A Classical Sequent Calculus with Dependent Types

Étienne Miquey

A Classical Sequent Calculus with Dependent Types

Étienne Miquey

πr^2 (INRIA), IRIF, Univ. Paris Diderot
IMERL, FING, Univ. de la República, Montevideo
emiquey@pps.univ-paris-diderot.fr

Abstract

Dependent types are a key feature of type systems, typically used in the context of both richly-typed programming languages and proof assistants. Control operators, which are connected with classical logic along the proof-as-program correspondence, are known to misbehave in the presence of dependent types, unless dependencies are restricted to values. We place ourselves in the context of the sequent calculus which has the ability to smoothly provide control under the form of the $\mu\alpha$ operator dual to the common `let` operator, as well as to smoothly support abstract machine and continuation-passing style interpretations.

We start from the call-by-value version of the $\mu\tilde{\mu}$ language and design a minimal language with a value restriction and a type system that includes a list of explicit dependencies and maintains type safety. We then show how to relax the value restriction and introduce delimited continuations to directly prove the consistency by means of a continuation-passing-style translation. Finally, we relate our calculus to a similar system by Lepigre [17], and present a methodology to transfer properties from this system to our own.

Categories and Subject Descriptors F-4.1 [*Mathematical logic*]: Proof theory

Keywords dependent types, sequent calculus, classical logic, control operators, call-by-value, delimited continuations, continuation-passing style translation, value restriction

Introduction

Control operators and dependent types Originally created to deepen the connection between programming and logic, dependent types are now a key feature of numerous functional programing languages. On the programming side, they allow for the expression of very precise specifications, while on the logical side, they permit definitions of proof terms for axioms like the full axiom of choice. This is the case in Coq or Agda, two of the most actively developed proof assistants, which both provide dependent types. However, both of them rely on a constructive type theory (Coquand and Huet's calculus of constructions for Coq [4], and Martin-Löf's type theory [18] for Agda), and lack classical logic.

In 1990, Griffin discovered [10] that the control operator `call/cc` (short for *call with current continuation*) of the Scheme programming language could be typed by Peirce's $((A \to B) \to A) \to A$), thus extending the formulæ-as-types interpretation [15]. As Peirce's law is known to imply, in an intuitionistic framework, all the other forms of classical reasoning (excluded middle, *reductio ad absurdum*, double negation elimination, etc.), this discovery opened the way for a direct computational interpretation of classical proofs, using control operators and their ability to *backtrack*. Several calculi were born from this idea, such as Parigot's $\lambda\mu$-calculus [20], Barbanera and Berardi's symmetric λ-calculus [1], Krivine's λ_c-calculus [16] or Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [5].

Nevertheless, dependent types are known to misbehave in the presence of control operators, causing a degeneracy of the domain of discourse [12]. Some restrictions on the dependent types are thus necessary to make them compatible with classical logic. Although dependent types and classical logic have been deeply studied separately, the question to know how to design a system compatible with both features does not have yet a general and definitive answer. Recent works from Herbelin [13] and Lepigre [17] proposed some restrictions on the dependent types to tackle the issue in the case of a proof system in natural deduction, while Blot [3] designed a hybrid realizability model where dependent types are restricted to an intuitionistic fragment.

Call-by-value In languages enjoying the Church-Rosser property (like the λ-calculus or Coq), the order of evaluation is irrelevant, and any reduction path will ultimately lead to the same value. In particular, the call-by-name and call-by-value evaluation strategies will always give the same result. However, this is no longer the case in presence of side-effects. Indeed, consider the simple case of a function applied to a term producing some side-effects (for instance increasing a reference). In call-by-name, the computation of the argument is delayed to the time of its effective use, while in call-by-value the argument is reduced to a value before performing the application. If, for instance, the function never uses its argument, the call-by-name evaluation will not generate any side-effect, and if it uses it twice, the side-effect will occurs twice (and the reference will have its value increased by two). On the contrary, in both cases the call-by-value evaluation generates the side-effect exactly once (and the reference has its value increased by one).

In this paper, we present a language following the call-by-value reduction strategy, which is as much a design choice as a goal in itself. Indeed, when considering a language with control operators, soundness often turns out to be subtle to preserve in call-by-value, and the loss of subject reduction we mention in the next paragraph is precisely due to the evaluation strategy we choose.

Value restriction The first issues in call-by-value in the presence of side-effects were related to references [23] and polymorphism[11]. In both cases, a simple and elegant (but way too restrictive in practice [9, 17]) to solve the inconsistencies consists in a restriction to values for the problematic cases, restoring then a sound type system. Recently, Lepigre presented a proof system providing de-

pendent types and a control operator[17], whose consistency is preserved by means of a semantical value restriction defined for terms that behave as values up to observational equivalence.

In the present work, we will rather use a syntactic restriction to a fragment of proofs that allows slightly more than values. This restriction is inspired by the negative-elimination-free fragment of Herbelin's dPAω system [13].

A sequent calculus presentation The main achievement of this paper is to give a sequent calculus presentation of a call-by-value language with a control operator and dependent types, and to justify its soundness through a continuation-passing style translation. Our calculus is an extension of the $\mu\tilde{\mu}$-calculus [5] to dependent types. Amongst other motivations, such a calculus is close to an abstract machine, which makes it particularly suitable to define CPS translations or to be an intermediate language for compilation [6]. In particular, the system we develop might be a first step to allow the adaption of the well-understood continuation-passing style translations for ML in order to design a typed compilation of a system with dependent types such as Coq.

However, in addition to the simultaneous presence of control and dependent types, the sequent calculus presentation itself is responsible for another difficulty. As we will see in Section 1.4, the usual call-by-value strategy of the $\mu\tilde{\mu}$-calculus causes subject reduction to fail. The problem can be understood as a desynchronization of the type system with the reduction. It can be solved by the addition of an explicit list of dependencies in the type derivations.

Delimited continuations and CPS translation Yet, we will show that the compensation within the typing derivations does not completely fix the problem, and in particular that we are unable to derive a continuation-passing style translation. We present a way to solve this issue by introducing delimited continuations. It also justifies the relaxation of the value restriction to the negative-elimination-free fragment (Section 2). Finally, we design in Section 3 a continuation-passing style translation that preserves dependent types and allows for proving the soundness of our system.

1. A minimal classical language

The simultaneous presence of classical logic (*i.e.* of a control operator) and dependent types is known to cause a degeneracy of the domain of discourse. Even considering a minimal logic of Σ-types and equality, with terms and proofs defined by:

$$\begin{array}{lcl} t, u & \triangleq & n \in \mathbb{N} \mid \mathsf{wit}\, p \\ p, q & \triangleq & (t,p) \mid \mathsf{prf}\, p \mid \mathsf{refl} \mid \mathsf{subst}\, p\, q \end{array}$$

its classical extension with cc_k (the `call/cc` operator) and $\mathsf{throw}\, k$ permits to derive a proof of $0 = 1$ [12]. More precisely, the degeneracy relies on the following reduction rule:

$$\mathsf{wit}\,(\mathsf{cc}_k p) \to \mathsf{cc}_k(\mathsf{wit}\, p[k\,(\mathsf{wit}\,[])/k])$$

where $[k\,(\mathsf{wit}\,[])\,/k]$ denotes the capture-free substitution replacing every occurrence of $\mathsf{throw}\, k\, t$ with $\mathsf{throw}\, k\,(\mathsf{wit}\, t)$. Intuitively, the incoherence comes from the fact that if p is a classical proof of the form $\mathsf{cc}_k\,(0, \mathsf{throw}\, k\,(1, \mathsf{refl})) : \Sigma x.x = 1$, the previous rule will cause the term $\mathsf{wit}\, p$ to reduce to 0, while the reduction of $\mathsf{prf}\, p$ would have backtracked before giving 1 as a witness and the corresponding certificate.

In a sequent calculus presentation of the same logic, such a rule happens to be somehow unnatural:

$$\langle \mathsf{wit}\, \mu\alpha.c \| e\rangle \to \langle \mu\alpha.c[\tilde{\mu}a.\langle \mathsf{wit}\, a \| \alpha\rangle/\alpha] \| e\rangle.$$

A more intuitive way to define a reduction rule for the same rule is to first reduce $\mu\alpha.c$ to a value V and then reduce $\mathsf{wit}\ V$:

$$\langle \mathsf{wit}\, \mu\alpha.c \| e\rangle \to \langle \mu\alpha.c \| \tilde{\mu}a.\langle \mathsf{wit}\, a \| e\rangle\rangle.$$

This still leaves us with the problem of convertibility of $\mathsf{wit}\, \mu\alpha.c$ within types. The easiest and usual approach to prevent this is to impose a restriction to values for proofs appearing inside dependent types and operators. In this section we will focus on this solution in the same minimal framework, and show how it permits to keep the proof system coherent. We shall see further in Section 2 how to relax this constraint.

1.1 The language

We place ourselves in the classical framework of the $\mu\tilde{\mu}$-calculus [5], that we define to be the domain of *proofs*[1], and to which we add:

- *terms*[1] which contain an encoding[2] of the natural numbers,
- proof terms (t, p) to inhabit the strong existential $\exists x^T A$ together with the corresponding projections wit and prf ,
- a proof term refl for the equality of terms and a proof term subst for the convertibility of types over equal terms.

In order to remain as general as possible, throughout this paper we will consider terms of type T, when the only type T for terms is $\mathbb{N}$. We believe it does not introduce to much of a complication, and it anticipates future extensions of the domain of terms as discussed in Section 5.2. The syntax of the corresponding system, that we call dL, is given in Figure 1.

Terms	t	::=	$x \mid n \in \mathbb{N} \mid \mathsf{wit}\, p$
Proofs	p	::=	$V \mid \mu\alpha.c \mid (t,p) \mid \mathsf{prf}\, p \mid \mathsf{subst}\, p\, q$
Values	V	::=	$a \mid \lambda a.p \mid \lambda x.p \mid (t, V) \mid \mathsf{refl}$
Contexts	e	::=	$\alpha \mid p \cdot e \mid t \cdot e \mid \tilde{\mu}a.c$
Commands	c	::=	$\langle p \| e\rangle$

Figure 1. Language of dL

The formulas are defined by:

$$A, B \quad ::= \quad \top \mid \bot \mid t = u \mid \forall x^T.A \mid \exists x^T.A \mid \Pi_{a:A} B.$$

Note that we included a dependent product $\Pi_{a:A} B$ at the level of proof terms, but that in the case where $a \notin FV(B)$ this amounts to the usual implication $A \to B$.

1.2 Reduction rules

As explained in the introduction of this section, a backtracking proof might give place to different witnesses and proofs according to the context of reduction, leading to incoherences [12]. On the contrary, the call-by-value evaluation strategy forces a proof to reduce first to a value (thus furnishing a witness) and to share this value amongst all the commands. In particular, this maintains the value restriction along reduction, since only values are substituted.

The reduction rules, defined in Figure 13 (where $t \to t'$ denotes the reduction of terms and $c \rightsquigarrow c'$ the reduction of commands), follow the call-by-value evaluation principle. In particular one can see that whenever the command is of the shape $\langle C[p] \| e\rangle$ where $C[p]$ is a proof built on top of p which is not a value, it reduces to $\langle p \| \tilde{\mu}a.\langle C[a] \| e\rangle\rangle$, opening the construction to evaluate p [3].

Additionally, we denote by $A \equiv B$ the transitive-symetric closure of the relation $A \rhd B$, defined as a congruence over term reduction (*i.e.* if $t \to t'$ then $A[t] \rhd A[t']$) and by the rules:

$$\begin{array}{ll} 0 = 0 \rhd \top & 0 = S(u) \rhd \bot \\ S(t) = 0 \rhd \bot & S(t) = S(u) \rhd t = u. \end{array}$$

[1] *Terms* represent mathematical objects while *proofs* or *proof terms* represent mathematical proofs.

[2] The nature of the representation is irrelevant here as we will not compute over it. We can for instance add one constant for each natural number.

[3] The reader might recognize the rule (ς) of Wadler's sequent calculus [22].

$$\frac{\Gamma \vdash p : A \mid \Delta;\sigma \quad \Gamma \mid e : A' \vdash \Delta;\sigma\{\cdot|p\} \quad A' \in A_\sigma}{\langle p \| e\rangle : \Gamma \vdash \Delta;\sigma}\ \textsc{Cut}$$

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash a : A \mid \Delta;\sigma}\ \textsc{Ax}^+ \qquad \frac{(\alpha : A) \in \Delta}{\Gamma \mid \alpha : A \vdash \Delta;\sigma\{\cdot|p\}}\ \textsc{Ax}^-$$

$$\frac{c : (\Gamma \vdash \Delta, \alpha : A;\sigma)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta;\sigma}\ \mu \qquad \frac{c : (\Gamma, a : A \vdash \Delta;\sigma\{a|p\})}{\Gamma \mid \tilde{\mu}a.c : A \vdash \Delta;\sigma\{\cdot|p\}}\ \tilde{\mu}$$

$$\frac{\Gamma, a : A \vdash p : B \mid \Delta;\sigma}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta;\sigma}\ \to_I \qquad \frac{\Gamma \vdash q : A \mid \Delta;\sigma \quad \Gamma \mid e : B[q/a] \vdash \Delta;\sigma\{\cdot|\dagger\} \quad q \notin \mathcal{D} \to a \notin FV(B)}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta;\sigma\{\cdot|p\}}\ \to_E$$

$$\frac{\Gamma, x : T \vdash p : A \mid \Delta;\sigma}{\Gamma \vdash \lambda x.p : \forall x^T A \mid \Delta;\sigma}\ \forall_l \qquad \frac{\Gamma \vdash t : T \vdash \Delta;\sigma \quad \Gamma \mid e : A[t/x] \vdash \Delta;\sigma\{\cdot|\dagger\}}{\Gamma \mid t \cdot e : \forall x^T A \vdash \Delta;\sigma\{\cdot|p\}}\ \forall_r$$

$$\frac{\Gamma \vdash t : T \mid \Delta;\sigma \quad \Gamma \vdash p : A(t) \mid \Delta;\sigma}{\Gamma \vdash (t, p) : \exists x^T A(x) \mid \Delta;\sigma}\ \exists \qquad \frac{\Gamma \vdash p : \exists x^T A(x) \mid \Delta;\sigma \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{prf}\, p : A(\mathsf{wit}\, p) \mid \Delta;\sigma}\ \mathsf{prf}$$

$$\frac{\Gamma \vdash p : A \mid \Delta;\sigma \quad A \equiv B}{\Gamma \vdash p : B \mid \Delta;\sigma}\ \textsc{Conv}^+ \qquad \frac{\Gamma \mid e : A \vdash \Delta;\sigma \quad A \equiv B}{\Gamma \mid e : B \vdash \Delta;\sigma}\ \textsc{Conv}^-$$

$$\frac{\Gamma \vdash p : t = u \mid \Delta;\sigma \quad \Gamma \vdash q : B[t/x] \mid \Delta;\sigma}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta;\sigma}\ \mathsf{subst} \qquad \frac{\Gamma \vdash t : T \mid \Delta;\sigma}{\Gamma \vdash \mathsf{refl} : t = t \mid \Delta;\sigma}\ \mathsf{refl}$$

$$\frac{}{\Gamma, x : T \vdash x : T \mid \Delta;\sigma} \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash n : \mathbb{N} \mid \Delta;\sigma} \qquad \frac{\Gamma \vdash p : \exists x A(x) \mid \Delta;\sigma \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{wit}\, p : \mathbb{N} \mid \Delta;\sigma}\ \mathsf{wit}$$

Figure 3. Typing rules

$$\begin{array}{rcll}
\langle \mu\alpha.c \| e\rangle & \rightsquigarrow & c[e/\alpha] & \\
\langle V \| \tilde{\mu}a.c\rangle & \rightsquigarrow & c[V/a] & \\
\langle \lambda a.p \| q \cdot e\rangle & \rightsquigarrow & \langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle & \\
\langle \lambda x.p \| t \cdot e\rangle & \rightsquigarrow & \langle p[t/x] \| e\rangle & \\
\langle (t,p) \| e\rangle & \rightsquigarrow & \langle p \| \tilde{\mu}a.\langle (t,a) \| e\rangle\rangle & (p \notin V) \\
\langle \mathsf{prf}\ (t, V) \| e\rangle & \rightsquigarrow & \langle V \| e\rangle & \\
\langle \mathsf{subst}\ \mathsf{refl}\ q \| e\rangle & \rightsquigarrow & \langle q \| e\rangle & \\
\langle \mathsf{subst}\ p\, q \| e\rangle & \rightsquigarrow & \langle p \| \tilde{\mu}a.\langle \mathsf{subst}\ a\, q \| e\rangle\rangle & (p \notin V) \\
\mathsf{wit}\ (t, V) & \to & t & \\
t \to t' & \Rightarrow & c[t] \rightsquigarrow c[t'] &
\end{array}$$

Figure 2. Reduction rules

1.3 Typing rules

As we previously explained, in this section we will limit ourselves to the simple case where dependent types are restricted to values, to make them compatible with classical logic. But even with this restriction, defining the type system in the most naive way leads to a system in which subject reduction will fail. Having a look at the β-reduction rule gives us an insight of what happens. Let us consider a proof $\lambda a.p : \Pi_{a:A} B$ and a context $q \cdot e : \Pi_{a:A} B$ (with q a value). A typing derivation of the corresponding command is of the form:

$$\frac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta} \qquad \dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta}}{\langle \lambda a.p \| q \cdot e\rangle : \Gamma \vdash \Delta}$$

while the command will reduce as follows:

$$\langle \lambda a.p \| q \cdot e\rangle \rightsquigarrow \langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle$$

On the right side, we see that p, whose type is $B[a]$, is now cut with e which type is $B[q]$. Consequently we are not able to derive a typing judgment for this command any more.

The intuition is that in the full command, a has been linked to q at a previous level of the typing judgment. However, the command is still safe, since the head-reduction imposes that the command $\langle p \| e\rangle$ will not be executed until the substitution of a by q[4], and by then the problem would have been solved. Somehow, this phenomenon can be seen as a desynchronization of the typing process with respect to the computation. The synchronization can be re-established by making explicit a *dependencies list* in the typing rules, allowing this typing derivation:

$$\frac{\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \qquad \dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B[a] \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma, a : A \mid e : B[q] \vdash \Delta;\{a|q\}}}{\langle p \| e\rangle : \Gamma, a : A \vdash \Delta;\{a|q\}}}{\Gamma \mid \tilde{\mu}a.\langle p \| e\rangle : A \vdash \Delta;\{.|q\}}}{\langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle : \Gamma \vdash \Delta;\varepsilon}$$

Formally, we denote by $\mathcal{D}$ the set of proofs we authorize in dependent types, and define it for the moment as the set of values:

$$\mathcal{D} \triangleq V.$$

We define a dependencies list σ as a list binding pairs of proof terms[5]:

$$\sigma ::= \varepsilon \mid \sigma\{p|q\},$$

[4] Note that even if we were not restricting ourselves to values, this would still hold: if at some point the command $\langle p \| e\rangle$ is executed, it is necessarily after that q has produced a value to substitute for a.

