A Classical Sequent Calculus with Dependent Types

Étienne Miquey

A Classical Sequent Calculus with Dependent Types

ÉTIENNE MIQUEY, INRIA, Équipe Gallinette

Dependent types are a key feature of the proof assistants based on the Curry-Howard isomorphism. It is well-known that this correspondence can be extended to classical logic by enriching the language of proofs with control operators. However, they are known to misbehave in the presence of dependent types, unless dependencies are restricted to values. Moreover, while sequent calculi naturally support 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 $\lambda\mu\tilde{\mu}$-calculus. We design a minimal language with a value restriction and a type system that includes a list of explicit dependencies to 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, and present a methodology to transfer properties from this system to our own.

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 programming languages. From the point of view of programming, dependent types provide more precise types—and thus more precise specifications—to existing programs. From a logical perspective, they permit definitions of proof terms for statements like the full axiom of choice. Dependent types are provided by Coq or Agda, two of the most actively developed proof assistants. They both rely on constructive type theories: the calculus of inductive constructions for Coq [6], and Martin-Löf's type theory for Agda [24]. Yet, both systems lack support for classical logic and more generally for side effects, which make them impractical as programming languages.

In practice, effectful languages give the programmer a more explicit access to low-level control (that is: to the way the program is executed on the available hardware), and make some algorithms easier to implement. Common effects, such as the explicit manipulation of the memory, the generation of random numbers and input/output facilities are available in most practical programming languages (*e.g.*, OCaml, C++, Python, Java).

In 1990, Griffin discovered that the control operator `call/cc` (short for *call with current continuation*) could be typed by Peirce's law ($((A \to B) \to A) \to A$) [15], thus extending the formulas-as-types interpretation. Indeed, Peirce's law is known to imply, in an intuitionistic framework, all the other forms of classical reasoning (excluded middle, *reductio ad absurdum*, double negation elimination, etc.). This discovery opened the way for a direct computational interpretation of classical proofs, using control operators and their ability to *backtrack*. Several calculi were born from this idea, for example Parigot's $\lambda\mu$-calculus [31], Barbanera and Berardi's symmetric λ-calculus [3], Krivine's λ_c-calculus [21], Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [7].

Nevertheless, dependent types are known to misbehave in the presence of control operators, and lead to logical inconsistencies [17]. Since the same problem arises with a wider class of effects, it seems that we are facing the following dilemma: either we choose an effectful language (allowing us to write more programs) while accepting the lack of dependent types, or we choose a dependently typed language (allowing us to write finer specifications) and give up effects.

Many works have tried to fill the gap between effectful programming languages and logic, by accommodating weaker forms of dependent types with computational effects (*e.g.*, divergence, I/O, local references, exceptions). Amongst other works, we can cite the recent works by Ahman *et al.* [1], by Vákár [35, 36] or by Pédrot and Tabareau who proposed a systematical way to add effects to type theory [33]. Side effects—that are impure computations in functional programming—are often interpreted by means of monads. Interestingly, control operators can be interpreted similarly through the continuation monad, but the continuation monad generally lacks the properties necessary to fit these frameworks.

Although dependent types and classical logic have been deeply studied separately, the problem of accommodating both features[1] in one and the same system has not found a completely satisfying answer yet. Recent works from Herbelin [18] and Lepigre [22] proposed some restrictions on dependent types to make them compatible with a classical proof system, while Blot [5] designed a hybrid realizability model where dependent types are restricted to an intuitionistic fragment.

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 occur 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 kinds 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 [39] and polymorphism [16]. In both cases, a simple solution (but often unnecessarily restrictive in practice [14, 22]) to solve the inconsistencies consists in the introduction of a value restriction for the problematic cases, restoring then a sound type system. Recently, Lepigre presented a proof system providing dependent types and a control operator [22], 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

[1]Aside from strictly logical considerations as in [18], there are motivating examples of programs that could only be written and specified in such a setting. Consider for instance the infinite tape lemma that states that from any infinite sequence of natural numbers, one can extract either an infinite sequence of odd numbers, or an infinite sequence of even numbers. Its proof deeply relies on classical logic, and the corresponding program (which, given as input a stream of integers, returns a stream that consists either only of odd integers or only of even ones) can only be written in a classical setting and requires dependent types to be specified. See [23, Section 7.8] for more details.

that allows slightly more than values. As we will see, the restriction that arises naturally coincides with 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[2] of a call-by-value language with classical control 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] with 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 design of a program translation for Herbelin's dPAω system (that already encompasses control operators and dependent types) to justify its soundness. However, this calculus was presented in a natural deduction style, making such a translation hard to obtain. We thus developed the framework presented in this paper to have an intermediate language more suitable for a continuation-passing style translation at our disposal.

Additionally, while we consider in this paper 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, which would already happen in an intuitionistic type theory. We claim that the solutions we give in this paper also works in the intuitionistic case. In particular, the system we develop might be a first step towards the adaption of the well-understood continuation-passing style translations for ML to design a (dependently) typed compilation of a system with dependent types such as Coq.

1.4 Delimited continuations and CPS translation

The main challenge in designing a sequent calculus with dependent types lies in the fact that the natural relation of reduction one would expect in such a framework is not safe with respect to types. As we will discuss in Section 2.6, the problem can be understood as a desynchronization of the type system with respect to the reduction. A simple solution, presented in Section 2, consists in the addition of an explicit list of dependencies in typing derivations. This has the advantage of leaving the computational part of the original calculus unchanged. However, it is not suitable for obtaining a continuation-passing style translation.

We thus present a second way to solve this issue by introducing delimited continuations [2], which are used to force the purity needed for dependent types in an otherwise non purely functional language. It also justifies the relaxation of the value restriction and leads to the definition of the negative-elimination-free fragment (Section 3). In addition, it allows for the design, in Section 4, of a continuation-passing style translation that preserves dependent types and permits us to prove the soundness of our system. Finally, it also provides us with a way to embed our calculus into Lepigre's calculus [22], as we shall see in Section 5. This embedding has in particular the benefit of furnishing us with a realizability interpretation for free.

1.5 Contributions of the paper

Our main contributions in this paper can be listed as follows:

- We soundly combine dependent types and control operators by means of a syntactic restriction to the negative-elimination-free fragment;

[2]In the sense of a formulas-as-types interpretation of a sequent calculus *à la* Hilbert (as Curien-Herbelin's $\lambda\mu\tilde{\mu}$-calculus [7] or Munch-Maccagnoni's system L [29]), as opposed to traditional type systems given in a natural deduction style.

- 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 list of dependencies 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;
- We relate our system to Lepigre's calculus, which gives us a realizability interpretation for free and 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 subset of proofs $\mathcal{V}$ and a subset 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 $\langle p \| e\rangle$ can be understood as a state of an abstract machine, representing the evaluation of a proof p (the program) against a co-proof e (the stack) that we call *context*. The μ operator comes from Parigot's $\lambda\mu$-calculus [31], $\mu\alpha$ binds a context to a context variable α in the same way that $\tilde{\mu}a$ 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. In contrast with Gentzen's original presentation of sequent calculus, the type system of the $\lambda\mu\tilde{\mu}$-calculus explicitly identifies at any time which formula is being worked on. In a nutshell, this presentation distinguishes between three kinds of sequents:

(1) sequents of the form $\Gamma \vdash p : A \mid \Delta$ for typing proofs, where the focus is put on the (right) formula A;
(2) sequents of the form $\Gamma \mid e : A \vdash \Delta$ for typing contexts, where the focus is put on the (left) formula A;
(3) sequents of the form $c : (\Gamma \vdash \Delta)$ for typing commands, where no focus is set.

In a right (resp. left) sequent $\Gamma \vdash p : A \mid \Delta$, the singled out formula[3] A reads as the conclusion *"where the proof shall continue"* (resp. hypothesis *"where it happened before"*).

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 $q \cdot e$:

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

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

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

The difference between call-by-name and call-by-value can be characterized by how this critical pair[4] is solved, by defining $\mathcal{V}$ and $\mathcal{E}$ such that the two rules do not overlap. Defining the subcategories

[3]This formula is often referred to as the formula in the *stoup*, a terminology due to Girard.

[4]Observe that this critical pair can be also interpreted in terms of non-determinism. Indeed, we can define a fork instruction by $\pitchfork \triangleq \lambda ab.\mu\alpha.\langle \mu\_.\langle a\|\alpha\rangle \| \tilde{\mu}\_.\langle b\|\alpha\rangle\rangle$, which verifies indeed that $\langle \pitchfork \| p_0 \cdot p_1 \cdot e\rangle \to \langle p_0 \| e\rangle$ and $\langle \pitchfork \| p_0 \cdot p_1 \cdot e\rangle \to \langle p_1 \| e\rangle$.

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

(a) Syntax

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

(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

of values $V \subset p$ and co-values $E \subset e$ by:

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

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

Remark 2.1 (Application). The reader unfamiliar with the $\lambda\mu\tilde{\mu}$-calculus might be puzzled by the absence of a syntactic construction for the application of proof terms. Intuitively, the usual application $p\,q$ of the λ-calculus is replaced by the application of the proof p to a stack of the shape $q \cdot e$ as in an abstract machine[5]. The usual application can thus be recovered through the following shorthand:

$$p\,q \triangleq \mu\alpha.\langle p \| q \cdot \alpha \rangle$$

Finally, it is worth noting that the μ binder is a *control operator*, since it allows for catching evaluation contexts and backtracking further in the execution. This is 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 Krivine's λ_c-calculus [21]. 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 (*e.g.*, the law of excluded middle, the double negation elimination).

Let us observe the behavior of `call/cc` (in call-by-name evaluation strategy, as in Krivine λ_c-calculus): in front of a context of the shape $q \cdot e$ with e of type A, it will catch the context e thanks to the $\mu\alpha$ binder and reduce as follows:

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

[5]To pursue the analogy with the λ-calculus, the rest of the stack e can be viewed as a context $C_e[\,]$ surrounding the application $p\,q$, the command $\langle p \| q \cdot e \rangle$ thus being identified with the term $C_e[p\,q]$. Similarly, the whole stack can be seen as the context $C_{q \cdot e}[\,] = C_e[[\,]q]$, whence the terminology.

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

(where $\bullet$ is used to shorten useless parts of typing contexts.)

Fig. 2. Proof term for Peirce's law

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:

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

This computational behavior corresponds exactly to the usual reduction rule for `call/cc` in the Krivine machine [21]:

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

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 highlighting this phenomenon [17].

Let us adopt here a stratified presentation of dependent types, by syntactically distinguishing *terms*—that represent mathematical objects—from *proof terms*—that represent mathematical proofs. In other words, we syntactically separate the categories corresponding to witnesses and proofs in dependent sum types. Consider a minimal logic of strong existentials and equality, whose formulas, terms (only representing natural number) and proofs are defined as follows:

$$\begin{array}{llll}\textbf{Formulas} & A,B & ::= & t=u \mid \exists x^{\mathbb{N}}.A\\ \textbf{Terms} & t,u & ::= & n\mid \mathsf{wit}\,p\mid x \qquad\qquad (n\in\mathbb{N})\\ \textbf{Proofs} & p,q & ::= & \mathsf{refl}\mid\mathsf{subst}\,p\,q\mid(t,p)\mid\mathsf{prf}\,p\end{array}$$

Let us explain the different proof terms by presenting their typing rules. First of all, 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 $A[t/x]$. This implies that both formulas and proofs are dependent on terms, which is usual in mathematics. What is less usual in mathematics is that, as in Martin-Löf's type theory, dependent types also allow for terms (and thus for formulas) to be dependent on proofs, by means of the constructors $\mathsf{wit}\,p$ and $\mathsf{prf}\,p$. Typing rules are given with separate typing judgments for terms, which can only be of type $\mathbb{N}$:

$$\dfrac{\Gamma\vdash p:A(t)\quad\Gamma\vdash t:\mathbb{N}}{\Gamma\vdash(t,p):\exists x^{\mathbb{N}}.A}(\exists_I)\qquad \dfrac{\Gamma\vdash(t,p):\exists x^{\mathbb{N}}.A}{\Gamma\vdash\mathsf{prf}\,p:A[\mathsf{wit}\,p/x]}(\mathsf{prf})\qquad \dfrac{\Gamma\vdash t:\exists x^{\mathbb{N}}.A}{\Gamma\vdash\mathsf{wit}\,t:\mathbb{N}}(\mathsf{wit})\qquad \dfrac{n\in\mathbb{N}}{\Gamma\vdash n:\mathbb{N}}$$

Then, refl is a proof term for equality, and $\mathsf{subst}\, p\, q$ allows us to use a proof of an equality $t = u$ to convert a formula $A(t)$ into $A(u)$:

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

The reduction rules for this language, which are safe with respect to typing, are then:

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

Starting from this (sound) minimal language, Herbelin showed that its classical extension with the control operators $\mathtt{call/cc}_k$ and $\mathsf{throw}\, k$ (that are similar to those presented in the previous section) permits to derive a proof of $0 = 1$ [17]. The $\mathtt{call/cc}_k$ operator, which is a binder for the variable k, is intended to catch its surrounding evaluation context. On the contrary, $\mathsf{throw}\, k$ discards the current context and restores the context captured by $\mathtt{call/cc}_k$. The addition to the type system of the typing rules for these operators:

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

allows the definition of the following proof:

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

Intuitively such a proof catches the context, gives 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.

