computer Science **8** (2012), no. 1.

[25] J. Lambek and P. J. Scott, *Introduction to higher order categorical logic*, Cambridge University Press, New York, NY, USA, 1986.

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

[27] Paul Blain Levy, *Adjunction models for call-by-push-value with stacks*, Proc. Cat. Th. and Comp. Sci., ENTCS, vol. 69, 2005.

[28] Marek Materzok and Dariusz Biernacki, *A dynamic interpretation of the CPS hierarchy*, APLAS, 2012, pp. 296–311.

[29] Paul-André Melliès, *Dialogue categories and chiralities*, Submitted to a journal.

[30] Guillaume Munch-Maccagnoni, *Focalisation and Classical Realisability*, Proc. CSL '09, LNCS, Springer-Verlag, 2009.

[31] _____, *λ-calcul, machines et orthogonalité*, Unpublished manuscript, October 2011.

[32] _____, *Syntax and models of a non-associative composition of programs and proofs*, Ph.D. thesis, Univ. Paris Diderot, 2013.

[33] _____, *Models of a non-associative composition*, 17th International Conference on Foundations of Software Science and Computation Structures (FoSSaCs) (A. Muscholl, ed.), LNCS, vol. 8412, Springer, 2014, pp. 397–412.

[34] Chetan R. Murthy, *Extracting constructive content from classical proofs*, Ph.D. thesis, Cornell University, 1990.

[35] _____, *An evaluation semantics for classical proofs*, LICS, 1991, pp. 96–107.

[36] _____, *A Computational Analysis of Girard's Translation and LC*, LICS, IEEE Computer Society, 1992, pp. 90–101.

[37] Tobias Nipkow, *Higher-Order Critical Pairs*, Proc. 6th IEEE Symp. Logic in Computer Science, IEEE Press, 1991, pp. 342–349.

[38] Pierre-Marie Pédrot, *A functional functional interpretation*, Proceedings of the joint meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS), 2014.

[39] Alexis Saurin, *Separation with Streams in the Λμ-calculus*, LICS, IEEE Computer Society, 2005, pp. 356–365.

[40] Chung-Chieh Shan, *A static simulation of dynamic delimited control*, Higher-Order and Symbolic Computation **20** (2007), no. 4, 371–401.

[41] Thomas Streicher and Bernhard Reus, *Classical Logic, Continuation Semantics and Abstract Machines*, J. Funct. Program. **8** (1998), no. 6, 543–572.

[42] Philip Wadler, *Call-by-value is dual to call-by-name*, SIGPLAN Not. **38** (2003), no. 9, 189–201.