A Classical Sequent Calculus with Dependent Types

Étienne Miquey

A Classical Sequent Calculus with Dependent Types

ÉTIENNE MIQUEY, Université Paris-Diderot and Universidad de la República

Dependent types are a key feature of the type systems used by the proof assistants based on the intuitionistic Curry-Howard correspondence. On the other hand, this correspondence can be extended to classical logic provided the language of proofs is enriched with control operators. Alas, control operators are known to misbehave in the presence of dependent types, unless dependencies are restricted to values. Besides, while sequent calculi smoothly support abstract machine and continuation-passing style interpretations, there is no such presentation of a language with dependent types. 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.

We start from the call-by-value version of the $\lambda\mu\tilde{\mu}$ 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 [23], and present a methodology to transfer properties from this system to our own.

CCS Concepts: • **Theory of computation → Type theory**; *Proof theory*;

Additional Key Words and Phrases: dependent types, sequent calculus, classical logic, control operators, call-by-value, delimited continuations, continuation-passing style translation, value restriction

ACM Reference format:
Étienne Miquey. 2017. A Classical Sequent Calculus with Dependent Types. 1, 1, Article 1 (May 2017), 44 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

1.1 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 Paulin-Mohring's calculus of inductive constructions for Coq [6], and Martin-Löf's type theory [24] for Agda), and lack classical logic.

In 1990, Griffin discovered [15] that the control operator `call/cc` (short for *call with current continuation*) of the Scheme programming language could be typed by Peirce's law $((A \to B) \to A) \to A$), thus extending the formulæ-as-types interpretation [20]. As Peirce's law is known to imply, in an intuitionistic framework, all the other forms of classical reasoning (excluded middle,

XXXX-XXXX/2017/5-ART1 $15.00
https://doi.org/0000001.0000001

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 [28], Barbanera and Berardi's symmetric λ-calculus [3], Krivine's λ_c-calculus [22], Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [7]...

Nevertheless, dependent types are known to misbehave in the presence of control operators, causing a degeneracy of the domain of discourse [17]. 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 [18] and Lepigre [23] proposed some restrictions on the dependent types to tackle the issue in the case of a proof system in natural deduction, while Blot [5] designed a hybrid realizability model where dependent types are restricted to an intuitionistic fragment.

Interestingly, the problems are due to control operators in presence of dependent types are very similar to the problems that might be cause to side-effects in functional programming. Specifically, control operators can be understood as one particular side-effect. Indeed, side-effects—that are impure computations in functional programming—are interpreted by means of monads, in the same way as control operators can be interpreted through the continuation monad. The more general interplay of dependent types and several computational effects (*e.g.* divergence, I/O, local references, exceptions) has been studied by Ahman et al [1] or Vákár [31]. More recently, an article by Pédrot and Tabareau proposed a systematical way to add effects to type theory [30], but the continuation monads lacks the properties to fit the pictures.

1.2 Call-by-value and value restriction

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 (or other kind of side-effects), soundness often turns out to be subtle to preserve in call-by-value. The first issues in call-by-value in the presence of side-effects were related to references [33] and polymorphism[16]. In both cases, a simple and elegant solution (but way too restrictive in practice [14, 23]) 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 dependent types and a control operator[23], 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 [18].

1.3 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 $\lambda\mu\tilde{\mu}$-calculus [7] 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 [8]. As a matter of fact, the original motivation for this work was the seek for a program translation for Herbelin's dPAω system (that already encompassed control operators and dependent types) to justify its soundness. However, this calculus was presented in a natural deduction style, and such a translation was really harsh to obtain. We thus developed the framework presented in this paper to dispose of an intermediate language more suitable for a continuation-passing style translation.

Besides, while we consider in this article the specific case of a calculus with classical logic, the sequent calculus presentation itself is responsible for another difficulty. As we will see, the usual call-by-value strategy of the $\lambda\mu\tilde{\mu}$-calculus causes subject reduction to fail, and this would already happen in an intuitionistic type theory. We claim that the solutions we give in this article also provides us with solutions in the intuitionistic case. 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 (dependently) typed compilation of a system with dependent types such as Coq.

1.4 Delimited continuations and CPS translation

The main difficulty to design a sequent calculus with dependent types resides in the fact that the natural reduction is not safe with respect with the type. As we discuss in Section 2.6, the problem can be understood as a desynchronization of the type system with the reduction. A simple solution to resolve this, presented in Section 2, consists in the addition of an explicit list of dependencies in the type derivations. This solution has the advantage to be give a calculus that is very close to the original, but is not enough to derive the continuation-passing style translation.

We thus present a second way to solve this issue introducing delimited continuations [2], which are used to force the purity needed for dependent types in an otherwise impure language. It also justifies the relaxation of the value restriction and leads to the definition of the negative-elimination-free fragment (Section 3). Besides, it permits the design in Section 4 of a continuation-passing style translation that preserves dependent types and allows for proving the soundness of our system. Finally, it also provides us with a way to embed our calculus into Lepigre's calculus, as we shall see in Section 5.

1.5 Contributions of the paper

Our main contributions in this paper are:

- we soundly combine dependent types and control operators by mean of a syntactic restriction to the negative-elimination-free fragment;
- we give a sequent calculus presentation and solve the type-soundness issues it raises in two different ways;
- our first solution simply relies on a dependencies list that is added to the type system
- our second solution uses delimited continuations to ensure consistency with dependent types and provides us with a CPS translation (carrying dependent types) to a calculus without control operator;

$$\begin{array}{rrcl} \text{Proofs} & p & ::= & a \mid \lambda a.p \mid \mu\alpha.c \\ \text{Contexts} & e & ::= & \alpha \mid p \cdot e \mid \tilde{\mu}a.c \\ \text{Commands} & c & ::= & \langle p \| e \rangle \end{array}$$

(a) Syntax

$$\begin{array}{lcll} \langle p \| \tilde{\mu}a.c \rangle & \to & c[a := p] & p \in \mathcal{V} \\ \langle \mu\alpha.c \| e \rangle & \to & c[\alpha := e] & e \in \mathcal{E} \\ \langle \lambda a.p \| u \cdot e \rangle & \to & \langle u \| \tilde{\mu}a.\langle p \| e \rangle \rangle & \end{array}$$

(b) Reduction rules

$$\frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta}{\langle t \| e \rangle : (\Gamma \vdash \Delta)}\ \textsc{Cut}$$

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash a : A \mid \Delta}\ \mathrm{Ax}_r \qquad \frac{\Gamma, a : A \vdash p : B \mid \Delta}{\Gamma \vdash \lambda a.p : A \to B \mid \Delta}\ \to_r \qquad \frac{c : (\Gamma \vdash \Delta, \alpha : A)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\ \mu$$

$$\frac{(\alpha : A) \in \Delta}{\Gamma \mid \alpha : A \vdash \Delta}\ \mathrm{Ax}_l \qquad \frac{\Gamma \vdash p : A \mid \Delta \qquad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid p \cdot e : A \to B \vdash \Delta}\ \to_l \qquad \frac{c : (\Gamma, a : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}a.c : A \vdash \Delta}\ \tilde{\mu}$$

(c) Typing rules

Fig. 1. The $\lambda\mu\tilde{\mu}$-calculus

- we relate our system to Lepigre's calculus, which offers an additional way of proving the consistency of our system.

This paper is an extended and revised version of the article presented at ESOP 2017 [26].

2 A MINIMAL CLASSICAL LANGUAGE WITH DEPENDENT TYPES

2.1 A short primer to the $\lambda\mu\tilde{\mu}$-calculus

We recall here the spirit of the $\lambda\mu\tilde{\mu}$-calculus, for further details and references please refer to the original article [7]. The syntax and reduction rules (parameterized over a sets of proofs $\mathcal{V}$ and a set of evaluation contexts $\mathcal{E}$) are given in Figure 1 where $\tilde{\mu}a.c$ can be read as a context **let** $a = [\,]$ **in** c. A command can be understood as a state of an abstract machine, representing the evaluation of a proof (the program) against a co-proof (the stack) that we call context. The μ operator comes from Parigot's $\lambda\mu$-calculus [28], $\mu\alpha$ binds a context to a context variable α in the same way $\tilde{\mu}a$ binds a proof to some proof variable a.

The $\lambda\mu\tilde{\mu}$-calculus can be seen as a proof-as-program correspondence between sequent calculus and abstract machines. Right introduction rules correspond to typing rules for proofs, while left introduction are seen as typing rules for evaluation contexts. For example, the left introduction rule of implication can be seen as a typing rule for pushing an element q on a stack e leading to the new stack $q \cdot e$:

$$\frac{\Gamma \vdash q : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid q \cdot e : A \to B \vdash \Delta}\ \to_l$$

Note that this presentation of sequent calculus involves three kinds of judgments: one with a focus on the right for programs, one with a focus on the left for contexts and one with no focus for states, as reflected on the Cut typing rule.

As for the reduction rules, we can see that there is a critical pair if $\mathcal{V}$ and $\mathcal{E}$ are not restricted enough:

$$c[\alpha := \tilde{\mu}x.c'] \quad \longleftarrow \quad \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \quad \longrightarrow \quad c'[x := \mu\alpha.c].$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a : (A \to B) \to A \vdash a : (A \to B) \to A \mid \bullet}\,\mathrm{Ax}_r \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\bullet, a' : A \vdash a' : A \mid \bullet}\,\mathrm{Ax}_r \quad \dfrac{}{\bullet \mid \alpha : A \vdash \alpha : A, \bullet}\,\mathrm{Ax}_l}{\langle a' \| \alpha \rangle : (\bullet, a' : A \vdash \alpha : A, \beta : B)}\,\mathrm{Cut}}{\bullet, a' : A \vdash \mu\beta.\langle a' \| \alpha \rangle : B \mid \alpha : A)}\,\mu}{\bullet \vdash \lambda a'.\mu\beta.\langle a' \| \alpha \rangle \mid \alpha : A}\,\to_r \quad \dfrac{}{\mid \alpha : A \vdash \alpha : A}\,\mathrm{Ax}_l}{\bullet \mid \lambda a'.\mu\beta.\langle a' \| \alpha \rangle \cdot \alpha : (A \to B) \to A \vdash \alpha : A)}\,\to_l}{\langle a \| \lambda a'.\mu\beta.\langle a' \| \alpha \rangle \cdot \alpha \rangle : (a : (A \to B) \to A \vdash \alpha : A)}\,\mathrm{Cut}$$
$$\dfrac{a : (A \to B) \to A \vdash \mu\alpha.\langle a \| \lambda a'.\mu\beta.\langle a' \| \alpha \rangle \cdot \alpha \rangle : A \mid}{\vdash \lambda a.\mu\alpha.\langle a \| \lambda a'.\mu\beta.\langle a' \| \alpha \rangle \cdot \alpha \rangle : ((A \to B) \to A) \to A \mid}\,\mu \quad \to_r$$

Fig. 2. Proof term for Peirce's law

The difference between call-by-name and call-by-value can be characterized by how this critical pair is solved, by defining $\mathcal{V}$ and $\mathcal{E}$ such that the two rules do not overlap. This justifies the definition of the subcategories of values $V \subset p$ and co-values $E \subset e$:

$$\text{(Values)} \qquad V ::= a \mid \lambda a.p \qquad\qquad \text{(Co-values)} \qquad E ::= \alpha \mid q \cdot e$$

The call-by-name evaluation strategy amounts to the case where $\mathcal{V} \triangleq \textit{Proofs}$ and $\mathcal{E} \triangleq \textit{Co-values}$, while call-by-value corresponds to $\mathcal{V} \triangleq \textit{Values}$ and $\mathcal{E} \triangleq \textit{Contexts}$. Both strategies can also been characterized through different CPS translations [7, Section 8].

Finally, it is worth nothing than the μ binder is a *control operator*, since it allows for catching evaluation contexts and backtracking further in the execution. This is then the key ingredient that makes the $\lambda\mu\tilde{\mu}$-calculus a proof system for classical logic. To illustrate this, let us draw the analogy with the `call/cc` operator of the Krivine λ_c-calculus [22]. Let us define the following proof terms:

$$\texttt{call/cc} \triangleq \lambda a.\mu\alpha.\langle a \| \boldsymbol{k}_\alpha \cdot \alpha \rangle \qquad\qquad \boldsymbol{k}_e \triangleq \lambda a'.\mu\beta.\langle a' \| e \rangle$$

The proof $\boldsymbol{k}_e$ can be understood as a proof term where the context e has been encapsulated. As expected, `call/cc` is a proof for Peirce's law (see Figure 2), which is known to imply other forms of classical reasoning.

Let observe the behavior of `call/cc`: in front of a context of the right shape (that is a stack $q \cdot e$ with e of type A), it will catch the context e thanks to the $\mu\alpha$ binder and reduce as follows:

$$\langle \lambda a.\mu\alpha.\langle a \| \boldsymbol{k}_\alpha \cdot \alpha \rangle \| q \cdot e \rangle \quad \to \quad \langle \mu\alpha.\langle q \| \boldsymbol{k}_\alpha \cdot \alpha \rangle \| e \rangle \quad \to \quad \langle q \| \boldsymbol{k}_e \cdot e \rangle$$

We notice that the proof term $\boldsymbol{k}_e = \lambda a'\mu\beta\langle a' \| e \rangle$ on top of the stack (which, if e was of type A, is of type $A \to B$, see Figure 2) contains a second binder $\mu\beta$. In front of a stack $q' \cdot e'$, this binder will now catch the context e' and replace it by the former context e:

$$\langle \lambda a'.\mu\beta.\langle a' \| e \rangle \| q' \cdot e' \rangle \quad \to \quad \langle \mu\beta.\langle q' \| e \rangle \| e' \rangle \quad \to \quad \langle q' \| e \rangle$$

This computation behavior corresponds exactly to the usual reduction rule for `call/cc` in the Krivine machine [22]:

$$\begin{aligned} \texttt{call/cc} \star t \cdot \pi \quad &\succ \quad t \star \boldsymbol{k}_\pi \cdot \pi \\ \boldsymbol{k}_\pi \star t \cdot \pi' \quad &\succ \quad t \star \pi \end{aligned}$$

2.2 Inconsistency of classical logic with dependent types

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. Let us shortly recap the argument of Herbelin [17] highlighting this phenomenon.