If, besides, the following reduction rules[6] are added:

$$\begin{array}{rcll} \mathsf{wit}\,(\mathtt{call/cc}_k\, p) & \to & \mathtt{call/cc}_k(\mathsf{wit}\,(p[k(\mathsf{wit}\,\{\ \})/k])) & \\ \mathtt{call/cc}_k\, t & \to & t & (k \notin FV(t)) \end{array}$$

then we can formally derive a proof of $1 = 0$. Indeed, the term $\mathsf{wit}\, p_0$ will reduce to $\mathtt{call/cc}_k\, 0$, which itself reduces to 0. The proof term refl is thus a proof of $\mathsf{wit}\, p_0 = 0$, and we obtain the following proof of $1 = 0$:

$$\frac{\dfrac{\vdash p_0 : \exists x^{\mathbb{N}}.x = 1}{\vdash \mathsf{prf}\, p_0 : \mathsf{wit}\, p_0 = 1}\ (\mathsf{prf}) \qquad \dfrac{\mathsf{wit}\, p_0 \to 0}{\vdash \mathsf{refl} : \mathsf{wit}\, p_0 = 0}\ (\mathsf{refl})}{\vdash \mathsf{subst}\,(\mathsf{prf}\, p_0)\ \mathsf{refl} : 1 = 0}\ (\mathsf{subst})$$

The bottom line of this example is that the same proof p_0 is behaving differently in different contexts thanks to control operators, causing inconsistencies between the witness and its certificate. The easiest and usual approach (in natural deduction) to prevent this is to impose a restriction to values (which are already reduced) for proofs appearing inside dependent types and within the operators wit and prf, together with a call-by-value discipline. In the present example, this would prevent us from writing $\mathsf{wit}\, p_0$ and $\mathsf{prf}\, p_0$.

[6]Technically this requires to extend the language to authorize the construction of terms $\mathtt{call/cc}_k\, t$ and of proofs $\mathsf{throw}\, t$. The first rule expresses that $\mathtt{call/cc}_k$ captures the context $\mathsf{wit}\,\{\ \}$ and replaces every occurrence of $\mathsf{throw}\, k\, t$ with $\mathsf{throw}\, k\ (\mathsf{wit}\, t)$. The second one just expresses the fact that $\mathtt{call/cc}_k$ can be dropped when applied to a term t which does not contain the variable k.

2.3 A minimal language with value restriction

In this section, we will focus on value restriction in a similar framework, and show that the obtained proof system is coherent. We will then see, in Section 3, how to relax this constraint. We follow here the stratified presentation[7] from the previous section. We place ourselves in the framework of the $\lambda\mu\tilde{\mu}$-calculus to which we add:

- a language of *terms* which contain an encoding[8] 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, called respectively 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 6.2. The syntax of the corresponding system, that we call dL, is given by:

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

The formulas are defined by:

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

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 inconsistencies [17]. The substitution at different places of a proof which can backtrack, as the call-by-name evaluation strategy does, is thus an unsafe operation. 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 $c \leadsto c'$ the reduction of commands), follow the call-by-value evaluation principle. In particular one can see that whenever a 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[9].

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:

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

[7]This design choice is usually a matter of taste and might seem unusual for some readers. However, it has the advantage of exhibiting the different treatments for terms and proofs through the CPS in the next sections.
[8]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.
[9]The reader might recognize the rule (ς) of Wadler's sequent calculus [38].

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

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

Fig. 3. Reduction rules of dL

2.5 Typing rules

As we explained before, 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 imagine that the type system of the $\lambda\mu\tilde\mu$-calculus has been extended to allow dependent products instead of implications. and consider a proof $\lambda a.p : \Pi_{a:A}B$ in front of a context $q\cdot e : \Pi_{a:A}B$. A typing derivation of the corresponding command would be of the form:

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

while this command would 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[10] for this command anymore:

$$\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A \mid \Delta} \qquad \dfrac{\dfrac{\Gamma, a:A \vdash p : \cancel{B[a]} \mid \Delta \quad \Gamma, a:A \mid e : \cancel{B[q]} \vdash \Delta}{\langle p\|e\rangle : \Gamma, a:A \vdash \Delta}\, \textit{Mismatch}}{\Gamma \mid \tilde\mu a.\langle p\|e\rangle : A \vdash \Delta}\,(\tilde\mu)}{\langle q \| \tilde\mu a.\langle p\|e\rangle\rangle : \Gamma \vdash \Delta}\,(\textsc{Cut})$$

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[11] is performed, and by then the problem would be solved. This phenomenon can be seen as a desynchronization of the typing process with respect to computation. The synchronization can be re-established by making explicit a *list of dependencies* σ in the typing rules, which links $\tilde\mu$ variables (here a) to the associated proof term on

[10]Observe that the problem here arises independently of the value restriction (that is whether we consider that q is a value or not), and is peculiar to the sequent calculus presentation.

[11]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 the case that q has produced a value to substitute for a.

$$\dfrac{\Gamma \vdash p : A \mid \Delta;\sigma \quad \Gamma \mid e : B \vdash \Delta;\sigma\{\cdot|p\} \quad B \in A_\sigma}{\langle p \| e\rangle : \Gamma \vdash \Delta;\sigma}\ (\textsc{Cut})$$

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

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

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

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

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

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

$$\dfrac{\Gamma \vdash p : t = u \mid \Delta;\sigma \quad \Gamma \vdash q : B[t/x] \mid \Delta;\sigma}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta;\sigma}\ (\mathsf{subst}) \qquad \dfrac{\Gamma \vdash t : \mathbb{N} \mid \Delta;\sigma}{\Gamma \vdash \mathsf{refl} : t = t \mid \Delta;\sigma}\ (\mathsf{refl})$$

$$\dfrac{}{\Gamma, x : \mathbb{N} \vdash x : \mathbb{N} \mid \Delta;\sigma}\ (\mathrm{Ax}_t) \qquad \dfrac{n \in \mathbb{N}}{\Gamma \vdash \overline{n} : \mathbb{N} \mid \Delta;\sigma}\ (\mathrm{Ax}_n) \qquad \dfrac{\Gamma \vdash p : \exists x.A(x) \mid \Delta;\sigma \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{wit}\, p : \mathbb{N} \mid \Delta;\sigma}\ (\mathsf{wit})$$

Fig. 4. Typing rules of dL

the left-hand side of the command (here q). We can now obtain the following typing derivation:

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

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:

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

We define a list of dependencies σ as a list binding pairs of proof terms[12]:

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

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

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

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

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

$$\frac{\Gamma \vdash p : A \mid \Delta; \sigma \qquad \Gamma \mid e : B \vdash \Delta; \sigma\{\cdot|p\} \qquad B \in A_\sigma}{\langle p \| e \rangle : \Gamma \vdash \Delta; \sigma}\ (\textsc{Cut})$$

Remark 2.2. The reader familiar with explicit substitutions [11] can think of the list of dependencies as a fragment of the substitution that is available when a command c is reduced. Another remark is that the design choice for the (Cut) rule is arbitrary, in the sense that we chose 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 occurrences of p have been replaced by q.

Furthermore, when typing a stack with the $(\rightarrow_l)$ and $(\forall_l)$ rules, we need to drop the open binding in the list of dependencies[13]. 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 trick spares us from defining a second type of sequents $\Gamma \mid e : A \vdash \Delta; \sigma$ 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 list of dependencies. 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 this means that a type in Δ might depend on a variable previously introduced in Γ and vice versa, so that the split into two contexts makes us lose track of the order of introduction of the hypotheses. 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 hypotheses have been introduced.

Example 2.3. The proof $p_1 \triangleq \text{subst}\,(\text{prf}\,p_0)\,\text{refl}$ which was of type $1 = 0$ in Section 2.2 is now incorrect since the backtracking proof p_0, defined by $\mu\alpha.(0, \mu\_.\langle(1,\text{refl})\|\alpha\rangle)$ in our framework, is not a value in $\mathcal{D}$. The proof p_1 should rather be defined by[14] $\mu\alpha.\langle p_0 \| \tilde{\mu}a.\langle \text{subst}\,(\text{prf}\,a)\,\text{refl} \| \alpha\rangle\rangle$ which can only be given the type $1 = 1$.

2.6 Subject reduction

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

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

[13]It is easy to convince ourselves 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.

[14]That is to say **let** $a = p_0$ **in** subst (prf a) refl in natural deduction.

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 2.4 (Dependencies weakening). *For any list of dependencies 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 one 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 † can indeed be replaced by any proof term when typing a context.

Corollary 2.5. *If $\sigma \Rrightarrow \sigma'$, then for any p, e, Γ, Δ:*

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

Proof. Assume that e is of the form $\tilde\mu a.c$ (other cases are trivial), then we have $c : \Gamma \vdash \Delta; \sigma\{a|\dagger\}$. By definition of † and from the hypothesis, we get that $\sigma\{a|\dagger\} \Rrightarrow \sigma'$, *i.e.* that $c : \Gamma \vdash \Delta; \sigma'$ is derivable. By applying the previous Lemma, we get that $c : \Gamma \vdash \Delta; \sigma'\{a|p\}$ is derivable for any proof p, whence the result. □

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

Lemma 2.6 (Safe term substitution). *If $\Gamma \vdash t : \mathbb{N} \mid \Delta; \varepsilon$ then:*

(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 2.7 (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 2.8 (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) $\Gamma \mid e : B \vdash \Delta, \alpha : A, \Delta'; \sigma \Rightarrow \Gamma \mid e[e/\alpha] : B \vdash \Delta, \Delta'; \sigma$,
(4) $\Gamma \vdash u : \mathbb{N} \mid \Delta, \alpha : A, \Delta'; \sigma \Rightarrow \Gamma \vdash u : \mathbb{N} \mid \Delta, \Delta'; \sigma]$.

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

We can now prove the preservation of typing through reduction, using the previous lemmas for rules which perform a substitution, and the list of dependencies to resolve local desynchronizations for dependent types.

Theorem 2.9 (Subject reduction). *If c, c' are two commands of* dL *such that $c : (\Gamma \vdash \Delta; \varepsilon)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta; \varepsilon)$.*

Proof. The proof is done by induction on the typing derivation of $c : (\Gamma \vdash \Delta; \varepsilon)$, assuming that for each typing proof, the conversion rules are always pushed down and right as much as possible.

To save some space, we sometimes omit the list of dependencies when empty, writing $c : \Gamma \vdash \Delta$ instead of $c : \Gamma \vdash \Delta; \varepsilon$, and we denote the composition of consecutive rules $(\equiv_l)$ as:

$$\frac{\Gamma \mid e : B \vdash \Delta; \sigma}{\Gamma \mid e : A \vdash \Delta; \sigma}\,(\equiv_l)$$

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:

$$\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, x : \mathbb{N} \vdash p : A \mid \Delta}}{\Gamma \vdash \lambda x.p : \forall x^{\mathbb{N}}.A \mid \Delta}\,(\forall_r) \qquad \dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma \vdash t : \mathbb{N} \mid \Delta} \quad \dfrac{\Pi_e}{\Gamma \mid e : B[t/x] \vdash \Delta; \{\cdot|\dagger\}}}{\Gamma \mid t \cdot e : \forall x^{\mathbb{N}}.B \vdash \Delta; \{\cdot|\lambda x.p\}}\,(\forall_l)}{\Gamma \mid t \cdot e : \forall x^{\mathbb{N}}.A \vdash \Delta; \{\cdot|\lambda x.p\}}\,(\equiv_l)}{\langle \lambda x.p \| t \cdot e \rangle : \Gamma \vdash \Delta}\,(\textsc{Cut})$$

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 the fact that $\Gamma, x : \mathbb{N} \vdash p : A \mid \Delta$ and $\Gamma \vdash t : \mathbb{N} \mid \Delta$, by Lemma 2.6 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:

$$\dfrac{\dfrac{\Pi'_p}{\Gamma \vdash p[t/x] : A[t/x] \mid \Delta} \qquad \dfrac{\dfrac{\Pi_e}{\Gamma \mid e : B[t/x] \vdash \Delta; \{\cdot|p[t/x]\}}}{\Gamma \mid e : A[t/x] \vdash \Delta; \{\cdot|p[t/x]\}}\,(\equiv_l)}{\langle p[t/x] \| e \rangle : \Gamma \vdash \Delta}\,(\textsc{Cut})$$

using Corollary 2.5 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:

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

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 cases, we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\Pi_q}{\Gamma \vdash q : A' \mid \Delta}}{\Gamma \vdash q : A \mid \Delta}\,(\equiv_l) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, a : A \vdash p : B \mid \Delta}}{\Gamma, a : A \vdash p : B' \mid \Delta}\,(\equiv_r) \quad \dfrac{\Pi_e}{\Gamma, a : A \mid e : B'_q \vdash \Delta; \{a|q\}\{\cdot|p\}} \quad B'_q \in B'_{\{a|q\}}}{\langle p \| e \rangle : \Gamma, a : A \vdash \Delta; \{a|q\}}\,(\textsc{Cut})}{\Gamma \mid \tilde{\mu} a.\langle p \| e \rangle : A \vdash \Delta; \{.|q\}}\,(\tilde{\mu})}{\langle q \| \tilde{\mu} a.\langle p \| e \rangle \rangle : \Gamma \vdash \Delta}\,(\textsc{Cut})$$

using Corollary 2.5 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:

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

We get a proof that $c[e/\alpha]:\Gamma\vdash\Delta$ is valid by Lemma 2.8.

