7- A classical sequent calculus with dependent types

Side-effects and dependent types

In Chapter 5, we introduced dependent types from the point of view of logic, in the realm of Martin-Löf type theory, but actually, as a programming features, restricted form of dependent types were anterior to this. For instance, in the 60s the programming language FORTRAN IV already allowed programmers to define arrays of a given dimension, and in this sense, (restricted form of) dependent types are as old as high-level programming languages.

From the point of view of programming, dependent types allow us to assign more precise types—and thus more precise specifications—to existing programs. Dependent types are provided by Coq or Agda, two of the most actively developed proof assistants, which both rely on a constructive type theory: Coquand and Paulin-Mohring's calculus of inductive constructions for Coq [29], and Martin-Löf's type theory [114] for Agda. Yet, both systems lack of classical logic and more generally of side-effects, which make them impractical as programming languages.

In practice, effectful languages give to 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 all practical programming languages (*e.g.* OCaml, C++, Python, Java,...).

As we saw in Section 5.1.2.2,dependent types misbehave in the presence of control operators, and lead to logical inconsistencies. 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 real programming languages and logic, by accomodating 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 [156] or by Pédrot and Tabareau who proposed a systematical way to add effects to type theory [141]. Side-effects—that are impure computations in functional programming—are interpreted by means of monads. Interestingly, control operators can be interpreted in a similar way through the continuation monad, but the continuation monads generally lacks the properties necessary to fit the picture.

Although dependent types and classical logic have been deeply studied separately, the problem of accomodating both features in one and the same system has not found a completely satisfying answer yet. Recent works from Herbelin [70] and Lepigre [108] proposed some restrictions on dependent types to make them compatible with a classical proof system, while Blot [17] designed a hybrid realizability model where dependent types are restricted to an intuitionistic fragment.

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 chapter, we present a language following the call-by-value reduction strategy. While this design choice is strongly related with our long term perspective of giving a sequent calculus presentation of dPA$^\omega$(following the call-by-value strategy but for the lazy parts), this also constitutes 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 [162] and polymorphism [67]. In both cases, a simple and elegant solution (but unnecessarily restrictive in practice [55, 108]) to solve the inconsistencies consists to introduce 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 [108], whose consistency is preserved by means of a semantical value restriction defined for terms that behave as values up to observational equivalence. In the present work, we will rather use a syntactic restriction to a fragment of proofs that allows slightly more than values. As will see, the restriction that arises naturally coincides with the negative-elimination-free fragment of Herbelin's dPAω system [70].

A sequent calculus presentation

The main achievement of this chapter is to give a sequent calculus presentation 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 [32] 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 [39].

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

Delimited continuations and CPS translation

The main challenge in designing a sequent calculus with dependent types resides 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 7.1.4, the problem can be understood as a desynchronization of the type system with respect to the reduction. A simple solution to resolve this, presented in Section 7.1, consists to add an explicit list of dependencies in the typing derivations. This has the advantage of giving a calculus that is very close to the original. 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 [5], 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 7.2). Additionally, it allows for the design, in Section 8.3, of a continuation-passing style translation that preserves dependent types and allows for proving the soundness of our system. Finally, it also provides us with a way to embed our calculus into Lepigre's calculus [108], as we shall see in Section 7.4, and in particular it furnishes us a realizability interpretation.

7.1 A minimal classical language with dependent types

The easiest and usual approach to prevent inconsistencies to arise from the simultaneous presence of classical logic is to impose a restriction to values for proofs appearing inside dependent types and operators. In particular, this would prevent us from writing $\texttt{wit}\ p_0$ and $\texttt{prf}\ p_0$ in Herbelin's example.

In this section we will focus on value restriction in the framework of the $\lambda\mu\tilde{\mu}$-calculus, and show how it allows us to keep the proof system is consistent. We shall then see, in Section 7.2, how to relax this constraint.

7.1.1 A minimal language with value restriction

We follow here the stratified presentation[1] of dependent types 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[2] 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 $\texttt{wit}$ (for terms) and $\texttt{prf}$ (for proofs),
- a proof term $\texttt{refl}$ for the equality of terms and a proof term $\texttt{subst}$ for the convertibility of types over equal terms.