Let us adopt a stratified presentation of dependent types, by syntactically distinguishing *terms*—that represent mathematical objects—from *proof terms*–that represent mathematical proofs. We first consider a minimal logic of Σ-types and equality, that are formulas defined by::

$$A, B ::= t = u \mid \exists x^{\mathbb{N}}.A$$

where terms (only representing natural number) and proofs are defined by:

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

Let us explain the different proof terms by presenting their typing rules. First of all, refl is a proof term for equality, and subst $p\,q$ allows to use a proof of $t = u$ to convert a formula $A(t)$ into $A(u)$:

$$\frac{t \to u}{\Gamma \vdash \mathsf{refl} : t = u} \qquad \frac{\Gamma \vdash p : t = u \quad \Gamma \vdash q : B[t]}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u]}$$

Then, the pair (t, p) is a proof for an existential formula $\exists x^{\mathbb{N}} A$ where t is a witness for x and p is a certificate for $A(t)$. This implies that both formulas and proofs are dependent on terms which is usual in mathematics. What is less usual in mathematics is that dependent types also allow for terms to be dependent on proofs (and thus for formulas too), by means of the constructors wit p (that represents the witness of the proof p) and prf p (that represents the certificate of p):

$$\frac{\Gamma \vdash p : A(t)}{\Gamma \vdash (t, p) : \exists x^{\mathbb{N}} A} \qquad \frac{\Gamma \vdash (t, p) : \exists x^{\mathbb{N}} A}{\Gamma \vdash \mathsf{prf}\, p : A(\mathsf{wit}\, p)}$$

The reduction rules for this language, which are safe with respect to typing, are as one would expected:

$$\mathsf{wit}\,(t, p) \to t \qquad \mathsf{prf}\,(t, p) \to p \qquad \mathsf{subst}\,\mathsf{refl}\, p \to p$$

Starting from this minimal language, Herbelin showed that its classical extension with the control operators $\mathtt{call/cc}_k$ and throw k permits to derive a proof of $0 = 1$ [17]. Indeed, the addition to the type system of the typing rules for this operators (that are similar to those we presented in the previous section) :

$$\frac{\Gamma, k : \neg A \vdash p : A}{\Gamma \vdash \mathtt{call/cc}_k p : A} \qquad \frac{\Gamma, k : \neg A \vdash p : A}{\Gamma, k : \neg A \vdash \mathsf{throw}\, k\, p : B}$$

allows the definition of the following proof:

$$p_0 \triangleq \mathtt{call/cc}_k\,(0, \mathsf{throw}\, k\,(1, \mathsf{refl})) : \Sigma x.x = 1$$

Intuitively[1], such a proof catches the context, give 0 as witness (which is incorrect), and a certificate that will backtrack and give 1 as witness (which is correct) with a proof of the equality. Hence the seek of a witness by a term wit p_0 is likely[1] to reduce to 0, while the reduction of prf p_0 would backtracked before giving 1 as a witness and refl as certificate. As $p_0 : \exists x^{\mathbb{N}} x = 1$, we would have prf p_0 of type wit $p_0 = 1$, that is $0 = 1$.

The easiest and usual approach to prevent this is to impose a restriction to values for proofs appearing inside dependent types and operators. In particular, in the present example this would prevent us from writing wit p_0 and prf p_0.

[1]We do not want to enter into the details of the reduction rules etc., but rather focus on the intuition of the causes of the problem. For a detailed proof, please refer to [17, Section 2].

2.3 A minimal language with value restriction

In this section we will focus on this solution in the similar framework, and show how it permits to keep the proof system coherent. We shall see further in Section 3 how to relax this constraint. We follow here the stratified presentation[2]. from the previous section. We place ourselves in the framework of the $\lambda\mu\tilde{\mu}$-calculus to which we add:

- a category of *terms* which contain an encoding[3] of the natural numbers,
- proof terms (t,p) to inhabit the strong existential $\exists x^{\mathbb{N}} A$ together with the first and second projections, which we call wit (for terms) and prf (for proofs),
- a proof term refl for the equality of terms and a proof term subst for the convertibility of types over equal terms.

For simplicity reasons, we will only consider terms of type $\mathbb{N}$ throughout this paper. We address the question of extending the domain of terms in Section 6.1. The syntax of the corresponding system, that we call dL, is given by:

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

The formulas are defined by:

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

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 $A \to B$.

2.4 Reduction rules

As explained in Section 2.2, a backtracking proof might give place to different witnesses and proofs according to the context of reduction, leading to incoherences [17]. 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 3 (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[4].

Additionally, we denote by $A \equiv B$ the transitive-symmetric 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:

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

[2]This design choice is usually a matter of taste and might seem unusual for some readers. However, it has the advantage of clearly enlighten the different treatments for term and proofs through the CPS in the next sections.
[3]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.
[4]The reader might recognize the rule (ς) of Wadler's sequent calculus [32].

$$
\begin{array}{rcl|rcll}
\langle \mu\alpha.c \| e\rangle & \rightsquigarrow & c[e/\alpha] & \langle (t,p)\|e\rangle & \rightsquigarrow & \langle p\|\tilde{\mu}a.\langle (t,a)\|e\rangle\rangle & (p \notin V)\\
\langle V\|\tilde{\mu}a.c\rangle & \rightsquigarrow & c[V/a] & \langle \mathsf{prf}\ (t,V)\|e\rangle & \rightsquigarrow & \langle V\|e\rangle & \\
\langle \lambda a.p\|q\cdot e\rangle & \rightsquigarrow & \langle q\|\tilde{\mu}a.\langle p\|e\rangle\rangle & \langle \mathsf{subst}\ p\ q\|e\rangle & \rightsquigarrow & \langle p\|\tilde{\mu}a.\langle \mathsf{subst}\ a\ q\|e\rangle\rangle & (p \notin V)\\
\langle \lambda x.p\|t\cdot e\rangle & \rightsquigarrow & \langle p[t/x]\|e\rangle & \langle \mathsf{subst\ refl}\ q\|e\rangle & \rightsquigarrow & \langle q\|e\rangle &
\end{array}
$$

$$
\mathsf{wit}\ (t,V) \to t \qquad\qquad t \to t' \Rightarrow c[t] \rightsquigarrow c[t']
$$

Fig. 3. Reduction rules of dL

2.5 Typing rules

As we previously explained, in this section we 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). Imagine that the type system of the $\lambda\mu\tilde{\mu}$-calculus has been extended to allows dependent product rather than (non-dependent) implication. A typing derivation, using this type system, of the corresponding command would of the form:

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

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-hand side, we see that p, whose type is $B[a]$, is now cut with e whose 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 before the substitution of a by q[5] is performed e 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:

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

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:

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

[5]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.

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

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

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

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

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

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

$$\frac{\Gamma \vdash p : A \mid \Delta; \sigma \quad A \equiv B}{\Gamma \vdash p : B \mid \Delta; \sigma} \ \equiv_r \qquad \frac{\Gamma \mid e : A \vdash \Delta; \sigma \quad A \equiv B}{\Gamma \mid e : B \vdash \Delta; \sigma} \ \equiv_l$$

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

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

Fig. 4. Typing rules of dL

We define a dependencies list σ as a list binding pairs of proof terms[6]:

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

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 $q \in \mathcal{D}$:

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

The dependencies list is filled while going up in the typing tree, and can be used when typing a command $\langle p \| e \rangle$ to resolve a potential inconsistency between their types:

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

[6]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.

The reader familiar with type theories with explicit substitutions [11] can think of the dependencies list used when typing a command c as a fragment of the explicit substitution that will be available when c will reduce. Another remark is that our choice of design for the Cut is somehow arbitrary, in the sense that we choose to check whether B is in A_σ. We could equivalently have checked whether the condition $\sigma(A) = \sigma(B)$ holds, where $\sigma(A)$ refers to the type A where for each binding $p|q \in \sigma$ with $q \in \mathcal{D}$, all the occurences of p have been replaced by q.

Furthermore, when typing a stack ($\to_l$-rule), we need to drop the open binding in the dependencies list[7]. We introduce the notation $\Gamma \mid e : A \vdash \Delta; \sigma\{\cdot|\dagger\}$ to denote that the dependency to be produced is irrelevant and can be dropped. This avoid us 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 4, 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 $\lambda\mu\tilde{\mu}$-calculus [7]. 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 $(\alpha : A^{\perp\!\!\!\perp})$.

2.6 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':

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

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

Lemma 2.1 (Dependencies weakening). *For any dependencies list σ we have:*

$$1.\ \forall V (\sigma\{V|V\} \Rrightarrow \sigma) \qquad\qquad 2.\ \forall \sigma' (\sigma \Rrightarrow \sigma\sigma').$$

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

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

Corollary 2.2. *If $\sigma \Rrightarrow \sigma'$, then for any p, e, Γ, Δ:*

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

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

Lemma 2.3 (Safe term substitution). *If $\Gamma \vdash t : \mathbb{N} \mid \Delta; \varepsilon$ then:*

[7] It is easy to convince ourself that when typing a command $\langle p \| q \cdot \tilde{\mu} a.c \rangle$ with $\{\cdot|p\}$, the "correct" dependency within c should be $\{a|\mu\alpha\langle p \| q \cdot \alpha\rangle\}$, where the right proof is not a value. Furthermore, this dependency is irrelevant since there is no way to produce such a command where a type adjustment with respect to a needs to be made in c.

(1) $c : (\Gamma, x : \mathbb{N}, \Gamma' \vdash \Delta; \sigma) \Rightarrow c[t/x] : (\Gamma, \Gamma'[t/x] \vdash \Delta[t/x]; \sigma[t/x])$,
(2) $\Gamma, x : \mathbb{N}, \Gamma' \vdash q : B \mid \Delta; \sigma \Rightarrow \Gamma, \Gamma'[t/x] \vdash q[t/x] : B[t/x] \mid \Delta[t/x]; \sigma[t/x]$,
(3) $\Gamma, x : \mathbb{N}, \Gamma' \mid e : B \vdash \Delta; \sigma \Rightarrow \Gamma, \Gamma'[t/x] \mid e[t/x] : B[t/x] \vdash \Delta[t/x]; \sigma[t/x]$,
(4) $\Gamma, x : \mathbb{N}, \Gamma' \vdash u : \mathbb{N} \mid \Delta; \sigma \Rightarrow \Gamma, \Gamma'[t/x] \vdash u[t/x] : \mathbb{N} \mid \Delta[t/x]; \sigma[t/x]$.

Lemma 2.4 (Safe value substitution). *If* $\Gamma \vdash V : A \mid \Delta; \varepsilon$ *then:*

(1) $c : \Gamma, a : A, \Gamma' \vdash \Delta; \sigma \Rightarrow c[V/a] : \Gamma, \Gamma'[V/a] \vdash \Delta[V/a]; \sigma[V/a]$,
(2) $\Gamma, a : A, \Gamma' \vdash q : B \mid \Delta; \sigma \Rightarrow \Gamma, \Gamma'[V/a] \vdash q[V/a] : B[V/a] \mid \Delta[V/a]; \sigma[t/x]$,
(3) $\Gamma, a : A, \Gamma' \mid e : B \vdash \Delta; \sigma \Rightarrow \Gamma, \Gamma'[V/a] \mid e[V/a] : B[V/a] \vdash \Delta[V/a]; \sigma[V/a]$,
(4) $\Gamma, a : A, \Gamma' \vdash u : \mathbb{N} \mid \Delta; \sigma \Rightarrow \Gamma, \Gamma'[V/a] \vdash u[V/a] : \mathbb{N} \mid \Delta[V/a]; \sigma[V/a]$.

Lemma 2.5 (Safe context substitution). *If* $\Gamma \mid e : A \vdash \Delta; \varepsilon$ *then:*

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

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 2.6 (Subject reduction). *If* c, c' *are two commands of* dL *such that* $c : (\Gamma \vdash \Delta)$ *and* $c \rightsquigarrow c'$, *then* $c' : (\Gamma \vdash \Delta)$.

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, writing $c : \Gamma \vdash \Delta$ 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-hand side is of the form:

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

  We first deduce $A[t/x] \equiv B[t/x]$ from the hypothesis $\forall x^{\mathbb{N}}.A \equiv \forall x^{\mathbb{N}}.B$. Then using that $\Gamma, x : \mathbb{N} \vdash p : A \mid \Delta$ and $\Gamma \vdash t : \mathbb{N} \mid \Delta$, by Lemma 2.3 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 2.2 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-hand side is of the form:

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

  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:

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

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

  using Corollary 2.2 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-hand side 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 2.5.
- **Case** $\langle V \| \tilde{\mu} a.c \rangle \rightsquigarrow c[V/a]$.

  A typing proof for the command on the left-hand side is of the form:

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

  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 2.4. We finally get a proof for $c[V/a] : \Gamma \vdash \Delta; \varepsilon$ by Lemma 2.1.
- **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-hand side is of the form:

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

We can build the following derivation:

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

where $\Pi_{(t,a)}$ is as expected, observing that since $p \notin \mathcal{D}$, the binding $\{\cdot \mid (t,p)\}$ is the same as $\{\cdot \mid \dagger\}$, and we can apply Corollary 2.2 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-hand side is of the form:

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

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

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

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

  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.

□

2.7 Soundness

We give here a proof of the soundness of dL with a value restriction. The proof is based on an embedding to the $\lambda\mu\tilde{\mu}$-calculus extended, whose syntax and rules are given in Figure 5. A more interesting proof through a continuation-passing translation is presented in Section 4.

We first show that typed commands of dL normalize by translation to the simply-typed $\lambda\mu\tilde{\mu}$-calculus with pairs (*i.e.* extended with proofs of the form (p_1, p_2) and contexts of the form $\tilde{\mu}(a_1, a_2).c$).

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

(a) Syntax

$$\frac{\Gamma \vdash p_1 : A_1 \mid \Delta \quad \Gamma \vdash p_2 : A_2 \mid \Delta}{\Gamma \vdash (p_1,p_2) : A_1 \wedge A_2 \mid \Delta}\ \wedge_r \qquad \frac{c : \Gamma, a_1 : A_1, a_2 : A_2 \vdash \Delta}{\Gamma \mid \tilde{\mu}(a_1,a_2).c : A_1 \wedge A_2 \vdash \Delta}\ \wedge_l$$

(b) Typing rules

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

(c) Reduction rules

Fig. 5. $\lambda\mu\tilde{\mu}$-calculus with pairs

We do not consider here a particular reduction strategy, and take $\rightarrowtail$ to be the contextual closure of the rules given in Figure 5.

The translation essentially consists in erasing the dependencies in types, turning the dependent products into arrows and the dependent sum into a pair. The erasure procedure is defined by:

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

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

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

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

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 $\bar{n} \in V$. The distinct translations for $\mathsf{subst}\, V\, q$ and $\mathsf{subst}\, p\, q$ are just a hack to simplify the proof of Proposition 2.9.

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

Lemma 2.7. *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. □

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

Proposition 2.8. *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, $\mathrm{refl}^* = \lambda x.x : \mathbb{N}^* \to \mathbb{N}^*$.

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

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

then $\mathrm{subst}\, p\, q^* = \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha\rangle \| \alpha\rangle\rangle : B[u/x]^*$. According to Lemma 2.7, 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:

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

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

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

Every other cases follows directly from the induction hypothesis.