[5] In practice we will only bind a variable with a proof term, but it is convenient for proofs to consider this slightly more general definition

and we define A_σ as the set of types that can be obtained from A by replacing none or all occurrences of p by q for each binding $\{p|q\}$ in σ such that $q \in \mathcal{D}$:

$$A_\varepsilon \triangleq \{A\} \qquad A_{\sigma\{p|q\}} \triangleq \begin{cases} A_\sigma \cup (A[q/p])_\sigma & \text{if } q \in \mathcal{D} \\ A_\sigma & \text{otherwise.} \end{cases}$$

Furthermore, we introduce the notation $\Gamma \mid e : A \vdash \Delta; \{\cdot|\dagger\}$ to denote that any future binding $\{a|\dagger\}$ introduced when the definition of a second type of sequent $\Gamma \mid e : A \vdash \Delta; \sigma$ to type contexts when dropping the (open) binding $\{\cdot|p\}$. Alternatively, one can think of $\dagger$ as any proof term not in $\mathcal{D}$, which is the same with respect to the dependencies list. The resulting set of typing rules is given in Figure 3, where we assume that every variable bound in the typing context is bound only once (proofs and contexts are considered up to α-conversion).

Note that we work with two-sided sequents here to stay as close as possible to the original presentation of the $\mu\tilde{\mu}$-calculus [5]. In particular it means that a type in Δ might depend on a variable previously introduced in Γ and reciprocally, so that the split into two contexts makes us lose track of the order of introduction of the hypothesis. In the sequel, to be able to properly define a typed CPS translation, we consider that we can unify both contexts into a single one that is coherent with respect to the order in which the hypothesis have been introduced. We denote by $\Gamma \cup \Delta$ this context, where the assumptions of Γ remain unchanged, while the former assumptions $(\alpha : A)$ in Δ are denoted by $(\alpha : A^{\perp\!\!\!\perp})$.

1.4 Subject reduction

We start by proving a few technical lemmas we will use to prove the subject reduction property. First, we prove that typing derivations allow weakening on the dependencies list. For this purpose, we introduce the notation $\sigma \Rrightarrow \sigma'$ to denote that whenever a judgment is derivable with σ as dependencies list, then it is derivable using σ':

$$\sigma \Rrightarrow \sigma' \quad \triangleq \quad \forall c \forall \Gamma \forall \Delta (c : (\Gamma \vdash \Delta; \sigma) \Rightarrow c : (\Gamma \vdash \Delta; \sigma')).$$

This clearly implies that the same property holds when typing contexts, *i.e.* if $\sigma \Rrightarrow \sigma'$ then σ can be replaced by σ' in any derivation for typing a context.

Lemma 1. *For any dependencies list σ and value V, we have $\sigma\{V|V\} \Rrightarrow \sigma$.*

Lemma 2 (Dependencies weakening)**.** *For any dependencies lists σ and σ', we have $\sigma \Rrightarrow \sigma\sigma'$.*

Proof. The first lemma is obvious. The proof of the second is straightforward from the fact that for any p and q, by definition $A_\sigma \subset A_{\sigma\{a|q\}}$. □

As a corollary, we get that $\dagger$ can indeed be replaced by any proof term when typing a context.

Corollary 3. *If $\sigma \Rrightarrow \sigma'$, then for any p, e, Γ, Δ:*

$$\Gamma \mid e : A \vdash \Delta; \sigma\{\cdot|\dagger\} \quad \Rightarrow \quad \Gamma \mid e : A \vdash \Delta; \sigma'\{\cdot|p\}.$$

We now prove the usual lemmas that guarantee the safety of terms (resp. values, contexts) substitution.

Lemma 4 (Safe term substitution)**.** *If $\Gamma \vdash t : T \mid \Delta; \varepsilon$ then:*

1. *if $c : (\Gamma, x : T, \Gamma' \vdash \Delta; \sigma)$ then:*
$$c[t/x] : (\Gamma, \Gamma'[t/x] \vdash \Delta[t/x]; \sigma[t/x]),$$
2. *if $\Gamma, x : T, \Gamma' \vdash q : B \mid \Delta$ then:*
$$\Gamma, \Gamma'[t/x] \vdash q[t/x] : B[t/x] \mid \Delta[t/x],$$
3. *if $\Gamma, x : T, \Gamma' \mid e : B \vdash \Delta; \sigma$ then:*
$$\Gamma, \Gamma'[t/x] \mid e[t/x] : B[t/x] \vdash \Delta[t/x]; \sigma[t/x].$$

Lemma 5 (Safe value substitution)**.** *If $\Gamma \vdash V : A \mid \Delta; \varepsilon$ then:*

1. *if $c : \Gamma, a : A, \Gamma' \vdash \Delta; \sigma$ then:*
$$c[V/a] : \Gamma, \Gamma'[V/a] \vdash \Delta[V/a]; \sigma[V/a],$$
2. *if $\Gamma, a : A, \Gamma' \vdash q : B \mid \Delta$ then:*
$$\Gamma, \Gamma'[V/a] \vdash q[V/a] : B[V/a] \mid \Delta[V/a],$$
3. *if $\Gamma, a : A, \Gamma' \mid e : B \vdash \Delta; \sigma$ then:*
$$\Gamma, \Gamma'[V/a] \mid e[V/a] : B[V/a] \vdash \Delta[V/a]; \sigma[V/a].$$

Lemma 6 (Safe context substitution)**.** *If $\Gamma \mid e : A \vdash \Delta; \varepsilon$ then:*

1. *if $c : \Gamma \vdash \Delta, \alpha : A, \Delta'; \sigma$ then:*
$$c[e/\alpha] : \Gamma \vdash \Delta, \Delta'; \sigma,$$
2. *if $\Gamma \vdash q : B \mid \Delta, \alpha : A, \Delta'$ then:*
$$\Gamma \vdash q[e/\alpha] : B \mid \Delta, \Delta',$$
3. *if $\Gamma \mid e : B \vdash \Delta, \alpha : A, \Delta'; \sigma$ then:*
$$\Gamma \mid e[e/\alpha] : B \vdash \Delta, \Delta'; \sigma.$$

Proof. The proofs are done by induction on typing rules. □

We can now prove the safety of reduction, using the previous lemmas for rules performing a substitution and the dependencies lists to resolve local inconsistencies for dependent types.

Theorem 7 (Subject reduction)**.** *If $c : (\Gamma \vdash \Delta; \varepsilon)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta; \varepsilon)$.*

Proof. The proof is done by induction on the typing rules, assuming that for each typing proof, the CONV rules are always pushed down and right as much as possible. To save some space, we sometimes omit the dependencies list when empty, noting $c : \Gamma \vdash \Delta$ instead of $c : \Gamma \vdash \Delta; \varepsilon$, and we denote the CONV-rules by:

$$\frac{\Gamma \mid e : B \vdash \Delta; \sigma}{\Gamma \mid e : A \vdash \Delta; \sigma} \equiv$$

where the hypothesis $A \equiv B$ is implicit. We only give some key cases, the full proof is given in appendix.

- **Case** $\langle \lambda a.p \| q \cdot e \rangle \rightsquigarrow \langle q \| \tilde{\mu} a.\langle p \| e \rangle \rangle$.
A typing proof for the command on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta} \quad \dfrac{\Pi_q \quad \dfrac{\Pi_e}{\Gamma \mid e : B'[q/a] \vdash \Delta; \{\cdot|\dagger\}}}{\Gamma \mid q \cdot e : \Pi_{a:A'} B' \vdash \Delta; \{\cdot|\lambda a.p\}}}{\langle \lambda a.p \| q \cdot e \rangle : \Gamma \vdash \Delta} \equiv$$

If $q \notin \mathcal{D}$, we define $B'_q \triangleq B'$ which is the only type in $B'_{\{a|q\}}$. Otherwise, we define $B'_q \triangleq B'[q/a]$ which is a type in $B'_{\{a|q\}}$. In both case, we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A' \mid \Delta}}{\Gamma \vdash q : A \mid \Delta} \equiv \dfrac{\Pi_{\tilde{\mu}}}{\Gamma \mid \tilde{\mu} a.\langle p \| e \rangle : A \vdash \Delta; \{.|q\}}}{\langle q \| \tilde{\mu} a.\langle p \| e \rangle \rangle : \Gamma \vdash \Delta; \varepsilon}$$

$$\Pi_{\tilde{\mu}} = \dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma, a : A \vdash p : B' \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma, a : A \mid e : B'_q \vdash \Delta; \{a|q\}\{\cdot|p\}}}{\langle p \| e \rangle : \Gamma, a : A \vdash \Delta; \{a|q\}} }{\Gamma \mid \tilde{\mu} a.\langle p \| e \rangle : A \vdash \Delta; \{.|q\}}$$

using Corollary 3 to weaken the dependencies in Π_e.
- **Case** $\langle (t, p) \| e \rangle \rightsquigarrow \langle p \| \tilde{\mu} a.\langle (t, a) \| e \rangle \rangle$, with $p \notin V$.

A proof of the command on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma \vdash t : T \mid \Delta} \quad \dfrac{\Pi_p}{\Gamma \vdash p : A[t/x] \mid \Delta}}{\Gamma \vdash (t,p) : \exists x^T A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : \exists x^T A \vdash \Delta; \{\cdot \mid (t,p)\}}}{\langle (t,p) \| e \rangle : \Gamma \vdash \Delta}$$

We can build the following derivation:

$$\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : A[t/x] \mid \Delta} \quad \dfrac{\Pi_{(t,a)} \quad \dfrac{\Pi_e}{\Gamma \mid e : \exists x^T A \vdash \Delta; \{a|p\}\{\cdot|(t,a)\}}}{\langle (t,a) \| e \rangle : \Gamma, a : A[t/x] \vdash \Delta; \{a|p\}}}{\Gamma \mid \tilde{\mu} a.\langle (t,a) \| e \rangle : A[t/x] \vdash \Delta; \{\cdot|p\}}}{\langle p \| \tilde{\mu} a.\langle (t,a) \| e \rangle \rangle : \Gamma \vdash \Delta}$$

where $\Pi_{(t,a)}$ is as expected, observing that since $p \notin \mathcal{D}$, the binding $\{\cdot|(t,p)\}$ is the same as $\{\cdot|\dagger\}$, and we can apply Corollary 3 to weaken dependencies in Π_e.

- **Case** $\langle \mathsf{prf}\,(t,V) \| e \rangle \rightsquigarrow \langle V \| e \rangle$.
  This case is easy, observing that a derivation of the command on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_t \quad \dfrac{\Pi_V}{\Gamma \vdash V : A(t) \mid \Delta}}{\Gamma \vdash (t,V) : \exists x^T A(x) \mid \Delta}}{\Gamma \vdash \mathsf{prf}\,(t,V) : A(\mathsf{wit}\,(t,V)) \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\,(t,V)) \vdash \Delta; \{\cdot|\dagger\}}}{\langle \mathsf{prf}\,(t,V) \| e \rangle : \Gamma \vdash \Delta}$$

  Since by definition we have $A(\mathsf{wit}\,(t,V)) \equiv A(t)$, we can derive:

$$\dfrac{\dfrac{\Pi_V}{\Gamma \vdash V : A(t) \mid \Delta} \quad \dfrac{\dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\,(t,V)) \vdash \Delta; \{\cdot|V\}}}{\Gamma \mid e : A(V) \vdash \Delta; \{\cdot|V\}}}{\langle \mathsf{prf}\,(t,V) \| e \rangle : \Gamma \vdash \Delta} \equiv$$

□

1.5 Soundness

We sketch here a proof of the soundness of dL with a value restriction. As we do not give any detail of the proofs, we only state the results as claims. A more interesting proof through a continuation-passing translation is presented in Section 3. We first show that typed commands of dL normalize by translation to the simply-typed $\mu\tilde{\mu}$-calculus with pairs (*i.e.* extended with proofs of the form (p_1, p_2) and contexts of the form $\tilde{\mu}(a_1, a_2).c$). The translation essentially consists of erasing the dependencies in types, turning the dependent products into arrows and the dependent sum into a pair. The erasure procedure is defined by:

$$\begin{array}{lcl|lcl} (\forall x^T A)^* & \triangleq & T \to A^* & \top^* & \triangleq & \top \\ (\exists x^T A)^* & \triangleq & T \wedge A^* & \bot^* & \triangleq & \bot \\ (\Pi_{a:A} B)^* & \triangleq & A^* \to B^* & (t=u)^* & \triangleq & X \to X \end{array}$$

and the translation for proofs, terms, contexts and commands is defined by:

$$\begin{array}{lll|lll} x^* & \triangleq & x & (t,p)^* & \triangleq & \mu\alpha.\langle p^* \| \tilde{\mu} a.\langle (t^*, a) \| \alpha \rangle \rangle \\ n^* & \triangleq & \bar{n} & (\mathsf{prf}\,p)^* & \triangleq & \pi_2(p^*) \\ (\mathsf{wit}\,p)^* & \triangleq & \pi_1(p^*) & & & \\ a^* & \triangleq & a & \alpha^* & \triangleq & \alpha \\ (\lambda a.p)^* & \triangleq & \lambda a.p^* & (t \cdot e)^* & \triangleq & t^* \cdot e^* \\ (\lambda x.p)^* & \triangleq & \lambda x.p^* & (q \cdot e)^* & \triangleq & q^* \cdot e^* \\ (\mu\alpha.c)^* & \triangleq & \mu\alpha.c^* & (\tilde{\mu} a.c)^* & \triangleq & \tilde{\mu} a.c^* \\ \mathsf{refl}^* & \triangleq & \lambda x.x & \langle p \| e \rangle^* & \triangleq & \langle p^* \| e^* \rangle \end{array}$$

$$(\mathsf{subst}\,V\,q)^* \triangleq \mu\alpha.\langle q^* \| \alpha \rangle$$

$$(\mathsf{subst}\,p\,q)^* \triangleq \mu\alpha.\langle p^* \| \tilde{\mu}_-.\langle \mu\alpha.\langle q^* \| \alpha \rangle \| \alpha \rangle \rangle \qquad (p \notin V)$$

where $\pi_i(p) \triangleq \mu\alpha.\langle p \| \tilde{\mu}(a_1, a_2).\langle a_1 \| \alpha \rangle \rangle$. We define $\bar{n}$ as any encoding of the natural numbers with its type $\mathbb{N}^*$, the encoding being irrelevant here.

We can extend the erasure procedure to contexts, and show that it is adequate with respect to the translation of proofs.

Proposition 8. *If $c : \Gamma \vdash \Delta; \sigma$, then $c^* : \Gamma^* \vdash \Delta^*$. The same holds for proofs and contexts.*

We can then deduce the normalization of dL from the normalization of the $\mu\tilde{\mu}$-calculus, by showing that the translation preserves the normalization in the sense that if c does not normalize, neither does c^*.

Proposition 9. *If $c : (\Gamma \vdash \Delta; \varepsilon)$, then c normalizes.*

Using the normalization, we can finally prove the soundness of the system.

Theorem 10 (Soundness)**.** *For any $p \in$ dL, we have $\not\vdash p : \bot$.*

Proof. (Sketch) Proof by contradiction, assuming that there is a proof p such that $\vdash p : \bot$, we can form the well-typed command $\langle p \| \star \rangle : (\vdash \star : \bot)$ where $\star$ is any fresh α-variable, and use the normalization to reduce to a command $\langle V \| \star \rangle$. By subject reduction, V would be a value of type $\bot$, which is absurd. □

1.6 Toward a continuation-passing style translation

The difficulty we encountered while defining our system mostly came from the simultaneous presence of a control operator and dependent types. Removing one of these two ingredients leaves us with a sound system in both cases: without the part necessary for dependent types, our calculus amounts to the usual $\mu\tilde{\mu}$-calculus. Without control operator, we would obtain an intuitionistic dependent type theory that would be easy to prove sound.

To demonstrate the correctness of our system, we might be tempted to define a translation to a subsystem without dependent types or control operator. We will discuss later in Section 4 a solution to handle the dependencies. We will focus here on the possibility of removing the classical part from dL, that is to define a translation that gets rid of the control operator. The use of continuation-passing style translations to address this issue is very common, and it was already studied for the simply-typed $\mu\tilde{\mu}$-calculus [5]. However, as it is defined to this point, dL is not suitable for the design of a CPS translation.

Indeed, in order to fix the problem of desynchronization of typing with respect to the execution, we have added an explicit dependencies list to the type system of dL. Interestingly, if this solved the problem inside the type system, the very same phenomenon happens when trying to define a CPS-translation carrying the type dependencies.

Let us consider the same case of a command $\langle q \| \tilde{\mu} a.\langle p \| e \rangle \rangle$ with $p : B[a]$ and $e : B[q]$. Its translation is very likely to look like:

$$[\![q]\!]\,[\![\tilde{\mu} a.\langle p \| e \rangle]\!] = [\![q]\!]\,(\lambda a.([\![p]\!])[\![e]\!]),$$

where $[\![p]\!]$ has type $(B[a] \to \bot) \to \bot$ and $[\![e]\!]$ type $B[q] \to \bot$, hence the sub-term $[\![p]\!]\,[\![e]\!]$ will be ill-typed. Therefore the fix at the level of typing rules is not satisfactory, and we need to tackle the problem already within the reduction rules.

We follow the idea that the correctness is guaranteed by the head-reduction strategy, preventing $\langle p \| e \rangle$ from reducing before the substitution of a was made. We would like to ensure the same thing happens in the target language (that will also be equipped with a head-reduction strategy), namely that $[\![p]\!]$ cannot be applied to $[\![e]\!]$ before $[\![q]\!]$ has furnished a value to substitute for a. This would

correspond informally to the term[6]:

$$([\![q]\!](\lambda a.[\![p]\!]))[\![e]\!].$$

The first observation is that, if instead of a value, q was a classical proof throwing the current continuation away, for instance $\mu\alpha.c$ where $\alpha \notin FV(c)$, this would lead to an incorrect term $[\![c]\!]\,[\![e]\!]$. We thus need to at least restrict to proof terms that could not throw the current continuation.