For simplicity reasons, we will only consider terms of type $\mathbb{N}$ throughout this chapter. We address the question of extending the domain of terms in Section 7.5.2. The syntax of the corresponding system, that we call dL, is given by:

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

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

[1] This design choice is usually a matter of taste and convenient for us in the perspective of adapting dPA$^{\omega}$. However, it also has the advantage of clearly enlightening the different treatments for term and proofs through the CPS in the next sections.

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

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

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

Figure 7.1: Reduction rules of dL

7.1.2 Reduction rules

As explained in Section 5.1.2.2, a backtracking proof might give place to different witnesses and proofs according to the context of reduction, leading to inconsistencies [69]. 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 7.1 (where $t \to t'$ denotes the reduction of terms and $c \rightsquigarrow c'$ the reduction of commands), follow the call-by-value evaluation principle. In particular one can see that whenever the command is of the shape $\langle C[p]\|e\rangle$ where $C[p]$ is a proof built on top of p which is not a value, it reduces to $\langle p\|\tilde{\mu}a.\langle C[a]\|e\rangle\rangle$, opening the construction to evaluate p[3].

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

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

7.1.3 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 $\lambda a.p : \Pi a : A.B$ and 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:

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

[3]The reader might recognize the rule (ς) of Wadler's sequent calculus [161].

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[4] 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[5] is performed and by then the problem would have been solved. Roughly speaking, 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 *dependencies list* in the typing rules, which links $\tilde{\mu}$ variables (here a) to the associate proof term on the left-hand side of the command (here q). We can now obtain the following typing derivation:

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

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

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

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

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

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

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

The 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:

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

Remark 7.1. The reader familiar with explicit substitutions [52] 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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 7.2: Typing rules of dL

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 occurences of p have been replaced by q. ⌟

Furthermore, when typing a stack with the $(\to_l)$ rule, we need to drop the open binding in the list of dependencies[7]. We introduce the notation $\Gamma \mid e : A \vdash \Delta; \sigma\{\cdot|\dagger\}$ to denote that the dependency to be produced is irrelevant and can be dropped. This trick spares us from defining a second type of sequent $\Gamma \mid e : A \vdash \Delta; \sigma$ to type contexts when dropping the (open) binding $\{\cdot|p\}$. Alternatively, one can think of $\dagger$ as any proof term not in $\mathcal{D}$, which is the same with respect to the list of dependencies. The resulting set of typing rules is given in Figure 7.2, 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 [32]. 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

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

translation, we consider that we can unify both contexts into a single one that is coherent with respect to the order in which the hypothesis have been introduced[8]. 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 $(\alpha : A^{\perp\!\!\!\perp})$.

Example 7.2. The proof $p_1 \triangleq \mathtt{subst}\,(\mathtt{prf}\ p_0)\ \mathtt{refl}$ which was of type $1 = 0$ in Section 5.1.2.2 is now incorrect since the backtracking proof p_0, defined by $\mu\alpha.(0, \mu\_.\langle(1, \mathtt{refl})\|\alpha\rangle)$ in our framework, is not a value in $\mathcal{D}$. The proof p_1 should rather be defined by[9] $\mu\alpha.\langle p_0\|\tilde{\mu}a.\langle \mathtt{subst}\,(\mathtt{prf}\ a)\ \mathtt{refl}\|\alpha\rangle\rangle$ which can only be given the type $1 = 1$. ⌟

7.1.4 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':

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

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

Lemma 7.3 (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 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 7.4. *If $\sigma \Rrightarrow \sigma'$, then for any p, e, Γ, Δ:*

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

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

Lemma 7.5 (Safe term substitution)**.** *If $\Gamma \vdash t : \mathbb{N} \mid \Delta; \varepsilon$ then:*

1. $c : (\Gamma, x : \mathbb{N}, \Gamma' \vdash \Delta; \sigma) \quad \Rightarrow\ c[t/x] : (\Gamma, \Gamma'[t/x] \vdash \Delta[t/x]; \sigma[t/x])$,
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 7.6 (Safe value substitution)**.** *If $\Gamma \vdash V : A \mid \Delta; \varepsilon$ then:*