□

We can then deduce the normalization of dL from the normalization of the $\lambda\mu\tilde{\mu}$-calculus [29], by showing that the translation preserves the normalization in the sense that if c does not normalize, neither does c^*.

Proposition 2.9. *If c is a command such that c^* normalizes, then c normalizes.*

Proof. We will actually prove a slightly 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 wit $(t,V) \to t$:

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

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

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

Case $\langle \text{subst refl}\,q\|e\rangle \rightsquigarrow \langle q\|e\rangle$:

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

Case $\langle \text{subst}\,p\,q\|e\rangle \rightsquigarrow \langle p\|\tilde{\mu}a.\langle \text{subst}\,a\,q\|e\rangle\rangle$ (with $p \notin V$):

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

□

Theorem 2.10. *If $c : (\Gamma \vdash \Delta;\varepsilon)$, then c normalizes.*

Proof. Proof by contradiction: if c does not normalize, then by Proposition 2.9 neither does c^*. However, by Proposition 2.8 we have that $c^* : \Gamma^* \vdash \Delta^*$. This is absurd since any well-typed command of the $\lambda\mu\tilde{\mu}$-calculus normalizes [29]. □

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

Theorem 2.11 (Soundness). *For any $p \in$ dL, we have $\nvdash p : \bot$.*

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. □

2.8 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 $\lambda\mu\tilde{\mu}$-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 5 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 $\lambda\mu\tilde{\mu}$-calculus [7]. 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, as discussed in Section 2.5, the 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[8]:

$$(\llbracket q \rrbracket(\lambda a.\llbracket p \rrbracket))\llbracket e \rrbracket.$$

Assuming that q eventually produces a value V, the previous term would indeed reduce as follows:

$$(\llbracket q \rrbracket(\lambda a.\llbracket p \rrbracket))\llbracket e \rrbracket \to ((\lambda a.\llbracket p \rrbracket)\,\llbracket V \rrbracket)\,\llbracket e \rrbracket \to \llbracket p \rrbracket[\llbracket V \rrbracket/a]\,\llbracket e \rrbracket$$

Since $\llbracket p \rrbracket[\llbracket V \rrbracket/a]$ now has a type convertible to $(B[q] \to \bot) \to \bot$, the term that is produced in the end is well-typed.

The first observation is that if q, instead of producing a value, was a classical proof throwing the current continuation away (for instance $\mu\alpha.c$ where $\alpha \notin FV(c)$), this would lead to the unsafe reduction:

$$(\lambda\alpha.\llbracket c \rrbracket(\lambda a.\llbracket p \rrbracket))\llbracket e \rrbracket \to \llbracket c \rrbracket\,\llbracket e \rrbracket.$$

Indeed, through such a translation $\mu\alpha$ would only be able to catch the local continuation, and the term ends in $\llbracket c \rrbracket\llbracket e \rrbracket$ instead of $\llbracket c \rrbracket$. We thus need to restrict at least to proof terms that could not throw the current continuation.

The second observation is that such a term suggests the use of delimited continuations[9] to temporarily encapsulate the evaluation of q when reducing such a command:

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

This command is safe under the guarantee that q will not throw away the continuation $\tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle$, and will mimic the aforedescribed reduction:

$$\langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.\langle V \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.\langle p[V/a] \| \hat{\mathfrak{tp}} \rangle \| e \rangle \rightsquigarrow \langle p[V/a] \| e \rangle.$$

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 3, we will extend the language according to this intuition, and see how to design a continuation-passing style translation in Section 4.

3 EXTENSION OF THE SYSTEM

3.1 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 unsatisfactory. 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 $\lambda\mu\tilde{\mu}$-calculus with only one continuation variable (that we call $\star$ in Figure 6), and which do not contain application[10]. We insist on the fact that this defines a syntactic

[8] We will see in Section 4.4 that such a term could be typed by turning the type $A \to \bot$ of the continuation that $\llbracket q \rrbracket$ is waiting for into a (dependent) type $\Pi_{a:A}R[a]$ parameterized by R. This way we could have $\llbracket q \rrbracket : \forall R(\Pi_{a:A}R[a] \to R[q])$ instead of $\llbracket q \rrbracket : ((A \to \bot) \to \bot)$. 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 term $\llbracket p[q/a] \rrbracket\,\llbracket e \rrbracket$.

[9] We stick here to the presentations of delimited continuations in [2, 19], where $\hat{\mathfrak{tp}}$ is used to denote the top-level delimiter.

[10] 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.

subcategory of proofs, as $\star$ is only a notation to denote the possibility of defining a proof with only one continuation variable. For instance, $\mu\alpha.\langle\mu\beta\langle V\|\beta\rangle\|\alpha\rangle$ belongs to this category since:

$$\mu\alpha.\langle\mu\beta\langle V\|\beta\rangle\|\alpha\rangle =_\alpha \mu\star.\langle\mu\star\langle V\|\star\rangle\|\star\rangle$$

Interestingly, this corresponds exactly to the so-called *negative-elimination-free* (NEF) proofs of Herbelin [18]. 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.

Second, Lepigre defined in a recent work [23] 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ the system dL of Section 2 extended with delimited continuations, and define the fragment of *negative-elimination-free* proof terms (NEF). The syntax of both categories is given by Figure 6, the proofs in the NEF fragment are considered up to α-conversion for the context variables[11].

The reduction rules, given in Figure 6, are slightly different from the rules in Section 2. In the case $\langle\lambda a.p\|q\cdot e\rangle$ with $q\in$ NEF (resp. $\langle\mathrm{prf}\,p\|e\rangle$), 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 requiring that proof values only contain term values. We elude the problem of reducing terms, by defining meta-rules for them[12]. We add standard rules for delimited continuations [2, 19], 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.

3.2 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 4 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 2); and a last one to type commands $\langle p\|e\rangle$, where we observe that the premise for p is typed in regular mode.

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:

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

[11] 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.

[12] 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.

$$
\begin{array}{lll|ll}
\text{Proofs} & p ::= \cdots \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}} & & \textsc{nef} & p_N ::= V \mid (t, p_N) \mid \mu\star.c_N \\
\text{Delimited} & c_{\hat{\mathfrak{tp}}} ::= \langle p_N \| e_{\hat{\mathfrak{tp}}} \rangle \mid \langle p \| \hat{\mathfrak{tp}} \rangle & & \text{fragment} & \quad \mid \mathsf{prf}\, p_N \mid \mathsf{subst}\, p_N\, q_N \\
\text{continuations} & e_{\hat{\mathfrak{tp}}} ::= \tilde{\mu} a.c_{\hat{\mathfrak{tp}}} & & & c_N ::= \langle p_N \| e_N \rangle \\
 & & & & e_N ::= \star \mid \tilde{\mu} a.c_N
\end{array}
$$

(a) Language

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

where:

$$
V_t ::= x \mid n \qquad V_p ::= a \mid \lambda a.p \mid \lambda x.p \mid (V_t, V_p) \mid \mathsf{refl} \qquad c[t] ::= \langle (t, p) \| e \rangle \mid \langle \lambda x.p \| t \cdot e \rangle
$$

(b) Reduction rules

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

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

(c) Typing rules

Fig. 6. $\mathrm{dL}_{\hat{\mathfrak{tp}}}$: extension of dL with delimited continuations

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

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

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

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

By definition of the NEF proof terms, $\mu\star.c$ is of the general form $\mu\star.c = \mu\star.\langle p_1 \| \tilde{\mu} a_1.\langle p_2 \| \tilde{\mu} a_2.\langle \ldots \| \tilde{\mu} a_{n-1}.\langle p_n \| \star \rangle \rangle \rangle \rangle$. 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):

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

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:

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

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:

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

□

We can now prove subject reduction for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$.

Theorem 3.2 (Subject reduction). *If c, c' are two commands of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ such that $c : (\Gamma \vdash \Delta)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta)$.*

Proof. Actually, the proof is slightly easier than for Theorem 2.6, because most of the rules do not involve dependencies. We only give some of the key cases.

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

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

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

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

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

- **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:

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

  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:

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

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

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

- **Case** $\langle \mu \hat{\mathfrak{p}}.\langle p \| \hat{\mathfrak{p}} \rangle \| e \rangle \rightsquigarrow \langle p \| e \rangle$.
  This case is trivial, because in a typing derivation for the command on the left, $\hat{\mathfrak{p}}$ is typed with an empty dependencies list, thus the type of p,e and $\hat{\mathfrak{p}}$ coincides.

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

(a) Language and formulas

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

(b) Reduction rules

$$\frac{}{\Gamma \vdash \bar{n} : \mathbb{N}}\ \mathrm{Ax}_n \qquad \frac{(x:\mathbb{N}) \in \Gamma}{\Gamma \vdash x : \mathbb{N}}\ \mathrm{Ax}_t \qquad \frac{(a:A) \in \Gamma}{\Gamma \vdash a : A}\ \mathrm{Ax}_p$$

$$\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B}\ \to_i \qquad \frac{\Gamma \vdash p : \Pi_{a:A} B \quad \Gamma \vdash q : A}{\Gamma \vdash p\, q : B[q/a]}\ \to_E \qquad \frac{\Gamma, x : \mathbb{N} \vdash p : A}{\Gamma \vdash \lambda x.p : \forall x^{\mathbb{N}} A}\ \forall_i$$

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

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

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

(c) Type system

Fig. 7. Target language

- **Case** $\langle \mu\hat{\mathfrak{p}}.c \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.c' \| e \rangle$ with $c \rightsquigarrow c'$.
  This case corresponds exactly to Theorem 2.6, 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 3.1.

□

We invite the reader to check that interestingly, we could have already taken $\mathcal{D} \triangleq$ NEF in dL and still be able to prove the subject reduction. This shows that the relaxation to the NEF fragment is valid even without delimited continuations.

4 A CONTINUATION-PASSING STYLE TRANSLATION

We shall now see how to define a continuation-passing style translation from $\mathrm{dL}_{\hat{\mathfrak{p}}}$ to an intuitionistic type theory, and use this translation to prove the soundness of $\mathrm{dL}_{\hat{\mathfrak{p}}}$. Continuation-passing style translations are indeed very useful to embed languages with control operators into purely functional ones [7, 15]. From a logical point of view, they generally amount to negative translations that allow to embed classical logic into intuitionistic logic [9]. 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.

4.1 Target language

We choose for the target language an intuitionistic theory in natural deduction that has exactly the same elements as $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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, also presented in Figure 7, is defined as expected, with the addition of a second-order quantification that we will use in the sequel to refine the type of translations of terms and NEF proofs. As for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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:

$$0 = 0 \triangleright \top \qquad 0 = S(u) \triangleright \bot$$
$$S(t) = 0 \triangleright \bot \qquad S(t) = S(u) \triangleright t = u.$$

4.2 Translation of the terms

We can now define the continuation-passing style translation of terms, proofs, contexts and commands. The translation is given in Figure 8, in which $\boldsymbol{\lambda}^{\bullet}a.$ denotes the usual[13] term $\lambda a.t$. The translation for delimited continuation follows the intuition we presented in Section 2.8, and the definition for stacks $t \cdot e$ and $q \cdot e$ (with q NEF) inline the reduction producing a command with a delimited continuation. All the other rules are natural in the sense that they reflect the reduction rule $\rightsquigarrow$, except for the translation of pairs (t,p):

$$[\![(t,p)]\!]_p \triangleq \lambda k.[\![p]\!]_p([\![t]\!]_t\ (\lambda x a.k\ (x,a)))$$

The natural definition would have been $\lambda k.[\![t]\!]_t\ (\lambda u.[\![p]\!]_p\ \lambda q.k\ (u,q))$, however such a term would have been ill-typed (while this definition is correct, as we will see in the proof of Lemma 4.9). 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 $[\![t]\!]_t$. 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. The complete translation is given in Figure 8.

Before defining the translation of types, we first state a lemma expressing the fact that the translations of terms and NEF proof terms use the continuation they are given once and only once. In particular, it makes them compatible with delimited continuations and a parametric return type. This will allow us to refine the type of their translation.

LEMMA 4.1. *The translation satisfies the following properties:*

(1) *For any term t in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, there exists a term t^+ such that for any k we have $[\![t]\!]_t\ k \to^*_\beta k\ t^+$.*
(2) *For any NEF proof p_N, there exists a proof p_N^+ such that for any k we have $[\![p_N]\!]_p\ k \to^*_\beta k\ p_N^+$.*

In particular, we have :

$$[\![t]\!]_t\ \lambda x.x \to^*_\beta t^+ \qquad \text{and} \qquad [\![p_N]\!]_p\ \lambda a.a \to^*_\beta p_N^+$$

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

[13] The annotation is a technical trick that will be used for the proof of Proposition 4.2.

$$
\begin{array}{lll}
[\![V_t]\!]_t & \triangleq \lambda k.k\,[\![V_t]\!]_{V_t} & [\![x]\!]_{V_t} \triangleq x \\
[\![\mathsf{wit}\,p]\!]_t & \triangleq \lambda k.[\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet} q.k\,(\mathsf{wit}\,q)) & [\![n]\!]_{V_t} \triangleq \bar{n}
\end{array}
$$

$$
\begin{array}{lll}
[\![a]\!]_V & \triangleq a & [\![\mathsf{refl}]\!]_V \triangleq \mathsf{refl} \\
[\![\lambda a.p]\!]_V & \triangleq \boldsymbol{\lambda}^{\bullet} a.[\![p]\!]_p & [\![\lambda x.p]\!]_V \triangleq \boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p \\
[\![(V_t,V_p)]\!]_V & \triangleq ([\![V_t]\!]_V,[\![V]\!]_V) &
\end{array}
$$

$$
\begin{array}{lll}
[\![V]\!]_p & \triangleq \lambda k.k\,[\![V]\!]_V & [\![\mu\hat{\mathfrak{p}}.c]\!]_p \triangleq \lambda k.[\![c]\!]_{\hat{\mathfrak{p}}} k \\
[\![\mu\alpha.c]\!]_p & \triangleq \boldsymbol{\lambda}^{\bullet}\alpha.[\![c]\!]_c & \\
[\![\mathsf{prf}\,p]\!]_p & \triangleq \boldsymbol{\lambda}^{\bullet} k.([\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet} q\lambda k'.k'\,(\mathsf{prf}\,q)))\,k & \\
[\![(t,p)]\!]_p & \triangleq \boldsymbol{\lambda}^{\bullet} k.[\![p]\!]_p([\![t]\!]_t\,(\lambda x\boldsymbol{\lambda}^{\bullet} a.k\,(x,a))) & \\
[\![\mathsf{subst}\,V\,q]\!]_p & \triangleq \lambda k.[\![q]\!]_p(\boldsymbol{\lambda}^{\bullet} q'.k\,(\mathsf{subst}\,[\![V]\!]_V\,q'))) & \\
[\![\mathsf{subst}\,p\,q]\!]_p & \triangleq \lambda k.[\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet} p'.[\![q]\!]_p(\boldsymbol{\lambda}^{\bullet} q'.k\,(\mathsf{subst}\,p'\,q'))) & (p \notin V)
\end{array}
$$

$$
\begin{array}{lll}
[\![\alpha]\!]_e & \triangleq \alpha & [\![\tilde{\mu}a.c]\!]_e \triangleq \boldsymbol{\lambda}^{\bullet} a.[\![c]\!]_c \\
[\![t\cdot e]\!]_e & \triangleq \lambda p.([\![t]\!]_t\,(\boldsymbol{\lambda}^{\bullet} v.p\,v))\,[\![e]\!]_e & \\
[\![q_N\cdot e]\!]_e & \triangleq \lambda p.([\![q_N]\!]_p\,(\boldsymbol{\lambda}^{\bullet} v.p\,v))\,[\![e]\!]_e & (q_N \in \textsc{nef}) \\
[\![q\cdot e]\!]_e & \triangleq \boldsymbol{\lambda}^{\bullet} p.[\![q]\!]_p\,(\boldsymbol{\lambda}^{\bullet} v.p\,v\,[\![e]\!]_e) & (q \notin \textsc{nef})
\end{array}
$$

$$
\begin{array}{lll}
[\![\langle p\|e\rangle]\!]_c & \triangleq [\![e]\!]_e\,[\![p]\!]_p & [\![\langle p\|\hat{\mathfrak{p}}\rangle]\!]_{\hat{\mathfrak{p}}} \triangleq [\![p]\!]_p \\
[\![\langle p\|e\rangle]\!]_{\hat{\mathfrak{p}}} & \triangleq [\![p]\!]_p\,[\![e]\!]_{e_{\hat{\mathfrak{p}}}} \quad (e \neq \hat{\mathfrak{p}}) & [\![\tilde{\mu}a.c]\!]_{e_{\hat{\mathfrak{p}}}} \triangleq \boldsymbol{\lambda}^{\bullet} a.[\![c]\!]_{\hat{\mathfrak{p}}}
\end{array}
$$