- **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:

$$\dfrac{\dfrac{\Pi_V}{\Gamma\vdash V:A\mid\Delta}\qquad \dfrac{\dfrac{\dfrac{\Pi_c}{c:\Gamma,a:A'\vdash\Delta;\{a|V\}}}{\Gamma\mid\tilde\mu a.c:A'\vdash\Delta;\{\cdot|V\}}\,(\tilde\mu)}{\Gamma\mid\tilde\mu a.c:A\vdash\Delta;\{\cdot|V\}}\,(\equiv_l)}{\langle V\|\tilde\mu a.c\rangle:\Gamma\vdash\Delta}\,(\textsc{Cut})$$

We first observe that we can derive the following proof:

$$\dfrac{\dfrac{\Pi_V}{\Gamma\vdash V:A\mid\Delta}}{\Gamma\vdash V:A'\mid\Delta}\,(\equiv_l)$$

and we get a proof for $c[V/a]:\Gamma\vdash\Delta;\{V|V\}$ by Lemma 2.7. We finally get a proof for $c[V/a]:\Gamma\vdash\Delta$ by Lemma 2.4.

- **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:

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

We can build the following derivation:

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

where $\Pi_{(t,a)}$ is as expected, observing that since $p\notin\mathcal{D}$, the binding $\{\cdot|(t,p)\}$ is the same as $\{\cdot|\dagger\}$, and we can apply Corollary 2.5 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:

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

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

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

- **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 similar to the case $\langle (t,p)\|e\rangle$.

- **Case** $c[t] \rightsquigarrow c[t']$ with $t \to t'$.

Immediate by observing that by definition of the relation $\equiv$, we have $A[t] \equiv A[t']$ for any A.

□

2.7 Soundness

We here give a proof of the soundness of dL with a value restriction. The proof is based on an embedding into the $\lambda\mu\tilde{\mu}$-calculus extended with pairs, 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 $\tilde{\mu}(a_1,a_2).c$). 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[15], turning the dependent products into arrows and the dependent sum into a pair. The erasure procedure is defined by:

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

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

[15]The use of erasure functions is a very standard technique in the systems of the λ-cube, see for instance [32] or [37].

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

(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

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

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

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 $\bar n \in V$. Note that we translate differently $\mathsf{subst}\,V\,q$ and $\mathsf{subst}\,p\,q$ to simplify the proof of Proposition 2.12.

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

Lemma 2.10. *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 list of dependencies 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 2.11. *The following holds for any contexts Γ, Δ and any type A:*

(1) *For any command c, if $c : \Gamma \vdash \Delta; \sigma$, then $c^* : \Gamma^* \vdash \Delta^*$.*
(2) *For any proof p, if $\Gamma \vdash p : A \mid \Delta; \sigma$, then $\Gamma^* \vdash p^* : A^* \mid \Delta^*$.*
(3) *For any context e, if $\Gamma \mid e : A \vdash \Delta; \sigma$, then $\Gamma^* \mid e^* : A^* \vdash \Delta^*$.*

Proof. By induction on typing derivations. The fourth item of the previous lemma shows that the list of dependencies becomes useless: since $A \in B_\sigma$ implies $A^* = B^*$, it is no longer needed for the (cut)-rule. Consequently, it can also 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 that $\mathsf{refl}^* = \lambda x.x : \mathbb{N}^* \to \mathbb{N}^*$.

The only non-direct cases are $\mathsf{subst}\,p\,q$, with p not a value, and (t,p). To prove the former with $p \notin V$, we have to show that if:

$$\dfrac{\Gamma \vdash p : t = u \mid \Delta;\sigma \qquad \Gamma \vdash q : B[t/x] \mid \Delta;\sigma}{\Gamma \vdash \mathsf{subst}\,p\,q : B[u/x] \mid \Delta;\sigma}\ (\mathrm{subst})$$

then $\mathsf{subst}\,p\,q^* = \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha\rangle \| \alpha\rangle\rangle : B[u/x]^*$. According to Lemma 2.10, we have that $B[u/x]^* = B[t/x]^* = B^*$. By induction hypothesis, we have proofs of $\Gamma^* \vdash p^* : \mathbb{N}^* \to \mathbb{N}^* \mid \Delta^*$ and of $\Gamma^* \vdash q^* : B \mid \Delta^*$. Using the notation $\eta_{q^*} \triangleq \mu\alpha.\langle q^* \| \alpha\rangle$, we can derive:

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

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

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

□

We can then deduce the normalization of dL from the normalization of the $\lambda\mu\tilde{\mu}$-calculus [34], by showing that the translation preserves the normalization in the sense that if c does not normalize, then neither does c^*.

Proposition 2.12. *If c is a command such that c^* normalizes, then c normalizes.*

Proof. We prove this by contraposition, by showing that if c does not normalize (*i.e.* if it admits an infinite reduction path), then $c*$ does not normalize either. We will actually prove a slightly more precise statement, namely that each step of reduction is reflected into at least one step through the translation:

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

Assuming this 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.

We shall now prove the previous statement by case analysis of the reduction $c_1 \rightsquigarrow c_2$.

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

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

□

Theorem 2.13. *If $c : (\Gamma \vdash \Delta; \varepsilon)$, then c normalizes.*

Proof. Proof by contradiction: if c does not normalize, then by Proposition 2.12 neither does c^*. However, by Proposition 2.11 we have that $c^* : \Gamma^* \vdash \Delta^*$. This is absurd since any well-typed command of the $\lambda\mu\tilde\mu$-calculus normalizes [34]. □

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

Theorem 2.14 (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 exists such a command $c : (\vdash)$. By normalization, we can reduce 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 shows that p cannot drop the context $\star$ when reducing, since it would give rise to the command $c : (\vdash)$. We can still reduce $\langle p \| \star\rangle$ to a command c in normal form, and see that c has to be of the shape $\langle V \| \star\rangle$ (by the same kind of reasoning, using the fact that c cannot reduce and that $c : (\vdash \star : \bot)$ by subject reduction). Therefore, V is a value of type $\bot$. Since there is no typing rule that can give the type $\bot$ to a value, this is absurd. □

2.8 Toward a continuation-passing style translation

The difficulties we encountered while defining our system mostly came from the interaction between classical control and dependent types. Removing one of these two ingredients leaves us with a sound system in both cases. Without dependent types, our calculus amounts to the usual $\lambda\mu\tilde\mu$-calculus. And without classical control, we would obtain an intuitionistic dependent type theory that we could easily prove sound.

To prove the correctness of our system, we might be tempted to define a translation to a subsystem without dependent types, or without classical control. 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 classical control. 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 list of dependencies 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 that 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[16]:

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

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

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

Since $[\![p]\!][[\![V]\!]/a]$ 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:

$$(\lambda\alpha.[\![c]\!](\lambda a.[\![p]\!]))[\![e]\!] \to [\![c]\!]\,[\![e]\!].$$

Indeed, through such a translation, $\mu\alpha$ would only be able to catch the local continuation, and the term would end in $[\![c]\!][\![e]\!]$ instead of $[\![c]\!]$. We thus need to restrict ourselves 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[17] to temporarily encapsulate the evaluation of q when reducing such a command:

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

Under the guarantee that q will not throw away the continuation[18] $\tilde{\mu}a.\langle p \| \hat{\mathrm{tp}} \rangle$, this command is safe and will mimic the aforedescribed reduction:

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

This will also allow us to restrict the use of the list of dependencies to the derivation of judgments involving a delimited continuation, and to fully absorb the potential inconsistency in the type of $\hat{\mathrm{tp}}$. 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 will thus relax this restriction to allow more proof terms within dependent types.

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

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

[18]Otherwise, this could lead to an ill-formed command $\langle \mu\hat{\mathrm{tp}}.c \| e \rangle$ where c does not contain $\hat{\mathrm{tp}}$.

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

(a) Language

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

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

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

We can follow several intuitions. First, we saw at the end of the previous section that we could actually allow any proof term 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 write $\star$ in Figure 6), and which do not contain application[19]. We insist on the fact that this defines a syntactic subset of proofs. Indeed, $\star$ is only a notation and any proof defined with only one continuation variable is α-convertible to denote this continuation variable with $\star$. 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 recent work [22] 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, and hence is compatible with the dependencies of Lepigre's calculus.

From now on, we consider the system dL of Section 2 extended with delimited continuations, which we call $\mathrm{dL}_{\hat{\mathtt{tp}}}$, and we 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

[19]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.

to α-conversion for the context variables[20]. 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 $q \in \text{NEF}$ (resp. $\langle \text{prf}\, p \| e\rangle$), a delimited continuation is now produced during the reduction of the proof term q (resp. p) that is involved in the list of 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[21]. 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

Regarding the typing rules, which are given in Figure7, we 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 list of dependencies; plus the new rule $(\hat{\mathfrak{tp}}_I)$ for the introduction of delimited continuations. The dependent mode is used to type commands and contexts involving $\hat{\mathfrak{tp}}$, and we use the symbol $\vdash_d$ to denote these 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), we thus add the rules:

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

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

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

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

Proof. 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 a derivation for the hypothesis is of the form (we assume the

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

[21] 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 for reducing proofs embedded in terms, 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.

Regular mode:

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

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash a : A \mid \Delta}\ (\mathrm{Ax}_r) \qquad \frac{(\alpha : A) \in \Delta}{\Gamma \mid \alpha : A \vdash \Delta}\ (\mathrm{Ax}_l)$$

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

$$\frac{\Gamma, a : A \vdash p : B \mid \Delta}{\Gamma \vdash \lambda a.p : \Pi_{a:A}B \mid \Delta}\ (\to_r) \qquad \frac{\Gamma \vdash q : A \mid \Delta \quad \Gamma \mid e : B[q/a] \vdash \Delta \quad q \notin \mathcal{D} \Rightarrow a \notin FV(B)}{\Gamma \mid q \cdot e : \Pi_{a:A}B \vdash \Delta}\ (\to_l)$$

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

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

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

$$\frac{\Gamma \vdash p : t = u \mid \Delta \quad \Gamma \vdash q : B[t/x] \mid \Delta}{\Gamma \vdash \mathsf{subst}\, p\, q : B[u/x] \mid \Delta}\ (\mathsf{subst}) \qquad \frac{\Gamma \vdash t : \mathbb{N} \mid \Delta}{\Gamma \vdash \mathsf{refl} : t = t \mid \Delta}\ (\mathsf{refl})$$

$$\frac{}{\Gamma, x : \mathbb{N} \vdash x : \mathbb{N} \mid \Delta}\ (\mathrm{Ax}_t) \qquad \frac{n \in \mathbb{N}}{\Gamma \vdash \overline{n} : \mathbb{N} \mid \Delta}\ (\mathrm{Ax}_n) \qquad \frac{\Gamma \vdash p : \exists x A(x) \mid \Delta \quad p \in \mathcal{D}}{\Gamma \vdash \mathsf{wit}\, p : \mathbb{N} \mid \Delta}\ (\mathsf{wit})$$

Dependent mode:

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

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

Fig. 7. Type system for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$

conv-rules have been pushed to the left of cuts):

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1}{\Gamma \vdash p_1 : A_1 \mid \Delta, \star : A} \quad \dfrac{\dfrac{\dfrac{\Pi_2}{\Gamma, a_1 : A_1 \vdash p_2 : A \mid \Delta, \star : A} \quad \dfrac{}{\cdots \mid \star : A \vdash \Delta, \star : A}}{\langle p_2 \| \star \rangle : \Gamma, a_1 : A_1 \vdash \Delta, \star : A}\ (\textsc{Cut})}{\Gamma \mid \tilde{\mu}a_1.\langle p_2 \| \star \rangle : A_1 \vdash \Delta, \star : A}\ (\tilde{\mu})}{\langle p_1 \| \tilde{\mu}a_1.\langle p_2 \| \star \rangle\rangle : \Gamma \vdash \Delta, \star : A}\ (\textsc{Cut})}{\dfrac{\Gamma \vdash \mu\star.\langle p_1 \| \tilde{\mu}a_1.\langle p_2 \| \star \rangle\rangle : A \mid \Delta}{\Gamma \vdash \mu\star.c : A \mid \Delta}}\ (\mu) \quad \dfrac{\Pi_e}{\Gamma \mid e : A \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \{\cdot|\mu\star.c\}}}{\langle \mu\star.c \| e \rangle : \Gamma \vdash_d \Delta, \hat{\mathfrak{tp}} : B; \varepsilon}\ (\textsc{Cut})$$

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:

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

It suffices to prove that if the list of dependencies is 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 list of dependencies 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:

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

□

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 2.9, because most of the rules do not involve dependencies. We only give some 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:

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

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

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

- **Case** $\langle \mathrm{prf}\,p \| e\rangle \rightsquigarrow \langle \mu\hat{\mathtt{tp}}.\langle p \| \tilde\mu a.\langle \mathrm{prf}\,a \| \hat{\mathtt{tp}}\rangle\rangle \| e\rangle$.