1. $c : (\Gamma, a : A, \Gamma' \vdash \Delta; \sigma) \quad \Rightarrow\ c[V/a] : (\Gamma, \Gamma'[V/a] \vdash \Delta[V/a]; \sigma[V/a])$,
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. $\Gamma, a : A, \Gamma' \mid e : B \vdash \Delta; \sigma \ \Rightarrow\ \Gamma, \Gamma'[V/a] \mid e[V/a] : B[V/a] \vdash \Delta[V/a]; \sigma[V/a]$,

[8] See Section 4.2.3.2 for further details on this point.
[9] That is to say $\mathtt{let}\, a = p_0\, \mathtt{in}\, \mathtt{subst}\ a\ \mathtt{refl}$ in natural deduction.

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

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

Proof. The proofs are done by induction on the typing derivation. □

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

Theorem 7.8 (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 the consecutive $(\equiv_l)$ rules as:

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

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:

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

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 that $\Gamma, x : \mathbb{N} \vdash p : A \mid \Delta$ and $\Gamma \vdash t : \mathbb{N} \mid \Delta$, by Lemma 8.4 and the fact that $\Delta[t/x] = \Delta$ we get a proof Π'_p of $\Gamma \vdash p[t/x] : A[t/x] \mid \Delta$. We can thus build the following derivation:

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

using Corollary 7.4 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:

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

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:

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

using Corollary 7.4 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:

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

We get a proof that $c[e/\alpha] : \Gamma \vdash \Delta; \varepsilon$ is valid by Lemma 7.7.

- **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:

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

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 get a proof for $c[V/a] : \Gamma \vdash \Delta; \{V|V\}$ by Lemma 7.6. We finally get a proof for $c[V/a] : \Gamma \vdash \Delta; \varepsilon$ by Lemma 7.3.

- **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:

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

We can build the following derivation:

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

where $\Pi_{(t,a)}$ is as expected, observing that since $p \notin \mathcal{D}$, the binding $\{\cdot|(t,p)\}$ is the same as $\{\cdot|\dagger\}$, and we can apply Corollary 7.4 to weaken dependencies in Π_e.

- **Case** $\langle \mathtt{prf}\ (t,V) \| e\rangle \rightsquigarrow \langle V \| e\rangle$.

This case is easy, observing that a derivation of the command on the left-hand side is of the form:

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

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

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

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

This case is straightforward, observing that for any terms t, u, if we have $\mathtt{refl} : t = u$, then $A[t] \equiv A[u]$ for any A.

- **Case** $\langle \mathtt{subst}\ p\ q \| e\rangle \rightsquigarrow \langle p \| \tilde{\mu}a.\langle \mathtt{subst}\ a\ q \| e\rangle\rangle$.

This case is exactly the same as 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.

□

7.1.5 Soundness

We give here 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 7.3. A more interesting proof through a continuation-passing translation is presented in Section 8.3.

We first show that typed commands of dL normalize by translating them into the simply-typed $\lambda\mu\tilde{\mu}$-calculus with pairs, that is to say the $\lambda\mu\tilde{\mu}$-calculus extended[10] 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 7.3.

The translation essentially consists of erasing the dependencies in types[11], turning the dependent products into arrows and the dependent sum into a pair. The erasure procedure is defined by:

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

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

[10] This corresponds to the addition of pairs and projections in the λ-calculus to obtain the $\lambda^\times$-calculus in Section 2.4.1.

[11] The use of erasure functions is a very standard technique in the systems of the λ-cube, see for instance [132] or [157].

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

(a) Syntax

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

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

(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

Figure 7.3: $\lambda\mu\tilde{\mu}$-calculus with pairs

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

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

where $\pi_i(p) \triangleq \mu\alpha.\langle p \| \tilde{\mu}(a_1, a_2).\langle a_1 \| \alpha \rangle \rangle$. The term $\bar{n}$ is defined as any encoding of the natural number n with its type $\mathbb{N}^*$, the encoding being irrelevant here as long as $\bar{n} \in V$. Note that we translate differently $\texttt{subst}\ V\ q$ and $\texttt{subst}\ p\ q$ to simplify the proof of Proposition 7.11.