The second observation is such a term suggests the use of delimited continuations [14], to temporarily encapsulate the evaluation of q when reducing such a command:

$$\langle \lambda a.p \| q \cdot e \rangle \;\rightsquigarrow\; \langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle,$$

which is safe under the guarantee that q will not throw away the continuation $\tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle$. This will also allow us to restrict the use of the dependencies list to the derivation of judgments involving a delimited continuation, and to fully absorb the potential inconsistency in the type of $\hat{\mathfrak{tp}}$.

In Section 2, we will extend the language according to this intuition, and see how to design a continuation-passing style translation in Section 3.

2. Extension of the system

2.1 Limits of the value restriction

In the previous section, we strictly restricted the use of dependent types to proof terms that are values. In particular, even though a proof-term might be computationally equivalent to some value (say $\mu\alpha.\langle V \| \alpha \rangle$ and V for instance), we cannot use it to eliminate a dependent product, which is unsatisfying. We shall then relax this restriction to allow more proof terms within dependent types.

We can follow several intuitions. First, we saw in the previous section that we could actually allow any proof terms as long as its CPS translation uses its continuation and uses it only once. We do not have such a translation yet, but syntactically, these are the proof terms that can be expressed (up to α-conversion) in the $\mu\tilde{\mu}$-calculus with only one continuation variable $\star$ (see Figure 5), and which do not contain application[7]. Interestingly, this corresponds exactly to the so-called *negative-elimination-free* (NEF) proofs of Herbelin [13]. To interpret the axiom of dependent choice, he designed a classical proof system with dependent types in natural deduction, in which the dependent types allow the use of NEF proofs.

Finally, Lepigre defined in a recent work [17] a classical proof system with dependent types, where the dependencies are restricted to values. However, the type system allows derivations of judgments up to an observational equivalence, and thus any proof computationally equivalent to a value can be used. In particular, any proof in the NEF fragment is observationally equivalent to a value, hence is compatible with the dependencies of Lepigre's calculus.

From now on, we consider the system of Section 1 extended with delimited continuations, and define the fragment of *negative-elimination-free* proof terms (NEF). The syntax of both categories is given by Figure 5, the proofs in the NEF fragment are considered up to α-conversion for the context variables[8].

The reduction rules, given in Figure 4, are slightly different from the rules in Section 1. In the case $\langle \lambda a.p \| q \cdot e \rangle$ with $q \in$ NEF (resp. $\langle \mathsf{prf}\ p \| e \rangle$), a delimited continuation is now produced during the reduction of the proof term q (resp. p) that is involved in dependencies. As terms can now contain proofs which are not values, we enforce the call-by-value reduction by asking proof values to only contain term values. We elude the problem of reducing terms, by defining meta-rules for them[9]. We add standard rules for delimited continuations, expressing the fact that when a proof $\mu\hat{\mathfrak{tp}}.c$ is in active position, the current context is temporarily frozen until c is fully reduced.

$$\begin{array}{rcll}
\langle \mu\alpha.c \| e \rangle & \rightsquigarrow & c[e/\alpha] & \\
\langle \lambda a.p \| q \cdot e \rangle & \rightsquigarrow & \langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle & (q \in \text{NEF}) \\
\langle \lambda a.p \| q \cdot e \rangle & \rightsquigarrow & \langle q \| \tilde{\mu}a.\langle p \| e \rangle \rangle & (q \notin \text{NEF}) \\
\langle \lambda x.p \| V_t \cdot e \rangle & \rightsquigarrow & \langle p[V_t/x] \| e \rangle & \\
\langle V_p \| \tilde{\mu}a.c \rangle & \rightsquigarrow & c[V_p/a] & \\
\langle (V_t, p) \| e \rangle & \rightsquigarrow & \langle p \| \tilde{\mu}a.\langle (V_t, a) \| e \rangle \rangle & (p \notin V) \\
\langle \mathsf{prf}\ p \| e \rangle & \rightsquigarrow & \langle \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle & \\
\langle \mathsf{prf}\ (V_t, V_p) \| e \rangle & \rightsquigarrow & \langle V \| e \rangle & \\
\langle \mathsf{subst}\ p\ q \| e \rangle & \rightsquigarrow & \langle p \| \tilde{\mu}a.\langle \mathsf{subst}\ a\ q \| e \rangle \rangle & (p \notin V) \\
\langle \mathsf{subst}\ \mathsf{refl}\ q \| e \rangle & \rightsquigarrow & \langle q \| e \rangle & \\
\langle \mu\hat{\mathfrak{tp}}.\langle p \| \hat{\mathfrak{tp}} \rangle \| e \rangle & \rightsquigarrow & \langle p \| e \rangle & \\
c \to c' & \Rightarrow & \langle \mu\hat{\mathfrak{tp}}.c \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.c' \| e \rangle & \\
\mathsf{wit}\ p \to t & \Leftarrow & \forall \alpha, \langle p \| \alpha \rangle \rightsquigarrow \langle (t, p') \| \alpha \rangle & \\
t \to t' & \Rightarrow & c[t] \rightsquigarrow c[t'] &
\end{array}$$

where

$$\begin{array}{rcl} V_t & ::= & x \mid n \\ V_p & ::= & a \mid \lambda a.p \mid \lambda x.p \mid (V_t, V_p) \mid \mathsf{refl} \end{array}$$

Figure 4. Reduction rules

2.2 Delimiting the scope of dependencies

For the typing rules, we can extend the set $\mathcal{D}$ to be the NEF fragment:

$$\mathcal{D} \triangleq \text{NEF}$$

and we now distinguish two modes. The regular mode corresponds to a derivation without dependency issues whose typing rules are the same as in Figure 3 without the dependencies list (we do not recall them to save some space); plus the new rule of introduction of a delimited continuation $\hat{\mathfrak{tp}}_I$. The dependent mode is used to type commands and contexts involving $\hat{\mathfrak{tp}}$, and we use the sign $\vdash_d$ to denote the sequents. There are three rules: one to type $\hat{\mathfrak{tp}}$, which is the only one where we use the dependencies to unify dependencies; one to type context of the form $\tilde{\mu}a.c$ (the rule is the same as the former rule for $\tilde{\mu}a.c$ in Section 1); and a last one to type commands $\langle p \| e \rangle$, where we observe that the premise for p is typed in regular mode.

[6] We will see in Section 3.3 that such a term could be typed by turning the type $A \to \bot$ of the continuation that $[\![q]\!]$ is waiting for into a (dependent) type $\Pi_{a:A} R[a]$ parameterized by R. This way we could have $[\![q]\!] : \forall R(\Pi_{a:A} R[a] \to R[q])$ instead of $[\![q]\!] : ((A \to \bot) \to \bot)$. For $R[a] := (B(a) \to \bot) \to \bot$, the whole term is well-typed. Readers familiar with realizability will also note that such a term is realizable, since it eventually terminates on a correct position $[\![p[q/a]]\!]\,[\![e]\!]$.

[7] Indeed, $\lambda a.p$ is a value for any p, hence proofs like $\mu\alpha.\langle \lambda a.p \| q \cdot \alpha \rangle$ can drop the continuation in the end once p becomes the proof in active position.

[8] We actually even consider α-conversion for delimited continuations $\hat{\mathfrak{tp}}$, to be able to insert such terms inside a type, even though it might seem strange it will make sense when proving subject reduction

[9] Everything works as if when reaching a state where the reduction of a term is needed, we had an extra abstract machine to reduce it. Note that this abstract machine could possibly need another machine itself, etc... We could actually solve this by making the reduction of terms explicit, introducing for instance commands and contexts for terms with the appropriate typing rules. However, this is not necessary from a logical point of view and it would significantly increase the complexity of the proofs, therefore we rather chose to stick to the actual presentation.

Proofs	p	$::= \cdots \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}}$
Delimited continuations	$c_{\hat{\mathfrak{tp}}}$	$::= \langle p_N \| e_{\hat{\mathfrak{tp}}} \rangle \mid \langle p \| \hat{\mathfrak{tp}} \rangle$
	$e_{\hat{\mathfrak{tp}}}$	$::= \tilde{\mu} a.c_{\hat{\mathfrak{tp}}}$
NEF	p_N	$::= V \mid (t, p_N) \mid \mu\star.c_N \mid \mathsf{prf}\, p_N \mid \mathsf{subst}\, p_N\, q_N$
	c_N	$::= \langle p_N \| e_N \rangle$
	e_N	$::= \star \mid \tilde{\mu} a.c_N$

(a) Language

$$\frac{c : (\Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : A; \varepsilon)}{\Gamma \vdash \mu\hat{\mathfrak{tp}}.c : A \mid \Delta}\ \hat{\mathfrak{tp}}_I \qquad \frac{\Gamma \vdash p : A \mid \Delta \quad \Gamma \mid e : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{\cdot|p\}}{\langle p \| e \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma}$$

$$\frac{B \in A_\sigma}{\Gamma \mid \hat{\mathfrak{tp}} : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{\cdot|p\}}\ \hat{\mathfrak{tp}}_E \qquad \frac{c : (\Gamma, a : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{a|p\})}{\Gamma \mid \tilde{\mu} a.c : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{\cdot|p\}}$$

(b) Typing rules

Figure 5. Extension of the system with delimited continuations

Additionally, we need to extend the congruence to make it compatible with the reduction of NEF proof terms (that can now appear in types), thus we add the rules:

$$\begin{array}{rcll} A[p] & \triangleright & A[q] & \text{if } \forall \alpha\, (\langle p \| \alpha \rangle \rightsquigarrow \langle q \| \alpha \rangle) \\ A[\langle q \| \tilde{\mu} a.\langle p \| \star \rangle \rangle] & \triangleright & A[\langle p[q/a] \| \star \rangle] & \text{with } p, q \in \text{NEF} \end{array}$$

Due to the presence of NEF proof terms (which contain a delimited form of control) within types and dependencies lists, we need the following technical lemma to prove subject reduction.

Lemma 11. *For any context Γ, Δ, any type A and any e, $\mu\star.c$, if:*

$$\langle \mu\star.c \| e \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \varepsilon$$

then:

$$c[e/\star] : \Gamma \vdash_d \Delta; \varepsilon.$$

Proof. A derivation for the hypothesis is of the form:

$$\frac{\dfrac{\Pi_c}{\Gamma \vdash \mu\star.c : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \{\cdot|\mu\star.c\}}}{\langle \mu\star.c \| e \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \varepsilon}$$

By definition of the NEF proof terms, $\mu\star.c$ is of the general form $\mu\star.c = \mu\star.\langle p_1 \| \tilde{\mu} a_1.\langle p_2 \| \tilde{\mu} a_2.\langle \ldots \| \tilde{\mu} a_n.\langle p_n \| \star \rangle \rangle \rangle \rangle$. For simplicity reasons, we will only give the proof for the case $n = 2$, so that Π_c is of the shape (we assume the CONV-rules have been pushed to the left of cuts):

$$\frac{\dfrac{\Pi_1}{\Gamma \vdash p_1 : A_1 \mid \Delta, \star : A} \quad \dfrac{\dfrac{\dfrac{\Pi_2}{\Gamma, a_1 : A_1 \vdash p_2 : A \mid \Delta, \star : A} \quad \dfrac{}{\cdots \mid \star : A \vdash \Delta, \star : A}}{\langle p_2 \| \star \rangle : \Gamma, a_1 : A_1 \vdash \Delta, \star : A}}{\Gamma \mid \tilde{\mu} a_1.\langle p_2 \| \star \rangle : A_1 \vdash \Delta, \star : A}}{\Gamma \vdash \mu\star.\langle p_1 \| \tilde{\mu} a_1.\langle p_2 \| \star \rangle \rangle : A \mid \Delta}$$

Thus we have to show that we can turn Π_e into a derivation Π'_e of $\Gamma \mid e : A \vdash_d \Delta_{\hat{\mathfrak{tp}}}; \{a_1|p_1\}\{\cdot|p_2\}$ with $\Delta_{\hat{\mathfrak{tp}}} \triangleq \Delta, \hat{\mathfrak{tp}} : B$, since this would allow us to build the following derivation:

$$\frac{\dfrac{\Pi_1}{\Gamma \vdash p_1 : A_1 \mid \Delta} \quad \dfrac{\dfrac{\dfrac{\Pi_2}{\Gamma, a_1 : A_1 \vdash p_2 : A \mid \Delta} \quad \dfrac{\Pi'_e}{\cdots \mid e : A \vdash_d \Delta_{\hat{\mathfrak{tp}}}; \{a_1|p_1\}\{\cdot|p_2\}}}{\langle p_2 \| \star \rangle : \Gamma, a_1 : A_1 \vdash \Delta_{\hat{\mathfrak{tp}}}; \{a_1|p_1\}}}{\Gamma \mid \tilde{\mu} a_1.\langle p_2 \| e \rangle : A_1 \vdash_d \Delta_{\hat{\mathfrak{tp}}}; \{\cdot|p_1\}}}{\langle p_1 \| \tilde{\mu} a_1.\langle p_2 \| e \rangle \rangle : \Gamma \vdash_d \Delta_{\hat{\mathfrak{tp}}}; \varepsilon}$$

It suffices to prove that if the dependencies list was used in Π_e to type $\hat{\mathfrak{tp}}$, we can still give a derivation with the new one. In practice, it corresponds to showing that for any variable a and any σ:

$$\{a|\mu\star.c\}\sigma \Rrightarrow \{a_1|p_1\}\{a|p_2\}\sigma.$$

For any $A \in B_\sigma$, by definition we have:

$$\begin{aligned} A[\mu\star.\langle p_1 \| \tilde{\mu} a_1.\langle p_2 \| \star \rangle \rangle / b] &\equiv A[\mu\star.\langle p_2[p_1/a_1] \| \star \rangle / b] \\ &\equiv A[p_2[p_1/a_1]/b] = A[p_2/b][p_1/a_1]. \end{aligned}$$

Hence for any $A \in B_{\{a|\mu\star.c\}\sigma}$, there exists $A' \in B_{\{a_1|p_1\}\{a|p_2\}\sigma}$ such that $A \equiv A'$, and we can derive:

$$\frac{\dfrac{A' \in B_{\{a_1|p_1\}\{a|p_2\}\sigma}}{\Gamma \mid \hat{\mathfrak{tp}} : A' \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \{a_1|p_1\}\{b|p_2\}\sigma} \quad A \equiv A'}{\Gamma \mid \hat{\mathfrak{tp}} : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \{a_1|p_1\}\{b|p_2\}\sigma}$$

□

We can now prove subject reduction for the extended version of dL.

Theorem 12 (Subject reduction)**.** *If $c : (\Gamma \vdash \Delta)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta)$.*

Proof. Actually, the proof is slightly easier than for Theorem 7, because most of the rules do not involve dependencies. We only present one case here, other key cases are proved in the appendix.

- **Case** $\langle \lambda a.p \| q \cdot e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$ with $q \in \text{NEF}$.
  A typing derivation for the command on the left is of the form:

$$\frac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta} \quad \dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta}}{\langle \lambda a.p \| q \cdot e \rangle : \Gamma \vdash \Delta}$$

  We can thus build the following derivation for the command on the right:

$$\frac{\dfrac{\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_p}{\Gamma, a : A \vdash p : B[a] \mid \Delta}}{\langle q \| \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B[q]; \varepsilon}}{\Gamma \vdash \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle : \Gamma \vdash \Delta}$$

  $\Pi_p =$

$$\frac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B[a] \mid \Delta} \quad \dfrac{B[q] \in (B[a])_{\{a|q\}}}{\Gamma \mid \hat{\mathfrak{tp}} : B[a] \vdash_d \Delta, \hat{\mathfrak{tp}} : B[q]; \{a|q\}\{\cdot|\dagger\}}}{\dfrac{\langle p \| \hat{\mathfrak{tp}} \rangle : \Gamma, a : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B[q]; \{a|q\}}{\Gamma \mid \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}} \rangle : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B[q]; \{\cdot|q\}}}$$

- **Case** $\langle \mathsf{prf}\, p \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu} a.\langle \mathsf{prf}\, a \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$.
  We prove it in the most general case, that is when this reduction occurs under a delimited continuation. A typing derivation for the command on the left has to be of the form:

$$\frac{\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : \exists x.A(x) \mid \Delta}}{\Gamma \vdash \mathsf{prf}\, p : A(\mathsf{wit}\, p) \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\, p) \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{\cdot|\mathsf{prf}\, p\}}}{\langle \mathsf{prf}\, p \| e \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma}$$

  The proof p being NEF, so is $\mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu} a.\langle \mathsf{prf}\, a \| \hat{\mathfrak{tp}} \rangle \rangle$, and by definition of the reduction for types, we have for any type A

that:

$$A[\mathsf{prf}\ p] \triangleright A[\mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle\rangle],$$

so that we can prove that for any b:

$$\sigma\{b|\mathsf{prf}\ p\} \Rrightarrow \sigma\{b|\mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle\rangle\}.$$

Thus we can turn Π_e into Π'_e a derivation of the same sequent except for the dependencies lists that is changed to $\sigma\{\cdot|\mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle\rangle\}$. We conclude the proof of this case by giving the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : \exists x.A(x) \mid \Delta}}{\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle\rangle \Gamma \vdash_d \mid \Delta, \hat{\mathfrak{tp}} : A(\mathsf{wit}\ p); \varepsilon} \quad \Pi_{\hat{\mathfrak{tp}}}}{\Gamma \vdash \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle\rangle : A(\mathsf{wit}\ p) \mid \Delta}$$

with $\Pi_{\hat{\mathfrak{tp}}}$ the following derivation where we removed Γ and Δ when irrelevant:

$$\dfrac{\dfrac{\dfrac{}{a : \exists x.A \vdash a : \exists x.A}}{a : \exists x.A \vdash \mathsf{prf}\ a : A(\mathsf{wit}\ a)} \quad \dfrac{A(\mathsf{wit}\ p) \in (A(\mathsf{wit}\ a))_{\{a|p\}}}{\hat{\mathfrak{tp}} : A(\mathsf{wit}\ a) \vdash_d \hat{\mathfrak{tp}} : A(\mathsf{wit}\ p); \{a|p\}}}{\dfrac{\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle : \Gamma, a : \exists x.A(x) \vdash_d \Delta, \hat{\mathfrak{tp}} : A(\mathsf{wit}\ p); \{a|p\}}{\Gamma \mid \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{tp}}\rangle : \exists x.A(x) \vdash_d \Delta, \hat{\mathfrak{tp}} : A(\mathsf{wit}\ p); \{\cdot|p\}}}$$

□

3. A continuation-passing style translation

We shall now see how to define a continuation-passing style translation from dL to an intuitionistic type theory, and use this translation to prove the soundness of dL. Indeed, continuation-passing style translations are very useful to embed a language with a control operator into another one that is purely functional [5, 10]. From a logical point of view, they generally amount to negative translations that allow to embed classical logic into intuitionistic logic [7]. Yet, we know that removing the control operator (*i.e.* classical logic) of our language leaves us with a sound intuitionistic type theory. We will now see how to design a CPS translation for our language which will allow us to prove its soundness.