  We prove it in the most general case, that is when this reduction occurs under a delimited continuation. A typing derivation for the command on the left has to be of the form:

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

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

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

  so that we can prove that for any b:

$$\sigma\{b|\mathrm{prf}\,p\} \Rrightarrow \sigma\{b|\mu\hat{\mathtt{tp}}.\langle p \| \tilde\mu a.\langle \mathrm{prf}\,a \| \hat{\mathtt{tp}}\rangle\rangle\}.$$

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

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

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

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

- **Case** $\langle \mu\hat{\mathtt{tp}}.\langle p \| \hat{\mathtt{tp}}\rangle \| e\rangle \rightsquigarrow \langle p \| e\rangle$.

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

  This case corresponds exactly to Theorem 2.9, 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.

□

Remark 3.3. Interestingly, we could have already taken $\mathcal{D} \triangleq$ NEF in dL and still be able to prove the subject reduction property. The only difference would have been for the case $\langle \mu\alpha.c \| e\rangle \rightsquigarrow c[e/\alpha]$ when $\mu\alpha.c$ is NEF. Indeed, we would have had to prove that such a reduction step is compatible with the list of dependencies, as in the proof for $\mathrm{dL}_{\hat{\mathtt{tp}}}$, which essentially amounts to Lemma 3.1. This shows that the relaxation to the NEF fragment is valid even without delimited continuations.

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

(a) Language and formulas

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

(b) Reduction rules

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

$$
\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : \Pi_{a:A} B}\,(\rightarrow_I) \qquad \frac{\Gamma \vdash p : \Pi_{a:A} B \quad \Gamma \vdash q : A}{\Gamma \vdash p\, q : B[q/a]}\,(\rightarrow_E) \qquad \frac{\Gamma, x : \mathbb{N} \vdash p : A}{\Gamma \vdash \lambda x.p : \forall x^{\mathbb{N}} A}\,(\forall_I^1)
$$

$$
\frac{\Gamma \vdash p : \forall x^{\mathbb{N}}.A \quad \Gamma \vdash t : \mathbb{N}}{\Gamma \vdash p\, t : A[t/x]}\,(\forall_E^1) \qquad \frac{\Gamma \vdash p : A \quad X \notin FV(\Gamma)}{\Gamma \vdash p : \forall X.A}\,(\forall_I^2) \qquad \frac{\Gamma \vdash p : \forall X.A}{\Gamma \vdash p : A[P/X]}\,(\forall_E^2)
$$

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

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

(c) Type system

Fig. 8. Target language

To sum up, the restriction to NEF is sufficient to obtain a sound type system, but is not enough to obtain a calculus suitable for a continuation-passing style translation. As we will now see, delimited continuations are crucial for the soundness of the CPS translation. Observe that they also provide us with a type system in which the scope of dependencies is more delimited.

4 A CONTINUATION-PASSING STYLE TRANSLATION

We shall now see how to define a continuation-passing style translation from $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ to an intuitionistic type theory, and use this translation to prove the soundness of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. Continuation-passing style translations are indeed very useful to embed languages with classical control into purely functional ones [7, 15]. From a logical point of view, they generally amount to negative translations that allow us to embed classical logic into intuitionistic logic [9]. Yet, we know that removing classical control (*i.e.* classical logic) from 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 the target language to be an intuitionistic theory in natural deduction that has exactly the same elements as $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, except the classical control. The language distinguishes between terms (of type $\mathbb{N}$) and proofs, it also includes dependent sums and products for types referring to terms, as well as a dependent product at the level of proofs. As 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 8.

The type system, also presented in Figure 8, 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 in $\mathrm{dL}_{\hat{\text{tp}}}$, 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:

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

4.2 Translation of proofs and terms

We can now define the continuation-passing style translation of terms, proofs, contexts and commands. The translation is given in Figure 9, in which we tag some lambdas with a bullet $\lambda^\bullet$ for technical reasons. The translation of delimited continuations follows the intuition we presented in Section 2.8, and the definition for stacks $t \cdot e$ and $q \cdot e$ (with q NEF) inlines 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):

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

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 the former 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 somewhat strange definition corresponds to the intuition that we reduce $[\![t]\!]_t$ within a delimited continuation[22], 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 9.

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

In particular, we have :

$$[\![t]\!]_t\,\lambda x.x \rightarrow_\beta^* t^+ \qquad \textit{and} \qquad [\![p_N]\!]_p\,\lambda a.a \rightarrow_\beta^* p_N^+$$

PROOF. Straightforward mutual induction on the structure of terms and NEF proofs, adding similar induction hypothesis for NEF contexts and commands. The terms t^+ and proofs p^+ are given in Figure 10. We detail the case (t,p) with $p \in$ NEF to give an insight of the proof.

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

[22]In fact, we could define it formally, which would require a kind of co-delimited continuation.

$$
\begin{array}{llll}
[\![\mathsf{wit}\,p]\!]_t & \triangleq \lambda k.[\![p]\!]_p\,(\lambda^{\bullet} q.k\,(\mathsf{wit}\,q)) & [\![n]\!]_{V_t} & \triangleq \bar{n} \\
[\![V_t]\!]_{V_t} & \triangleq \lambda k.k\,V_t & [\![x]\!]_{V_t} & \triangleq x \\
\hline
[\![a]\!]_V & \triangleq a & [\![\mathsf{refl}]\!]_V & \triangleq \mathsf{refl} \\
[\![\lambda a.p]\!]_V & \triangleq \lambda^{\bullet} a.[\![p]\!]_p & [\![\lambda x.p]\!]_V & \triangleq \lambda^{\bullet} x.[\![p]\!]_p \\
[\![(V_t,V_p)]\!]_V & \triangleq ([\![V_t]\!]_{V_t},[\![V]\!]_V) & & \\
\hline
[\![V]\!]_p & \triangleq \lambda k.k\,[\![V]\!]_V & [\![\mu\hat{\mathtt{tp}}.c]\!]_p & \triangleq \lambda k.[\![c]\!]_{\hat{\mathtt{tp}}}k \\
[\![\mu\alpha.c]\!]_p & \triangleq \lambda^{\bullet}\alpha.[\![c]\!]_c & & \\
[\![\mathsf{prf}\,p]\!]_p & \triangleq \lambda^{\bullet} k.([\![p]\!]_p\,(\lambda^{\bullet} q\lambda k'.k'\,(\mathsf{prf}\,q)))\,k & & \\
[\![(t,p)]\!]_p & \triangleq \lambda^{\bullet} k.[\![p]\!]_p([\![t]\!]_t\,(\lambda x\lambda^{\bullet} a.k\,(x,a))) & & \\
[\![\mathsf{subst}\,V\,q]\!]_p & \triangleq \lambda k.[\![q]\!]_p(\lambda^{\bullet} q'.k\,(\mathsf{subst}\,[\![V]\!]_V\,q')) & & \\
[\![\mathsf{subst}\,p\,q]\!]_p & \triangleq \lambda k.[\![p]\!]_p\,(\lambda^{\bullet} p'.[\![q]\!]_p(\lambda^{\bullet} q'.k\,(\mathsf{subst}\,p'\,q'))) & & (p\notin V) \\
\hline
[\![\alpha]\!]_e & \triangleq \alpha & [\![\tilde{\mu}a.c]\!]_e & \triangleq \lambda^{\bullet} a.[\![c]\!]_c \\
[\![t\cdot e]\!]_e & \triangleq \lambda p.([\![t]\!]_t\,(\lambda^{\bullet} v.p\,v))\,[\![e]\!]_e & & \\
[\![q_N\cdot e]\!]_e & \triangleq \lambda p.([\![q_N]\!]_p\,(\lambda^{\bullet} v.p\,v))\,[\![e]\!]_e & & (q_N\in\textsc{nef}) \\
[\![q\cdot e]\!]_e & \triangleq \lambda^{\bullet} p.[\![q]\!]_p\,(\lambda^{\bullet} v.p\,v\,[\![e]\!]_e) & & (q\notin\textsc{nef}) \\
\hline
[\![\langle p\|e\rangle]\!]_c & \triangleq [\![e]\!]_e\,[\![p]\!]_p & [\![\langle p\|\hat{\mathtt{tp}}\rangle]\!]_{\hat{\mathtt{tp}}} & \triangleq [\![p]\!]_p \\
[\![\langle p\|e\rangle]\!]_{\hat{\mathtt{tp}}} & \triangleq [\![p]\!]_p\,[\![e]\!]_{e_{\hat{\mathtt{tp}}}} \quad (e\neq\hat{\mathtt{tp}}) & [\![\tilde{\mu}a.c]\!]_{e_{\hat{\mathtt{tp}}}} & \triangleq \lambda^{\bullet} a.[\![c]\!]_{\hat{\mathtt{tp}}}
\end{array}
$$

Fig. 9. Continuation-passing style translation

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

Fig. 10. Linearity of the translation for NEF proofs

□

Moreover, we can verify that the translation preserves the reduction:

Proposition 4.2. *If c,c' are two commands of* $\mathrm{dL}_{\hat{\mathtt{tp}}}$ *such that $c \rightsquigarrow c'$, then* $[\![c]\!]_c =_\beta [\![c']\!]_c$

Proof. Simple proof by induction on the reduction rules for $\rightsquigarrow$, using Lemma 4.1 for cases involving a term t. □

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

We can in fact prove a finer result to show that normalization is preserved through the translation. Namely, we want to prove that any infinite reduction sequence in $\mathrm{dL}_{\hat{\mathtt{tp}}}$ 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.

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 reductions;
(3) we deduce that every infinite sequence of reductions in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ gives rise 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 in a way that is *sufficient* to ensure the preservation of normalization (that is the third point). Instead of stating the result directly and giving a long and tedious proof of its correctness, we will rather sketch its main steps.

First of all, we split the reduction rule $\rightarrow_\beta$ 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 $\lambda^\bullet$'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 $\rightarrow_\beta$ 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 that for any reduction $c \rightsquigarrow c'$, through the translation we have:

$$\begin{tikzcd}
\llbracket c \rrbracket_c \arrow[dr, "*", "\beta"'] & & & & \llbracket c' \rrbracket_c \arrow[dl, "*"', "\mathsf{a}"] \\
& t_0 \arrow[r, "1", "\bullet"'] & t_1 \arrow[r, "*", "\beta"'] & t_2 &
\end{tikzcd}$$

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:

$$\begin{tikzcd}
\llbracket c_0 \rrbracket_c \arrow[dr, "*", "\beta"'] & & & & \llbracket c_1 \rrbracket_c \arrow[dl, "*"', "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & & \llbracket c_2 \rrbracket_c \arrow[dl, "*"', "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & \\
& t_{00} \arrow[r, "1", "\bullet"'] & t_{01} \arrow[r, "*", "\beta"'] & t_{02} & & t_{10} \arrow[r, "1", "\bullet"'] & t_{11} \arrow[r, "*", "\beta"'] & t_{12} & & t_{20} \arrow[r, "1", "\bullet"'] & t_{21} \arrow[r, dashed] & {}
\end{tikzcd}$$

Using the fact that all reductions are deterministic, and that the arrow from $\llbracket c_1 \rrbracket_c$ to t_{02} (and $\llbracket c_2 \rrbracket_c$ 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:

$$\begin{tikzcd}
\llbracket c_0 \rrbracket_c \arrow[d, "*", "\beta"'] & & \llbracket c_1 \rrbracket_c \arrow[d, "*", "\mathsf{a}"'] & & & \llbracket c_2 \rrbracket_c \arrow[d, "*", "\mathsf{a}"'] & & & \\
t_{00} \arrow[r, "1", "\bullet"'] & t_{01} \arrow[r, "*", "\beta"'] & t_{02} \arrow[r, "*", "\beta"'] & t_{10} \arrow[r, "1", "\bullet"'] & t_{11} \arrow[r, "*", "\beta"'] & t_{12} \arrow[r, "*", "\beta"'] & t_{20} \arrow[r, "1", "\bullet"'] & t_{21} \arrow[r, dashed] & {}
\end{tikzcd}$$

This directly implies that $\llbracket c_0 \rrbracket_c$ 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 that our goal is to preserve the existence of an infinite sequence of distinguished steps.