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

Lemma 7.9. *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 7.10. *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, $\texttt{refl}^* = \lambda x.x : \mathbb{N}^* \to \mathbb{N}^*$.

The only non-direct cases are `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:

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

then $\texttt{subst}\ p\,q^* = \mu\alpha.\langle p^* \| \tilde{\mu}\_.\langle \mu\alpha.\langle q^* \| \alpha\rangle \| \alpha\rangle\rangle : B[u/x]^*$. According to Lemma 7.9, we have $B[u/x]^* = B[t/x]^* = B^*$. By induction hypothesis, we have proofs of $\Gamma^* \vdash p^* : \mathbb{N}^* \to \mathbb{N}^* \mid \Delta^*$ and $\Gamma^* \vdash q^* : B \mid \Delta^*$. Using the notation $\eta_{q^*} \triangleq \mu\alpha.\langle q^* \| \alpha\rangle$, we can derive:

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

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

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

□

We can then deduce the normalization of dL from the normalization of the $\lambda\mu\tilde{\mu}$-calculus [140], by showing that the translation preserves the normalization in the sense that if c does not normalize, then neither does c^*.

Proposition 7.11. *If c is a command such that c^* normalizes, then c normalizes.*

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

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

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

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

- **Case** $\texttt{wit}\,(t,V) \to t$:.

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

- **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** $\langle \mathtt{prf}\ (t,V) \| e\rangle \rightsquigarrow \langle V \| e\rangle$:.

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

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

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

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

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

□

Theorem 7.12. *If* $c : (\Gamma \vdash \Delta; \varepsilon)$, *then* c *normalizes.*

Proof. Proof by contradiction: if c does not normalize, then by Proposition 7.11 neither does c^*. However, by Proposition 7.10 we have that $c^* : \Gamma^* \vdash \Delta^*$. This is absurd since any well-typed command of the $\lambda\mu\tilde{\mu}$-calculus normalizes [140]. □

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

Theorem 7.13 (Soundness). *For any $p \in$ dL, we have $\not\vdash p : \bot$.*

Proof. We actually start by proving by contradiction that a command $c \in$ dL cannot be well-typed with empty contexts. Indeed, let us assume that there is such a command $c : (\vdash)$. By normalization, we can 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 command $c : (\vdash)$. We can still reduce $\langle p \| \star \rangle$ to a command c in normal form, and see that c it has to be of the shape $\langle V \| \star \rangle$ (by the same kind of reasoning, using 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. □

7.1.6 Toward a continuation-passing style translation

The difficulty 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 classical control. We will discuss later in Section 7.4 a solution to handle the dependencies. We will focus here on the possibility of removing the classical part from dL, that is to define a translation that gets rid of the 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 [32]. 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 7.1.3, the case of a command $\langle q \| \tilde{\mu}a.\langle p \| e \rangle \rangle$ with $p : B[a]$ and $e : B[q]$. Its translation is very likely to look like:

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

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

We follow the idea that the correctness is guaranteed by the head-reduction strategy, preventing $\langle p \| e \rangle$ from reducing before the substitution of a was made. We would like to ensure the same thing happens in the target language (that will also be equipped with a head-reduction strategy), namely that $[\![p]\!]$ cannot be applied to $[\![e]\!]$ before $[\![q]\!]$ has furnished a value to substitute for a. This would correspond informally to the term[12]:

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

[12] We will see in Section 7.3.4 that such a term could be typed by turning the type $A \to \bot$ of the continuation that $[\![q]\!]$ is

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

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

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

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

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

Indeed, through such a translation, $\mu\alpha$ would only be able to catch the local continuation, and the term ends in $\llbracket c \rrbracket\llbracket e \rrbracket$ instead of $\llbracket c \rrbracket$. We thus need to restrict 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[13] to temporarily encapsulate the evaluation of q when reducing such a command:

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

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

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

This will also allow us to restrict the use of the list of dependencies to the derivation of judgments involving a delimited continuation, and to fully absorb the potential inconsistency in the type of $\hat{\mathfrak{tp}}$. In Section 7.2, we will extend the language according to this intuition, and see how to design a continuation-passing style translation in Section 8.3.