3.1 Target language

We choose as the target language an intuitionistic theory in natural deduction that has exactly the same elements as dL but the control operator: the language makes the difference between terms (of type $\mathbb{N}$) and proofs, it also includes dependent sums and products for type referring to term as well as a dependent product at the level of proofs. As it is common for CPS translations, the evaluation follows a head-reduction strategy. The syntax of the language and its reduction rules are given by Figure 7.

The type system, presented in Figure 6 is defined as expected, with the addition of a second-order quantification that we will use in the sequel to refine the type of the translation of terms and NEF proofs. As for dL the type system has a conversion rule, where the relation $A \equiv B$ is the symmetric-transitive closure of $A \triangleright B$, defined once again as the congruence over the reduction $\longrightarrow$ and by the rules:

$$\begin{array}{rlrl} 0 = 0 & \triangleright \top & 0 = S(u) & \triangleright \bot \\ S(t) = 0 & \triangleright \bot & S(t) = S(u) & \triangleright\ t = u. \end{array}$$

3.2 Translation of the terms

We can now define the translation of terms, proofs, contexts and commands. The translation for delimited continuation follows the intuition we presented in Section 1.6, and the definition for stacks $t \cdot e$ and $q \cdot e$ with q NEF inline the reduction producing a command

Terms	t	$::=$	$x \mid \bar{n} \mid \mathsf{wit}\ p \qquad (n \in \mathbb{N})$
Proofs	p	$::=$	$a \mid \lambda a.p \mid \lambda x.p \mid p\,q \mid p\,t$
			$\mid (t,p) \mid \mathsf{prf}\ p \mid \mathsf{refl} \mid \mathsf{subst}\ p\,q$
Formulas	A, B	$::=$	$\top \mid \bot \mid t = u \mid \Pi_{a:A} B$
			$\mid \forall x^T A \mid \exists x^T A \mid \forall X.A$

(a) Language

$$\begin{array}{rcll} (\lambda x.p)\,t & \longrightarrow & p[t/x] & \\ (\lambda a.p)\,q & \longrightarrow & p[q/a] & \\ p\,q & \longrightarrow & p'\,q & (\text{if } p \longrightarrow p') \\ k(\mathsf{wit}\ (t,p)) & \longrightarrow & k\,t & \\ \mathsf{prf}\ (t,p) & \longrightarrow & p & \\ \mathsf{subst}\ \mathsf{refl}\ q & \longrightarrow & q & \end{array}$$

(b) Reduction rules

Figure 7. Target language

with a delimited continuation. All the other rules are natural, except for the translation of pairs (t,p) in the NEF case:

$$[\![(t,p)]\!]_p \triangleq \lambda k.([\![t]\!]_t\ (\lambda ur.r\ (\lambda q.k\ (u,q))))[\![p]\!]_p$$

The natural definition is the one given in the non NEF case, but as we will see in the proof of Lemma 16, this definition is incompatible with the expected type for the translation of NEF proofs. This somehow strange definition corresponds to the intuition that we reduce $[\![t]\!]_t$ within a delimited continuation, in order to guarantee that we will not reduce $[\![p]\!]_p$ before $[\![t]\!]_t$ has returned a value to substitute for u. Indeed, the type of $[\![p]\!]_p$ depends on t, while the continuation $(\lambda q.k\,(u,q))$ depends on u, but both become compatible once u is substituted by the value return by $[\![t]\!]_t$. The complete translation is given in Figure 8.

$$\begin{array}{lll}
[\![x]\!]_t & \triangleq \lambda k.k\,x & \\
[\![n]\!]_t & \triangleq \lambda k.k\,\bar{n} & \\
[\![\mathsf{wit}\ p]\!]_t & \triangleq \lambda k.[\![p]\!]_p\ (\lambda q.k\ (\mathsf{wit}\ q)) & \\
[\![a]\!]_p & \triangleq \lambda k.k\,a & \\
[\![\lambda a.p]\!]_p & \triangleq \lambda k.k\ (\lambda a.[\![p]\!]_p) & \\
[\![\lambda x.p]\!]_p & \triangleq \lambda k.k\ (\lambda x.[\![p]\!]_p) & \\
[\![\mu\alpha.c]\!]_p & \triangleq \lambda k.[\![c]\!]_c[k/\alpha] & \\
[\![(t,p)]\!]_p & \triangleq \lambda k.([\![t]\!]_t\ (\lambda ur.r\ (\lambda q.k\ (u,q))))[\![p]\!]_p & (p \in \text{NEF}) \\
[\![(t,p)]\!]_p & \triangleq \lambda k.[\![t]\!]_t\ (\lambda u.[\![p]\!]_p\ \lambda q.k\ (u,q)) & (p \notin \text{NEF}) \\
[\![\mathsf{prf}\ p]\!]_p & \triangleq \lambda k.([\![p]\!]_p\ (\lambda q k'.k'\ (\mathsf{prf}\ q)))\ k & \\
[\![\mathsf{refl}]\!]_p & \triangleq \lambda k.k\ \mathsf{refl} & \\
[\![\mathsf{subst}\ p\,q]\!]_p & \triangleq\ \lambda k.[\![p]\!]_p\ (\lambda p'.[\![q]\!]_p(\lambda q'.k\ (\mathsf{subst}\ p'\ q'))) & \\
[\![\mu\hat{\mathfrak{tp}}.c]\!]_p & \triangleq \lambda k.[\![c]\!]_{\hat{\mathfrak{tp}}}k & \\
[\![\alpha]\!]_e & \triangleq \lambda p.\alpha\,p & \\
[\![q_N \cdot e]\!]_e & \triangleq \lambda p.([\![q_N]\!]_p\ (\lambda v.p\,v))\ [\![e]\!]_e & (q_N \in \text{NEF}) \\
[\![q \cdot e]\!]_e & \triangleq \lambda p.[\![q]\!]_p\ (\lambda v.p\,v\ [\![e]\!]_e) & (q \notin \text{NEF}) \\
[\![t \cdot e]\!]_e & \triangleq \lambda p.([\![t]\!]_t\ (\lambda v.p\,v))\ [\![e]\!]_e & \\
[\![\tilde{\mu}a.c]\!]_e & \triangleq \lambda a.[\![c]\!]_c & \\
[\![\langle p \| e\rangle]\!]_c & \triangleq [\![e]\!]_e\ [\![p]\!]_p & \\
[\![\langle p \| \hat{\mathfrak{tp}}\rangle]\!]_{\hat{\mathfrak{tp}}} & \triangleq [\![p]\!]_p & \\
[\![\langle p \| e\rangle]\!]_{\hat{\mathfrak{tp}}} & \triangleq [\![p]\!]_p\ [\![e]\!]_{e_{\hat{\mathfrak{tp}}}} & (e \neq \hat{\mathfrak{tp}}) \\
[\![\tilde{\mu}a.c]\!]_{e_{\hat{\mathfrak{tp}}}} & \triangleq \lambda a.[\![c]\!]_{\hat{\mathfrak{tp}}} &
\end{array}$$

Figure 8. Continuation-passing style translation

Before defining the translation of types, we first state a lemma expressing the fact that the translations of terms and NEF proof

$$\frac{}{\Gamma \vdash \bar{n} : \mathbb{N}}\ \mathrm{Ax}_n \qquad \frac{(x:T)\in\Gamma}{\Gamma\vdash x:T}\ \mathrm{Ax}_t \qquad \frac{(a:A)\in\Gamma}{\Gamma\vdash a:A}\ \mathrm{Ax}_p \qquad \frac{\Gamma, a:A\vdash p:B}{\Gamma\vdash \lambda a.p:\Pi_{a:A}B}\ \to_i \qquad \frac{\Gamma\vdash p:\Pi_{a:A}B \quad \Gamma\vdash q:A}{\Gamma\vdash p\,q:B[q/a]}\ \to_E$$

$$\frac{\Gamma, x:T\vdash p:A}{\Gamma\vdash \lambda x.p:\forall x^T A}\ \forall_i \qquad \frac{\Gamma\vdash p:\forall x^T A \quad \Gamma\vdash t:T}{\Gamma\vdash p\,t:A[t/x]}\ \forall_e \qquad \frac{\Gamma\vdash p:A \quad X\notin FV(\Gamma)}{\Gamma\vdash p:\forall X.A}\ \forall_I \qquad \frac{\Gamma\vdash p:\forall X.A}{\Gamma\vdash p:A[P/X]}\ \forall_E$$

$$\frac{\Gamma\vdash t:T \quad \Gamma\vdash p:A[u/x]}{\Gamma\vdash (t,p):\exists x^T A}\ \exists_i \qquad \frac{\Gamma\vdash p:\exists x^T A}{\Gamma\vdash \mathsf{prf}\,p:A(\mathsf{wit}\,p)}\ \mathsf{prf} \qquad \frac{\Gamma\vdash p:\exists x^T A}{\Gamma\vdash \mathsf{wit}\,p:T}\ \mathsf{wit}$$

$$\frac{}{\Gamma\vdash \mathsf{refl}:x=x}\ \mathsf{refl} \qquad \frac{\Gamma\vdash q:t=u \quad \Gamma\vdash q:A[t]}{\Gamma\vdash \mathsf{subst}\,p\,q:A[u]}\ \mathsf{subst} \qquad \frac{\Gamma\vdash p:A \quad A\equiv B}{\Gamma\vdash p:B}\ \textsc{Conv}$$

Figure 6. Typing rules of the target language

terms use the continuation they are given once and only once. In particular, this makes them compatible with delimited continuations and a parametric return type. This will allow us to refine the type of their translation.

Lemma 13. *The translation satisfies the following properties:*

1. *For any term t in* dL*, there exists a term t^+ such that for any k we have $[\![t]\!]_t\, k =_\beta k\, t^+$.*
2. *For any* NEF *proof term p, there exists a proof p^+ such that for any k we have $[\![p]\!]_p\, k =_\beta k\, p^+$.*

Proof. Straightforward mutual induction on the translation, adding similar induction hypothesis for NEF contexts and commands. The terms t^+ and proofs p^+ are given in Figure 9. We detail the case (t,p) with $p\in$ NEF to give an insight of the proof.

$$\begin{aligned}
[\![(t,p)]\!]_p\, k &=_\beta ([\![t]\!]_t\, (\lambda u r.r\, (\lambda q.k\, (u,q))))\, [\![p]\!]_p && \text{(by def.)}\\
&=_\beta ((\lambda u r.r\, (\lambda q.k\, (u,q)))\, t^+)\, [\![p]\!]_p && \text{(by induction)}\\
&=_\beta [\![p]\!]_p\, (\lambda q.k\, (t^+, q))\\
&=_\beta (\lambda q.k\, (t^+,q))\, p^+ && \text{(by induction)}\\
&=_\beta k\, (t^+, p^+)
\end{aligned}$$

□

x^+	$\triangleq x$	refl^+	$\triangleq \mathsf{refl}$
n^+	$\triangleq \bar{n}$	$(\mathsf{subst}\,p\,q)^+$	$\triangleq \mathsf{subst}\,p^+\,q^+$
$(\mathsf{wit}\,p)^+$	$\triangleq \mathsf{wit}\,p^+$	$(\mu\star.c)^+$	$\triangleq c^+$
a^+	$\triangleq a$	$(\mu\hat{\mathfrak{tp}}.c)^+$	$\triangleq c^+$
$(\lambda a.p)^+$	$\triangleq \lambda a.[\![p]\!]_p$	$(\langle p \| \star\rangle)^+$	$\triangleq p^+$
$(\lambda x.p)^+$	$\triangleq \lambda x.[\![p]\!]_p$	$(\langle p \| \hat{\mathfrak{tp}}\rangle)^+$	$\triangleq p^+$
$(t,p)^+$	$\triangleq (t^+, p^+)$	$(\langle p \| \tilde{\mu} a.c\rangle)^+$	$\triangleq c^+[p^+/a]$
$(\mathsf{prf}\,p)^+$	$\triangleq \mathsf{prf}\,p^+$		

Figure 9. Linearity of the translation for NEF proofs

Moreover, we easily verify that the translation preserves the reduction.

Proposition 14 (Preservation of reduction)**.** *Let c, c' be two commands of* dL*. If $c \rightsquigarrow c'$, then $[\![c]\!]_c =_\beta [\![c']\!]_c$*

Proof. By induction on the reduction rules for $\rightsquigarrow$, using the previous lemma for cases involving a term t. □

We could actually prove a finer result to show that any reduction step in dL is responsible for at least one step of reduction through the translation, and use the preservation of typing (Proposition 17) together with the normalization of the target language to prove the normalization of dL for typed proof terms.

Claim 15. *If $c : \Gamma \vdash \Delta$, then c normalizes.*

3.3 Translation of types

We can now define the translation of types, in order to show further that the translation $[\![p]\!]_p$ of a proof p of type A is of type $[\![A]\!]^*$, where $[\![A]\!]^*$ is the double-negation of a type $[\![A]\!]^+$ that depends on the structure of A. Thanks to the restriction of dependent types to NEF proof terms, we can interpret a dependency in p (resp. t) in dL by a dependency in p^+ (resp. t^+) in the target language. Lemma 13 indeed guarantees that the translation of a NEF proof p will eventually return p^+ to the continuation it is applied to. The translation is defined in Figure 3.3. Observe that types depending on a term of type T are translated to types depending on a term of the same type T, because terms can only be of type $\mathbb{N}$. As we shall discuss in Section 5.2, this will no longer be the case when extending the domain of terms.

$[\![A]\!]^*$	$\triangleq ([\![A]\!]^+ \to \bot) \to \bot$	$[\![t=u]\!]^+$	$\triangleq t^+ = u^+$
$[\![\forall x^T.A]\!]^+$	$\triangleq \forall x^{T^+}.[\![A]\!]^*$	$[\![\top]\!]^+$	$\triangleq \top$
$[\![\exists x^T.A]\!]^+$	$\triangleq \exists x^{T^+}.[\![A]\!]^+$	$[\![\bot]\!]^+$	$\triangleq \bot$
$[\![\Pi_{a:A}B]\!]^+$	$\triangleq \Pi_{a:[\![A]\!]^+}[\![B]\!]^*$	$\mathbb{N}^+$	$\triangleq \mathbb{N}$

Figure 10. Translation of types

We naturally extend the translation for types to the translation of contexts, where we consider unified contexts of the form $\Gamma\cup\Delta$:

$$\begin{aligned}
[\![\Gamma, a:A]\!]^+ &\triangleq [\![\Gamma]\!]^+, a:[\![A]\!]^+\\
[\![\Gamma, x:T]\!]^+ &\triangleq [\![\Gamma]\!]^+, x:T^+\\
[\![\Gamma, \alpha:A^{\perp\!\!\!\perp}]\!]^+ &\triangleq [\![\Gamma]\!]^+, \alpha:[\![A]\!]^+\to\bot.
\end{aligned}$$

As explained informally in Section 1.6 and stated by Lemma 13, the translation of a NEF proof term p of type A uses its continuation linearly. In particular, this allows us to refine its type to make it parametric in the return type of the continuation. From a logical point of view, it amounts to replace the double-negation $(A\to\bot)\to\bot$ by Friedman's translation [8]: $\forall R.(A\to R)\to R$. Moreover, we can even make the return type of the continuation dependent on its argument $\Pi_{a:A}{\to}R(a)$, so that the type of $[\![p]\!]_p$ will correspond to the elimination rule:

$$\forall R.(\Pi_{a:A}{\to}R(a))\to R(p^+).$$

This refinement will make the translation of NEF proofs compatible with the translation of delimited continuations.

Lemma 16. *The following holds:*

1. *If* $\Gamma \vdash t : T \mid \Delta$ *then:*
$$[\![\Gamma \cup \Delta]\!] \vdash [\![t]\!]_t : \forall X(\forall x^{T^+} X(x) \to X(t^+)).$$
2. *If* $p \in$ NEF *and* $\Gamma \vdash p : A \mid \Delta$ *then:*
$$[\![\Gamma \cup \Delta]\!] \vdash [\![p]\!]_p : \forall X.(\Pi_{a:[\![A]\!]^+} X(a) \to X(p^+)).$$
3. *If* $c \in$ NEF *and* $c : \Gamma \vdash \Delta, \star : B$ *then:*
$$[\![\Gamma \cup \Delta]\!], \star : \Pi_{b:B^+} X(b) \vdash [\![c]\!]_c : X(c^+).$$

Proof. The proof is done by mutual induction on the typing rule of dL for terms and NEF proof terms. We only give one case to give an insight of the proof, other key cases are given in the appendix.

- **Case** (t, p)**.**
In dL the typing rule for (t, p) is the following:
$$\frac{\Gamma \vdash t : T \mid \Delta \quad \Gamma \vdash p : A(t) \mid \Delta}{\Gamma \vdash (t, p) : \exists x^T A(x) \mid \Delta}\ \exists_i$$
Hence by induction we obtain, using the same notation Γ' for $[\![\Gamma \cup \Delta]\!]$:
$$\Gamma' \vdash [\![t]\!]_t : \forall X.(\forall x^{T^+} X(x) \to X(t^+))$$
$$\Gamma' \vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A(t^+)} Y(a) \to Y(p^+))$$
and we want to show that for any Z:
$$\Gamma' \vdash [\![(t, p)]\!]_p : \Pi_{a:\exists x^{T^+} A} Z(a) \to Z(t^+, p^+).$$
By definition, we have:
$$[\![(t, p)]\!]_p = \lambda k.([\![t]\!]_t\,(\lambda ur.r\,(\lambda q.k\,(u, q))))[\![p]\!]_p,$$
so we need to prove that:
$$\Gamma_k \vdash ([\![t]\!]_t\,(\lambda ur.r\,(\lambda q.k\,(u, q))))[\![p]\!]_p : Z(t^+, p^+)$$
where $\Gamma_k = \Gamma', k : \Pi_{a:\exists x^{T^+} A} Z(a)$. We let the reader check that such a type is derivable by using $X(x) \triangleq P(x) \to Z(x, a)$ in the type of $[\![t]\!]_p$ where $P(t^+)$ is the type of $[\![p]\!]_p$, and using $Y(a) \triangleq Z(t^+, a)$ in the type of $[\![p]\!]_p$. The crucial point is to see that the bounded variable r is abstracted with type $P(x)$ in the derivation, which would not have been possible in the definition of $[\![(t, p)]\!]_p$ with $p \notin$ NEF.