Fig. 8. Continuation-passing style translation

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

Fig. 9. Linearity of the translation for NEF proofs

case (t,p) with $p \in \textsc{nef}$ to give an insight of the proof.

$$
\begin{array}{lll}
[\![(t,p)]\!]_p\,k \to_\beta [\![p]\!]_p([\![t]\!]_t\,(\lambda x a.k\,(x,a))) & & \text{(by definition)} \\
\to_\beta ([\![t]\!]_t\,(\lambda x a.k\,(x,a)))\,p^+ & & \text{(by induction)} \\
\to_\beta (\lambda x a.k\,(x,a))\,t^+\,p^+ & & \\
\to_\beta (\lambda a.k\,(t^+,a))\,p^+ & & \text{(by induction)} \\
\to_\beta k\,(t^+,p^+) & &
\end{array}
$$

□

Moreover, we can verify by induction on the reduction rules for $\rightsquigarrow$ that the translation preserves the reduction:

PROPOSITION 4.2 (PRESERVATION OF REDUCTION). *Let c,c' be two commands of $\mathrm{dL}_{\hat{\mathfrak{p}}}$. If $c \rightsquigarrow c'$, then $[\![c]\!]_c =_\beta [\![c']\!]_c$*

As we shall see in the next section, we can in fact prove a finer result to show that any infinite reduction sequence in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ is responsible for an infinite reduction sequence through the translation. Using the preservation of typing (Proposition 4.10) together with the normalization of the target language, this will give us a proof of the normalization of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ for typed proof terms.

4.3 Normalization of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$

We will now prove that the translation is well-behaved with respect to the reduction. In practice, we are mainly interested in the preservation of normalization through the translation. Namely, we want to prove that if the image $[\![c]\!]_c$ of a command c is normalizing in the target language, then the command c is already normalizing in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. To this purpose, we roughly proceed as follows:

(1) we identify a set of reduction steps in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ which are directly reflected into a strictly positive number of reduction steps through the CPS;
(2) we show that the other steps alone can not form an infinite sequence of reduction;
(3) we deduce that every infinite sequence of reduction in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ give raises to an infinite sequence through the translation.

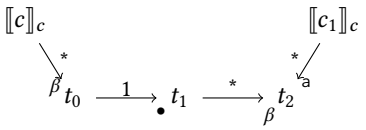
The first point corresponds thereafter to Proposition 4.5, the second one to the Proposition 4.6. As a matter of fact, the most difficult part is somehow anterior to these points. It consists in understanding *how* a reduction step can be reflected through the translation and why it is *enough* to ensure the preservation of normalization (that is the third point). Instead of stating the result directly and give a long and tedious proof of its correctness, we will rather sketch its main steps.

First of all, we split the reduction rule $\to_\beta$ into two different kinds of reduction steps:

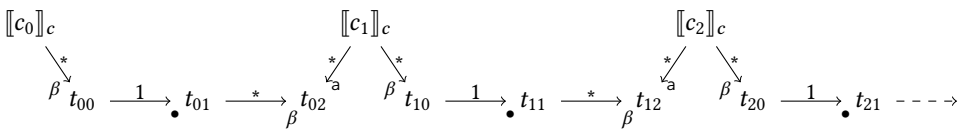
- *administrative reductions*, that we denote by $\longrightarrow_{\mathsf{a}}$, which correspond to continuation-passing and computationally irrelevant (w.r.t. to $\mathrm{dL}_{\hat{\mathfrak{tp}}}$) reduction steps. These are defined as the β-reduction steps of non-annotated λs.
- *distinguished reductions*, that we denote by $\longrightarrow_\bullet$, which correspond to the image of a reduction step through the translation. These are defined as every other rules, that is to say the β-reduction steps of annotated $\boldsymbol{\lambda}^\bullet$'s plus the rules corresponding to redexes formed with wit, prf and subst .

In other words, we define two deterministic reductions $\longrightarrow_\bullet$ and $\longrightarrow_{\mathsf{a}}$, such that the usual weak-head reduction $\to_\beta$ is equal to the union $\longrightarrow_\bullet \cup \longrightarrow_{\mathsf{a}}$. Our goal will be to prove that every infinite reduction sequence in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ will be reflected in the existence of an infinite reduction sequence for $\longrightarrow_\bullet$.

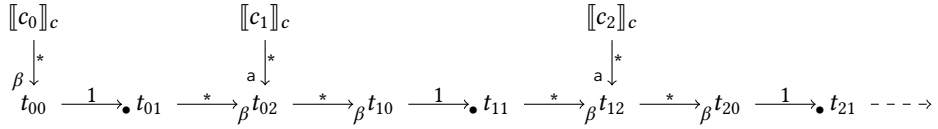
Second, let us assume for a while that we can show for any reduction $c \rightsquigarrow c'$ we obtain:



through the translation. Then by induction, it implies that if a command c_0 produces an infinite reduction sequence $c_0 \rightsquigarrow c_1 \rightsquigarrow c_2 \rightsquigarrow \ldots$, it is reflected through the translation by the following reduction scheme:



Using the fact that all reductions are deterministic, and that the arrow from $[\![c_1]\!]_c$ to t_{02} (and $[\![c_2]\!]_c$ to t_{12} and so on) can only contain steps of the reduction $\longrightarrow_{\mathsf{a}}$, the previous scheme in fact ensures us that we have:



This directly implies that $[\![c_0]\!]_c$ produces an infinite reduction sequence and thus is not normalizing. This would be the ideal situation, and if the aforementioned steps were provable as such, the proof would be over. Yet, our situation is more subtle, and we need to refine our analysis to tackle the problem. We shall briefly explain now why we can actually consider a slightly more general reduction scheme, while trying to remain concise on the justification. Keep in mind is that our goal is to preserve the existence of an infinite sequence of distinguished steps in spite of the generalization.

The first generalization is the possibility to allow distinguished reductions that are not in head positions, guided by the following proposition.

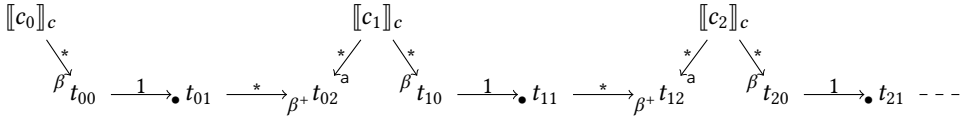
Proposition 4.3. *If $u \longrightarrow_\bullet u'$ and $t[u']$ does not normalize, then neither does $t[u]$.*

Proof. By induction on the structure of t, a very similar proof can be found in [21]. □

Following this idea, we define a new arrow $\xrightarrow{?}_\bullet$ by:

$$u \longrightarrow_\bullet u' \;\Rightarrow\; t[u] \xrightarrow{?}_\bullet t[u']$$

where $t[] ::= [] \mid t'(t[]) \mid \lambda x.t[]$, expressing the fact that a distinguished step can be perform somewhere in the term. We denote by $\longrightarrow_{\beta^+}$ reduction obtained by the union $\longrightarrow_\beta \cup \xrightarrow{?}_\bullet$ (which is no longer deterministic). Coming back to the thread thread scheme we described above, we can now generalize it with this arrow. Indeed, as we are only interested in getting an infinite reduction sequence from $[\![c_0]\!]_c$, the previous proposition ensures us that if t_{02} (t_{12}, etc.) does not normalize, it is enough to have an arrow $t_{01} \xrightarrow{*}_{\beta^+} t_{02}$ ($t_{11} \xrightarrow{*}_{\beta^+} t_{12}$, etc.) to deduce that t_{01} does not normalize either. Hence it is enough to prove that we have the following thread scheme, where we took advantage of this observation:

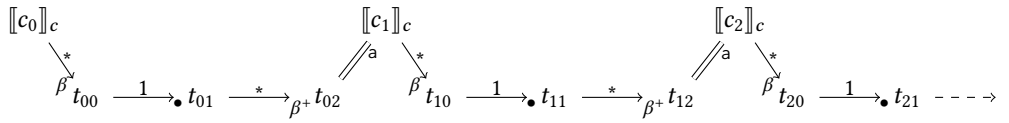


In the same spirit, if we define $=_{\mathsf{a}}$ to be the congruence over terms induced by the reduction $\longrightarrow_{\mathsf{a}}$, we can show that if a term has a redex for the distinguished relation in head position, then so does any (administratively) congruent term.

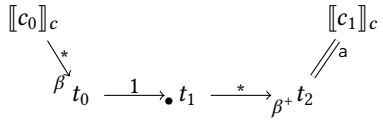
Proposition 4.4. *If $t \xrightarrow{1}_\bullet u$ and $t =_{\mathsf{a}} t'$, then there exists u' such that $t' \xrightarrow{1}_\bullet u'$ and $u =_{\mathsf{a}} u'$.*

Proof. By induction on t, observing that an administrative reduction can neither delete nor create redexes for $\longrightarrow_\bullet$. □

In other words, as we are only interested in the distinguished reduction steps, we can take the liberty to reason modulo the congruence $=_{\mathsf{a}}$. Notably, we can generalize one last time our reduction scheme, replacing the left (administrative) arrow from $[\![c_i]\!]_c$ by this congruence:



For all the reasons explained above, such a reduction scheme ensure us that there is an infinite reduction sequence from $[\![c_0]\!]_c$. So that, by induction, it is enough to show that for any reduction step $c_0 \rightsquigarrow c_1$, we have:



(1)

In fact, as explained in the preamble of this section, not all reduction steps can be reflected this way through the translation. There are indeed 4 reduction rules, that we identify hereafter, that might only be reflected into administrative reductions, and produce a scheme of this shape (which subsumes the former):

$$[\![c_0]\!]_c \longrightarrow_{\beta^+}^{*} t =_{\mathsf{a}} [\![c_1]\!]_c \tag{2}$$

This allows us to give a more precise statement about the preservation of reduction through the CPS translation.

Proposition 4.5. *Let c be two commands of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. If $c_0 \rightsquigarrow c_1$, then it is reflected through the translation into a reduction scheme (1), except for the rules:*

$$\begin{array}{rcl} \langle \mathsf{subst}\, p\, q \| e\rangle & \overset{p\notin V}{\rightsquigarrow} & \langle p \| \tilde{\mu} a.\langle \mathsf{subst}\, a\, q \| e\rangle\rangle \\ \langle \mathsf{subst}\ \mathsf{refl}\, q \| e\rangle & \rightsquigarrow & \langle q \| e\rangle \end{array} \qquad \begin{array}{rcl} \langle \mu \hat{\mathfrak{tp}}.\langle p \| \hat{\mathfrak{tp}}\rangle \| e\rangle & \rightsquigarrow & \langle p \| e\rangle \\ c[t] & \rightsquigarrow & c[t'] \end{array}$$

which are reflected in the reduction scheme (2).

Proof. The proof is done by induction on the reduction $\rightsquigarrow$ (see Figure 6). To ease the notations, we will often write $\boldsymbol{\lambda}^{\bullet} v.(\boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p)\, v \longrightarrow_{\bullet} \boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p$ where we perform α-conversion to identify $\boldsymbol{\lambda}^{\bullet} v.[\![p]\!]_p[v/x]$ and $\boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p$. Besides, to facilitate the comprehension of the steps corresponding to the congruence $=_{\mathsf{a}}$, we use an arrow $\overset{?}{\longrightarrow}_{\mathsf{a}}$ to denote the possibility of performing an administrative reduction not in head position, defined by:

$$u \longrightarrow_{\mathsf{a}} u' \;\Rightarrow\; t[u] \overset{?}{\longrightarrow}_{\mathsf{a}} t[u']$$

We write $\longrightarrow_{\mathsf{a}^+}$ the union $\longrightarrow_{\mathsf{a}} \cup \overset{?}{\longrightarrow}_{\mathsf{a}}$.