The first generalization consists in allowing distinguished reductions for redexes that are not in head positions. The safety of this generalization follows from this 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 [20]. □

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 performed somewhere in the term. We denote by $\longrightarrow_{\beta^+}$ the extended reduction relation defined as the union $\longrightarrow_\beta \cup \xrightarrow{?}_\bullet$, which is not deterministic. Coming back to the 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 $\llbracket c_0 \rrbracket_c$, 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:

$$\begin{tikzcd}
\llbracket c_0 \rrbracket_c \arrow[dr, "*", "\beta"'] & & & & \llbracket c_1 \rrbracket_c \arrow[dl, "*"', "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & & \llbracket c_2 \rrbracket_c \arrow[dl, "*"', "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & \\
& t_{00} \arrow[r, "1", "\bullet"'] & t_{01} \arrow[r, "*", "\beta^+"'] & t_{02} & & t_{10} \arrow[r, "1", "\bullet"'] & t_{11} \arrow[r, "*", "\beta^+"'] & t_{12} & & t_{20} \arrow[r, "1", "\bullet"'] & t_{21} \arrow[r, dashed] & {}
\end{tikzcd}$$

In the same spirit, if we define $=_\mathsf{a}$ to be the congruence over terms induced by administrative reductions $\longrightarrow_\mathsf{a}$, 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 $\llbracket c_i \rrbracket_c$ by this congruence:

$$\begin{tikzcd}
\llbracket c_0 \rrbracket_c \arrow[dr, "*", "\beta"'] & & & & \llbracket c_1 \rrbracket_c \arrow[dl, equal, "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & & \llbracket c_2 \rrbracket_c \arrow[dl, equal, "\mathsf{a}"] \arrow[dr, "*", "\beta"'] & & & \\
& t_{00} \arrow[r, "1", "\bullet"'] & t_{01} \arrow[r, "*", "\beta^+"'] & t_{02} & & t_{10} \arrow[r, "1", "\bullet"'] & t_{11} \arrow[r, "*", "\beta^+"'] & t_{12} & & t_{20} \arrow[r, "1", "\bullet"'] & t_{21} \arrow[r, dashed] & {}
\end{tikzcd}$$

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

$$\begin{tikzcd}
{[\![c_0]\!]_c} \arrow[d, "*"', "\beta"] & & & {[\![c_1]\!]_c} \arrow[d, equals, "\mathsf{a}"] \\
t_0 \arrow[r, "1"', "\bullet"] & t_1 \arrow[rr, "*", "\beta^+"'] & & t_2
\end{tikzcd} \tag{1}$$

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):

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

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

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

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

which are reflected into 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 $\lambda^{\bullet} v. (\lambda^{\bullet} x. [\![p]\!]_p)\, v \longrightarrow_{\bullet} \lambda^{\bullet} x. [\![p]\!]_p$ where we perform α-conversion to identify $\lambda^{\bullet} v. [\![p]\!]_p[v/x]$ and $\lambda^{\bullet} x. [\![p]\!]_p$. Additionally, to facilitate the comprehension of the steps corresponding to the congruence $=_{\mathsf{a}}$, we use an arrow $\xrightarrow{?}_{\mathsf{a}}$ to denote the possibility of performing an administrative reduction not in head position, defined by:

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

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

- **Case** $\langle \mu \alpha. c \| e \rangle \rightsquigarrow c[e/\alpha]$:
  We have:
  $$\begin{array}{rcl}
  [\![\langle \mu \alpha. c \| e \rangle]\!]_c & = & (\lambda^{\bullet} \alpha. [\![c]\!]_c) [\![e]\!]_e \\
  & \longrightarrow_{\bullet} & [\![c]\!]_c[[\![e]\!]_e/\alpha] = [\![c[e/\alpha]]\!]_c
  \end{array}$$
- **Case** $\langle \lambda a. p \| q \cdot e \rangle \rightsquigarrow \langle q \| \tilde{\mu} a. \langle p \| e \rangle \rangle$:
  We have:
  $$\begin{array}{rcl}
  [\![\langle \lambda a. p \| q \cdot e \rangle]\!]_c & = & (\lambda k. k\, (\lambda^{\bullet} a. [\![p]\!]_p))\, \lambda^{\bullet} p. [\![q]\!]_p\, (\lambda^{\bullet} v. p\, v\, [\![e]\!]_e) \\
  & \longrightarrow_{\mathsf{a}} & (\lambda^{\bullet} p. [\![q]\!]_p\, (\lambda^{\bullet} v. p\, v\, [\![e]\!]_e))\, \lambda^{\bullet} a. [\![p]\!]_p \\
  & \longrightarrow_{\bullet} & [\![q]\!]_p\, (\lambda^{\bullet} v. (\lambda^{\bullet} a. [\![p]\!]_p)\, v\, [\![e]\!]_e) \\
  & \xrightarrow{?}_{\bullet} & [\![q]\!]_p\, (\lambda^{\bullet} a. [\![p]\!]_p\, [\![e]\!]_e) = [\![\langle q \| \tilde{\mu} a. \langle p \| e \rangle \rangle]\!]_c
  \end{array}$$
- **Case** $\langle \lambda a. p \| q_N \cdot e \rangle \overset{q_N \in \mathrm{NEF}}{\rightsquigarrow} \langle \mu \hat{\mathtt{tp}}. \langle q_N \| \tilde{\mu} a. \langle p \| \hat{\mathtt{tp}} \rangle \rangle \| e \rangle$:
  We know by Lemma 4.1 that q_N being NEF, it will use, and use only once, 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 $\llbracket q_N \rrbracket_p\, k \longrightarrow_{\bullet} \llbracket q_N \rrbracket_p\, k'$ in the sense that it corresponds to performing now a reduction that would have been performed in the future. Using this remark, we have:

$$\begin{array}{rcl}
\llbracket \langle \lambda a.p \| q_N \cdot e \rangle \rrbracket_c & = & (\lambda k.k\,(\lambda^{\bullet} a.\llbracket p \rrbracket_p))\,\lambda p.(\llbracket q_N \rrbracket_p\,(\lambda^{\bullet} v.p\,v))\,\llbracket e \rrbracket_e \\
& \xrightarrow{2}_{\mathsf{a}} & (\llbracket q_N \rrbracket_p\,(\lambda^{\bullet} v.(\lambda^{\bullet} a.\llbracket p \rrbracket_p)\,v))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & (\llbracket q_N \rrbracket_p\,(\lambda^{\bullet} a.\llbracket p \rrbracket_p))\llbracket e \rrbracket_e \\
& {}_{\mathsf{a}}\!\longleftarrow & (\lambda k.(\llbracket q_N \rrbracket_p\,(\lambda^{\bullet} a.\llbracket p \rrbracket_p))\,k)\,\llbracket e \rrbracket_e = \llbracket \langle \mu \hat{\mathtt{tp}}.\langle q_N \| \tilde{\mu} a.\langle p \| \hat{\mathtt{tp}} \rangle \rangle \| e \rangle \rrbracket_c
\end{array}$$

- **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 $\llbracket V_t \rrbracket_t = \lambda k.k\,\llbracket V_t \rrbracket_{V_t}$. In particular, it is easy to deduce that $\llbracket p[V_t/x] \rrbracket_p = \llbracket p \rrbracket_p[\llbracket V_t \rrbracket_{V_t}/x]$, and then we have:

$$\begin{array}{rcl}
\llbracket \langle \lambda x.p \| V_t \cdot e \rangle \rrbracket_c & = & (\lambda k.k\,(\lambda^{\bullet} x.\llbracket p \rrbracket_p))\lambda p.(\llbracket V_t \rrbracket_t\,(\lambda^{\bullet} v.p\,v))\,\llbracket e \rrbracket_e \\
& \xrightarrow{2}_{\mathsf{a}} & (\llbracket V_t \rrbracket_t\,(\lambda^{\bullet} v.(\lambda^{\bullet} x.\llbracket p \rrbracket_p)\,v))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\mathsf{a}} & ((\lambda^{\bullet} v.(\lambda^{\bullet} x.\llbracket p \rrbracket_p)\,v)\,\llbracket V_t \rrbracket_{V_t})\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & ((\lambda^{\bullet} x.\llbracket p \rrbracket_p)\,\llbracket V_t \rrbracket_{V_t})\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & (\llbracket p \rrbracket_p[\llbracket V_t \rrbracket_{V_t}/x])\,\llbracket e \rrbracket_e = \llbracket p[V_t/x] \rrbracket_p\,\llbracket e \rrbracket_e = \langle p[V_t/x] \| e \rangle
\end{array}$$

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

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

- **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 :

$$\begin{array}{rcl}
\llbracket \langle (V_t,p) \| e \rangle \rrbracket_c & = & (\lambda^{\bullet} k.\llbracket p \rrbracket_p(\llbracket V_t \rrbracket_t\,(\lambda x \lambda^{\bullet} a.k\,(x,a)))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & \llbracket p \rrbracket_p\,(\llbracket V_t \rrbracket_t\,(\lambda x \lambda^{\bullet} a.\llbracket e \rrbracket_e\,(x,a))) \\
& \longrightarrow_{\mathsf{a}^+} & \llbracket p \rrbracket_p\,((\lambda x \lambda^{\bullet} a.\llbracket e \rrbracket_e\,(x,a))\,\llbracket V_t \rrbracket_{V_t}) \\
& \longrightarrow_{\mathsf{a}^+} & \llbracket p \rrbracket_p\,(\lambda^{\bullet} a.\llbracket e \rrbracket_e\,(\llbracket V_t \rrbracket_{V_t},a)) \\
& {}_{\mathsf{a}^+}\!\longleftarrow & \llbracket p \rrbracket_p\,(\lambda^{\bullet} a.\llbracket (V_t,a) \rrbracket_p\,\llbracket e \rrbracket_e) \\
& {}_{\mathsf{a}^+}\!\longleftarrow & (\lambda k\,\llbracket p \rrbracket_p\,(\lambda^{\bullet} a.\llbracket (V_t,a) \rrbracket_p\,k))\,\llbracket e \rrbracket_e = \llbracket \langle p \| \tilde{\mu} a.\langle (V_t,a) \| e \rangle \rangle \rrbracket_c
\end{array}$$

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

$$\begin{array}{rcl}
\llbracket \langle \mathsf{prf}\,p) \| e \rangle \rrbracket_c & = & \lambda^{\bullet} k.(\llbracket p \rrbracket_p\,(\lambda^{\bullet} a \lambda k'.k'\,(\mathsf{prf}\,a)))\,k)\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & (\llbracket p \rrbracket_p\,(\lambda^{\bullet} a.\lambda k'.k'\,(\mathsf{prf}\,a)))\,\llbracket e \rrbracket_e \\
& {}_{\mathsf{a}}\!\longleftarrow & (\lambda k.(\llbracket p \rrbracket_p\,(\lambda^{\bullet} a.\lambda k'.k'\,(\mathsf{prf}\,a)))\,k)\,\llbracket e \rrbracket_e = \llbracket \langle \mu \hat{\mathtt{tp}}.\langle p \| \tilde{\mu} a.\langle \mathsf{prf}\,a \| \hat{\mathtt{tp}} \rangle \rangle \| e \rangle \rrbracket_c
\end{array}$$

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

$$\begin{array}{rcl}
\llbracket \langle \mathsf{prf}\,(V_t,V_p) \| e \rangle \rrbracket_c & = & \lambda^{\bullet} k.((\lambda k.k\,(\llbracket V_t \rrbracket_V,\llbracket V_p \rrbracket_V))\,(\lambda^{\bullet} q \lambda k'.k'\,(\mathsf{prf}\,q)))\,k)\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & ((\lambda k.k\,(\llbracket V_t \rrbracket_V,\llbracket V_p \rrbracket_V))\,(\lambda^{\bullet} q \lambda k'.k'\,(\mathsf{prf}\,q)))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\mathsf{a}} & ((\lambda^{\bullet} q \lambda k'.k'\,(\mathsf{prf}\,q))(\llbracket V_t \rrbracket_V,\llbracket V_p \rrbracket_V))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\bullet} & (\lambda k'.k'\,(\mathsf{prf}\,(\llbracket V_t \rrbracket_V,\llbracket V_p \rrbracket_V)))\,\llbracket e \rrbracket_e \\
& \longrightarrow_{\mathsf{a}} & \llbracket e \rrbracket_e\,(\mathsf{prf}\,(\llbracket V_t \rrbracket_V,\llbracket V_p \rrbracket_V))) \\
& \xrightarrow{?}_{\bullet} & \llbracket e \rrbracket_e\,\llbracket V_p \rrbracket_V \;{}_{\mathsf{a}}\!\longleftarrow \llbracket \langle V_p \| e \rangle \rrbracket_c
\end{array}$$