7.2 Extension of the system

7.2.1 Limits of the value restriction

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

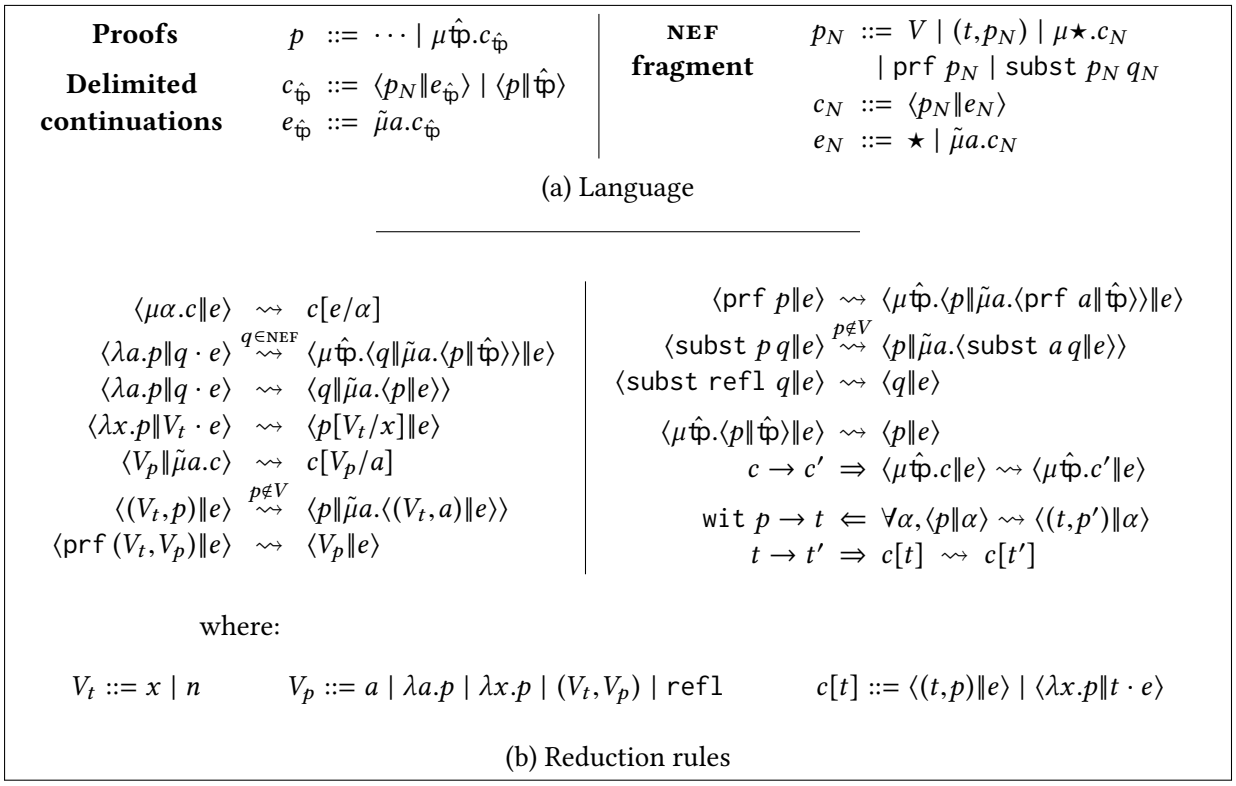
We can follow several intuitions. First, we saw in the previous section that we could actually allow any proof terms as long as its CPS translation uses its continuation and uses it only once. We do not have such a translation yet, but syntactically, these are the proof terms that can be expressed (up to α-conversion) in the $\lambda\mu\tilde{\mu}$-calculus with only one continuation variable (that we call $\star$ in Figure 7.4), and which do not contain application[14]. 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:

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

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

[13] We stick here to the presentations of delimited continuations in [71, 5], where $\hat{\mathfrak{tp}}$ is used to denote the top-level delimiter.

[14] 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.



(a) Language

(b) Reduction rules

Figure 7.4: $\mathrm{dL}_{\hat{\mathfrak{tp}}}$: extension of dL with delimited continuations

Interestingly, this corresponds exactly to the so-called *negative-elimination-free* (NEF) proofs of Herbelin [70]. To interpret the axiom of dependent choice, he designed a classical proof system with dependent types in natural deduction, in which the dependent types allow the use of NEF proofs.