- **Case** $\langle \mu\alpha.c \| e\rangle \rightsquigarrow c[e/\alpha]$:
  We have:
  $$\begin{array}{rcl} [\![\langle \mu\alpha.c \| e\rangle]\!]_c & = & (\boldsymbol{\lambda}^{\bullet}\alpha.[\![c]\!]_c)[\![e]\!]_e \\ & \longrightarrow_{\bullet} & [\![c]\!]_c[[\![e]\!]_e/\alpha] = [\![c[e/\alpha]]\!]_c \end{array}$$
- **Case** $\langle \lambda a.p \| q \cdot e\rangle \rightsquigarrow \langle q \| \tilde{\mu} a.\langle p \| e\rangle\rangle$:
  We have:
  $$\begin{array}{rcl} [\![\langle \lambda a.p \| q \cdot e\rangle]\!]_c & = & (\lambda k.k\, (\boldsymbol{\lambda}^{\bullet} a.[\![p]\!]_p))\, \boldsymbol{\lambda}^{\bullet} p.[\![q]\!]_p\, (\boldsymbol{\lambda}^{\bullet} v.p\, v\, [\![e]\!]_e) \\ & \longrightarrow_{\mathsf{a}} & (\boldsymbol{\lambda}^{\bullet} p.[\![q]\!]_p\, (\boldsymbol{\lambda}^{\bullet} v.p\, v\, [\![e]\!]_e))\, \boldsymbol{\lambda}^{\bullet} a.[\![p]\!]_p \\ & \longrightarrow_{\bullet} & [\![q]\!]_p\, (\boldsymbol{\lambda}^{\bullet} v.(\boldsymbol{\lambda}^{\bullet} a.[\![p]\!]_p)\, v\, [\![e]\!]_e) \\ & \overset{?}{\longrightarrow}_{\bullet} & [\![q]\!]_p\, (\boldsymbol{\lambda}^{\bullet} a.[\![p]\!]_p\, [\![e]\!]_e) = [\![\langle q \| \tilde{\mu} a.\langle p \| e\rangle\rangle]\!]_c \end{array}$$

- **Case** $\langle \lambda a.p \| q_N \cdot e \rangle \overset{q_N \in \text{NEF}}{\rightsquigarrow} \langle \mu\hat{\mathfrak{tp}}.\langle q_N \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$:
We know by Lemma 4.1 that q_N being NEF, it will use and use once only the continuation it is applied to. Thus we know that if $k \longrightarrow_\bullet k'$, we have that:

$$[\![q_N]\!]_p\, k \xrightarrow{*}_\beta k\, q_N^+ \longrightarrow_\bullet k'\, q_N^+ \;{}_\beta\!\longleftarrow [\![q_N]\!]_p\, k'$$

and we can legitimately write $[\![q_N]\!]_p\, k \longrightarrow_\bullet [\![q_N]\!]_p\, k'$ in the sens that it corresponds to performing now a reduction that would have been performed in the future. Using this remark, we have:

$$\begin{array}{rcl}
[\![\langle \lambda a.p \| q_N \cdot e \rangle]\!]_c & = & (\lambda k.k\, (\boldsymbol{\lambda}^\bullet a.[\![p]\!]_p))\, \lambda p.([\![q_N]\!]_p\, (\boldsymbol{\lambda}^\bullet v.p\, v))\, [\![e]\!]_e \\
& \xrightarrow{2}_{\mathsf{a}} & ([\![q_N]\!]_p\, (\boldsymbol{\lambda}^\bullet v.(\boldsymbol{\lambda}^\bullet a.[\![p]\!]_p)\, v))\, [\![e]\!]_e \\
& \longrightarrow_\bullet & ([\![q_N]\!]_p\, (\boldsymbol{\lambda}^\bullet a.[\![p]\!]_p))\, [\![e]\!]_e \\
& {}_{\mathsf{a}}\!\longleftarrow & (\lambda k.([\![q_N]\!]_p\, (\boldsymbol{\lambda}^\bullet a.[\![p]\!]_p))\, k)\, [\![e]\!]_e = [\![\langle \mu\hat{\mathfrak{tp}}.\langle q_N \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle]\!]_c
\end{array}$$

- **Case** $\langle \lambda x.p \| V_t \cdot e \rangle \rightsquigarrow \langle p[V_t/x] \| e \rangle$:
Since V_t is a value (*i.e.* x or n), we have $[\![V_t]\!]_t = \lambda k.k\, [\![V_t]\!]_{V_t}$. In particular, it is easy to deduce that $[\![p[V_t/x]]\!]_p = [\![p]\!]_p[[\![V_t]\!]_{V_t}/x]$, and then we have:

$$\begin{array}{rcl}
[\![\langle \lambda x.p \| V_t \cdot e \rangle]\!]_c & = & (\lambda k.k\, (\boldsymbol{\lambda}^\bullet x.[\![p]\!]_p))\lambda p.([\![V_t]\!]_t\, (\boldsymbol{\lambda}^\bullet v.p\, v))\, [\![e]\!]_e \\
& \xrightarrow{2}_{\mathsf{a}} & ([\![V_t]\!]_t\, (\boldsymbol{\lambda}^\bullet v.(\boldsymbol{\lambda}^\bullet x.[\![p]\!]_p)\, v))\, [\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & ((\boldsymbol{\lambda}^\bullet v.(\boldsymbol{\lambda}^\bullet x.[\![p]\!]_p)\, v)\, [\![V_t]\!]_{V_t})\, [\![e]\!]_e \\
& \longrightarrow_\bullet & ((\boldsymbol{\lambda}^\bullet x.[\![p]\!]_p)\, [\![V_t]\!]_{V_t})\, [\![e]\!]_e \\
& \longrightarrow_\bullet & ([\![p]\!]_p[[\![V_t]\!]_{V_t}/x])\, [\![e]\!]_e = [\![p[V_t/x]]\!]_p\, [\![e]\!]_e = \langle p[V_t/x] \| e \rangle
\end{array}$$

- **Case** $\langle V \| \tilde{\mu}a.c \rangle \rightsquigarrow c[V_p/a]$:
Similarly to the previous case, we have $[\![V]\!]_p = \lambda k.k\, [\![V]\!]_V$ and thus $[\![c[V/x]]\!]_c = [\![p]\!]_p[[\![V]\!]_V/a]$.

$$\begin{array}{rcl}
[\![\langle V_p \| \tilde{\mu}a.c \rangle]\!]_c & = & (\lambda k.k\, [\![V]\!]_V)\boldsymbol{\lambda}^\bullet a.[\![c]\!]_c \\
& \longrightarrow_{\mathsf{a}} & (\boldsymbol{\lambda}^\bullet a.[\![c]\!]_c)\, [\![V]\!]_V \\
& \longrightarrow_\bullet & [\![c]\!]_c[[\![V]\!]_V/a] = [\![c[V/a]]\!]_c
\end{array}$$

- **Case** $\langle (V_t, p) \| e \rangle \overset{p \notin V}{\rightsquigarrow} \langle p \| \tilde{\mu}a.\langle (V_t, a) \| e \rangle \rangle$:
We have :

$$\begin{array}{rcl}
[\![\langle (V_t, p) \| e \rangle]\!]_c & = & (\boldsymbol{\lambda}^\bullet k.[\![p]\!]_p([\![V_t]\!]_t\, (\lambda x \boldsymbol{\lambda}^\bullet a.k\, (x, a)))\, [\![e]\!]_e \\
& \longrightarrow_\bullet & [\![p]\!]_p\, ([\![V_t]\!]_t\, (\lambda x \boldsymbol{\lambda}^\bullet a.[\![e]\!]_e\, (x, a))) \\
& \longrightarrow_{\mathsf{a}^+} & [\![p]\!]_p\, ((\lambda x \boldsymbol{\lambda}^\bullet a.[\![e]\!]_e\, (x, a))\, [\![V_t]\!]_{V_t}) \\
& \longrightarrow_{\mathsf{a}^+} & [\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a.[\![e]\!]_e\, ([\![V_t]\!]_{V_t}, a)) \\
& {}_{\mathsf{a}^+}\!\longleftarrow & [\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a.[\![(V_t, a)]\!]_p\, [\![e]\!]_e) \\
& {}_{\mathsf{a}^+}\!\longleftarrow & (\lambda k\, [\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a.[\![(V_t, a)]\!]_p\, k))\, [\![e]\!]_e = [\![\langle p \| \tilde{\mu}a.\langle (V_t, a) \| e \rangle \rangle]\!]_c
\end{array}$$

- **Case** $\langle \text{prf}\, p \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \text{prf}\, a \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$:
We have:

$$\begin{array}{rcl}
[\![\langle \text{prf}\, p \| e \rangle]\!]_c & = & \boldsymbol{\lambda}^\bullet k.([\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a \lambda k'.k'\, (\text{prf}\, a)))\, k)\, [\![e]\!]_e \\
& \longrightarrow_\bullet & ([\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a.\lambda k'.k'\, (\text{prf}\, a)))\, [\![e]\!]_e \\
& {}_{\mathsf{a}}\!\longleftarrow & (\lambda k.([\![p]\!]_p\, (\boldsymbol{\lambda}^\bullet a.\lambda k'.k'\, (\text{prf}\, a)))\, k)\, [\![e]\!]_e = [\![\langle \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}a.\langle \text{prf}\, a \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle]\!]_c
\end{array}$$

- **Case** $\langle \mathsf{prf}\,(V_t, V_p) \| e\rangle \rightsquigarrow \langle V_p \| e\rangle$:
  We have:

$$\begin{array}{rcl}
[\![\langle \mathsf{prf}\,(V_t,V_p)\|e\rangle]\!]_c & = & \boldsymbol{\lambda}^{\bullet}k.((\lambda k.k\,([\![V_t]\!]_V,[\![V_p]\!]_V))\,(\boldsymbol{\lambda}^{\bullet}q\lambda k'.k'\,(\mathsf{prf}\,q)))\,k)\,[\![e]\!]_e \\
& \longrightarrow_{\bullet} & ((\lambda k.k\,([\![V_t]\!]_V,[\![V_p]\!]_V))\,(\boldsymbol{\lambda}^{\bullet}q\lambda k'.k'\,(\mathsf{prf}\,q)))\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & ((\boldsymbol{\lambda}^{\bullet}q\lambda k'.k'\,(\mathsf{prf}\,q))([\![V_t]\!]_V,[\![V_p]\!]_V))\,[\![e]\!]_e \\
& \longrightarrow_{\bullet} & (\lambda k'.k'\,(\mathsf{prf}\,([\![V_t]\!]_V,[\![V_p]\!]_V)))\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & [\![e]\!]_e\,(\mathsf{prf}\,([\![V_t]\!]_V,[\![V_p]\!]_V)) \\
& \xrightarrow{?}_{\bullet} & [\![e]\!]_e\,[\![V_p]\!]_V \;{}_{\mathsf{a}}\!\longleftarrow [\![\langle V_p\|e\rangle]\!]_c
\end{array}$$

- **Case** $\langle \mathsf{subst}\,p\,q\|e\rangle \overset{p\notin V}{\rightsquigarrow} \langle p\|\tilde{\mu}a.\langle \mathsf{subst}\,a\,q\|e\rangle\rangle$:
  We have:

$$\begin{array}{rcl}
[\![\langle \mathsf{subst}\,p\,q\|e\rangle]\!]_c & = & (\lambda k.[\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet}a.[\![q]\!]_p(\boldsymbol{\lambda}^{\bullet}q'.k\,(\mathsf{subst}\,a\,q'))))\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & [\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet}a.[\![q]\!]_p(\boldsymbol{\lambda}^{\bullet}q'.[\![e]\!]_e\,(\mathsf{subst}\,a\,q'))) \\
& {}_{\mathsf{a}}\!\overset{?}{\longleftarrow} & [\![p]\!]_p\,(\boldsymbol{\lambda}^{\bullet}a.(\lambda k.[\![q]\!]_p(\boldsymbol{\lambda}^{\bullet}q'.k\,(\mathsf{subst}\,a\,q')))\,[\![e]\!]_e) \\
& = & [\![\langle p\|\tilde{\mu}a.\langle \mathsf{subst}\,a\,q\|e\rangle\rangle]\!]_c
\end{array}$$

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

$$\begin{array}{rcl}
[\![\langle \mathsf{subst}\,\mathsf{refl}\,q\|e\rangle]\!]_c & = & (\lambda k.[\![q]\!]_p\,(\boldsymbol{\lambda}^{\bullet}q'.k\,(\mathsf{subst}\,\mathsf{refl}\,q')))\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & [\![q]\!]_p\,(\boldsymbol{\lambda}^{\bullet}q'.[\![e]\!]_e\,(\mathsf{subst}\,\mathsf{refl}\,q')) \\
& \xrightarrow{?}_{\bullet} & [\![q]\!]_p\,(\boldsymbol{\lambda}^{\bullet}q'.[\![e]\!]_e\,q') \\
& \xrightarrow{?}_{\bullet} & [\![q]\!]_p\,[\![e]\!]_e = [\![\langle q\|e\rangle]\!]_c
\end{array}$$

- **Case** $\langle \mu\hat{\mathfrak{p}}.\langle p\|\hat{\mathfrak{p}}\rangle\|e\rangle \rightsquigarrow \langle p\|e\rangle$:
  We have:

$$\begin{array}{rcl}
[\![\langle \mu\hat{\mathfrak{p}}.\langle p\|\hat{\mathfrak{p}}\rangle\|e\rangle]\!]_c & = & (\lambda k.[\![p]\!]_p k)\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & [\![p]\!]_p\,[\![e]\!]_e = [\![\langle p\|e\rangle]\!]_c
\end{array}$$

- **Case** $c \rightsquigarrow c' \Rightarrow \langle \mu\hat{\mathfrak{p}}.c\|e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.c'\|e\rangle$:
  By induction hypothesis, we get that $[\![c]\!]_c \overset{*}{\longrightarrow}_{\beta^+} t =_{\mathsf{a}} [\![c']\!]_c$ for some term t. Therefore we have:

$$\begin{array}{rcl}
\langle \mu\hat{\mathfrak{p}}.c\|e\rangle & = & (\lambda k.[\![c]\!]_c\,k)\,[\![e]\!]_e \\
& \longrightarrow_{\mathsf{a}} & [\![c]\!]_c\,[\![e]\!]_e \\
& \overset{*}{\longrightarrow}_{\beta^+} & t\,[\![e]\!]_e \\
& =_{\mathsf{a}} & [\![c']\!]_c\,[\![e]\!]_e \\
& {}_{\mathsf{a}}\!\longleftarrow & (\lambda k.[\![c']\!]_c\,k)\,[\![e]\!]_e = \langle \mu\hat{\mathfrak{p}}.c'\|e\rangle
\end{array}$$

- **Case** $t \to t' \Rightarrow c[t] \;\rightsquigarrow\; c[t']$:
  As such, the translation does not allow an analysis of this case, mainly because we did not give an explicit small-step semantics for terms, and defined terms reduction through a big-step semantics:

$$\forall \alpha, \langle p\|\alpha\rangle \overset{*}{\rightsquigarrow} \langle (t,q)\|\alpha\rangle \Rightarrow \mathsf{wit}\,p \to t$$