□

Using the previous Lemma, we can now prove that the CPS translation is well-typed in the general case.

Proposition 17 (Preservation of typing)**.** *The translation is well-typed, i.e. the following holds:*

1. *if* $\Gamma \vdash p : A \mid \Delta$ *then* $[\![\Gamma \cup \Delta]\!] \vdash [\![p]\!]_p : [\![A]\!]^*$,
2. *if* $\Gamma \mid e : A \vdash \Delta$ *then* $[\![\Gamma \cup \Delta]\!] \vdash [\![e]\!]_e : [\![A]\!]^+ \to \bot$,
3. *if* $c : \Gamma \vdash \Delta$ *then* $[\![\Gamma \cup \Delta]\!] \vdash [\![c]\!]_c : \bot$.

Proof. The proof is done by induction on the typing rules of dL. It is clear that for the NEF cases, Lemma 16 implies the result by taking $X(a) = \bot$. The rest of the cases are straightforward, except for the delimited continuations that we detail hereafter. We consider a command $\langle \mu\hat{\mathfrak{p}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}} \rangle \rangle \| e \rangle$ produced by the reduction of the command $\langle \lambda a.p \| q \cdot e \rangle$ with $q \in$ NEF. Both commands are translated by a proof reducing to $([\![q]\!]_p\,(\lambda a.[\![p]\!]_p))\,[\![e]\!]_e$. The corresponding typing derivation in dL is of the form:
$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta} \quad \dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta}}{\langle \lambda a.p \| q \cdot e \rangle : \Gamma \vdash \Delta}$$
By induction hypothesis for e and p we obtain:
$$\begin{array}{rl} \Gamma' & \vdash [\![e]\!]_e : [\![B[q^+]]\!]^+ \to \bot \\ \Gamma', a : A^+ & \vdash [\![p]\!]_p : [\![B[a]]\!]^* \\ \Gamma' & \vdash \lambda a.[\![p]\!]_p : \Pi_{a:A^+} [\![B[a]]\!]^*, \end{array}$$
where $\Gamma' = [\![\Gamma \cup \Delta]\!]$. Applying Lemma 16 for $q \in$ NEF we can derive:
$$\frac{\Gamma' \vdash [\![q]\!]_p : \forall X.(\Pi_{a:A^+} X(a) \to X(q^+))}{\Gamma' \vdash [\![q]\!]_p : (\Pi_{a:A^+} [\![B[a]]\!]^* \to [\![B[q^+]]\!]^*}\ \forall_E$$
We can thus build derive that:
$$\Gamma' \vdash [\![q]\!]_p\,(\lambda a.[\![p]\!]_p) : [\![B[q^+]]\!]^*,$$
and finally conclude that:
$$\Gamma' \vdash ([\![q]\!]_p\,(\lambda a.[\![p]\!]_p))\,[\![e]\!]_e : \bot.$$

□

We can finally deduce the correctness of dL through the translation, since a closed proof term of type $\bot$ would be translated in a closed proof of $(\bot \to \bot) \to \bot$. The correctness of the target language guarantees that such proof cannot exist.

Theorem 18 (Soundness)**.** *For any* $p \in$ dL*, we have:* $\not\vdash p : \bot$.

4. Embedding in Lepigre's calculus

In a recent paper [17], Lepigre presented a classical system allowing the use of dependent types with a semantic value restriction. In practice, the type system of his calculus does not contain a dependent product $\Pi_{a:A} B$ strictly speaking, but it contains a predicate $a \in A$ allowing the decomposition of the dependent product into
$$\forall a((a \in A) \to B)$$
as it is usual in Krivine's classical realizability [16]. In his system, the relativization $a \in A$ is restricted to values, so that we can only type $V : V \in A$, but the typing judgments are defined up to observational equivalence, so that if t is observationally equivalent to V, one can derive the judgment $t : V \in A$.

Interestingly, as highlighted through the CPS translation by Lemma 13, any NEF proof $p : A$ is observationally equivalent to some value p^+, so that we could derive $p : (p \in A)$ from $p^+ : (p^+ \in A)$. The NEF fragment is thus compatible with the semantical value restriction. The converse is obviously false, since observational equivalence allows us to type realizers that would otherwise be untyped terms.

We will sketch here an embedding of dL into Lepigre's calculus, and explain how to transfer normalization and correctness properties along this translation. Actually, his language is much more expressive than ours, since it contains records and pattern-matching (we will only use pairs, *i.e.* records with two fields), but it is not stratified: no distinction is made between a language of terms and a language of proofs.

We only recall here the syntax for the fragment of Lepigre's calculus we will use; for the reduction rules and the type system

the reader should refer to [17].

$$\begin{array}{rcl} v, w & ::= & x \mid \lambda x.t \mid \{l_1 = v_1, l_2 = v_2\} \\ t, u & ::= & a \mid v \mid t\,u \mid \mu\alpha.t \mid p \mid v.l_i \\ \pi, \rho & ::= & \alpha \mid v \cdot \pi \mid [t]\pi \\ p, q & ::= & t * \pi \\ A, B & ::= & X_n(t_1, \ldots, t_n) \mid A \to B \mid \forall a.A \mid \exists a.A \\ & \mid & \forall X_n.A \mid \{l_1 : A_1, l_2 : A_2\} \mid t \in A \end{array}$$

Even though records are only defined for values, we can define pairs and projections as syntactic sugar:

$$\begin{array}{lcl} (p_1, p_2) & \triangleq & (\lambda v_1 v_2.\{l_1 = v_1, l_2 = v_2\})\, p_1\, p_2 \\ \pi_i(p) & \triangleq & (\lambda x.(x.l_i))\, p \\ A_1 \wedge A_2 & \triangleq & \{l_1 : A_1, l_2 : A_2\} \end{array}$$

We first define the translation for types (extended for typing contexts) in Figure 11, where $[\![t]\!]_t$ is the translation of the term t defined thereafter.

$$\begin{array}{rcl} (\forall x^{\mathbb{N}}.A)^* & \triangleq & \forall x(\mathsf{Nat}(x) \to A^*) \\ (\exists x^{\mathbb{N}}.A)^* & \triangleq & \exists x(\mathsf{Nat}(x) \wedge A^*) \\ (t = u)^* & \triangleq & \forall X(Xt^* \to Xu^*) \\ \top^* & \triangleq & \forall X(X \to X) \\ \bot^* & \triangleq & \forall XY(X \to Y) \\ (\Pi_{a:A} B)^* & \triangleq & \forall a((a \in A^*) \to B^*) \\ (\Gamma, x : \mathbb{N})^* & \triangleq & \Gamma^*, x : \mathsf{Nat}(x) \\ (\Gamma, a : A)^* & \triangleq & \Gamma^*, a : A^* \\ (\Gamma, \alpha : A^{\perp\!\!\!\perp})^* & \triangleq & \Gamma^*, \alpha : \neg A^* \end{array}$$

Figure 11. Translation of types

The predicate $\mathsf{Nat}(x)$ is defined as usual in second-order logic:

$$\mathsf{Nat}(x) \triangleq \forall X(X(0) \to \forall y(X(y) \to X(S(y))) \to X(x))$$

Note that the equality is mapped to Leibniz equality, and that the definitions of $\bot^*$ and $\top^*$ are completely ad hoc, in order to make the conversion rule admissible through the translation.

We then give the definitions of the translation for terms, proofs, contexts and commands of dL. The translation, given in Figure 12 is pretty straightforward, we only want to draw the reader's attention on a few points:

- the equality is being translated as Leibniz equality, refl is translated as the identity $\lambda a.a$, which also matches with $\top^*$,
- the strong existential is encoded as a pair, hence wit (resp. prf) is mapped to the projection π_1 (resp. π_2).

$$\begin{array}{lll|lll} [\![x]\!]_t & \triangleq & x & [\![\mathsf{refl}]\!]_p & \triangleq & \lambda a.a \\ [\![n]\!]_t & \triangleq & \lambda z s.s^n(z) & [\![\mathsf{subst}\, p\, q]\!]_p & \triangleq & [\![p]\!]_p\, [\![q]\!]_p \\ [\![\mathsf{wit}\, p]\!]_t & \triangleq & \pi_1([\![p]\!]_p) & [\![\alpha]\!]_e & \triangleq & \alpha \\ [\![a]\!]_p & \triangleq & a & [\![q \cdot e]\!]_e & \triangleq & [\![q]\!]_p \cdot [\![e]\!]_e \\ [\![\lambda a.p]\!]_p & \triangleq & \lambda a.[\![p]\!]_p & [\![t \cdot e]\!]_e & \triangleq & [\![t]\!]_t \cdot [\![e]\!]_e \\ [\![\lambda x.p]\!]_p & \triangleq & \lambda x.[\![p]\!]_p & [\![\tilde{\mu} a.c]\!]_e & \triangleq & [\lambda a.[\![c]\!]_c]\star \\ [\![(t,p)]\!]_p & \triangleq & ([\![t]\!]_t, [\![p]\!]_p) & [\![\langle p \| e \rangle]\!]_c & \triangleq & [\![p]\!]_p * [\![e]\!]_e \\ [\![\mu\alpha.c]\!]_p & \triangleq & \mu\alpha.[\![c]\!]_c & [\![\mu\hat{\mathfrak{p}}.c]\!]_p & \triangleq & \mu\alpha.[\![c]\!]_{\hat{\mathfrak{p}}} \\ [\![\mathsf{prf}\, p]\!]_p & \triangleq & \pi_2([\![p]\!]_p) & [\![\langle p \| \hat{\mathfrak{p}} \rangle]\!]_{\hat{\mathfrak{p}}} & \triangleq & [\![p]\!]_p * \alpha \end{array}$$

$$[\![\langle p \| \tilde{\mu} a.c \rangle]\!]_{\hat{\mathfrak{p}}} \triangleq (\mu\alpha.[\![p]\!]_p * [\lambda a.[\![c]\!]_{\hat{\mathfrak{p}}}]\alpha) * \alpha$$

Figure 12. Translation of proof terms

In [17], the coherence of the system is justified by a realizability model, and the type system does not allow us to type stacks. Thus we cannot formally prove that the translation preserves the typing, unless we extend the type system[10] in which case this would imply the adequacy. We might also directly prove the adequacy of the realizability model (through the translation) with respect to the typing rules of dL. We detail a proof of adequacy using the first technique in the appendix.

Proposition 19 (Adequacy)**.** *If* $\Gamma \vdash p : A \mid \Delta$ *and* σ *is a substitution realizing* $(\Gamma \cup \Delta)^*$*, then* $[\![p]\!]_p \sigma \in [\![A^*]\!]_\sigma^{\perp\perp}$.

This immediately implies the soundness of dL, since a closed proof p of type $\bot$ would be translated as a realizer of $\top \to \bot$, so that $[\![p]\!]_p\, \lambda x.x$ would be a realizer of $\bot$, which is impossible. Furthermore, the translation clearly preserves normalization (that is that for any c, if c does not normalize then neither does $[\![c]\!]_c$), and thus the normalization of dL is consequence of adequacy.

It is worth noting that without delimited continuations, we would not have been able to define an adequate translation, since we would have encountered the same problem as for the CPS translation (see Section 1.6).

5. Further extensions

As we explained in the preamble of Section 1, we defined dL as the minimal language containing all the potential sources of inconsistency we wanted to mix: a control operator, dependent types, and a sequent calculus presentation. It had the benefit to focus our attention on the difficulties inherent to the issue, but on the other hand, the language we obtain is far from being as expressive as other usual proof systems. We claimed our system to be extensible, thus we shall now discuss this matter.

5.1 Adding expressiveness

From the point of view of the proof language (that is of the tools we have to build proofs), dL only enjoys the presence of a dependent sum and a dependent product over terms, as well as a dependent product at the level of proofs (which subsume the non-dependent implication). If this is obviously enough to encode the usual constructors for pairs (p_1, p_2) (of type $A_1 \wedge A_2$), injections $\iota_i(p)$ (of type $A_1 \vee A_2$), etc..., it seems reasonable to wonder whether such constructors can be directly defined in the language of proofs. Actually this is a case, and we claim that is possible to define the constructors for proofs (for instance (p_1, p_2)) together with their destructors in the contexts (in that case $\tilde{\mu}(a_1, a_2).c$), with the appropriate typing rules. In practice, it is enough to:

- extend the definitions of the NEF fragment according to the chosen extension,
- extend the call-by-value reduction system, opening if needed the constructors to reduce it to a value,
- in the dependent typing mode, make some pattern-matching within the dependencies list for the destructors. For instance, for the case of the pairs, the corresponding rule would be:

$$\frac{c : \Gamma, a_1 : A_1, a_2 : A_2 \vdash_d \Delta, \hat{\mathfrak{p}} : B; \sigma\{(a_1, a_2)|p\}}{\Gamma \mid \tilde{\mu}(a_1, a_2).c : (A_1 \wedge A_2) \vdash_d \Delta, \hat{\mathfrak{p}} : B; \sigma\{\cdot|p\}}\ \wedge_E$$

The soundness of such extensions can be justified either by extending the CPS translation, or by defining a translation to Lepigre's calculus (which already allows records and pattern-matching over general constructors) and proving the adequacy of the translation with respect to the realizability model.

[10] In fact, the proof of adequacy [17, Theorem 6] suggests a way to extend the type system to include typed stacks. Indeed, any typing rule for typing stacks is valid as long as it is adequate with the realizability model.

5.2 Extending the domain of terms

Throughout the article, we only worked with terms of a unique type $\mathbb{N}$, hence it is natural to wonder whether it is possible to extend the domain of terms in dL, for instance with terms in the simply-typed λ-calculus. A good way to understand the situation is to observe what happens through the CPS translation. We see that a *term* of type $T = \mathbb{N}$ is translated into a *proof* of type $\neg\neg T^+ = \neg\neg\mathbb{N}$, from which we can extract a *term* of type $\mathbb{N}$. However, if T was for instance the function type $\mathbb{N} \to \mathbb{N}$, we would only be able to extract a *proof* of type $T^+ = \mathbb{N} \to \neg\neg\mathbb{N}$. In particular, such a proof would be of the form $\lambda x.p$, where p might backtrack to a former position, for instance before it was extracted, and furnish another proof. This accounts for a well-know phenomenon in classical logic, where witness extraction is limited to formulas in the Σ_0^1-fragment [19]. It also corresponds to the type we obtain for the image of a dependent product $\Pi_{a:A}B$, that is translated to a type $\neg\neg\Pi_{a:A^+}B^*$ where the dependence is in a proof of type A^+. This phenomenon is not surprising and was already observed for other CPS translations for type theories with dependent types [2].

In other words, there is no hope to define a correct translation of $(t_f, p) : \exists f^{\mathbb{N}\to\mathbb{N}} A$ that would allow the extracttion of a strong pair $([\![t_f]\!], [\![p]\!]) : \exists f^{\mathbb{N}\to\mathbb{N}} A^*$. More precisely, the proof $[\![t_f]\!]$ is no longer a witness in the usual sense, but a realizer of $f \in \mathbb{N} \to \mathbb{N}$ in the sense of Krivine classical realizability.

This does not mean that we cannot extend the domain of terms in dL (in particular, it should affect neither the subject reduction nor the soundness), but it rather means that the stratification between terms and proofs is to be lost through a CPS translation. However, it should still be possible to express the fact that the image of a function through the CPS is a realizer corresponding to this function, by cleverly adapting the predicate $f \in \mathbb{N} \to \mathbb{N}$ to make it stick to the intuition of a realizer.

5.3 A fully sequent-style dependent calculus

While the aim of this paper was to design a sequent-style calculus embedding dependent types, we only present the Π-type in sequent-style. Indeed, we wanted to ensure ourselves in a first time that we were able of having the key ingredients of dependent types in our language, even presented in a natural deduction spirit. Rather than having left-rules, we presented the existential type and the equality type with the following elimination rules:

$$\frac{\Gamma \vdash p : \exists x^T A(x) \mid \Delta;\sigma \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{prf}\, p : A(\mathsf{wit}\, p) \mid \Delta;\sigma}\ \mathsf{prf}$$

$$\frac{\Gamma \vdash p : t = u \mid \Delta;\sigma \quad \Gamma \vdash q : B[t/x] \mid \Delta;\sigma}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta;\sigma}\ \mathsf{subst}$$

However, it is now easy to have both rules in a sequent calculus fashion, for instance we could rather have contexts of the shape $\tilde{\mu}(x,a).c$ (dual to proofs (t,p)) with the left rule:

$$\frac{c : \Gamma, x : T, a : A(x) \vdash \Delta;\sigma\{(x,a)|p\}}{\Gamma \mid \tilde{\mu}(x,a).c \vdash \Delta;\sigma\{\cdot|p\}}\ \exists$$

and define $\mathsf{prf}\, p \triangleq \mu\alpha.\langle p \| \tilde{\mu}(x,a).\langle a\|\alpha\rangle\rangle$. For any $p \in$ NEF and any variables a, α, we have $A(\mathsf{wit}\, p) \in A(\mathsf{wit}\,(x,a))_{\{(x,a)|p\}}$ which allows us to derive (using this in the $(\mathtt{cut})$-rule):

$$\Pi = \dfrac{\dfrac{\dfrac{}{a : A(x) \vdash a : A(x)}}{a : A(x) \vdash a : A(\mathsf{wit}\,(x,a))} \equiv \dfrac{}{\Gamma \mid \alpha : A(\mathsf{wit}\, p) \vdash \alpha : A(\mathsf{wit}\, p) \mid \Delta}}{\dfrac{\langle a\|\alpha\rangle : \Gamma, x : T, a : A(x) \vdash \Delta, \alpha : A(\mathsf{wit}\, p);\sigma\{(x,a)|p\}}{\Gamma \mid \tilde{\mu}(x,a).\langle a\|\alpha\rangle : \exists x^T A \vdash \Delta, \alpha : A(\mathsf{wit}\, p);\sigma\{\cdot|p\}}}$$

and makes the former prf -rule admissible:

$$\dfrac{\Gamma \vdash p : \exists x^T.A \mid \Delta;\sigma \qquad \Pi}{\dfrac{\langle p\|\tilde{\mu}(x,a).\langle a\|\alpha\rangle\rangle : \Gamma \vdash \Delta, \alpha : A(\mathsf{wit}\, p);\sigma\{\cdot|p\}}{\Gamma \vdash \mu\alpha.\langle p\|\tilde{\mu}(x,a).\langle a\|\alpha\rangle\rangle : A(\mathsf{wit}\, p) \mid \Delta;\sigma}}$$