Second, Lepigre defined in a recent work [108] 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ the system dL of Section 7.1 extended with delimited continuations, and define the fragment of *negative-elimination-free* proof terms (NEF). The syntax of both categories is given by Figure 7.4, the proofs in the NEF fragment are considered up to α-conversion for the context variables[15]. The reduction rules, given in Figure 7.4, are slightly different from the rules in Section 7.1. In the case $\langle \lambda a.p \| q \cdot e \rangle$ with $q \in \text{NEF}$ (resp. $\langle \mathtt{prf}\ p \| e \rangle$), 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[16]. We add standard rules for delimited continuations [71, 5], 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.

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

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

Regular mode:

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

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

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

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

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

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

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

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

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

Dependent mode:

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

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

Figure 7.5: Type system for $\mathrm{dL}_{\hat{\mathfrak{p}}}$

7.2.2 Delimiting the scope of dependencies

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

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

and we now distinguish two modes. The regular mode corresponds to a derivation without dependency issues whose typing rules are the same as in Figure 7.2 without the list of dependencies; plus the new rule of introduction of a delimited continuation $\hat{\mathfrak{tp}}_I$. The dependent mode is used to type commands and contexts involving $\hat{\mathfrak{tp}}$, and we use the symbol $\vdash_d$ to denote the sequents. There are three rules: one to type $\hat{\mathfrak{tp}}$, which is the only one where we use the dependencies to unify dependencies; one to type context of the form $\tilde{\mu}a.c$ (the rule is the same as the former rule for $\tilde{\mu}a.c$ in Section 7.1); and a last one to type commands $\langle p \| e \rangle$, where we observe that the premise for p is typed in regular mode.

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

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

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

Lemma 7.14. *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 CONV-rules have been pushed to the left of cuts):

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

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:

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

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

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

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

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

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

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

□

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

Theorem 7.15 (Subject reduction)**.** *If c, c' are two commands of $\mathrm{dL}_{\hat{\mathfrak{p}}}$ 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 7.8, because most of the rules do not involve dependencies. We only give some key cases.

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

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

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

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

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

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

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

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

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

so that we can prove that for any b:

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

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

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

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

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

- **Case** $\langle \mu\hat{\mathfrak{tp}}.\langle p\|\hat{\mathfrak{tp}}\rangle\|e\rangle \rightsquigarrow \langle p\|e\rangle$.
  This case is trivial, because in a typing derivation for the command on the left, $\hat{\mathfrak{tp}}$ is typed with an empty list of dependencies, thus the type of p,e and $\hat{\mathfrak{tp}}$ coincides.
- **Case** $\langle \mu\hat{\mathfrak{tp}}.c\|e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{tp}}.c'\|e\rangle$ with $c \rightsquigarrow c'$.
  This case corresponds exactly to Theorem 7.8, 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 7.14.

□

Remark 7.16. 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, which essentially amounts to Lemma 7.14. This shows that the relaxation to the NEF fragment is valid even without delimited continuations.

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. ⌟

7.3 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 [62, 32]. From a logical point of view, they generally amount to negative translations that allow to embed classical logic into intuitionistic logic [42]. Yet, we know that removing classical control (*i.e.* classical logic) of our language leaves us with a sound intuitionistic type theory. We will now see how to design a CPS translation for our language which will allow us to prove its soundness.

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

(a) Language and formulas

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

(b) Reduction rules

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

$$\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : \Pi a : A.B}\ (\to_I) \qquad \frac{\Gamma \vdash p : \Pi a : A.B \quad \Gamma \vdash q : A}{\Gamma \vdash p\,q : B[q/a]}\ (\to_E) \qquad \frac{\Gamma, x : \mathbb{N} \vdash p : A}{\Gamma \vdash \lambda x.p : \forall x^{\mathbb{N}} A}\ (\forall^1_I)$$

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

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

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

(c) Type system

Figure 7.6: Target language

7.3.1 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 it is common for CPS translations, the evaluation follows a head-reduction strategy. The syntax of the language and its reduction rules are given by Figure 7.6.