  However, we claim that we could have extended the language of $\mathrm{dL}_{\hat{\mathfrak{p}}}$ with commands for terms:

$$c_t ::= \langle t\|e_t\rangle \qquad e_t ::= \tilde{\mu}x.c[t] \qquad c[\,] ::= \langle ([\,],p)\|e\rangle \mid \langle \lambda x.p\|[\,]\cdot e\rangle$$

and adding dual operators for (co-)delimited continuations to allow for a small-step definition of terms reduction:

$$\begin{array}{l|r}
\langle \lambda x.p \| t \cdot e \rangle \rightsquigarrow \langle \mu \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle & \langle V_t \| \tilde{\mu} x.c_t \rangle \rightsquigarrow c_t[V_t/x] \\
\langle \mathsf{wit}\, p \| e_t \rangle \rightsquigarrow \langle p \| \tilde{\mu} a.\langle \mathsf{wit}\, a \| e_t \rangle \rangle & \langle \mathsf{wit}\, (V_t, V_p) \| e_t \rangle \rightsquigarrow \langle V_t \| e_t \rangle \\
\langle (t,p) \| e \rangle \rightsquigarrow \langle p \| \tilde{\mu} \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle \hat{\mathfrak{tp}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle \rangle & \langle V_p \| \tilde{\mu} \hat{\mathfrak{tp}}.\langle \hat{\mathfrak{tp}} \| e \rangle \rangle \rightsquigarrow \langle V_p \| e \rangle
\end{array}$$

$$c \rightsquigarrow c' \Rightarrow \langle p \| \tilde{\mu} \hat{\mathfrak{tp}}.c \rangle \rightsquigarrow \langle p \| \tilde{\mu} \hat{\mathfrak{tp}}.c' \rangle$$

It is worth noting that these rules simulate the big-step definitions we had before while preserving the global call-by-value strategy. Defining the translation for terms in the extended syntax:

$$\begin{array}{ll}
[\![\mathsf{wit}\, V_t]\!]_t \triangleq \lambda k.k\, (\mathsf{wit}\, [\![V_t]\!]_{V_t}) & [\![\tilde{\mu} x.c]\!]_t \triangleq \boldsymbol{\lambda}^{\bullet} x.[\![c]\!]_c \\
[\![\mathsf{wit}\, p]\!]_t \triangleq \lambda k.[\![p]\!]_p\, (\boldsymbol{\lambda}^{\bullet} q.k\, (\mathsf{wit}\, q)) & [\![\langle t \| e_t \rangle]\!]_t \triangleq [\![t]\!]_t\, [\![e_t]\!]_t \\
[\![\tilde{\mu} \hat{\mathfrak{tp}}.c_t]\!]_t \triangleq [\![c_t]\!]_t & [\![\hat{\mathfrak{tp}}]\!]_p \triangleq \boldsymbol{\lambda}^{\bullet} k.k
\end{array}$$

We can then prove that each reduction rule satisfies the expected scheme.
Case $\langle \lambda x.p \| t \cdot e \rangle \rightsquigarrow \langle \mu \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$:
We have:

$$\begin{array}{rcl}
\langle \lambda x.p \| t \cdot e \rangle & = & (\boldsymbol{\lambda}^{\bullet} k.k\, \boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p)\, (\lambda p.([\![t]\!]_t\, (\boldsymbol{\lambda}^{\bullet} v.p\, v))\, [\![e]\!]_e) \\
& \longrightarrow_{\bullet} & (\lambda p.([\![t]\!]_t\, (\boldsymbol{\lambda}^{\bullet} v.p\, v))\, [\![e]\!]_e)\, \boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p \\
& \longrightarrow_{\mathsf{a}} & ([\![t]\!]_t\, (\boldsymbol{\lambda}^{\bullet} v.(\boldsymbol{\lambda}^{\bullet} x.[\![p]\!]_p)\, v))\, [\![e]\!]_e \\
& \xrightarrow{?}_{\bullet} & ([\![t]\!]_t\, (\boldsymbol{\lambda}^{\bullet} x.[\![p]\!]))\, [\![e]\!]_e \\
& {}_{\mathsf{a}^+}\longleftarrow & \lambda k.(([\![t]\!]_t\, (\boldsymbol{\lambda}^{\bullet} x.[\![p]\!]))\, k)\, [\![e]\!]_e = [\![\langle \mu \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle]\!]_c
\end{array}$$

Case $\langle (t,p) \| e \rangle \rightsquigarrow \langle p \| \tilde{\mu} \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle \hat{\mathfrak{tp}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle \rangle$:
We have:

$$\begin{array}{rcl}
\langle (t,p) \| e \rangle & = & (\boldsymbol{\lambda}^{\bullet} k.[\![p]\!]_p\, ([\![t]\!]_t\, (\lambda x.\boldsymbol{\lambda}^{\bullet} a.k\, (x,a))))\, [\![e]\!]_e \\
& \longrightarrow_{\bullet} & [\![p]\!]_p\, ([\![t]\!]_t\, (\lambda x.\boldsymbol{\lambda}^{\bullet} a.[\![e]\!]_e\, (x,a))) \\
& {}_{\mathsf{a}^+}\longleftarrow & [\![p]\!]_p\, ([\![t]\!]_t\, (\lambda x.(\lambda k.k) \boldsymbol{\lambda}^{\bullet} a.[\![e]\!]_e\, (x,a))) \\
& {}_{\mathsf{a}^+}\longleftarrow & [\![p]\!]_p\, ([\![t]\!]_t\, (\lambda x.(\lambda k.k) \boldsymbol{\lambda}^{\bullet} a.(\lambda k.k\, (x,a))\, [\![e]\!]_e)) \\
& = & [\![\langle p \| \tilde{\mu} \hat{\mathfrak{tp}}.\langle t \| \tilde{\mu} x.\langle \hat{\mathfrak{tp}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle \rangle]\!]_c
\end{array}$$

Case $\langle \mathsf{wit}\, p \| e_t \rangle \rightsquigarrow \langle p \| \tilde{\mu} a.\langle \mathsf{wit}\, a \| e_t \rangle \rangle$:
We have:

$$\begin{array}{rcl}
[\![\mathsf{wit}\, p]\!]_t [\![e_t]\!]_t & = & (\lambda k.[\![p]\!]_p\, (\boldsymbol{\lambda}^{\bullet} a.k\, (\mathsf{wit}\, a)))\, [\![e_t]\!]_t \\
& \longrightarrow_{\mathsf{a}} & [\![p]\!]_p\, (\boldsymbol{\lambda}^{\bullet} a.[\![e_t]\!]_t\, (\mathsf{wit}\, a))) \\
& {}_{\mathsf{a}^+}\longleftarrow & [\![p]\!]_p\, (\boldsymbol{\lambda}^{\bullet} a.(\lambda k.k\, (\mathsf{wit}\, a))\, [\![e_t]\!]_t) = [\![\langle p \| \tilde{\mu} a.\langle \mathsf{wit}\, a \| e_t \rangle \rangle]\!]_c
\end{array}$$

Case $\langle V_t \| \tilde{\mu} x.c_t \rangle \rightsquigarrow c_t[V_t/x]$:
We have:

$$\begin{array}{rcl}
[\![\mathsf{wit}\, (V_t, V_p)]\!]_t [\![e_t]\!]_t & = & (\lambda k.k\, (\mathsf{wit}\, ([\![V_t]\!]_{V_t}, [\![V_p]\!]_V)))\, [\![e_t]\!]_t \\
& \longrightarrow_{\mathsf{a}} & [\![e_t]\!]_t\, (\mathsf{wit}\, ([\![V_t]\!]_{V_t}, [\![V_p]\!]_V)) \\
& \longrightarrow_{\bullet} & [\![e_t]\!]_t\, [\![V_t]\!]_{V_t} \\
& {}_{\mathsf{a}}\longleftarrow & (\lambda k.k\, [\![V_t]\!]_{V_t})\, [\![e_t]\!]_t = [\![V_t]\!]_t e_t
\end{array}$$

Case $\langle \mathsf{wit}\, (V_t, V_p) \| e_t \rangle \rightsquigarrow \langle V_t \| e_t \rangle$:
We have:

$$\begin{array}{rcl}
[\![V_t]\!]_t [\![\tilde{\mu} x.c]\!]_t & = & (\lambda k.k\, [\![V_t]\!]_{V_t})\, \boldsymbol{\lambda}^{\bullet} x.[\![c]\!]_c \\
& \longrightarrow_{\mathsf{a}} & (\boldsymbol{\lambda}^{\bullet} x.[\![c]\!]_c)\, [\![V_t]\!]_{V_t} \\
& \longrightarrow_{\bullet} & [\![c]\!]_c [[\![V_t]\!]_{V_t}/x] = [\![c[V_t/x]]\!]_c
\end{array}$$

Case $\langle V \| \tilde{\mu}\hat{\mathfrak{p}}.\langle \hat{\mathfrak{p}} \| e\rangle\rangle \rightsquigarrow \langle V \| e\rangle$:
We have:

$$\begin{array}{rcl} [\![V]\!]_p[\![\tilde{\mu}\hat{\mathfrak{p}}.\langle \hat{\mathfrak{p}} \| e\rangle]\!]_e & = & (\lambda k.k\,[\![V]\!]_V)\,((\lambda k.k)[\![e]\!]_e) \\ & \longrightarrow_{\mathsf{a}} & ((\lambda k.k)[\![e]\!]_e)\,[\![V]\!]_V \\ & \longrightarrow_{\mathsf{a}} & [\![e]\!]_e\,[\![V]\!]_V \\ & {}_{\mathsf{a}}\longleftarrow & (\lambda k.k\,[\![V]\!]_V)\,[\![e]\!]_e = [\![\langle V \| e\rangle]\!]_c \end{array}$$

Case $c \rightsquigarrow c' \Rightarrow \langle V \| \tilde{\mu}\hat{\mathfrak{p}}.c\rangle \rightsquigarrow \langle V \| \tilde{\mu}\hat{\mathfrak{p}}.c'\rangle$:
This case is similar to the case for delimited continuations proved before, we only need to use the induction hypothesis for $[\![c]\!]_c$ to get:

$$\begin{array}{rcl} [\![V]\!]_p[\![\tilde{\mu}\hat{\mathfrak{p}}.c]\!]_e & = & (\lambda k.k\,[\![V]\!]_V)\,[\![c]\!]_c \\ & \longrightarrow_{\mathsf{a}} & [\![c]\!]_c\,[\![V]\!]_V \\ & \overset{*}{\longrightarrow}_{\beta^+} & t\,[\![V]\!]_V \\ & =_{\mathsf{a}} & [\![c']\!]_c\,[\![V]\!]_V \\ & {}_{\mathsf{a}^+}\longleftarrow & (\lambda k.k\,[\![V]\!]_V)\,[\![c']\!]_c = [\![V]\!]_p[\![\tilde{\mu}\hat{\mathfrak{p}}.c']\!]_e \end{array}$$

□

Proposition 4.6. *There is no infinite sequence only made of reduction:*

(1)	$\langle \mathsf{subst}\, p\, q \| e\rangle$	$\overset{p\notin V}{\rightsquigarrow}$	$\langle p \| \tilde{\mu}a.\langle \mathsf{subst}\, a\, q \| e\rangle\rangle$	(3)	$\langle \mu\hat{\mathfrak{p}}.\langle p \| \hat{\mathfrak{p}}\rangle \| e\rangle$	$\rightsquigarrow$	$\langle p \| e\rangle$
(2)	$\langle \mathsf{subst}\ \mathsf{refl}\, q \| e\rangle$	$\rightsquigarrow$	$\langle q \| e\rangle$	(4)	$c[t]$	$\rightsquigarrow$	$c[t']$

Proof. It is sufficient to observe if we define the following quantities:

(1) the quantity of $\mathsf{subst}\, p\, q$ with p not a value within a command,
(2) the quantity of subst within a command,
(3) the quantity of $\hat{\mathfrak{p}}$ within a command,
(4) the quantity of wit terms within a command.

then the rule (1) makes decrease the quantity (1) while preserving the others, (2) makes decrease (2) and preserves the other, and so one. All in all, we have a bound on the maximal number of steps for the reduction restricted to this 4 rules. □

Proposition 4.7 (Preservation of normalization). *If $[\![c]\!]_c$ normalizes, then c is also normalizing*

Proof. We prove the contraposition, that is that is c is not normalizing, neither is $[\![c]\!]_c$. In any infinite reduction sequence from c, according to the previous proposition, there need to be infinitely many often some steps that are reflected through the CPS into at least one distinguished step (Proposition 4.5). Thus there is an infinite reduction sequence from $[\![c]\!]_c$ too. □

Using the preservation of typing (Proposition 4.10), we know that if c is typed in $\mathrm{dL}_{\hat{\mathfrak{p}}}$, then its image $[\![c]\!]_c$ is also typed. Using the fact that typed terms of the target language are normalizing, we finally can apply the previous proposition to deduce that c normalizes.

Theorem 4.8 (Normalization). *If $c : \Gamma \vdash \Delta$, then c normalizes.*

4.4 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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$ by a dependency in p^+ (resp. t^+) in the target language.

Lemma 4.1 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 by:

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

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 6.1, this will no longer be the case when extending the domain of terms. We naturally extend the translation for types to the translation of contexts, where we consider unified contexts of the form $\Gamma \cup \Delta$:

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

As explained informally in Section 2.8 and stated by Lemma 4.1, 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 [12]: $\forall R.(A \to R) \to R$. It is worth noticing the correspondences with the continuation monad [10] and the codensity monad, and that we make plain use here of the fact that the NEF fragment is intuitionistic, so to speak. Indeed, it would be impossible to attribute this type to the translation of a (really) classical proof.

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 4.9. *The following holds:*

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

PROOF. The proof is done by mutual induction on the typing rule of $\mathrm{dL}_{\hat{\mathfrak{ph}}}$ for terms and NEF proofs. We only give the key cases of the proof.