- **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:
  $$\begin{array}{rcl} [\![\langle \mathsf{subst}\, p\, q \| e\rangle]\!]_c & = & (\lambda k.[\![p]\!]_p\ (\lambda^{\bullet} a.[\![q]\!]_p (\lambda^{\bullet} q'.k\ (\mathsf{subst}\, a\, q'))))\ [\![e]\!]_e \\ & \longrightarrow_{\mathsf{a}} & [\![p]\!]_p\ (\lambda^{\bullet} a.[\![q]\!]_p (\lambda^{\bullet} q'.[\![e]\!]_e\ (\mathsf{subst}\, a\, q'))) \\ & {}_{\mathsf{a}}\xleftarrow{?} & [\![p]\!]_p\ (\lambda^{\bullet} a.(\lambda k.[\![q]\!]_p (\lambda^{\bullet} q'.k\ (\mathsf{subst}\, a\, q')))\ [\![e]\!]_e) \\ & = & [\![\langle p \| \tilde{\mu} a.\langle \mathsf{subst}\, a\, q \| e\rangle\rangle]\!]_c \end{array}$$
- **Case** $\langle \mathsf{subst}\, \mathsf{refl}\, q \| e\rangle \rightsquigarrow \langle q \| e\rangle$:
  We have:
  $$\begin{array}{rcl} [\![\langle \mathsf{subst}\, \mathsf{refl}\, q \| e\rangle]\!]_c & = & (\lambda k.[\![q]\!]_p\ (\lambda^{\bullet} q'.k\ (\mathsf{subst}\, \mathsf{refl}\, q')))\ [\![e]\!]_e \\ & \longrightarrow_{\mathsf{a}} & [\![q]\!]_p\ (\lambda^{\bullet} q'.[\![e]\!]_e\ (\mathsf{subst}\, \mathsf{refl}\, q')) \\ & \xrightarrow{?}_{\bullet} & [\![q]\!]_p\ (\lambda^{\bullet} q'.[\![e]\!]_e\ q') \\ & \xrightarrow{?}_{\bullet} & [\![q]\!]_p\ [\![e]\!]_e = [\![\langle q \| e\rangle]\!]_c \end{array}$$
- **Case** $\langle \mu \hat{\mathrm{tp}}.\langle p \| \hat{\mathrm{tp}}\rangle \| e\rangle \rightsquigarrow \langle p \| e\rangle$:
  We have:
  $$\begin{array}{rcl} [\![\langle \mu \hat{\mathrm{tp}}.\langle p \| \hat{\mathrm{tp}}\rangle \| e\rangle]\!]_c & = & (\lambda k.[\![p]\!]_p k)\ [\![e]\!]_e \\ & \longrightarrow_{\mathsf{a}} & [\![p]\!]_p\ [\![e]\!]_e = [\![\langle p \| e\rangle]\!]_c \end{array}$$
- **Case** $c \rightsquigarrow c' \Rightarrow \langle \mu \hat{\mathrm{tp}}.c \| e\rangle \rightsquigarrow \langle \mu \hat{\mathrm{tp}}.c' \| e\rangle$:
  By induction hypothesis, we get that $[\![c]\!]_c \xrightarrow{*}_{\beta^+} t =_{\mathsf{a}} [\![c']\!]_c$ for some term t. Therefore, we have:
  $$\begin{array}{rcl} \langle \mu \hat{\mathrm{tp}}.c \| e\rangle & = & (\lambda k.[\![c]\!]_c\ k)\ [\![e]\!]_e \\ & \longrightarrow_{\mathsf{a}} & [\![c]\!]_c\ [\![e]\!]_e \\ & \xrightarrow{*}_{\beta^+} & t\ [\![e]\!]_e \\ & =_{\mathsf{a}} & [\![c']\!]_c\ [\![e]\!]_e \\ & {}_{\mathsf{a}}\longleftarrow & (\lambda k.[\![c']\!]_c\ k)\ [\![e]\!]_e = \langle \mu \hat{\mathrm{tp}}.c' \| e\rangle \end{array}$$
- **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 $\mathrm{dL}_{\hat{\mathrm{tp}}}$ 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 $\check{\mathrm{tp}}/\tilde{\mu}\check{\mathrm{tp}}$ for (co-)delimited continuations to allow for a small-step definition of terms reduction:
  $$\begin{array}{rcl|rcl} \langle \lambda x.p \| t \cdot e\rangle & \rightsquigarrow & \langle \mu \hat{\mathrm{tp}}.\langle t \| \tilde{\mu} x.\langle p \| \hat{\mathrm{tp}}\rangle\rangle \| e\rangle & \langle V_t \| \tilde{\mu} x.c_t\rangle & \rightsquigarrow & c_t[V_t/x] \\ \langle \mathsf{wit}\, p \| e_t\rangle & \rightsquigarrow & \langle p \| \tilde{\mu} a.\langle \mathsf{wit}\, a \| e_t\rangle\rangle & \langle \mathsf{wit}\,(V_t, V_p) \| e_t\rangle & \rightsquigarrow & \langle V_t \| e_t\rangle \\ \langle (t,p) \| e\rangle & \rightsquigarrow & \langle p \| \tilde{\mu}\check{\mathrm{tp}}.\langle t \| \tilde{\mu} x.\langle \check{\mathrm{tp}} \| \tilde{\mu} a.\langle (x,a) \| e\rangle\rangle\rangle\rangle & \langle V_p \| \tilde{\mu}\check{\mathrm{tp}}.\langle \check{\mathrm{tp}} \| e\rangle\rangle & \rightsquigarrow & \langle V_p \| e\rangle \end{array}$$
  $$c \rightsquigarrow c' \Rightarrow \langle p \| \tilde{\mu}\check{\mathrm{tp}}.c\rangle \rightsquigarrow \langle p \| \tilde{\mu}\check{\mathrm{tp}}.c'\rangle$$

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

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:

$$\begin{aligned}
\langle \lambda x.p \| t \cdot e\rangle &= (\lambda^{\bullet} k.k\, \lambda^{\bullet} x.\llbracket p\rrbracket_p)\, (\lambda p.(\llbracket t\rrbracket_t\, (\lambda^{\bullet} v.p\, v))\, \llbracket e\rrbracket_e) \\
&\longrightarrow_{\bullet} (\lambda p.(\llbracket t\rrbracket_t\, (\lambda^{\bullet} v.p\, v))\, \llbracket e\rrbracket_e)\, \lambda^{\bullet} x.\llbracket p\rrbracket_p \\
&\longrightarrow_{\mathsf{a}} (\llbracket t\rrbracket_t\, (\lambda^{\bullet} v.(\lambda^{\bullet} x.\llbracket p\rrbracket_p)\, v))\, \llbracket e\rrbracket_e \\
&\overset{?}{\longrightarrow}_{\bullet} (\llbracket t\rrbracket_t\, (\lambda^{\bullet} x.\llbracket p\rrbracket))\, \llbracket e\rrbracket_e \\
&{}_{\mathsf{a}^+}\!\!\longleftarrow \lambda k.((\llbracket t\rrbracket_t\, (\lambda^{\bullet} x.\llbracket p\rrbracket))\, k)\, \llbracket e\rrbracket_e = \llbracket\langle \mu\hat{\mathfrak{tp}}.\langle t \| \tilde{\mu}x.\langle p \| \hat{\mathfrak{tp}}\rangle\rangle \| e\rangle\rrbracket_c
\end{aligned}$$

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

$$\begin{aligned}
\langle (t,p) \| e\rangle &= (\lambda^{\bullet} k.\llbracket p\rrbracket_p\, (\llbracket t\rrbracket_t\, (\lambda x.\lambda^{\bullet} a.k\, (x,a))))\, \llbracket e\rrbracket_e \\
&\longrightarrow_{\bullet} \llbracket p\rrbracket_p\, (\llbracket t\rrbracket_t\, (\lambda x.\lambda^{\bullet} a.\llbracket e\rrbracket_e\, (x,a))) \\
&{}_{\mathsf{a}^+}\!\!\longleftarrow \llbracket p\rrbracket_p\, (\llbracket t\rrbracket_t\, (\lambda x.(\lambda k.k)\lambda^{\bullet} a.\llbracket e\rrbracket_e\, (x,a))) \\
&{}_{\mathsf{a}^+}\!\!\longleftarrow \llbracket p\rrbracket_p\, (\llbracket t\rrbracket_t\, (\lambda x.(\lambda k.k)\lambda^{\bullet} a.(\lambda k.k\, (x,a))\, \llbracket e\rrbracket_e)) \\
&= \llbracket\langle p \| \tilde{\mu}\check{\mathfrak{tp}}.\langle t \| \tilde{\mu}x.\langle \check{\mathfrak{tp}} \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle\rangle\rrbracket_c
\end{aligned}$$

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

$$\begin{aligned}
\llbracket \mathsf{wit}\, p\rrbracket_t \llbracket e_t\rrbracket_t &= (\lambda k.\llbracket p\rrbracket_p\, (\lambda^{\bullet} a.k\, (\mathsf{wit}\, a)))\, \llbracket e_t\rrbracket_t \\
&\longrightarrow_{\mathsf{a}} \llbracket p\rrbracket_p\, (\lambda^{\bullet} a.\llbracket e_t\rrbracket_t\, (\mathsf{wit}\, a))) \\
&{}_{\mathsf{a}^+}\!\!\longleftarrow \llbracket p\rrbracket_p\, (\lambda^{\bullet} a.(\lambda k.k\, (\mathsf{wit}\, a))\, \llbracket e_t\rrbracket_t) = \llbracket\langle p \| \tilde{\mu}a.\langle \mathsf{wit}\, a \| e_t\rangle\rangle\rrbracket_c
\end{aligned}$$

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

$$\begin{aligned}
\llbracket \mathsf{wit}\, (V_t, V_p)\rrbracket_t \llbracket e_t\rrbracket_t &= (\lambda k.k\, (\mathsf{wit}\, (\llbracket V_t\rrbracket_{V_t}, \llbracket V_p\rrbracket_V)))\, \llbracket e_t\rrbracket_t \\
&\longrightarrow_{\mathsf{a}} \llbracket e_t\rrbracket_t\, (\mathsf{wit}\, (\llbracket V_t\rrbracket_{V_t}, \llbracket V_p\rrbracket_V)) \\
&\longrightarrow_{\bullet} \llbracket e_t\rrbracket_t\, \llbracket V_t\rrbracket_{V_t} \\
&{}_{\mathsf{a}}\!\!\longleftarrow (\lambda k.k\, \llbracket V_t\rrbracket_{V_t})\, \llbracket e_t\rrbracket_t = \llbracket V_t\rrbracket_t e_t
\end{aligned}$$

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

$$\begin{aligned}
\llbracket V_t\rrbracket_t \llbracket \tilde{\mu}x.c\rrbracket_t &= (\lambda k.k\, \llbracket V_t\rrbracket_{V_t})\, \lambda^{\bullet} x.\llbracket c\rrbracket_c \\
&\longrightarrow_{\mathsf{a}} (\lambda^{\bullet} x.\llbracket c\rrbracket_c)\, \llbracket V_t\rrbracket_{V_t} \\
&\longrightarrow_{\bullet} \llbracket c\rrbracket_c[\llbracket V_t\rrbracket_{V_t}/x] = \llbracket c[V_t/x]\rrbracket_c
\end{aligned}$$

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

$$\begin{aligned}
\llbracket V\rrbracket_p \llbracket \tilde{\mu}\hat{\mathfrak{tp}}.\langle \hat{\mathfrak{tp}} \| e\rangle\rrbracket_e &= (\lambda k.k\, \llbracket V\rrbracket_V)\, ((\lambda k.k)\llbracket e\rrbracket_e) \\
&\longrightarrow_{\mathsf{a}} ((\lambda k.k)\llbracket e\rrbracket_e)\, \llbracket V\rrbracket_V \\
&\longrightarrow_{\mathsf{a}} \llbracket e\rrbracket_e\, \llbracket V\rrbracket_V \\
&{}_{\mathsf{a}}\!\!\longleftarrow (\lambda k.k\, \llbracket V\rrbracket_V)\, \llbracket e\rrbracket_e = \llbracket\langle V \| e\rangle\rrbracket_c
\end{aligned}$$

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

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

□

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

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

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

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

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

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

Proof. Reasoning by contraposition, let us assume that c is not normalizing. Then in any infinite reduction sequence from c, according to the previous proposition, there are infinitely many 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. □

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

Proof. Using the preservation of typing that we shall prove in the next section (Proposition 4.10), we know that if c is typed in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, then its image $[\![c]\!]_c$ is also typed. Using the fact that typed terms of the target language are normalizing, we can finally apply the previous proposition to deduce that 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 $[\![A]\!]^*$. The type $[\![A]\!]^*$ 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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:

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

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 6.2, this will no longer be the case when extending the domain of terms.

To extend the translation for types to the translation of contexts, we consider that we can unify left and right contexts into a single one that is coherent with respect to the order in which the hypotheses have been introduced. We denote this context by $\Gamma \cup \Delta$, where the assumptions of Γ remain unchanged, while the former assumptions $(\alpha : A)$ in Δ are denoted by $(\alpha : A^{\perp\!\!\perp})$. The translation of unified contexts is given by:

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

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 replacing 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]. Also, 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[23] to the translation of a (really) classical proof.

Moreover, we can even make the return type of the continuation dependent on its argument (that is a type of the shape $\Pi_{a:A} R(a)$), so that the type of $[\![p]\!]_p$ will correspond to the elimination rule:

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

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

Lemma 4.9 (Typing translation for NEF proofs). *The following holds:*

(1) *For any term t, if $\Gamma \vdash t : \mathbb{N} \mid \Delta$ then $[\![\Gamma \cup \Delta]\!] \vdash [\![t]\!]_t : \forall X.(\forall x^{\mathbb{N}}.X(x) \to X(t^+))$.*
(2) *For any NEF proof p, if $\Gamma \vdash p : A \mid \Delta$ then $[\![\Gamma \cup \Delta]\!] \vdash [\![p]\!]_p : \forall X.(\Pi_{a:[\![A]\!]^+} X(a) \to X(p^+)))$.*
(3) *For any NEF command c, if $c : (\Gamma \vdash \Delta, \star : B)$ then $[\![\Gamma \cup \Delta]\!], \star : \Pi_{b:B^+} X(b) \vdash [\![c]\!]_c : X(c^+))$.*

Proof. The proof is done by induction on typing derivations. We only give the key cases of the proof.