In the same spirit, we can define contexts of the shape $\tilde{\mu}{=}.c$ to be dual to refl, extend the dependencies list to terms (to compensate the conversion from $A[t]$ to $A[u]$) and give the following left-rule for the existential type:

$$\frac{c : \Gamma \vdash \Delta;\sigma\{t|u\}}{\Gamma \mid \tilde{\mu}{=}.c : t = u \vdash \Delta;\sigma\{\cdot|p\}}\ =_l$$

We can then let $\mathsf{subst}\, p\, q$ be syntactic sugar for: $\mathsf{subst}\, p\, q \triangleq \mu\alpha.\langle p\|\tilde{\mu}{=}.\langle q\|\alpha\rangle\rangle$. Using the fact that $B[u] \in (B[t])_{\sigma\{t|u\}}$, we get that the former (subst)-rule is admissible:

$$\dfrac{\dfrac{\Gamma \vdash p : t = u \mid \Delta;\sigma \qquad \dfrac{\Gamma \vdash q : B[t] \mid \Delta;\sigma \qquad \dfrac{}{\Gamma \mid \alpha : B[u] \vdash \alpha : B[u] \mid \Delta}}{\langle q\|\alpha\rangle : \Gamma \vdash \Delta, \alpha : B[u];\sigma\{t|u\}}}{\Gamma \mid \tilde{\mu}{=}.\langle q\|\alpha\rangle : t = u \vdash \Delta, \alpha : B[u];\sigma\{\cdot|p\}}}{\dfrac{\langle p\|\tilde{\mu}{=}.\langle q\|\alpha\rangle\rangle : \Gamma \vdash \Delta, \alpha : B[u];\sigma}{\Gamma \vdash \mu\alpha.\langle p\|\tilde{\mu}{=}.\langle q\|\alpha\rangle\rangle : B[u] \mid \Delta;\sigma}}\ .$$

As for the reduction rules, we can define the following natural (call-by-value) reductions:

$$\begin{array}{rcl} \langle (V_t, V)\|\tilde{\mu}(x,a).c\rangle & \rightsquigarrow & c[V_t/x][V/a] \\ \langle \mathsf{refl}\|\tilde{\mu}{=}.c\rangle & \rightsquigarrow & c \end{array}$$

and check that they advantageously simulate the previous rules (the expansion rules become useless):

$$\begin{array}{rcll} \langle \mathsf{prf}\, p\|e\rangle & \rightsquigarrow & \langle \mu\hat{\mathfrak{tp}}.\langle p\|\tilde{\mu}a.\langle \mathsf{prf}\, a\|\hat{\mathfrak{tp}}\rangle\rangle\|e\rangle & \\ \langle \mathsf{prf}\,(V_t, V_p)\|e\rangle & \rightsquigarrow & \langle V\|e\rangle & \\ \langle \mathsf{subst}\, p\, q\|e\rangle & \rightsquigarrow & \langle p\|\tilde{\mu}a.\langle \mathsf{subst}\, a\, q\|e\rangle\rangle & (p \notin V) \\ \langle \mathsf{subst}\, \mathsf{refl}\, q\|e\rangle & \rightsquigarrow & \langle q\|e\rangle & \end{array}\ .$$

Acknowledgments

The author wish to thanks Pierre-Marie Pédrot for a discussion that led to the idea of using delimited continuations, Gabriel Scherer for the constant interest he showed for this work as well as the anonymous referees of an earlier version of this paper.

References

[1] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Inf. Comput.*, 125(2):103–117, 1996.

[2] G. Barthe, J. Hatcliff, and M. H. B. Sørensen. CPS translations and applications: The cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, 1999. ISSN 1573-0557. doi: 10.1023/A:1010000206149. URL `http://dx.doi.org/10.1023/A:1010000206149`.

[3] V. Blot. Hybrid realizability for intuitionistic and classical choice. In *LICS 2016, New York, USA, July 5-8, 2016*, 2016.

[4] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2):95 – 120, 1988. ISSN 0890-5401. doi: http://dx.doi.org/10.1016/0890-5401(88)90005-3. URL `http://www.sciencedirect.com/science/article/pii/0890540188900053`.

[5] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of ICFP 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000. ISBN 1-58113-202-6.

[6] P. Downen, L. Maurer, Z. M. Ariola, and S. P. Jones. Sequent calculus as a compiler intermediate language. In *ICFP 2016*, 2016. URL `http://research.microsoft.com/en-us/um/people/simonpj/papers/sequent-core/scfp\_ext.pdf`.

[7] G. Ferreira and P. Oliva. On various negative translations. In S. van Bakel, S. Berardi, and U. Berger, editors, *Proceedings Third International Workshop on Classical Logic and Computation, CL&C 2010, Brno, Czech Republic, 21-22 August 2010.*, volume 47 of *EPTCS*, pages 21–33, 2010. doi: 10.4204/EPTCS.47.4. URL `http://dx.doi.org/10.4204/EPTCS.47.4`.

[8] H. Friedman. *Classically and intuitionistically provably recursive functions*, pages 21–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978. ISBN 978-3-540-35749-0. doi: 10.1007/BFb0103100. URL `http://dx.doi.org/10.1007/BFb0103100`.

[9] J. Garrigue. Relaxing the value restriction. In Y. Kameyama and P. J. Stuckey, editors, *Functional and Logic Programming, 7th International Symposium, FLOPS 2004, Nara, Japan, April 7-9, 2004, Proceedings*, volume 2998 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2004. ISBN 3-540-21402-X. doi: 10.1007/978-3-540-24754-8_15. URL `http://dx.doi.org/10.1007/978-3-540-24754-8_15`.

[10] T. G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 47–58, New York, NY, USA, 1990. ACM. ISBN 0-89791-343-4. doi: 10.1145/96709.96714. URL `http://doi.acm.org/10.1145/96709.96714`.

[11] R. Harper and M. Lillibridge. Polymorphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6(3):361–379, 1993.

[12] H. Herbelin. On the degeneracy of sigma-types in presence of computational classical logic. In P. Urzyczyn, editor, *Proceedings of TLCA 2005*, volume 3461 of *LNCS*, pages 209–220. Springer, 2005. ISBN 3-540-25593-1.

[13] H. Herbelin. A constructive proof of dependent choice, compatible with classical logic. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 365–374. IEEE Computer Society, 2012. ISBN 978-1-4673-2263-8. doi: 10.1109/LICS.2012.47. URL `http://dx.doi.org/10.1109/LICS.2012.47`.

[14] H. Herbelin and S. Ghilezan. An approach to call-by-name delimited continuations. In G. C. Necula and P. Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 383–394. ACM, Jan. 2008. ISBN 978-1-59593-689-9.

[15] W. A. Howard. The formulae-as-types notion of construction. Privately circulated notes, 1969.

[16] J.-L. Krivine. Realizability in classical logic. In interactive models of computation and program behaviour. *Panoramas et synthèses*, 27: 197–229, 2009.

[17] R. Lepigre. A classical realizability model for a semantical value restriction. In P. Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 476–502. Springer, 2016. ISBN 978-3-662-49497-4.

[18] P. Martin-Löf. Constructive mathematics and computer programming. In *Proc. Of a Discussion Meeting of the Royal Society of London on Mathematical Logic and Programming Languages*, pages 167–184, Upper Saddle River, NJ, USA, 1985. Prentice-Hall, Inc. ISBN 0-13-561465-1. URL `http://dl.acm.org/citation.cfm?id=3721.3731`.

[19] A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7 (2), 2011. doi: 10.2168/LMCS-7(2:2)2011. URL `http://dx.doi.org/10.2168/LMCS-7(2:2)2011`.

[20] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. Symb. Log.*, 62(4):1461–1479, 1997.

[21] E. Polonovski. Strong normalization of lambda-bar-mu-mu-tilde-calculus with explicit substitutions. In *FOSSACS*, volume 2987 of *Lecture Notes in Computer Science*, pages 423–437, Barcelona, Spain, 2004. Springer-Verlag. URL `https://hal.archives-ouvertes.fr/hal-00004321`.

[22] P. Wadler. Call-by-value is dual to call-by-name. In C. Runciman and O. Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, pages 189–201. ACM, 2003. ISBN 1-58113-756-7. doi: 10.1145/944705.944723. URL `http://doi.acm.org/10.1145/944705.944723`.

[23] A. Wright. Simple imperative polymorphism. In *LISP and Symbolic Computation*, pages 343–356, 1995.

A. Proofs of Section 1

A.1 Subject reduction

We give here the full proof for subject reduction in Section 1.

Theorem 7 (Subject reduction)**.** *If* $c : (\Gamma \vdash \Delta; \varepsilon)$ *and* $c \rightsquigarrow c'$*, then* $c' : (\Gamma \vdash \Delta; \varepsilon)$.

Proof. The proof is done by induction on the typing rules, assuming that for each typing proof, the CONV rules are always pushed down and right as much as possible. To save some space, we sometimes omit the dependencies list when empty, noting $c : \Gamma \vdash \Delta$ instead of $c : \Gamma \vdash \Delta; \varepsilon$, and we denote the CONV-rules by:

$$\frac{\Gamma \mid e : B \vdash \Delta; \sigma}{\Gamma \mid e : A \vdash \Delta; \sigma} \equiv$$

where the hypothesis $A \equiv B$ is implicit.

- **Case** $\langle \lambda x.p \| t \cdot e \rangle \rightsquigarrow \langle p[t/x] \| e \rangle$.
  A typing proof for the command on the left is of the form:

$$\dfrac{\dfrac{\Pi_p}{\dfrac{\Gamma, x : T \vdash p : A \mid \Delta}{\Gamma \vdash \lambda x.p : \forall x^T A \mid \Delta}} \quad \dfrac{\dfrac{\Pi_t}{\Gamma \vdash t : T \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[t/x] \vdash \Delta; \{\cdot \mid \dagger\}}}{\dfrac{\Gamma \mid t \cdot e : \forall x^T B \vdash \Delta; \{\cdot \mid \lambda x.p\}}{\Gamma \mid t \cdot e : \forall x^T A \vdash \Delta; \{\cdot \mid \lambda x.p\}}}}{\langle \lambda x.p \| t \cdot e \rangle : \Gamma \vdash \Delta; \varepsilon} \equiv$$

  We first deduce $A[t/x] \equiv B[t/x]$ from the hypothesis $\forall x^T.A \equiv \forall x^T.B$. Then using that $\Gamma, x : T \vdash p : A \mid \Delta$ and $\Gamma \vdash t : T \mid \Delta$, by Lemma 4 and the fact that $\Delta[t/x] = \Delta$ we get a proof Π'_p of $\Gamma \vdash p[t/x] : A[t/x] \mid \Delta$. We can thus build the following derivation:

$$\dfrac{\dfrac{\Pi'_p}{\Gamma \vdash p[t/x] : A[t/x] \mid \Delta} \quad \dfrac{\dfrac{\Pi_e}{\Gamma \mid e : B[t/x] \vdash \Delta; \{\cdot \mid p[t/x]\}}}{\Gamma \mid e : A[t/x] \vdash \Delta; \{\cdot \mid p[t/x]\}}}{\langle p[t/x] \| e \rangle : \Gamma \vdash \Delta} \equiv$$

  using Corollary 3 to weaken the binding to $p[t/x]$ in Π_e.
- **Case** $\langle \lambda a.p \| q \cdot e \rangle \rightsquigarrow \langle q \| \tilde{\mu} a. \langle p \| e \rangle \rangle$.
  A typing proof for the command on the left is of the form:

$$\dfrac{\dfrac{\Pi_p}{\dfrac{\Gamma, a : A \vdash p : B \mid \Delta}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta}} \quad \dfrac{\Pi_q \quad \dfrac{\Pi_e}{\Gamma \mid e : B'[q/a] \vdash \Delta; \{\cdot \mid \dagger\}}}{\dfrac{\Gamma \mid q \cdot e : \Pi_{a:A'} B' \vdash \Delta; \{\cdot \mid \lambda a.p\}}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta; \{\cdot \mid \lambda a.p\}}}}{\langle \lambda a.p \| q \cdot e \rangle : \Gamma \vdash \Delta} \equiv$$

  If $q \notin \mathcal{D}$, we define $B'_q \triangleq B'$ which is the only type in $B'_{\{a|q\}}$. Otherwise, we define $B'_q \triangleq B'[q/a]$ which is a type in $B'_{\{a|q\}}$. In both case, we can build the following derivation:

$$\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A' \mid \Delta}}{\Gamma \vdash q : A \mid \Delta} \equiv \dfrac{\Pi_{\tilde{\mu}}}{\Gamma \mid \tilde{\mu} a. \langle p \| e \rangle : A \vdash \Delta; \{. \mid q\}}$$
$$\langle q \| \tilde{\mu} a. \langle p \| e \rangle \rangle : \Gamma \vdash \Delta; \varepsilon$$

$$\Pi_{\tilde{\mu}} = \dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma, a : A \vdash p : B' \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma, a : A \mid e : B'_q \vdash \Delta; \{a|q\}\{\cdot|p\}}}{\dfrac{\langle p \| e \rangle : \Gamma, a : A \vdash \Delta; \{a|q\}}{\Gamma \mid \tilde{\mu} a. \langle p \| e \rangle : A \vdash \Delta; \{. \mid q\}}}$$

  using Corollary 3 to weaken the dependencies in Π_e.
- **Case** $\langle \mu\alpha.c \| e \rangle \rightsquigarrow c[e/\alpha]$.
  A typing proof for the command on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_c}{c : \Gamma \vdash \Delta, \alpha : A}}{\Gamma \vdash \mu\alpha.c : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A \vdash \Delta; \{\cdot \mid \mu\alpha.c\}}}{\langle \mu\alpha.c \| e \rangle : \Gamma \vdash \Delta}$$

  We get a proof that $c[e/\alpha] : \Gamma \vdash \Delta; \varepsilon$ is valid by Lemma 6.
- **Case** $\langle V \| \tilde{\mu} a.c \rangle \rightsquigarrow c[V/a]$.
  A typing proof for the command on the left is of the form:

$$\dfrac{\dfrac{\Pi_V}{\Gamma \vdash V : A \mid \Delta} \quad \dfrac{\dfrac{\dfrac{\Pi_c}{c : \Gamma, a : A' \vdash \Delta; \{a|V\}}}{\Gamma \mid \tilde{\mu} a.c : A' \vdash \Delta; \{\cdot|V\}}}{\Gamma \mid \tilde{\mu} a.c : A \vdash \Delta; \{\cdot|V\}}}{\langle V \| \tilde{\mu} a.c \rangle : \Gamma \vdash \Delta} \equiv$$

  We first observe that we can derive the following proof:

$$\dfrac{\dfrac{\Pi_V}{\Gamma \vdash V : A \mid \Delta}}{\Gamma \vdash V : A' \mid \Delta} \equiv$$

  and get a proof for $c[V/a] : \Gamma \vdash \Delta; \{V|V\}$ by Lemma 5. We finally get a proof for $c[V/a] : \Gamma \vdash \Delta; \varepsilon$ by Lemma 1.
- **Case** $\langle (t,p) \| e \rangle \rightsquigarrow \langle p \| \tilde{\mu} a. \langle (t,a) \| e \rangle \rangle$, with $p \notin V$.
  A proof of the command on the left is of the form:

$$\dfrac{\dfrac{\Pi_t}{\Gamma \vdash t : T \mid \Delta} \quad \dfrac{\Pi_p}{\Gamma \vdash p : A[t/x] \mid \Delta}}{\Gamma \vdash (t,p) : \exists x^T A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : \exists x^T A \vdash \Delta; \{\cdot|(t,p)\}}$$
$$\langle (t,p) \| e \rangle : \Gamma \vdash \Delta$$

  We can build the following derivation:

$$\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : A[t/x] \mid \Delta} \quad \dfrac{\Pi_{(t,a)} \quad \dfrac{\Pi_e}{\Gamma \mid e : \exists x^T A \vdash \Delta; \{a|p\}\{\cdot|(t,a)\}}}{\dfrac{\langle (t,a) \| e \rangle : \Gamma, a : A[t/x] \vdash \Delta; \{a|p\}}{\Gamma \mid \tilde{\mu} a. \langle (t,a) \| e \rangle : A[t/x] \vdash \Delta; \{\cdot|p\}}}}{\langle p \| \tilde{\mu} a. \langle (t,a) \| e \rangle \rangle : \Gamma \vdash \Delta}$$

  where $\Pi_{(t,a)}$ is as expected, observing that since $p \notin \mathcal{D}$, the binding $\{\cdot|(t,p)\}$ is the same as $\{\cdot|\dagger\}$, and we can apply Corollary 3 to weaken dependencies in Π_e.
- **Case** $\langle \mathsf{prf}\,(t,V) \| e \rangle \rightsquigarrow \langle V \| e \rangle$.
  This case is easy, observing that a derivation of the command on the left is of the form:

$$\dfrac{\dfrac{\Pi_t \quad \dfrac{\Pi_V}{\Gamma \vdash V : A(t) \mid \Delta}}{\Gamma \vdash (t,V) : \exists x^T A(x) \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\,(t,V)) \vdash \Delta; \{\cdot \mid \dagger\}}}{\langle \mathsf{prf}\,(t,V) \| e \rangle : \Gamma \vdash \Delta}$$

  Since by definition we have $A(\mathsf{wit}\,(t,V)) \equiv A(t)$, we can derive:

$$\dfrac{\dfrac{\Pi_V}{\Gamma \vdash V : A(t) \mid \Delta} \quad \dfrac{\dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\,(t,V)) \vdash \Delta; \{\cdot|V\}}}{\Gamma \mid e : A(V) \vdash \Delta; \{\cdot|V\}}}{\langle \mathsf{prf}\,(t,V) \| e \rangle : \Gamma \vdash \Delta} \equiv$$

- **Case** $\langle \mathsf{subst\ refl}\ q \| e \rangle \rightsquigarrow \langle q \| e \rangle$.
  This case is straightforward, observing that for any terms t, u, if we have $\mathsf{refl} : t = u$, then $A[t] \equiv A[u]$ for any A.
- **Case** $\langle \mathsf{subst}\,p\,q \| e \rangle \rightsquigarrow \langle p \| \tilde{\mu} a. \langle \mathsf{subst}\,a\,q \| e \rangle \rangle$.
  This case is exactly the same than the case $\langle (t,p) \| e \rangle$.
- **Case** $c[t] \rightsquigarrow c[t']$ with $t \to t'$.
  Obvious by observing that by definition of the relation $\equiv$, we have $A[t] \equiv A[t']$ for any A. □