- **Case** $\mathsf{wit}\, p$::
  In $\mathrm{dL}_{\hat{\mathfrak{ph}}}$ the typing rule for $\mathsf{wit}\, p$ is the following:

$$\frac{\Gamma \vdash p : \exists x^{\mathbb{N}} A(x) \mid \Delta \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{wit}\, p : \mathbb{N} \mid \Delta}\ \mathsf{wit}.$$

  We set $\Gamma' = [\![\Gamma \cup \Delta]\!]$. We want to show that:

$$\Gamma' \vdash \lambda k.[\![p]\!]_p\ (\lambda a.k(\mathsf{wit}\, a)) : \forall X(\forall x^{\mathbb{N}} X(x) \to X(t^+))$$

  where $t^+ \triangleq \mathsf{wit}\, p^+$. But by induction hypothesis, we have:

$$\Gamma' \vdash [\![p]\!]_p : \forall Z.(\Pi_{a:\exists x^{\mathbb{N}}[\![A]\!]^+} Z(a) \to Z(p^+)),$$

hence it amounts to showing that for any X we can build the following derivation, where $\Gamma_k = \Gamma', k : \forall x^{\mathbb{N}} X(x)$:

$$\dfrac{\dfrac{}{\Gamma_k \vdash k : \forall x^{\mathbb{N}} X(x)} \quad \dfrac{\dfrac{}{\Gamma_k, a : \exists x^{\mathbb{N}} [\![A]\!]^+ \vdash a : \exists x^{\mathbb{N}} [\![A]\!]^+}}{\Gamma_k, a : \exists x^{\mathbb{N}} [\![A]\!]^+ \vdash \mathsf{wit}\, a : \mathbb{N}}}{\dfrac{\Gamma_k, a : \exists x^{\mathbb{N}} [\![A]\!]^+ \vdash k\ (\mathsf{wit}\, a) : X(\mathsf{wit}\, a)}{\Gamma_k \vdash \lambda a.k(\mathsf{wit}\, a) : \Pi_{a:\exists x^{\mathbb{N}} [\![A]\!]^+} X(\mathsf{wit}\, a)}}$$

- **Case** (t,p)::
  in $\mathrm{dL}_{\hat{\Phi}}$ the typing rule for (t,p) is the following:

$$\dfrac{\Gamma \vdash t : \mathbb{N} \mid \Delta \quad \Gamma \vdash p : A(t) \mid \Delta}{\Gamma \vdash (t,p) : \exists x^{\mathbb{N}} A(x) \mid \Delta}\ \exists_i$$

  Hence we obtain by induction, using the same notation Γ' for $[\![\Gamma \cup \Delta]\!]$:

$$\Gamma' \vdash [\![t]\!]_t : \forall X.(\forall x^{\mathbb{N}} X(x) \to X(t^+)) \qquad (IH_t)$$
$$\Gamma' \vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A(t^+)} Y(a) \to Y(p^+)) \qquad (IH_p)$$

  and we want to show that for any Z:

$$\Gamma' \vdash \lambda k.[\![p]\!]_p([\![t]\!]_t\ (\lambda x a.k\ (x,a))) : \Pi_{a:\exists x^{\mathbb{N}} A} Z(a) \to Z(t^+, p^+).$$

  So we need to prove that:

$$\Gamma', k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q) \vdash [\![p]\!]_p([\![t]\!]_t\ (\lambda x a.k\ (x,a))) : Z(t^+, p^+)$$

  where $\Gamma_k = \Gamma', k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q)$. We let the reader check that such a type is derivable by using $X(x) \triangleq \Pi_{a:A(x)} Z(x,a)$ in the type of $[\![t]\!]_p$, and using $Y(a) \triangleq Z(t^+, a)$ in the type of $[\![p]\!]_p$:

$$\dfrac{\Gamma' \vdash [\![p]\!]_p : \ldots \qquad \dfrac{\Gamma' \vdash [\![t]\!]_t : \ldots \qquad \dfrac{\dfrac{}{k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q) \vdash k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q)} \quad \dfrac{}{x : \mathbb{N}, a : A(x) \vdash (x,a) : \exists x^{\mathbb{N}} A}}{k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(), x : \mathbb{N}, a : A(x) \vdash k\ (x,a) : Z(x,a)}}{k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q) \vdash \lambda x a.k\ (x,a) : \forall x.\Pi_{a:A(x)} Z(x,a)}}{\Gamma', k : \Pi_{a:\exists x^{\mathbb{N}} A} Z(a) \vdash [\![t]\!]_t\ (\lambda x a.k\ (x,a)) : \Pi_{a:A(t^+)} Z(t^+, a)}}{\Gamma', k : \Pi_{q:\exists x^{\mathbb{N}} A} Z(q) \vdash [\![p]\!]_p([\![t]\!]_t\ (\lambda x a.k\ (x,a))) : Z(t^+, p^+)}$$

- **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 $\mathrm{dL}_{\hat{\Phi}}$:

$$\dfrac{\dfrac{\Pi_p}{\Gamma \vdash p : A \mid \Delta} \qquad \dfrac{\dfrac{\Pi_q}{\Gamma, a : A \vdash q : B \mid \Delta} \quad \dfrac{}{\cdots \mid \star : B \vdash \Delta, \star : B}}{\langle q \| \star \rangle : \Gamma, a : A \vdash \Delta, \star : B}}{\Gamma \mid \tilde{\mu} a.\langle q \| \star \rangle : A \vdash \Delta, \star : B}}{\dfrac{\langle p \| \tilde{\mu} a.\langle q \| \star \rangle\rangle : \Gamma \mid \Delta, \star : B}{\Gamma \vdash \mu\star.\langle p \| \tilde{\mu} a.\langle q \| \star \rangle \mid \Delta\rangle : B}}$$

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:

$$\begin{array}{c}\Gamma' \vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A^+} Y(a) \to Y(p^+)) \\ \Gamma', a : A^+ \vdash [\![q]\!]_t : \forall Z.(\Pi_{b:B^+} Z(b) \to Z(q^+)),\end{array}$$

so that by choosing $Z(b) \triangleq X(b)$ and $Y(a) \triangleq X(q^+)$, we get the expected derivation:

$$\dfrac{\Gamma' \vdash [\![p]\!]_p : \ldots \qquad \dfrac{\dfrac{\Gamma', a : A^+ \vdash [\![q]\!]_p : \ldots \qquad \overline{k : \Pi_{b:B} X(b) \vdash k : k : \Pi_{b:B} X(b)}}{\Gamma', k : \Pi_{b:B} X(b), a : A^+ \vdash [\![q]\!]_p\ k : X(q^+)}}{\Gamma', k : \Pi_{b:B} X(b) \vdash \lambda a.[\![q]\!]_p\ k : \Pi_{a:A^+} X(q^+)}}{\Gamma', k : \Pi_{b:B} X(b) \vdash [\![p]\!]_p\ (\lambda a.[\![q]\!]_p\ k) : X(q^+[p^+/a])}$$

□

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

Proposition 4.10 (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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$. It is clear that for the nef cases, Lemma 4.9 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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$ is of the form:

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

By induction hypothesis for e and p we obtain:

$$\begin{array}{c}\Gamma' \vdash [\![e]\!]_e : [\![B[q^+]]\!]^+ \to \bot \\ \Gamma', a : A^+ \vdash [\![p]\!]_p : [\![B[a]]\!]^* \\ \Gamma' \vdash \lambda a.[\![p]\!]_p : \Pi_{a:A^+} [\![B[a]]\!]^*,\end{array}$$

where $\Gamma' = [\![\Gamma \cup \Delta]\!]$. Applying Lemma 4.9 for $q \in$ nef we can derive:

$$\dfrac{\Gamma' \vdash [\![q]\!]_p : \forall X.(\Pi_{a:A^+} X(a) \to X(q^+))}{\Gamma' \vdash [\![q]\!]_p : (\Pi_{a:A^+} [\![B[a]]\!]^* \to [\![B[q^+]]\!]^*}\ \forall_E$$

We can thus 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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 4.11 (Soundness). *For any $p \in \mathrm{dL}_{\hat{\mathfrak{tp}}}$, we have: $\nvdash p : \bot$.*

5 EMBEDDING IN LEPIGRE'S CALCULUS

In a recent paper [23], 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 [22]. 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 4.1, 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, observational equivalence allowing us to type realizers that would otherwise be untyped[14].

We shall detail now an embedding of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ into Lepigre's calculus, and explain how to transfer normalization and correctness properties along this translation. Actually, his language is 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 use, for the reduction rules and the type system the reader should refer to [23]:

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

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

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

We first define the translation for types (extended for typing contexts) where the predicate $\mathrm{Nat}(x)$ is defined as usual in second-order logic:

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

and $[\![t]\!]_t$ is the translation of the term t given in Figure 10:

[14] In particular, Lepigre's semantical restriction is so permissive that it is not decidable, while it is easy to decide wheter a proof term of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ is in NEF.

$$
\begin{array}{lll|lll|lll}
[\![x]\!]_t & \triangleq x & & [\![(t,p)]\!]_p & \triangleq ([\![t]\!]_t, [\![p]\!]_p) & & [\![q \cdot e]\!]_e & \triangleq [\![q]\!]_p \cdot [\![e]\!]_e \\
[\![n]\!]_t & \triangleq \lambda zs.s^n(z) & & [\![\mu\alpha.c]\!]_p & \triangleq \mu\alpha.[\![c]\!]_c & & [\![t \cdot e]\!]_e & \triangleq [\![t]\!]_t \cdot [\![e]\!]_e \\
[\![\mathsf{wit}\, p]\!]_t & \triangleq \pi_1([\![p]\!]_p) & & [\![\mathsf{prf}\, p]\!]_p & \triangleq \pi_2([\![p]\!]_p) & & [\![\tilde{\mu}a.c]\!]_e & \triangleq [\lambda a.[\![c]\!]_c]\bullet \\
[\![a]\!]_p & \triangleq a & & [\![\mathsf{refl}]\!]_p & \triangleq \lambda a.a & & [\![\langle p \| e \rangle]\!]_c & \triangleq [\![p]\!]_p * [\![e]\!]_e \\
[\![\lambda a.p]\!]_p & \triangleq \lambda a.[\![p]\!]_p & & [\![\mathsf{subst}\, p\, q]\!]_p & \triangleq [\![p]\!]_p\, [\![q]\!]_p & & [\![\mu \hat{\mathfrak{p}}.c]\!]_p & \triangleq \mu\alpha.[\![c]\!]_{\hat{\mathfrak{p}}} \\
[\![\lambda x.p]\!]_p & \triangleq \lambda x.[\![p]\!]_p & & [\![\alpha]\!]_e & \triangleq \alpha & & [\![\langle p \| \hat{\mathfrak{p}} \rangle]\!]_{\hat{\mathfrak{p}}} & \triangleq [\![p]\!]_p \\
\multicolumn{9}{c}{[\![\langle p \| \tilde{\mu}a.c \rangle]\!]_{\hat{\mathfrak{p}}} \triangleq (\mu\alpha.[\![p]\!]_p * [\lambda a.[\![c]\!]_{\hat{\mathfrak{p}}}]\alpha) * \alpha}
\end{array}
$$

Fig. 10. Translation of proof terms to Lepigre's calculus

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

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

Fig. 11. Extension of Lepigre's typing rules for stacks

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

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.

The translation for terms, proofs, contexts and commands of $\mathrm{dL}_{\hat{\mathfrak{p}}}$, given in Figure 10 is almost straightforward. We only want to draw the reader's attention on a few points:

- the equality 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).

In [23], 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 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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$. We will detail here a proof of adequacy using the former method in the appendix. We then need to extend Lepigre's system to be able to type stacks. In fact, the proof of adequacy [23, Theorem 6] suggests a way to do so, since any typing rule for typing stacks is valid as long as it is adequate with the realizability model.

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 2.5). We also add a distinguished bottom stack $\bullet$ to the syntax, which is given the most general type $\bot^{\perp\!\!\!\perp}$. We change the rules $(*)$ and (μ) of the original type system in [23] and add rules for stacks, whose definitions are guided by the proof of the adequacy [23, Theorem 6] in particular by the $(\Rightarrow_e)$-case. These rules are given in Figure 11.

We shall now show that these rules are adequate with respect with the realizability model defined in [23, Section 2].

Proposition 5.1 (Adequacy). *Let Γ be a (valid) context, A be a formula with $FV(A) \subset dom(\Gamma)$ and σ be a substitution realizing Γ. The following statements hold:*

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

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 [23, Theorem 6]).

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

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

($*$). We need to show that $t\sigma * \pi\sigma \in [\![B]\!]^{\perp\perp}_\sigma$, so we take $\rho \in [\![B]\!]^\perp_\sigma$ 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 $t\sigma \in [\![A]\!]^{\perp\perp}_\sigma$ and $\pi\sigma \in [\![A]\!]^\perp_\sigma$.

(μ). The proof is the very same as in [23, Theorem 6].

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

($\Rightarrow_l$). If t is a value v, by induction hypothesis, we have that $v\sigma \in [\![A]\!]_\sigma$ and $\pi\sigma \in [\![B]\!]^\perp_\sigma$ and we need to show that $v\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\perp_\sigma$. The proof is already done in the case ($\Rightarrow_e$) (see [23, Theorem 6]). Otherwise, by induction hypothesis, we have that $t\sigma \in [\![A]\!]^{\perp\perp}_\sigma$ and $\pi\sigma \in [\![B]\!]^\perp_\sigma$ and we need to show that $t\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\perp_\sigma$. 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 $[\lambda x.u]\pi \in [\![A]\!]^\perp_\sigma$, 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 step 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 $t\sigma \in [\![A \Rightarrow B]\!]^{\perp\perp}_\sigma$ and $\pi\sigma \in [\![B]\!]^\perp_\sigma$, thus $v \cdot \pi\sigma \in [\![A \Rightarrow B]\!]^\perp_\sigma$.

□

It only remains to show that the translation we defined in Figure 10 preserve typing to conclude the proof of Proposition 5.3.

Lemma 5.2. *If $\Gamma \vdash p : A \mid \Delta$ (in $\mathrm{dL}_{\hat{\Phi}}$), 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 a delimited continuation somehow decompiles it to simulate the reduction in a natural deduction fashion of the applications of functions to stacks (that could have generated the same delimited continuations in $\mathrm{dL}_{\hat{\Phi}}$), while maintaining the frozen context (at top-level) outside of the active command (just like a delimited continuation would do). This trick allows us to avoid the problem of dependencies conflict in the typing derivation. For instance, assuming that $[\![q_1]\!]_p$ ($[\![q_2]\!]_p$) reduces to a value V_1

(V_2) we have:

$$
\begin{aligned}
&[\![\langle \mu\hat{\mathfrak{p}}.\langle q_1 \| \tilde{\mu}a_1.\langle q_2 \| \tilde{\mu}a_2.\langle p \| \hat{\mathfrak{p}}\rangle\rangle\rangle \| e\rangle]\!]_c \\
&= \mu\alpha.(\mu\alpha.([\![q_1]\!]_p * [\lambda a_1.[\![\langle q_2 \| \tilde{\mu}a_2.\langle p \| \hat{\mathfrak{p}}\rangle\rangle]\!]_{\hat{\mathfrak{p}}}]\alpha) * \alpha) * [\![e]\!]_e \\
&\succ \mu\alpha.([\![q_1]\!]_p * [\lambda a_1.[\![\langle q_2 \| \tilde{\mu}a_2.\langle p \| \hat{\mathfrak{p}}\rangle\rangle]\!]_{\hat{\mathfrak{p}}}]\alpha) * [\![e]\!]_e \\
&\succ [\![q_1]\!]_p * [\lambda a_1.[\![\langle q_2 \| \tilde{\mu}a_2.\langle p \| \hat{\mathfrak{p}}\rangle\rangle]\!]_{\hat{\mathfrak{p}}}][\![e]\!]_e \\
&\succ^* [\![q_2]\!]_p * [\lambda a_2.[\![p]\!]_p[V_1/a_1]][\![e]\!]_e \\
&\succ^* [\![p]\!]_p[[\![V_1]\!]_p/a_1][[\![V_2]\!]_p/a_2] * [\![e]\!]_e \\
&{}^*\prec [\![q_2]\!]_p * [\lambda a_2.[\![p]\!]_p[V_1/a_1]][\![e]\!]_e \\
&{}^*\prec [\![q_1]\!]_p * [\lambda a_1 a_2.[\![p]\!]_p)][\![q_2]\!]_p \cdot [\![e]\!]_e \\
&{}^*\prec (\lambda a_1 a_2.[\![p]\!]_p) * [\![q_1]\!]_p \cdot [\![q_2]\!]_p \cdot [\![e]\!]_e = [\![\langle \lambda a_1 \lambda a_2.p \| q_1 \cdot q_2 \cdot e\rangle]\!]_c
\end{aligned}
$$

where we observe that $[\![e]\!]_e$ is always kept outside of the computations, and where each command $\langle q_i \| \tilde{\mu}a_i.c_{\hat{\mathfrak{p}}}\rangle$ is decompiled into $(\mu\alpha.[\![q_i]\!]_p * [\lambda a_i.[\![c_{\hat{\mathfrak{p}}}]\!]_{\hat{\mathfrak{p}}}].\alpha) * [\![e]\!]_e$, simulating the (natural deduction style) reduction of $\lambda a_i.[\![c_{\hat{\mathfrak{p}}}]\!]_{\hat{\mathfrak{p}}} * [\![q_i]\!]_p \cdot [\![e]\!]_e$. These terms correspond somehow to the translations of former commands typable without types dependencies.