- **Case** (wit). In $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ the typing rule for $\mathrm{wit}\, p$ is the following:

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

We want to show that:

$$[\![\Gamma \cup \Delta]\!] \vdash \lambda k.[\![p]\!]_p\, (\lambda a.k(\mathrm{wit}\, a)) : \forall X.(\forall x^{\mathbb{N}}.X(x) \to X(\mathrm{wit}\, p^+))$$

By induction hypothesis, we have:

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

[23] A classical proof might backtrack, thus it translation might use a former continuation. The return type of continuations thus need to be uniform (usually $\bot$) and can not be parametrized by $\forall R$.

hence, it amounts to showing that for any X we can build the following derivation:

$$\dfrac{\dfrac{}{\llbracket\Gamma\cup\Delta\rrbracket,k:\forall x^{\mathbb{N}}.X(x)\vdash k:\forall x^{\mathbb{N}}.X(x)}\,(\mathrm{Ax}_p)\qquad \dfrac{\dfrac{}{\llbracket\Gamma\cup\Delta\rrbracket,k:\forall x^{\mathbb{N}}.X(x),a:\exists x^{\mathbb{N}}.\llbracket A\rrbracket^+\vdash a:\exists x^{\mathbb{N}}.\llbracket A\rrbracket^+}\,(\mathrm{Ax}_p)}{\llbracket\Gamma\cup\Delta\rrbracket,k:\forall x^{\mathbb{N}}.X(x),a:\exists x^{\mathbb{N}}.\llbracket A\rrbracket^+\vdash \mathsf{wit}\,a:\mathbb{N}}\,(\mathrm{wit})}{\dfrac{\llbracket\Gamma\cup\Delta\rrbracket,k:\forall x^{\mathbb{N}}.X(x),a:\exists x^{\mathbb{N}}.\llbracket A\rrbracket^+\vdash k\,(\mathsf{wit}\,a):X(\mathsf{wit}\,a)}{\llbracket\Gamma\cup\Delta\rrbracket,k:\forall x^{\mathbb{N}}.X(x)\vdash \lambda a.k(\mathsf{wit}\,a):\Pi_{a:\exists x^{\mathbb{N}}.\llbracket A\rrbracket^+}X(\mathsf{wit}\,a)}\,(\to_I)}\,(\forall_E^1)$$

- **Case** $(\exists_I)$. In $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ the typing rule for (t,p) is the following:

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

Hence, we obtain by induction:

$$\begin{aligned}&\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket t\rrbracket_t:\forall X.(\forall x^{\mathbb{N}}.X(x)\to X(t^+)) && (IH_t)\\ &\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket p\rrbracket_p:\forall Y.(\Pi_{a:A(t^+)}Y(a)\to Y(p^+)) && (IH_p)\end{aligned}$$

and we want to show that for any Z:

$$\llbracket\Gamma\cup\Delta\rrbracket\vdash\lambda k.\llbracket p\rrbracket_p(\llbracket t\rrbracket_t\,(\lambda xa.k\,(x,a))):\Pi_{a:\exists x^{\mathbb{N}}.A}Z(a)\to Z(t^+,p^+).$$

So we need to prove that:

$$\llbracket\Gamma\cup\Delta\rrbracket,k:\Pi_{q:\exists x^{\mathbb{N}}.A}Z(q)\vdash\llbracket p\rrbracket_p(\llbracket t\rrbracket_t\,(\lambda xa.k\,(x,a))):Z(t^+,p^+)$$

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 $\llbracket t\rrbracket_p$, and using $Y(a)\triangleq Z(t^+,a)$ in the type of $\llbracket p\rrbracket_p$:

$$\dfrac{\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket p\rrbracket_p:\dots\qquad \dfrac{\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket t\rrbracket_t:\dots\qquad \dfrac{\dfrac{\dfrac{}{k:\Pi_{q:\exists x^{\mathbb{N}}.A}Z(q)\vdash k:\Pi_{q:\exists x^{\mathbb{N}}A}Z(q)}\,(\mathrm{Ax}_p)\quad\dfrac{}{x:\mathbb{N},a:A(x)\vdash(x,a):\exists x^{\mathbb{N}}.A}\,(\exists_I)}{k:\Pi_{q:\exists x^{\mathbb{N}}.A}Z(q),x:\mathbb{N},a:A(x)\vdash k\,(x,a):Z(x,a)}\,(\to_E)}{k:\Pi_{q:\exists x^{\mathbb{N}}.A}Z(q)\vdash\lambda xa.k\,(x,a):\forall x.\Pi_{a:A(x)}Z(x,a)}\,(\forall_I)}{\llbracket\Gamma\cup\Delta\rrbracket,k:\Pi_{a:\exists x^{\mathbb{N}}.A}Z(a)\vdash\llbracket t\rrbracket_t\,(\lambda xa.k\,(x,a)):\Pi_{a:A(t^+)}Z(t^+,a)}\,(\to_E)}{\llbracket\Gamma\cup\Delta\rrbracket,k:\Pi_{q:\exists x^{\mathbb{N}}.A}Z(q)\vdash\llbracket p\rrbracket_p(\llbracket t\rrbracket_t\,(\lambda xa.k\,(x,a))):Z(t^+,p^+)}\,(\to_E)$$

- **Case** (μ). 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$:

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

We want to show that for any X we can derive:

$$\llbracket\Gamma\cup\Delta\rrbracket\vdash\lambda k.\llbracket p\rrbracket_p\,(\lambda a.\llbracket q\rrbracket_p\,k):\Pi_{b:B}X(b)\to X(q^+[p^+/a]).$$

By induction, we have:

$$\begin{array}{c}\llbracket\Gamma\cup\Delta\rrbracket \vdash \llbracket p\rrbracket_p : \forall Y.(\Pi_{a:A^+}Y(a)\to Y(p^+))\\ \llbracket\Gamma\cup\Delta\rrbracket, a:A^+ \vdash \llbracket q\rrbracket_t : \forall Z.(\Pi_{b:B^+}Z(b)\to Z(q^+)),\end{array}$$

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

$$\dfrac{\llbracket\Gamma\cup\Delta\rrbracket \vdash \llbracket p\rrbracket_p : \ldots \qquad \dfrac{\dfrac{\llbracket\Gamma\cup\Delta\rrbracket, a:A^+ \vdash \llbracket q\rrbracket_p : \ldots \qquad \overline{k:\Pi_{b:B}X(b)\vdash k : k:\Pi_{b:B}X(b)}}{\llbracket\Gamma\cup\Delta\rrbracket, k:\Pi_{b:B}X(b), a:A^+ \vdash \llbracket q\rrbracket_p\, k : X(q^+)}\,(\to_E)}{\llbracket\Gamma\cup\Delta\rrbracket, k:\Pi_{b:B}X(b) \vdash \lambda a.\llbracket q\rrbracket_p\, k : \Pi_{a:A^+}X(q^+)}\,(\to_I)}{\llbracket\Gamma\cup\Delta\rrbracket, k:\Pi_{b:B}X(b) \vdash \llbracket p\rrbracket_p\,(\lambda a.\llbracket q\rrbracket_p\, k) : X(q^+[p^+/a])}\,(\to_E)$$

□

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* $\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket p\rrbracket_p : \llbracket A\rrbracket^*$,
(2) *if* $\Gamma\mid e:A\vdash\Delta$ *then* $\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket e\rrbracket_e : \llbracket A\rrbracket^+\to\bot$,
(3) *if* $c:\Gamma\vdash\Delta$ *then* $\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket c\rrbracket_c : \bot$.

Proof. The proof is done by induction on the typing derivation, distinguishing cases according to the typing rule used in the conclusion. 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 delimited continuations that we detail hereafter. We consider a command $\langle\mu\hat{\text{tp}}.\langle q\|\tilde\mu a.\langle p\|\hat{\text{tp}}\rangle\rangle\|e\rangle$ 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 $(\llbracket q\rrbracket_p\,(\lambda a.\llbracket p\rrbracket_p))\,\llbracket e\rrbracket_e$. The corresponding typing derivation in $\text{dL}_{\hat{\text{tp}}}$ is of the form:

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

By induction hypothesis for e and p we obtain:

$$\begin{array}{c}\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket e\rrbracket_e : \llbracket B[q^+]\rrbracket^+\to\bot\\ \llbracket\Gamma\cup\Delta\rrbracket, a:A^+\vdash\llbracket p\rrbracket_p : \llbracket B[a]\rrbracket^*\\ \llbracket\Gamma\cup\Delta\rrbracket\vdash\lambda a.\llbracket p\rrbracket_p : \Pi_{a:A^+}\llbracket B[a]\rrbracket^*,\end{array}$$

Applying Lemma 4.9 for $q\in$ NEF we can derive:

$$\dfrac{\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket q\rrbracket_p : \forall X.(\Pi_{a:A^+}X(a)\to X(q^+))}{\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket q\rrbracket_p : (\Pi_{a:A^+}\llbracket B[a]\rrbracket^*\to\llbracket B[q^+]\rrbracket^*}\,(\forall^2_E)$$

We can thus derive that:

$$\llbracket\Gamma\cup\Delta\rrbracket\vdash\llbracket q\rrbracket_p\,(\lambda a.\llbracket p\rrbracket_p) : \llbracket B[q^+]\rrbracket^*,$$

and finally conclude that:

$$\llbracket\Gamma\cup\Delta\rrbracket\vdash(\llbracket q\rrbracket_p\,(\lambda a.\llbracket p\rrbracket_p))\,\llbracket e\rrbracket_e : \bot\,.$$

□

We can finally deduce the correctness of $\text{dL}_{\hat{\text{tp}}}$ through the translation:

Theorem 4.11 (Soundness). *For any* $p\in\text{dL}_{\hat{\text{tp}}}$, *we have:* $\nvdash p:\bot$.

Proof. Any 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 a proof cannot exist. □

5 EMBEDDING INTO LEPIGRE'S CALCULUS

In a recent paper [22], 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 [21]. In his system, the relativization $a \in A$ is restricted to values, so that we can only type $V : V \in A$:

$$\frac{\Gamma \vdash_{val} V : A}{\Gamma \vdash_{val} V : V \in A}\ \exists_i$$

However, typing judgments are defined up to observational equivalence, so that if t is observationally equivalent to V, one can derive the judgment $t : t \in A$.

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 be untyped otherwise[24].

We shall now detail an embedding of $\mathrm{dL}_{\hat{tp}}$ into Lepigre's calculus, and explain how to transfer normalization and correctness properties along this translation. Additionally, this has the benefits of providing us with a realizability interpretation for our calculus. While we do not use it in the current paper, we take advantage of this interpretation (and in particular of the interpretation of dependent types) in [28] to prove the normalization of dLPA^{ω}, the sequent calculus which originally motivated this work and whose construction relies on $\mathrm{dL}_{\hat{tp}}$.

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 and the reduction rules for the fragment of Lepigre's calculus we use, for the type system we refer the reader to [22]:

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

The reduction $\succ$ is defined as the smallest relation satisfying:

$$\begin{array}{rcl@{\qquad}rcl}
t\,u * \pi & \succ & u * [t]\pi & \mu\alpha.t * \pi & \succ & t[\alpha := \pi] * \pi \\
v * [t]\pi & \succ & t * v \cdot \pi & p * \pi & \succ & p \\
\lambda x.t * v \cdot \pi & \succ & t[x := v] * \pi & (v_1, v_2).l_i & \succ & v_i
\end{array}$$

It is worth noting that the call-by-value strategy is obtained via the construction $[t]\pi$ which allows to evaluate the argument of t to a value before pushing it onto the stack.

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

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

Fig. 11. Translation of proof terms into Lepigre's calculus

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

$$
\begin{array}{lcl}
(t_1,t_2) & \triangleq & (\lambda v_1v_2.\{l_1=v_1,l_2=v_2\})\,t_1\,t_2 \\
\mathtt{fst}(t) & \triangleq & (\lambda x.(x.l_1))\,t \\
\mathtt{snd}(t) & \triangleq & (\lambda x.(x.l_2))\,t \\
A_1\wedge A_2 & \triangleq & \{l_1:A_1,l_2:A_2\}
\end{array}
$$

Similarly, only values can be pushed on stacks, but we can define processes[25] with stacks of the shape $t\cdot\pi$ as syntactic sugar:

$$t*u\cdot\pi \quad\triangleq\quad tu*\pi$$

We first define the translation for types (extended for typing contexts) where the predicate $\mathsf{Nat}(x)$ is defined[26] as usual in second-order logic:

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

and $[\![t]\!]_t$ is the translation of the term t given in Figure 11.

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

Note that the equality is mapped to Leibniz equality, and that the definitions of $\bot^*$ and $\top^*$ respectively correspond to $(0=1)^*$ and $(0=0)^*$ in order to make the conversion rule admissible through the translation.

The translation for terms, proofs, contexts and commands of $\mathrm{dL}_{\hat{\mathtt{tp}}}$, given in Figure 11 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 [22], 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 typing, unless we extend the type system in which case this would imply the adequacy. We might

[25]This will allow us to ease the definition of the translation to handle separately proofs and contexts. Otherwise, we would need formally to define $[\![\langle p\|q\cdot e\rangle]\!]_c$ all together by $[\![p]\!]_p[\![q]\!]_p * [\![e]\!]_e$.

[26]Where 0 is defined as $\lambda zs.z$ and $S(t)$ as $(\lambda zs.s(tzs))$, *i.e.* as the translation of the corresponding 0 and successor from $\mathrm{dL}_{\hat{\mathtt{tp}}}$.