A.2 Soundness

We detail here the proof of soundness in Section 1.5. The proof is based on an embedding to the $\mu\tilde{\mu}$-calculus extended, whose syntax and rules are given below. We do not consider here a particular reduction strategy, and take $\rightarrowtail$ to be the contextual closure of the rules given below.

$$\begin{array}{lrcl} \text{Proofs} & p & ::= & V \mid \mu\alpha.c \mid (p_1, p_2) \\ \text{Values} & V & ::= & a \mid \lambda a.p \mid (V_1, V_2) \\ \text{Contexts} & e & ::= & \alpha \mid p \cdot e \mid \tilde{\mu}a.c \mid \tilde{\mu}(a_1, a_2).c \\ \text{Commands} & c & ::= & \langle p \| e \rangle \end{array}$$

(a) Syntax

$$\begin{array}{rcl} \langle \mu\alpha.c \| e \rangle & \rightarrowtail & c[e/\alpha] \\ \langle \lambda a.p \| q \cdot e \rangle & \rightarrowtail & \langle q \| \tilde{\mu}a.\langle p \| e \rangle \rangle \\ \langle p \| \tilde{\mu}a.c \rangle & \rightarrowtail & c[p/a] \\ \langle (p_1, p_2) \| \tilde{\mu}(a_1, a_2).c \rangle & \rightarrowtail & c[p_1/a_1][p_2/a_2] \\ \mu\alpha.\langle p \| \alpha \rangle & \rightarrowtail & p \\ \tilde{\mu}a.\langle a \| e \rangle & \rightarrowtail & e \end{array}$$

(b) Reduction rules

$$\frac{\Gamma \vdash p_1 : A_1 \mid \Delta \quad \Gamma \vdash p_2 : A_2 \mid \Delta}{\Gamma \vdash (p_1, p_2) : A_1 \wedge A_2 \mid \Delta} \wedge_r$$

$$\frac{c : \Gamma, a_1 : A_1, a_2 : A_2 \vdash \Delta}{\Gamma \mid \tilde{\mu}(a_1, a_2).c : A_1 \wedge A_2 \vdash \Delta} \wedge_l$$

(c) Typing rules

Figure 13. $\mu\tilde{\mu}$-calculus with pairs

We briefly recall the erasure procedure:

$$\begin{array}{lcl|lcl} (\forall x^{\mathbb{N}} A)^* & \triangleq & \mathbb{N}^* \to A^* & \top^* & \triangleq & \mathbb{N}^* \to \mathbb{N}^* \\ (\exists x^{\mathbb{N}} A)^* & \triangleq & \mathbb{N}^* \wedge A^* & \bot^* & \triangleq & \mathbb{N}^* \to \mathbb{N}^* \\ (\Pi_{a:A} B)^* & \triangleq & A^* \to B^* & (t = u)^* & \triangleq & \mathbb{N}^* \to \mathbb{N}^* \end{array}$$

and the corresponding translation for terms, proofs, contexts and commands:

$$\begin{array}{lll|lll} x^* & \triangleq & x & (t,p)^* & \triangleq & \mu\alpha.\langle p^* \| \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle \rangle \\ n^* & \triangleq & \bar{n} & (\mathsf{prf}\, p)^* & \triangleq & \pi_2(p^*) \\ (\mathsf{wit}\, p)^* & \triangleq & \pi_1(p^*) & & & \\ a^* & \triangleq & a & \alpha^* & \triangleq & \alpha \\ (\lambda a.p)^* & \triangleq & \lambda a.p^* & (t \cdot e)^* & \triangleq & t^* \cdot e^* \\ (\lambda x.p)^* & \triangleq & \lambda x.p^* & (q \cdot e)^* & \triangleq & q^* \cdot e^* \\ (\mu\alpha.c)^* & \triangleq & \mu\alpha.c^* & (\tilde{\mu}a.c)^* & \triangleq & \tilde{\mu}a.c^* \\ \mathsf{refl}^* & \triangleq & \lambda x.x & \langle p \| e \rangle^* & \triangleq & \langle p^* \| e^* \rangle \end{array}$$

$$(\mathsf{subst}\, V\, q)^* \triangleq \mu\alpha.\langle q^* \| \alpha \rangle$$

$$(\mathsf{subst}\, p\, q)^* \triangleq \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha \rangle \| \alpha \rangle \rangle \qquad (p \notin V)$$

where $\pi_i(p) \triangleq \mu\alpha.\langle p \| \tilde{\mu}(a_1, a_2).\langle a_1 \| \alpha \rangle \rangle$. The term $\bar{n}$ is defined as any encoding of the natural number n with its type $\mathbb{N}^*$, the encoding being irrelevant here as long as $\bar{n} \in V$.

We first show that the erasure procedure is adequate with respect to the previous translation.

Lemma 20. *The following holds for any types A and B:*

1. *For any terms t and u, $(A[t/u])^* = A^*$.*
2. *For any proofs p and q, $(A[p/q])^* = A^*$.*
3. *If $A \equiv B$ then $A^* = B^*$.*
4. *For any dependencies list σ, if $A \in B_\sigma$, then $A^* = B^*$.*

Proof. Straightforward: 1 and 2 are direct consequences of the erasure of terms (and thus proofs) from types. 3 follows from 1,2 and the fact that $(t = u)^* = \top^* = \bot^*$. 4 follows from 2. □

Proposition 8. *If $c : \Gamma \vdash \Delta; \sigma$, then $c^* : \Gamma^* \vdash \Delta^*$. The same holds for proofs and contexts.*

Proof. By induction on typing rules. The fourth item of the previous lemma shows that the dependencies list becomes useless for the (CUT)-rule, and consequently it can be dropped for all the other cases. The case of the conversion rule is a direct consequence of the third case. For refl, we have by definition, $\mathsf{refl}^* = \lambda x.x : \mathbb{N}^* \to \mathbb{N}^*$.

The only non-direct cases are the cases for (t, p) and $\mathsf{subst}\, p\, q$. To prove the latter with $p \notin V$, we have to show that if:

$$\frac{\Gamma \vdash p : t = u \mid \Delta; \sigma \quad \Gamma \vdash q : B[t/x] \mid \Delta; \sigma}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta; \sigma}\ \mathsf{subst}$$

then $\mathsf{subst}\, p\, q^* = \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha \rangle \| \alpha \rangle \rangle : B[u/x]^*$. According to Lemma 20, we have $B[u/x]^* = B[t/x]^* = B^*$. By induction hypothesis, we have proofs of $\Gamma^* \vdash p^* : \mathbb{N}^* \to \mathbb{N}^* \mid \Delta^*$ and $\Gamma^* \vdash q^* : B \mid \Delta^*$. Using the notation $\eta_{q^*} \triangleq \mu\alpha.\langle q^* \| \alpha \rangle$, we can derive:

$$\dfrac{\dfrac{\Gamma^* \vdash p^* : \mathbb{N}^* \to \mathbb{N}^* \mid \Delta^* \quad \dfrac{\dfrac{\Gamma^* \vdash q^* : B^* \mid \Delta^*}{\Gamma^* \vdash \eta_{q^*} : B^* \mid \Delta^*} \quad \overline{\alpha : B^* \vdash \alpha : B^*}}{\langle \eta_{q^*} \| \alpha \rangle : \Gamma \vdash \Delta^*, \alpha : B^*}}{\Gamma^* \mid \tilde{\mu}\_.\langle \eta_{q^*} \| \alpha \rangle : B^* \vdash \Delta^*, \alpha : B^*}}{\dfrac{\langle p^* \| \tilde{\mu}\_.\langle \eta_{q^*} \| \alpha \rangle \rangle : \Gamma^* \vdash \Delta^*, \alpha : B^*}{\Gamma^* \vdash \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \eta_{q^*} \| \alpha \rangle \rangle : B^* \mid \Delta^*}}$$

The case $\mathsf{subst}\, V\, q$ is easy since $(\mathsf{subst}\, V\, q)^* = [\![q]\!]_p$ has type B^* by induction. Similarly the proof for the case (t, p) corresponds to the following derivation:

$$\dfrac{\dfrac{\dfrac{\Gamma^* \vdash t^* : \mathbb{N}^* \mid \Delta^* \quad \overline{a : A^* \vdash a : A^*}}{\Gamma^*, a : A^* \vdash (t^*, a) : \mathbb{N}^* \wedge A^* \mid \Delta^*} \quad \overline{\alpha : \mathbb{N}^* \wedge A^* \vdash \alpha : \mathbb{N}^* \wedge A^*}}{\langle (t^*, a) \| \alpha \rangle : \Gamma, a : A^* \vdash \Delta^*, \alpha : \mathbb{N}^* \wedge A^*} \quad \overline{\Gamma^* \mid \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle : A^* \vdash \Delta^*, \alpha : \mathbb{N}^* \wedge A^*}}{\dfrac{\Gamma^* \vdash p^* : A^* \mid \Delta^* \quad \overline{\Gamma^* \mid \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle : A^* \vdash \Delta^*, \alpha : \mathbb{N}^* \wedge A^*}}{\langle p^* \| \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle \rangle : \Gamma^* \vdash \Delta^*, \alpha : \mathbb{N}^* \wedge A^*}}{\Gamma^* \vdash \mu\alpha.\langle p^* \| \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle \rangle : \mathbb{N}^* \wedge A^* \mid \Delta^*}$$

Every other cases follows directly from the induction hypothesis. □

We can then deduce the normalization of dL from the normalization of the $\mu\tilde{\mu}$-calculus [21], by showing that the translation preserves the normalization in the sense that if c does not normalize, neither does c^*.

Proposition 21. *If c is a command such that c^* normalizes, then c normalizes.*

Proof. We will actually prove a slighty more precise statement:

$$\forall c_1, c_2, (c_1 \overset{1}{\rightsquigarrow} c_2 \Rightarrow \exists n \geq 1, (c_1)^* \overset{n}{\rightarrowtail} (c_2)^*).$$

Assuming it holds, we get from any infinite reduction path (for $\rightsquigarrow$) starting from c another infinite reduction path (for $\rightarrowtail$) from c^*. Thus the normalization of c^* implies the one of c.

It remains to prove the previous statement, that is an easy induction on the reduction rule $\rightsquigarrow$.

Case $\mathsf{wit}\,(t, V) \to t$**:**

$$\begin{array}{rcl} (\mathsf{wit}\,(t, V))^* & = & \pi_1(\mu\alpha.\langle V^* \| \tilde{\mu}a.\langle (t^*, a) \| \alpha \rangle \rangle) \\ & \rightarrowtail & \pi_1(\mu\alpha.\langle (t^*, V^*) \| \alpha \rangle) \\ & \rightarrowtail & \pi_1(t^*, V^*) \\ & = & \mu\alpha.\langle (t^*, t^*) \| \tilde{\mu}(a_1, a_2).\langle a_1 \| \alpha \rangle \rangle \\ & \rightarrowtail & \mu\alpha.\langle t^* \| \alpha \rangle \rightarrowtail t^*. \end{array}$$

Case $\langle \mu\alpha.c \| e\rangle \rightsquigarrow c[e/\alpha]$***:***
$(\langle \mu\alpha.c \| e\rangle)^* = \langle \mu\alpha.c^* \| e^*\rangle \rightarrowtail c^*[e^*/\alpha] = c[e/\alpha]^*$.

Case $\langle V \| \tilde{\mu}a.c\rangle \rightsquigarrow c[V/a]$***:***
$(\langle V \| \tilde{\mu}a.c\rangle)^* = \langle V^* \| \tilde{\mu}a.c^*\rangle \rightarrowtail c^*[V^*/a] = c[V/a]^*$.

Case $\langle \lambda a.p \| q \cdot e\rangle \rightsquigarrow \langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle$***:***

$$\begin{array}{rcl}(\langle \lambda a.p \| q \cdot e\rangle)^* & = & \langle \lambda a.p^* \| q^* \cdot e^*\rangle \\ & \rightarrowtail & \langle q^* \| \tilde{\mu}a.\langle p^* \| e^*\rangle\rangle \\ & = & (\langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle)^*.\end{array}$$

Case $\langle \lambda x.p \| t \cdot e\rangle \rightsquigarrow \langle p[t/x] \| e\rangle$***:***

$$\begin{array}{rcl}\langle \lambda x.p \| t \cdot e\rangle^* & = & \langle \lambda x.p^* \| t^* \cdot e^*\rangle \\ & \rightarrowtail & \langle t^* \| \tilde{\mu}x.\langle p^* \| e^*\rangle\rangle \\ & \rightarrowtail & \langle p^*[t^*/x] \| e^*\rangle = (\langle p[t/x] \| e\rangle)^*.\end{array}$$

Case $\langle (t,p) \| e\rangle \rightsquigarrow \langle p \| \tilde{\mu}a.\langle (t,a) \| e\rangle\rangle$***:***

$$\begin{array}{rcl}(\langle (t,p) \| e\rangle)^* & = & \langle \mu\alpha.\langle p^* \| \tilde{\mu}a.\langle (t^*,a) \| \alpha\rangle\rangle \| e^*\rangle \\ & \rightarrowtail & \langle p^* \| \tilde{\mu}a.\langle (t^*,a) \| e^*\rangle\rangle \\ & = & (\langle p \| \tilde{\mu}a.\langle (t,a) \| e\rangle\rangle)^*.\end{array}$$

Case $\langle \mathsf{prf}\ (t,V) \| e\rangle \rightsquigarrow \langle V \| e\rangle$***:***

$$\begin{array}{rcl}(\langle \mathsf{prf}\ (t,V) \| e\rangle)^* & = & \langle \pi_2(\mu\alpha.\langle V^* \| \tilde{\mu}a.\langle (t^*,a) \| \alpha\rangle\rangle) \| e^*\rangle \\ & \rightarrowtail & \langle \pi_2(\mu\alpha.\langle (t^*,V^*) \| \alpha\rangle) \| e^*\rangle \\ & \rightarrowtail & \langle \pi_2(t^*,V^*) \| e^*\rangle \\ & = & \langle \mu\alpha.\langle (t^*,V^*) \| \tilde{\mu}(a_1,a_2).\langle a_2 \| \alpha\rangle\rangle \| e^*\rangle \\ & = & \langle (t^*,V^*) \| \tilde{\mu}(a_1,a_2).\langle a_2 \| e^*\rangle\rangle \\ & \rightarrowtail & \langle V^* \| e^*\rangle = (\langle V \| e\rangle)^*.\end{array}$$

Case $\langle \mathsf{subst\ refl}\ q \| e\rangle \rightsquigarrow \langle q \| e\rangle$***:***

$$\begin{array}{rcl}(\langle \mathsf{subst\ refl}\ q \| e\rangle)^* & = & \langle \mu\alpha.\langle q^* \| \alpha\rangle \| e^*\rangle \\ & \rightarrowtail & \langle q^* \| e^*\rangle = (\langle q \| e\rangle)^*\end{array}$$

Case $\langle \mathsf{subst}\ p\ q \| e\rangle \rightsquigarrow \langle p \| \tilde{\mu}a.\langle \mathsf{subst}\ a\ q \| e\rangle\rangle \quad (p \notin V)$***:***

$$\begin{array}{rcl}(\langle \mathsf{subst}\ p\ q \| e\rangle)^* & = & \langle \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha\rangle \| \alpha\rangle\rangle \| e^*\rangle \\ & \rightarrowtail & \langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha\rangle \| e^*\rangle\rangle \\ & \rightarrowtail & \langle \mu\alpha.\langle q^* \| \alpha\rangle \| e^*\rangle = (\langle \mathsf{subst}\ a\ q \| e\rangle)^*\end{array}$$

□

Proposition 9. *If* $c : (\Gamma \vdash \Delta; \varepsilon)$*, then* c *normalizes.*

Proof. Proof by contradiction: if c does not normalize, then by Proposition 21 neither does c^*. However, by Proposition 8 we have that $c^* : \Gamma^* \vdash \Delta^*$. This is absurd since any well-typed command of the $\mu\tilde{\mu}$-calculus normalizes [21]. □

Using the normalization, we can finally prove the soundness of the system.

Theorem 10. *For any proof* p *in* dL*, we have:* $\not\vdash p : \bot$.

Proof. We actually start by proving by contradiction that a command $c \in$ dL cannot be well-typed with empty contexts. Indeed, let us assume that there is such a command $c : (\vdash)$. By normalization, we can reduced it to $c' = \langle p' \| e'\rangle$ in normal form and for which we have $c' : (\vdash)$ by subject reduction. Since c' cannot reduce and is well-typed, p' is necessarily a value and cannot be a free variable. Thus e' cannot be of the shape $\tilde{\mu}a.c''$ and every other possibility is either ill-typed or admits a reduction, which are both absurd.

We can now prove the soundness by contradiction. Assuming that there is a proof p such that $\vdash p : \bot$, we can form the well-typed command $\langle p \| \star\rangle : (\vdash \star : \bot)$ where $\star$ is any fresh α-variable. The previous result showe that p cannot drop the context $\star$ when reducing, since it would give raise to command $c : (\vdash)$. We can still reduce $\langle p \| \star\rangle$ to a command c in normal form, and see that c it has to be of the shape $\langle V \| \star\rangle$ (by the same kind of reasoning, using that c cannot reduce and that $c : (\vdash \star : \bot)$ by subject reduction). Therefore V is a value of type $\bot$, which is absurd. □

B. Proofs of Section 3

B.1 Subject reduction

Theorem 12 (Subject reduction)**.** *If* $c : (\Gamma \vdash \Delta)$ *and* $c \rightsquigarrow c'$*, then* $c' : (\Gamma \vdash \Delta)$.

Proof. Actually, the proof is slightly easier than for Theorem 7, because most of the rules do not involve dependencies. We only give some of the key cases.