□

As a corollary we get a proof of the adequacy of $\mathrm{dL}_{\hat{\mathfrak{p}}}$ typing rules with respect to Lepigre's realizability model.

Proposition 5.3 (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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$, 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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$ is a consequence of adequacy.

Theorem 5.4 (Soundness). *For any proof* p *in* $\mathrm{dL}_{\hat{\mathfrak{p}}}$, *we have:* $\nvdash p : \bot$.

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 2.8).

6 FURTHER EXTENSIONS

As we explained in the preamble of Section 2, we defined dL and $\mathrm{dL}_{\hat{\mathfrak{p}}}$ as minimal languages 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.

6.1 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 $\mathrm{dL}_{\hat{\mathfrak{p}}}$, 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^1_0-fragment [25]. 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 [4].

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 extraction of a strong pair $(\llbracket t_f \rrbracket, \llbracket p \rrbracket) : \exists f^{\mathbb{N}\to\mathbb{N}} A^*$. More precisely, the proof $\llbracket t_f \rrbracket$ 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ (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.

Nevertheless, this discussion only makes sense as long as we consider a language classical logic. Indeed, we explained in the introduction that $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ also provides us with a sequent calculus presentation for an intuitionistic theory with dependent types. In fact, we pretend that it is enough to restrict the syntax of proofs to allow only one continuation variable to obtain an intuitionistic calculus. In particular, in such a setting, all proofs will be NEF, and every result we obtained will still hold. Even more, in this case we should be able to extend the syntax of terms to includes higher-order terms, and still be able to perform witness extraction to gets a CPS to a stratified target language.

6.2 Adding expressiveness

From the point of view of the proof language (that is of the tools we have to build proofs), $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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.

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.

For instance, for the case of the pairs, we can extend the syntax with:

$$p ::= \cdots \mid (p_1, p_2) \qquad\qquad e ::= \cdots \mid \tilde{\mu}(a_1, a_2).c$$

We then need to add the corresponding typing rules (plus a third rule to type $\tilde{\mu}(a_1, a_2).c$ in regular mode=t:

$$\frac{\Gamma \vdash p_1 A_1 \mid \Delta \quad \Gamma \vdash p_2 : A_2) \mid \Delta}{\Gamma \vdash (p_1, p_2) : (A_1 \wedge A_2) \mid \Delta} \wedge_r \qquad\qquad \frac{c : \Gamma, a_1 : A_1, a_2 : A_2 \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{(a_1, a_2)|p\}}{\Gamma \mid \tilde{\mu}(a_1, a_2).c : (A_1 \wedge A_2) \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \sigma\{\cdot|p\}} \wedge_l$$

and the reduction rules:

$$\langle (p_1, p_2) \| e \rangle \rightsquigarrow \langle p_1 \| \tilde{\mu} a_1 . \langle p_2 \| \tilde{\mu} a_2 . \langle (a_1, a_2) \| e \rangle \rangle \rangle \qquad\qquad \langle (V_1, V_2) \| \tilde{\mu}(a_1, a_2).c \rangle \rightsquigarrow c[V_1/a_1, V_2/a_2]$$

We let the reader check that these rules preserve subject reduction, and suggest for a the following CPS translations:

$$\begin{array}{rcl} [\![(p_1, p_2)]\!]_p & \triangleq & \boldsymbol{\lambda}^{\bullet} k . [\![p_1]\!]_p \, (\boldsymbol{\lambda}^{\bullet} a_1 . [\![p_2]\!]_p \, (\boldsymbol{\lambda}^{\bullet} a_2 . k \, (a_1, a_2))) \\ [\![(V_1, V_2)]\!]_V & \triangleq & \boldsymbol{\lambda}^{\bullet} k . k \, ([\![V_1]\!]_V, [\![V_2]\!]_V) \\ [\![\tilde{\mu}(a_1, a_2).c]\!]_e & \triangleq & \lambda p . \, \mathtt{split}\ p\ \mathtt{as}\ (a_1, a_2)\ \mathtt{in}\ [\![c]\!]_c \end{array}$$

which allows to prove that the calculus remain correct with these extension.

We claim that this is methodical approach furnishes in first approximation to the question "*Can I extend this with ... ?*". In particular, it should be enough to get closer to a realistic programming language and extend the language with inductive fix-points, operators.

6.3 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:

$$\frac{\Gamma \vdash p : \exists x^{\mathbb{N}} A(x) \mid \Delta; \sigma \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{prf}\, p : A(\mathsf{wit}\, p) \mid \Delta; \sigma} \ \mathsf{prf} \qquad\qquad \frac{\Gamma \vdash p : t = u \mid \Delta; \sigma \quad \Gamma \vdash q : B[t/x] \mid \Delta; \sigma}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta; \sigma} \ \mathsf{subst}$$

However, it is now easy to have both rules in a sequent calculus fashion, for instance we could rather have contexts of the shape $\tilde{\mu}(x, a).c$ (to be dual to proofs (t, p)) and $\tilde{\mu}{=}.c$ (dual to refl). We could then define the following typing rules (where we add another dependencies list δ for terms, to compensate the conversion from $A[t]$ to $A[u]$ in the former (subst)-rule)):

$$\frac{c : \Gamma, x : \mathbb{N}, a : A(x) \vdash_d \Delta; \sigma\{(x, a)|p\}}{\Gamma \mid \tilde{\mu}(x, a).c : \exists x^{\mathbb{N}} A(x) \vdash -d\Delta; \sigma\{\cdot|p\}} \ \exists \qquad\qquad \frac{c : \Gamma \vdash \Delta; \delta\{t|u\}}{\Gamma \mid \tilde{\mu}{=}.c : t = u \vdash \Delta; \delta} \ =_l$$

and define $\mathsf{prf}\, p$ and $\mathsf{subst}\, p\, q$ as syntactic sugar:

$$\mathsf{prf}\, p \triangleq \mu \hat{\mathfrak{tp}} . \langle p \| \tilde{\mu}(x, a) . \langle a \| \hat{\mathfrak{tp}} \rangle \rangle \qquad\qquad \mathsf{subst}\, p\, q \triangleq \mu \alpha . \langle p \| \tilde{\mu}{=}. \langle q \| \alpha \rangle \rangle .$$

Observe that $\mathsf{prf}\, p$ is now only definable if p is a NEF proof term. For any $p \in$ NEF and any variables a, α, $A(\mathsf{wit}\, p)$ is in $A(\mathsf{wit}\, (x, a))_{\{(x,a)|p\}}$ which allows us to derive (using this in the CUT-rule) the admissibility of the former prf -rule:

$$\dfrac{\dfrac{\dfrac{}{a : A(x) \vdash a : A(x)}}{a : A(x) \vdash a : A(\mathsf{wit}\,(x,a))} \equiv \dfrac{A(\mathsf{wit}\,p) \in A(\mathsf{wit}\,(x,a))_{\{(x,a)|p\}}}{\Gamma \mid \hat{\mathfrak{p}} : A(\mathsf{wit}\,(x,a)) \vdash_d \hat{\mathfrak{p}} : A(\mathsf{wit}\,p) \mid \Delta}}{\dfrac{\langle a \| \alpha \rangle : \Gamma, x : \mathbb{N}, a : A(x) \vdash_d \Delta, \hat{\mathfrak{p}} : A(\mathsf{wit}\,p); \sigma\{(x,a)|p\}}{\Gamma \mid \tilde{\mu}(x,a).\langle a \| \hat{\mathfrak{p}} \rangle : \exists x^{\mathbb{N}} A \vdash_d \Delta, \hat{\mathfrak{p}} : A(\mathsf{wit}\,p); \sigma\{\cdot|p\}}}\ \mathsf{cut}$$

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

Using the fact that $\delta(B[u]) = \delta(B[t])$, we get that the former (subst)-rule is admissible:

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

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

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

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

$$\langle \mathsf{subst}\ \mathsf{refl}\ q \| e \rangle \rightsquigarrow \langle q \| e \rangle \qquad \langle \mathsf{subst}\ p\ q \| e \rangle \overset{p \notin V}{\rightsquigarrow} \langle p \| \tilde{\mu}a.\langle \mathsf{subst}\ a\ q \| e \rangle \rangle$$
$$\langle \mathsf{prf}\ (V_t, V_p) \| e \rangle \rightsquigarrow \langle V \| e \rangle \qquad \langle \mathsf{prf}\ p \| e \rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}a.\langle \mathsf{prf}\ a \| \hat{\mathfrak{p}} \rangle \rangle \| e \rangle.$$

7 CONCLUSION

In this paper, we presented dL, a sequent calculus that combine dependent types and control operators by mean of a syntactic restriction to values. We proved in Section 2 the normalization of dL for typed terms, as well as its soundness. This calculus can be extended with delimited continuations, which permits to extend the syntactic restriction for dependent types to the fragment of negative-elimination-free proofs. The resulting calculus $\mathrm{dL}_{\hat{\mathfrak{p}}}$, that we presented in Section 3, is suitable for the definition of a dependently typed translation to an intuitionistic type theory. As shown in Section 4, this translation guarantees both the normalization and the soundness of $\mathrm{dL}_{\hat{\mathfrak{p}}}$. Furthermore, a similar translation can be designed to embedded $\mathrm{dL}_{\hat{\mathfrak{p}}}$ to Lepigre's calculus. As explained in Section 5, this provides an alternative way of proving the soundness of $\mathrm{dL}_{\hat{\mathfrak{p}}}$.

Several directions remain to be explored. In future work, we plan to investigate possible extensions of the syntactic restriction we defined, and its connection with notions such as with Fürhmann's *thunkability* [13] or Munch-Maccagnoni's *linearity* [27]. Furthermore, it might be of interest to check whether this restriction could make dependent types compatible with other side-effects, in presence of classical logic or not. More generally, we would like to better understand the possible connections between our calculus with the categorical models for dependently typed theory. On a different perspective, the continuation-passing style translation we defined is at the best of our knowledge a novelty, event without considering the classical part. In particular, our translation allows to use computations (as in the call-by-push value terminology) within dependent types with a call-by-value evaluation strategy, and without any thunking construction. It might be the case

that this translation can be adapted to justifies extensions of other dependently typed calculi, or provide typed translations between them.

Last but not least, we plan to use the $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ to solve the problem that originally motivated us to design such a calculus. We contemplate the possibility of proving the normalization and soundness of Herbelin's dPA^{ω} calculus by designing an equivalent sequent calculus inspired from $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. Hopefully, we might be able to prove the soundness of such a system by means of a continuation-passing style translation or a realizability model inspired from Lepigre's construction.

Acknowledgments. The author wish to thanks Pierre-Marie Pédrot for a discussion that led to the idea of using delimited continuations, Gabriel Scherer for his accurate observations and the constant interest he showed for this work, Hugo Herbelin who provided valuable helping all along the writing of this paper, as well as Alexandre Miquel and anonymous referees of earlier versions of this paper for their constructive remarks.

REFERENCES

[1] Danel Ahman, Neil Ghani, and Gordon D. Plotkin. *Dependent Types and Fibred Computational Effects*, pages 36–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[2] Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-theoretic foundation of delimited continuations. *Higher-Order and Symbolic Computation*, 22(3):233–273, 2009.

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

[4] Gilles Barthe, John Hatcliff, and Morten Heine B. Sørensen. CPS translations and applications: The cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, 1999.

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

[6] Thierry Coquand and Christine Paulin. *Inductively defined types*, pages 50–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[7] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of ICFP 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000.

[8] Paul Downen, Luke Maurer, Zena M. Ariola, and Simon Peyte Jones. Sequent calculus as a compiler intermediate language. In *ICFP 2016*, 2016.

[9] Gilda Ferreira and Paulo Oliva. On various negative translations. In Steffen van Bakel, Stefano Berardi, and Ulrich 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.

[10] Andrzej Filinski. Representing monads. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 446–457. ACM Press, 1994.

[11] Daniel Fridlender and Miguel Pagano. Pure type systems with explicit substitutions. *J. Funct. Program.*, 25, 2015.

[12] Harvey Friedman. *Classically and intuitionistically provably recursive functions*, pages 21–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[13] Carsten Führmann. Direct models for the computational lambda calculus. *Electr. Notes Theor. Comput. Sci.*, 20:245–292, 1999.

[14] Jacques Garrigue. Relaxing the value restriction. In Yukiyoshi Kameyama and Peter 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.

[15] Timothy 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.

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

[17] Hugo Herbelin. On the degeneracy of sigma-types in presence of computational classical logic. In Pawel Urzyczyn, editor, *Proceedings of TLCA 2005*, volume 3461 of *LNCS*, pages 209–220. Springer, 2005.

[18] Hugo 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.

[19] Hugo Herbelin and Silvia Ghilezan. An approach to call-by-name delimited continuations. In George C. Necula and Philip 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, January 2008.

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

[21] Felix Joachimski and Ralph Matthes. Short proofs of normalization for the simply- typed λ-calculus, permutative conversions and gödel's t. *Archive for Mathematical Logic*, 42(1):59–87, 2003.

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

[23] Rodolphe Lepigre. A classical realizability model for a semantical value restriction. In Peter 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.

[24] 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.

[25] Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7(2), 2011.

[26] Étienne Miquey. A classical sequent calculus with dependent types. In Hongseok Yang, editor, *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings*, pages 777–803, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[27] Guillaume Munch-Maccagnoni. *Models of a Non-associative Composition*, pages 396–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

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

[29] Emmanuel 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.

[30] Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. In *Proc. Of LICS2017*, 2017.

[31] Matthijs Vákár. A framework for dependent types and effects. *CoRR*, abs/1512.08009, 2015.

[32] Philip Wadler. Call-by-value is dual to call-by-name. In Colin Runciman and Olin 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.

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