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

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

Fig. 12. Extension of Lepigre's typing rules for stacks

also directly prove the adequacy of the realizability model (through the translation) with respect to the typing rules of $\mathrm{dL}_{\hat{\mathrm{tp}}}$. We will detail here a proof of adequacy using the former method. We then need to extend Lepigre's system to be able to type stacks. In fact, his proof of adequacy [22, 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 used 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 $\bot^{\perp\!\!\perp}$. Finally, we change the rule $(*)$ of the original type system in [22] and add rules for stacks, whose definitions are guided by the proof of the adequacy [22, Theorem 6] in particular by the $(\Rightarrow_e)$-case. These rules are given in Figure 12.

We shall now show that these rules are adequate with respect to the realizability model defined in [22, 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 $\pi\sigma \in [\![A]\!]_\sigma^{\perp}$;*
- *if $\Gamma \vdash t : A$ then $t\sigma \in [\![A]\!]_\sigma^{\perp\perp}$.*

Proof. The proof is done by induction on typing derivations, we only need to do the proof for the rules we defined above (all the other cases correspond to the proof of [22, Theorem 6]).

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

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

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

(μ). The proof is the very same as in [22, Theorem 6].

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

($\Rightarrow_l$). If t is a value v, by induction hypothesis, we have that $v\sigma \in [\![A]\!]_\sigma$ and $\pi\sigma \in [\![B]\!]_\sigma^{\perp}$, and we need to show that $v\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^{\perp}$. The proof is already done in the case $(\Rightarrow_e)$ (see [22, Theorem 6]). Otherwise, by induction hypothesis, we have that $t\sigma \in [\![A]\!]_\sigma^{\perp\perp}$ and $\pi\sigma \in [\![B]\!]_\sigma^{\perp}$, and we need to show that $t\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^{\perp}$. 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 $[\lambda x.u]\pi \in [\![A]\!]_\sigma^\perp$, which is already proven in the case $(\Rightarrow_e)$.

(`let`*).* We need to show that for all $v \in [\![A]\!]_\sigma$, $v * [t\sigma]\pi\sigma \in \perp\!\!\!\perp$. 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 $t\sigma \in [\![A \Rightarrow B]\!]_\sigma^{\perp\perp}$ and $\pi\sigma \in [\![B]\!]_\sigma^\perp$, thus $v \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^\perp$.

□

It only remains to show that the translation we defined in Figure 11 preserves typing to conclude the proof of Proposition 5.3.

Lemma 5.2. *If $\Gamma \vdash p : A \mid \Delta$ (in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$), 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 $(\Gamma \cup \Delta)^* \vdash [\![c]\!]_c : \perp$.*

Proof. The proof is an easy induction on the typing derivation $\Gamma \vdash p : A \mid \Delta$. Note that in a way, the translation of a delimited continuation decompiles it to simulate in a natural deduction fashion the reduction of the applications of functions to stacks (that could have generated the same delimited continuations in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$), 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 $[\![q_1]\!]_p$ (resp. $[\![q_2]\!]_p$) reduces to a value V_1 (resp. V_2) we have:

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

where we observe that $[\![e]\!]_e$ is always kept outside of the computations, and where each command $\langle q_i \| \tilde{\mu}a_i.c_{\hat{\mathfrak{tp}}}\rangle$ is decompiled into $(\mu\alpha.[\![q_i]\!]_p * [\lambda a_i.[\![c_{\hat{\mathfrak{tp}}}]\!]_{\hat{\mathfrak{tp}}}].\alpha) * [\![e]\!]_e$, simulating the (natural deduction style) reduction of $\lambda a_i.[\![c_{\hat{\mathfrak{tp}}}]\!]_{\hat{\mathfrak{tp}}} * [\![q_i]\!]_p \cdot [\![e]\!]_e$. 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$:

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

Proof. By contradiction, if we had a closed proof p of type $\perp$, it would be translated as a realizer of $\top \to \perp$. Therefore, $[\![p]\!]_p\ \lambda x.x$ would be a realizer of $\perp$, which is impossible. □

Furthermore, the translation clearly preserves normalization (in the sense that for any c, if c does not normalize then neither does $[\![c]\!]_c$), and thus the normalization of $\mathrm{dL}_{\hat{\mathrm{tp}}}$ is a consequence of adequacy. It is worth noting that without delimited continuations, we would not have been able to define an adequate translation, since we would have encountered the same problem[27] than with a naive CPS translation (see Section 2.8).

6 FURTHER EXTENSIONS

As we explained in the preamble of Section 2, we defined dL and $\mathrm{dL}_{\hat{\mathrm{tp}}}$ as small languages containing all the potential sources of inconsistency we wanted to mix: classical control, 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 Intuitionistic sequent calculus

There is not much to say on this topic, but it is worth mentioning that dL and $\mathrm{dL}_{\hat{\mathrm{tp}}}$ could be easily restricted to obtain an intuitionistic framework. Indeed, just like for the passage from LK to LJ, it is enough to restrict the syntax of proofs to allow only one continuation variable (that is one conclusion on the right-hand side of sequent) to obtain an intuitionistic calculus. In particular, in such a setting, all proofs will be NEF, and every result we obtained will still hold.

6.2 Extending the domain of terms

Throughout the paper, 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 $\mathrm{dL}_{\hat{\mathrm{tp}}}$, 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 saw that a *term* t of type $T = \mathbb{N}$ is translated into a *proof* t^* which is roughly of type $T^* = \neg\neg T^+ = \neg\neg\mathbb{N}$, from which we can extract a *term* t^+ of type $\mathbb{N}$.

However, if T was for instance the function type $\mathbb{N} \to \mathbb{N}$ (resp. $T \to U$), we would only be able to extract a *proof* of type $T^+ = \mathbb{N} \to \neg\neg\mathbb{N}$ (resp. $T^+ \to U^*$). There is no hope in general to extract a function $f : \mathbb{N} \to \mathbb{N}$ from such a term, since such a proof could 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. Such a proof is no longer a witness in the usual sense, but rather a realizer of $f \in \mathbb{N} \to \mathbb{N}$ in the sense of Krivine classical realizability. 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].

Nevertheless, if the extraction is not possible in the general case, our situation is more specific. Indeed, we only need to consider proofs that are obtained as translation of terms, which can only contains NEF proofs in $\mathrm{dL}_{\hat{\mathrm{tp}}}$. In particular, such proofs cannot drop continuations (remember that this was the whole point of the restriction to the NEF fragment). Therefore, we could again refine the translation of types, similarly to what we did in Lemma 4.9. Once more, this refinement would also coincide with a computational property similar to Lemma 4.1, expressing the fact that the extraction can be done simply by passing the identity as a continuation[28]. This witnesses the fact

[27]That is, the translation $[\![q]\!]_p * [\lambda a.[\![p]\!]_p * [\![e]\!]_e]\bullet$ of a command $\langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle$ (where e is of type $B[q]$ and p of type $B[a]$) would have been ill-typed (because $[\![p]\!]_p * [\![e]\!]_e$ is).

[28]To be precise, for each arrow in the type, a double-negation (or its refinement) would be inserted. For instance, to recover a function of type $\mathbb{N} \to \mathbb{N}$ from a term $t : \neg\neg(\mathbb{N} \to \neg\neg\mathbb{N})$ (where $\neg\neg A$ is in fact more precise, at least $\forall R.(A \to R) \to R$),

that for any function t in the source language, there exists a term t^+ in the target language which represents the same function, even though the translation of t is a proof $[\![t]\!]$.

To sum up, this means that we can extend the domain of terms in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ (in particular, it should affect neither the subject reduction property nor the soundness), but the stratification between terms and proofs is to be lost through a CPS translation. If the target language is a non-stratified type theory (most of the presentations of type theory correspond to this case), then it becomes possible to force the extraction of terms through the translation.

Another solution would consist in the definition of a separate translation for terms. Indeed, as it was reflected by Lemma 4.1, since neither terms nor NEF proofs may contain continuations, they can be directly translated. The corresponding translation is actually an embedding which maps every pure term (without $\mathrm{wit}\,p$) to itself, and which performs the reduction of NEF proofs p to proofs p^+ so as to eliminate every μ binder. Such a translation would intuitively reflect an abstract machine where the reduction of terms (and the NEF proofs inside) is performed in an external machine. If this solution is arguably a bit *ad hoc*, it is nonetheless correct and it is maybe a good way to take advantage of the stratified presentation.

6.3 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 subsumes 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. In fact, this is the 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 them to a value,
- in the dependent typing mode, make some pattern-matching within the list of dependencies 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):

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

and the reduction rules:

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

the continuation needs to be forced at each level: $\lambda x.t\,I\,x\,I : \mathbb{N} \to \mathbb{N}$. We do not want to enter into to much details on this here, as it would lead us to much more than a paragraph to define the objects formally, but we claim that we could reproduce the results obtained for terms of type $\mathbb{N}$ in a language with terms representing arithmetic functions in finite types.

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

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

which allow us to prove that the calculus remains correct with these extensions.

We claim that this methodology furnishes a good approach to handle the question "*Can I extend the language with ... ?*". In particular, it should be enough to get closer to a realistic programming language and extend the language with inductive fixed point operators[29].

6.4 A fully sequent-style dependent calculus

While the aim of this paper was to design a sequent-style calculus embedding dependent types, we only presented the Π-type in sequent-style. Indeed, we wanted to be sure above all that it was possible to define a sound sequent-calculus with the key ingredients of dependent types (*i.e.* dependent pairs and dependently-typed functions). In particular, rather than having left-rules (as in sequent calculi) for every syntactic constructors, we presented the existential type and the equality type with the following elimination rules (as in natural deduction):

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

However, it is now easy to replace both elimination rules (and thus the corresponding destructors) by equivalent left-rules (and thus syntactic constructors for contexts). 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:

$$\frac{c : \Gamma, x : \mathbb{N}, a : A(x) \vdash_d \Delta;\sigma\{(x,a)|p\}}{\Gamma \mid \tilde{\mu}(x,a).c : \exists x^{\mathbb{N}}.A(x) \vdash_d \Delta;\sigma\{\cdot|p\}}\ \exists_l \qquad\qquad \frac{\Gamma \vdash p : A \mid \Delta \quad \Gamma \mid e : A[u/t] \vdash \Delta}{\Gamma \mid \tilde{\mu}^{=}.\langle p \| e\rangle : t = u \vdash \Delta;\delta}\ (=_l)$$

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

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

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

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

Similarly, we get that the former (subst)-rule is admissible:

[29]The interested reader could see for instance [28] where a similar language with pairs, patter-matching, inductive and coinductive fixed points is defined.

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

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

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

and check that they advantageously[30] simulate the previous rules:

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

7 CONCLUSION

Several directions remain to be explored. We plan to investigate possible extensions of the syntactic restriction we defined, and its connections with notions such as Fürhmann's *thunkability* [13] or Munch-Maccagnoni's *linearity* [30]. Moreover, 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 and 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 novel contribution, even without considering the classical part. In particular, our translation allows us 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 could be adapted to justify extensions of other dependently typed calculi, or provide typed translations between them.

Last but not least, we extended $\mathrm{dL}_{\hat{\mathrm{tp}}}$ to solve the problem that was our original motivation to design such a calculus. In [28], we present dLPA^{ω}, a sequent calculus equivalent to Herbelin's dPA^{ω} [18] whose presentation is inspired from $\mathrm{dL}_{\hat{\mathrm{tp}}}$. This leads to the definition of a realizability model inspired from Lepigre's construction and from another technique developed with Herbelin [27] to give a realizability interpretation to calculi with laziness and memory sharing (two features of dPA^{ω}). As a consequence, we deduce the normalization and the soundness of the resulting system.

Acknowledgments. The author wishes to thank 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 help all along the writing of this paper, Rodolphe Lepigre for the example of the infinite tape lemma, Théo Winterhalter for helping me with typos, as well as Alexandre Miquel and anonymous referees 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.

[30] The expansion rules (*i.e.* for $\mathsf{prf}\ p$ or $\mathsf{subst}\ p\ q$ with p not a value) become useless.

[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] 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.

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

[22] 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.

[23] Rodolphe Lepigre. *Semantics and Implementation of an Extension of ML for Proving Programs.* PhD thesis, Université Savoie Mont Blanc, 2017.

[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] Étienne Miquey and Hugo Herbelin. Realizability interpretation and normalization of typed call-by-need λ-calculus with control. In *Foundations of Software Science and Computation Structures: 21th International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, Proceedings*, 2018.

[28] Étienne Miquey. A sequent calculus with dependent types for classical arithmetic. In *Proceedings of the 33nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2018.

[29] Guillaume Munch-Maccagnoni. Focalisation and Classical Realisability. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic '09*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, Heidelberg, 2009.

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

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

[32] C. Paulin-Mohring. Extracting F_ω's programs from proofs in the calculus of constructions. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 89–104, New York, NY, USA, 1989. ACM.

[33] Pierre-Marie Pédrot and Nicolas Tabareau. An Effectful Way to Eliminate Addiction to Dependence. In *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*, page 12, Reykjavik, Iceland, June 2017.

[34] 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.

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

[36] Matthijs Vákár. *In Search of Effectful Dependent Types*. PhD thesis, University of Oxford, 2017.

[37] Steffen van Bakel, Luigi Liquori, Simona Ronchi della Rocca, and Pawel Urzyczyn. Comparing cubes of typed and type assignment systems. *Annals of Pure and Applied Logic*, 86(3):267 – 303, 1997.

[38] 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.

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