- **Case** $\langle \lambda a.p \| q \cdot e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle$ with $q \in$ NEF.
A typing derivation for the command on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B \mid \Delta} \quad \dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\Gamma \mid q \cdot e : \Pi_{a:A} B \vdash \Delta}}{\langle \lambda a.p \| q \cdot e\rangle : \Gamma \vdash \Delta}$$

We can thus build the following derivation for the command on the right:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \quad \dfrac{\Pi_p}{\langle p \| \hat{\mathfrak{p}}\rangle : \Gamma, a : A \vdash_d \Delta, \hat{\mathfrak{p}} : B[q]; \varepsilon}}{\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle\rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{p}} : B[q]; \varepsilon}}{\Gamma \vdash \mu\hat{\mathfrak{p}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[q/a] \vdash \Delta}}{\langle \mu\hat{\mathfrak{p}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle : \Gamma \vdash \Delta}$$

$$\Pi_p = \dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B[a] \mid \Delta} \quad \dfrac{B[q] \in (B[a])_{\{a|q\}}}{\Gamma \mid \hat{\mathfrak{p}} : B[a] \vdash_d \Delta, \hat{\mathfrak{p}} : B[q]; \{a|q\}\{\cdot|\dagger\}}}{\dfrac{\langle p \| \hat{\mathfrak{p}}\rangle : \Gamma, a : A \vdash_d \Delta, \hat{\mathfrak{p}} : B[q]; \{a|q\}}{\Gamma \mid \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle : A \vdash_d \Delta, \hat{\mathfrak{p}} : B[q]; \{\cdot|q\}}}$$

- **Case** $\langle \mathsf{prf}\ p \| e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle$.
We prove it in the most general case, that is when this reduction occurs under a delimited continuation. A typing derivation for the command on the left has to be of the form:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : \exists x.A(x) \mid \Delta}}{\Gamma \vdash \mathsf{prf}\ p : A(\mathsf{wit}\, p) \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : A(\mathsf{wit}\, p) \vdash_d \Delta, \hat{\mathfrak{p}} : B; \sigma\{\cdot|\mathsf{prf}\ p\}}}{\langle \mathsf{prf}\ p \| e\rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{p}} : B; \sigma}$$

The proof p being NEF, so is $\mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle$, and by definition of the reduction for types, we have for any type A that:

$$A[\mathsf{prf}\ p] \triangleright A[\mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle],$$

so that we can prove that for any b:

$$\sigma\{b|\mathsf{prf}\ p\} \Rrightarrow \sigma\{b|\mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle\}.$$

Thus we can turn Π_e into Π'_e a derivation of the same sequent except for the dependencies lists that is changed to $\sigma\{\cdot|\mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle\}$. We conclude the proof of this case by giving the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : \exists x.A(x) \mid \Delta}}{\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle \Gamma \vdash_d \mid \Delta, \hat{\mathfrak{p}} : A(\mathsf{wit}\, p); \varepsilon} \quad \Pi_{\hat{\mathfrak{p}}}}{\Gamma \vdash \mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle\rangle : A(\mathsf{wit}\, p) \mid \Delta}$$

with $\Pi_{\hat{\mathfrak{p}}}$ the following derivation where we removed Γ and Δ when irrelevant:

$$\dfrac{\dfrac{\overline{a : \exists x.A \vdash a : \exists x.A}}{a : \exists x.A \vdash \mathsf{prf}\ a : A(\mathsf{wit}\, a)} \quad \dfrac{A(\mathsf{wit}\, p) \in (A(\mathsf{wit}\, a))_{\{a|p\}}}{\hat{\mathfrak{p}} : A(\mathsf{wit}\, a) \vdash_d \hat{\mathfrak{p}} : A(\mathsf{wit}\, p); \{a|p\}}}{\dfrac{\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle : \Gamma, a : \exists x.A(x) \vdash_d \Delta, \hat{\mathfrak{p}} : A(\mathsf{wit}\, p); \{a|p\}}{\Gamma \mid \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}}\rangle : \exists x.A(x) \vdash_d \Delta, \hat{\mathfrak{p}} : A(\mathsf{wit}\, p); \{\cdot|p\}}}$$

- **Case** $\langle \mu\hat{\mathfrak{tp}}.\langle p \| \hat{\mathfrak{tp}}\rangle \| e\rangle \rightsquigarrow \langle p \| e\rangle$.
  This case is trivial, because in a typing derivation for the command on the left, $\hat{\mathfrak{tp}}$ is typed with an empty dependencies list, thus the type of p,e and $\hat{\mathfrak{tp}}$ coincides.
- **Case** $\langle \mu\hat{\mathfrak{tp}}.c \| e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.c' \| e\rangle$ with $c \rightsquigarrow c'$.
  This case corresponds exactly to Theorem 7, except for the rule $\langle \mu\alpha.c \| e\rangle \rightsquigarrow c[e/\alpha]$, since $\mu\alpha.c$ is a NEF proof term (remember we are inside a delimited continuation), but this corresponds precisely to Lemma 11.

□

B.2 Soundness

Lemma 16. *The following holds:*

1. *If* $\Gamma \vdash t : T \mid \Delta$ *then:*
$$\Gamma \cup \Delta^* \vdash [\![t]\!]_t : \forall X(\forall x^{T^+} X(x) \to X(t^+)).$$
2. *If* $p \in$ NEF *and* $\Gamma \vdash p : A \mid \Delta$ *then:*
$$\Gamma \cup \Delta^* \vdash [\![p]\!]_p : \forall X.(\Pi_{a:[\![A]\!]^+} X(a) \to X(p^+)).$$
3. *If* $c \in$ NEF *and* $c : \Gamma \vdash \Delta, \star : B$ *then:*
$$\Gamma \cup \Delta^*, \star : \Pi_{b:B^+} X(b) \vdash [\![c]\!]_c : X(c^+).$$

Proof. The proof is done by mutual induction on the typing rule of dL for terms and NEF proof terms. We only give here a few cases to give an insight of the proof.

- **Case** $\mathsf{wit}\, p$**.**
  In dL the typing rule for $\mathsf{wit}\, p$ is the following:
$$\frac{\Gamma \vdash p : \exists x^T A(x) \mid \Delta \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{wit}\, p : \mathbb{N} \mid \Delta}\ \mathsf{wit}.$$
  We set $\Gamma' = \Gamma \cup \Delta^*$. We want to show that:
$$\Gamma' \vdash \lambda k.[\![p]\!]_p\,(\lambda a.k(\mathsf{wit}\, a)) : \forall X(\forall x^{T^+} X(x) \to X(t^+))$$
  where $t^+ \triangleq \mathsf{wit}\, p^+$. But by induction hypothesis, we have:
$$\Gamma' \vdash [\![p]\!]_p : \forall Z.(\Pi_{a:\exists x^{T^+}[\![A]\!]^+} Z(a) \to Z(p^+)),$$
  hence it amounts to showing that for any X we can build the following derivation, where $\Gamma_k = \Gamma', k : \forall x^{T^+} X(x)$:
$$\dfrac{\dfrac{}{\Gamma_k \vdash k : \forall x^{T^+} X(x)} \quad \dfrac{\overline{\Gamma_k, a : \exists x^{T^+}[\![A]\!]^+ \vdash a : \exists x^{T^+}[\![A]\!]^+}}{\Gamma_k, a : \exists x^{T^+}[\![A]\!]^+ \vdash \mathsf{wit}\, a : T^+}}{\dfrac{\Gamma_k, a : \exists x^{T^+}[\![A]\!]^+ \vdash k\,(\mathsf{wit}\, a) : X(\mathsf{wit}\, a)}{\Gamma_k \vdash \lambda a.k(\mathsf{wit}\, a) : \Pi_{a:\exists x^{T^+}[\![A]\!]^+} X(\mathsf{wit}\, a)}}$$

- **Case** (t, p)**.**
  In dL the typing rule for (t, p) is the following:
$$\frac{\Gamma \vdash t : T \mid \Delta \quad \Gamma \vdash p : A(t) \mid \Delta}{\Gamma \vdash (t, p) : \exists x^T A(x) \mid \Delta}\ \exists_i$$
  Hence by induction we obtain, using the same notation Γ' for $\Gamma \cup \Delta^*$:
$$\Gamma' \vdash [\![t]\!]_t : \forall X.(\forall x^{T^+} X(x) \to X(t^+))$$
$$\Gamma' \vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A(t^+)} Y(a) \to Y(p^+))$$
  and we want to show that for any Z:
$$\Gamma' \vdash [\![(t, p)]\!]_p : \Pi_{a:\exists x^{T^+} A} Z(a) \to Z(t^+, p^+).$$
  By definition, we have:
$$[\![(t, p)]\!]_p = \lambda k.([\![t]\!]_t\,(\lambda ur.r\,(\lambda q.k\,(u, q))))[\![p]\!]_p,$$
  so we need to prove that:
$$\Gamma_k \vdash ([\![t]\!]_t\,(\lambda ur.r\,(\lambda q.k\,(u, q))))[\![p]\!]_p : Z(t^+, p^+)$$
  where $\Gamma_k = \Gamma', k : \Pi_{a:\exists x^{T^+}{}_A} Z(a)$. We let the reader check that such a type is derivable by using $X(x) \triangleq P(x) \to Z(x, a)$ in the type of $[\![t]\!]_p$ where $P(t^+)$ is the type of $[\![p]\!]_p$, and using $Y(a) \triangleq Z(t^+, a)$ in the type of $[\![p]\!]_p$. The crucial point is to see that the bounded variable r is abstracted with type $P(x)$ in the derivation, which would not have been possible in the definition of $[\![(t, p)]\!]_p$ with $p \notin$ NEF.
- **Case** $\mu\star.c$
  For this case, we could actually conclude directly using the induction hypothesis for c. Rather than that, we do the full proof for the particular case $\mu\star.\langle p \| \tilde{\mu}a.\langle q \| \star\rangle\rangle$, which condensates the proofs for $\mu\star c$ and the two possible cases $\langle p_N \| e_N\rangle$ and $\langle p_N \| \star\rangle$ of NEF commands. This case corresponds to the following typing derivation in dL:
$$\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : A \mid \Delta} \quad \dfrac{\dfrac{\dfrac{\Pi_q}{\Gamma, a : A \vdash q : B \mid \Delta} \quad \overline{\cdots \mid \star : B \vdash \Delta, \star : B}}{\langle q \| \star\rangle : \Gamma, a : A \vdash \Delta, \star : B}}{\Gamma \mid \tilde{\mu}a.\langle q \| \star\rangle : A \vdash \Delta, \star : B}}{\dfrac{\langle p \| \tilde{\mu}a.\langle q \| \star\rangle\rangle : \Gamma \mid \Delta, \star : B}{\Gamma \vdash \mu\star.\langle p \| \tilde{\mu}a.\langle q \| \star\rangle \mid \Delta\rangle : B}}$$
  We want to show that for any X we can derive:
$$\Gamma' \vdash \lambda k.[\![p]\!]_p\,(\lambda a.[\![q]\!]_p\,k) : \Pi_{b:B} X(b) \to X(q^+[p^+/a]).$$
  By induction, we have:
$$\Gamma' \vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A^+} Y(a) \to Y(p^+))$$
$$\Gamma', a : A^+ \vdash [\![q]\!]_t : \forall Z.(\Pi_{b:B^+} Z(b) \to Z(q^+)),$$
  so that by choosing $Z(b) \triangleq X(b)$ and $Y(a) \triangleq X(q^+)$, we get the expected derivation:
$$\dfrac{\dfrac{\Gamma', a : A^+ \vdash [\![q]\!]_p : \ldots \quad \overline{k : \Pi_{b:B} X(b) \vdash k : k : \Pi_{b:B} X(b)}}{\Gamma', k : \Pi_{b:B} X(b), a : A^+ \vdash [\![q]\!]_p\,k : X(q^+)}}{\dfrac{\Gamma', k : \Pi_{b:B} X(b) \vdash \lambda a.[\![q]\!]_p\,k : \Pi_{a:A^+} X(q^+)}{\dfrac{[\![p]\!]_p : \ldots \quad \Gamma', k : \Pi_{b:B} X(b) \vdash \lambda a.[\![q]\!]_p\,k : \Pi_{a:A^+} X(q^+)}{\Gamma', k : \Pi_{b:B} X(b) \vdash [\![p]\!]_p\,(\lambda a.[\![q]\!]_p\,k) : X(q^+[p^+/a])}}}$$

□

C. Proof of the embedding into Lepigre's system

We begin by extending Lepigre's system to be able to type stacks. We denote by $A^{\perp\!\!\!\perp}$ the type A when typing a stack, in the same fashion we use to go from a type A in a left rule of two-sided sequent to the type $A^{\perp\!\!\!\perp}$ in a one-sided sequent (see the remark at the end of Section 1.3). We also add a distinguished bottom stack $\bullet$ to the syntax, which is given the most general type $\perp^{\perp\!\!\!\perp}$. We change the rules $(*)$ and (μ) of the original type system in [17] and add rules for stacks, whose definitions are guided by the proof of the adequacy [17, Theorem 6] in particular by the $(\Rightarrow_e)$-case.

We shall now show that these rules are adequate with respect with the realizability model defined in [17, Section 2].

Proposition 22 (Adequacy)**.** *Let* Γ *be a (valid) context,* A *be a formula with* $FV(A) \subset dom(\Gamma)$ *and* σ *be a substitution realizing* Γ.

- *If* $\Gamma \vdash_{val} v : A$ *then* $v\sigma \in [\![A]\!]_\sigma$,
- *if* $\Gamma \vdash \pi : A^{\perp\!\!\!\perp}$ *then* $\pi\sigma \in [\![A]\!]_\sigma^{\perp}$,
- *if* $\Gamma \vdash t : A$ *then* $t\sigma \in [\![A]\!]_\sigma^{\perp\perp}$.

$$\frac{}{\Gamma \vdash \bullet : \bot^{\perp\!\!\!\perp}}\ \bullet \qquad \frac{}{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash \alpha : A^{\perp\!\!\!\perp}}\ \alpha$$

$$\frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash t : A}{\Gamma \vdash \mu\alpha.t : A}\ \mu \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash \pi : A^{\perp\!\!\!\perp}}{\Gamma \vdash t * \pi : B}\ *$$

$$\frac{\Gamma \vdash \pi : (A[x := t])^{\perp\!\!\!\perp}}{\Gamma \vdash \pi : (\forall x A)^{\perp\!\!\!\perp}}\ \forall_l \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash \pi : B^{\perp\!\!\!\perp}}{\Gamma \vdash t \cdot \pi : (A \Rightarrow B)^{\perp\!\!\!\perp}}\ \Rightarrow_l$$

$$\frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash \pi : B^{\perp\!\!\!\perp}}{\Gamma \vdash [t]\pi : A^{\perp\!\!\!\perp}}\ \texttt{let}$$

Figure 14. Extension of Lepigre's typing rules for stacks

Proof. The proof is done by induction on the typing rules, we only need to do the proof for the rules we define above (all the other cases correspond to the proof of [17, Theorem 6]).

(•) By definition, we have $[\![\bot]\!]_\sigma = [\![\forall X.X]\!]_\sigma = \emptyset$, thus for any stack π, we have $\pi \in [\![\bot]\!]^\bot_\sigma = \Pi$. In particular, $\bullet \in [\![\bot]\!]^\bot_\sigma$.

(α) By hypothesis, σ realizes $\Gamma, \alpha : A^{\perp\!\!\!\perp}$ from which we obtain $\alpha\sigma = \sigma(\alpha) \in [\![A]\!]^\bot_\sigma$.

($*$) We need to show that $t\sigma * \pi\sigma \in [\![B]\!]^{\bot\bot}_\sigma$, so we take $\rho \in [\![B]\!]^\bot_\sigma$ and show that $(t\sigma * \pi\sigma) * \rho \in \perp\!\!\!\perp$. By anti-reduction, it is enough to show that $(t\sigma * \pi\sigma) \in \perp\!\!\!\perp$. This is true by induction hypothesis, since $t\sigma \in [\![A]\!]^{\bot\bot}_\sigma$ and $\pi\sigma \in [\![A]\!]^\bot_\sigma$.

(μ) The proof is the very same as in [17, Theorem 6].

($\forall_l$) By induction hypothesis, we have that $\pi\sigma \in [\![A[x := t]]\!]^\bot_\sigma$. We need to show that $[\![A[x := t]]\!]^\bot_\sigma \subseteq [\![\forall x A]\!]^\bot_\sigma$, which follows from the fact $[\![\forall x A]\!]_\sigma \subseteq [\![A[x := t]]\!]_\sigma$.

($\Rightarrow_l$) If t is a value v, by induction hypothesis, we have that $v\sigma \in [\![A]\!]_\sigma$ and $\pi\sigma \in [\![B]\!]^\bot_\sigma$ and we need to show that $v\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\bot_\sigma$. The proof is already done in the case ($\Rightarrow_e$) (see [17, Theorem 6]). Otherwise, by induction hypothesis, we have that $t\sigma \in [\![A]\!]^{\bot\bot}_\sigma$ and $\pi\sigma \in [\![B]\!]^\bot_\sigma$ and we need to show that $t\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\bot_\sigma$. So we consider $\lambda x.u \in [\![A \Rightarrow B]\!]_\sigma$, and show that $\lambda x.u * t\sigma \cdot \pi\sigma \in \perp\!\!\!\perp$. We can take a reduction step, and prove instead that $t\sigma * [\lambda x.u]\pi\sigma \in \perp\!\!\!\perp$. This amounts to showing that $[\lambda x.u]\pi \in [\![A]\!]^\bot_\sigma$, which is already proven in the case ($\Rightarrow_e$).

(`let`) We need to show that $v \in [\![A]\!]_\sigma$, $v * [t\sigma]\pi\sigma \in \perp\!\!\!\perp$. Taking a steap of reduction, it is enough to have $t\sigma * v \cdot \pi\sigma \in \perp\!\!\!\perp$. This is true since by induction hypothesis, we have $t\sigma \in [\![A \Rightarrow B]\!]^{\bot\bot}_\sigma$ and $\pi\sigma \in [\![B]\!]^\bot_\sigma$, thus $v \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\bot_\sigma$.

□

It only remains to show that the translation we defined in Figure 12 preserve typing to conclude the proof of Proposition 19.

Lemma 23. *If $\Gamma \vdash p : A \mid \Delta$ (in* dL*), then $\Gamma \cup \Delta \vdash [\![p]\!]_p : A^*$ (in Lepigre's extended system). The same holds for contexts, and if $c : \Gamma \vdash \Delta$ then $\Gamma \cup \Delta \vdash [\![c]\!]_c : \bot$.*

Proof. The proof is an induction over the typing rules. Note that the translation of delimited continuations allows us to avoid dependencies conflict in the typing derivation. □

As a corollary we get a proof of the adequacy of dL typing rules with respect to Lepigre's realizability model.

Proposition 19. *If $\Gamma \vdash p : A \mid \Delta$ and σ is a substitution realizing $(\Gamma \cup \Delta)^*$, then $[\![p]\!]_p\sigma \in [\![A^*]\!]^{\bot\bot}_\sigma$.*

This immediately implies the soundness of dL, since a closed proof p of type $\bot$ would be translated in a realizer of $\top \to \bot$, so that $[\![p]\!]_p\ \lambda x.x$ would be a realizer of $\bot$, which is impossible. Furthermore, the translation clearly preserves the normalization (that is that for any c, if c does not normalize then neither does $[\![c]\!]_c$), and thus the adequacy has for consequence the normalization of dL.

Theorem 18. *For any proof p in* dL*, we have: $\not\vdash p : \bot$.*