The type system, also presented in Figure 7.6, is defined as expected, with the addition of a second-order quantification that we will use in the sequel to refine the type of translations of terms and NEF proofs. As for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ the type system has a conversion rule, where the relation $A \equiv B$ is the symmetric-transitive closure of $A \triangleright B$, defined once again as the congruence over the reduction $\longrightarrow$ and by the rules:

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

7.3.2 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 7.7, in which we tag some lambdas with a bullet $\lambda^\bullet$ for technical reasons. The translation for delimited continuation follows the intuition we presented in Section 7.1.6, and the definition for stacks $t \cdot e$ and $q \cdot e$ (with q NEF) inlines the reduction producing a command with

$$
\begin{array}{lll}
[\![\mathtt{wit}\ p]\!]_t \triangleq \lambda k.[\![p]\!]_p\ (\lambda^\bullet q.k\ (\mathtt{wit}\ q)) & & [\![n]\!]_{V_t} \triangleq \bar{n} \\
[\![x]\!]_t \triangleq \lambda k.k\ x & & \\
\hline
[\![a]\!]_V \triangleq a & & [\![\mathtt{refl}]\!]_V \triangleq \mathtt{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,[\![V]\!]_V) & & \\
\hline
[\![V]\!]_p \triangleq \lambda k.k\ [\![V]\!]_V & & [\![\mu\hat{\mathfrak{p}}.c]\!]_p \triangleq \lambda k.[\![c]\!]_{\hat{\mathfrak{p}}} k \\
[\![\mu\alpha.c]\!]_p \triangleq \lambda^\bullet \alpha.[\![c]\!]_c & & \\
[\![\mathtt{prf}\ p]\!]_p \triangleq \lambda^\bullet k.([\![p]\!]_p\ (\lambda^\bullet q \lambda k'.k'\ (\mathtt{prf}\ q)))\ k & & \\
[\![(t,p)]\!]_p \triangleq \lambda^\bullet k.[\![p]\!]_p([\![t]\!]_t\ (\lambda x \lambda^\bullet a.k\ (x,a))) & & \\
[\![\mathtt{subst}\ V\ q]\!]_p \triangleq \lambda k.[\![q]\!]_p(\lambda^\bullet q'.k\ (\mathtt{subst}\ [\![V]\!]_V\ q'))) & & \\
[\![\mathtt{subst}\ p\ q]\!]_p \triangleq \lambda k.[\![p]\!]_p\ (\lambda^\bullet p'.[\![q]\!]_p(\lambda^\bullet q'.k\ (\mathtt{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{\mathfrak{p}} \rangle]\!]_{\hat{\mathfrak{p}}} \triangleq [\![p]\!]_p \\
[\![\langle p \| e \rangle]\!]_{\hat{\mathfrak{p}}} \triangleq [\![p]\!]_p\ [\![e]\!]_{e_{\hat{\mathfrak{p}}}} \quad (e \neq \hat{\mathfrak{p}}) & & [\![\tilde{\mu}a.c]\!]_{e_{\hat{\mathfrak{p}}}} \triangleq \lambda^\bullet a.[\![c]\!]_{\hat{\mathfrak{p}}}
\end{array}
$$

Figure 7.7: Continuation-passing style translation

a delimited continuation. All the other rules are natural[17] in the sense that they reflect the reduction rule $\rightsquigarrow$, except for the translation of pairs (t,p):

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

The natural definition would have been $\lambda k.[\![t]\!]_t\ (\lambda u.[\![p]\!]_p\ \lambda q.k\ (u,q))$, however such a term would have been ill-typed (while this definition is correct, as we will see in the proof of Lemma 7.25). 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[18], 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 7.7.

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

Lemma 7.17. *The translation satisfies the following properties:*

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

[17] As usual, we actually obtained the translation from an intermediate step consisting in the definition of an context-free abstract machine. The reader will recognize the usual descent (in call-by-value) through the levels of p, e, V.

[18] In fact, we will see in the next chapter that this requires a kind of co-delimited continuation.

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

Figure 7.8: Linearity of the translation for NEF proofs

In particular, we have :

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

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 7.8. We detail the case (t,p) with $p \in \text{NEF}$ to give an insight of the proof.

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

□

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

Proposition 7.18. *If c,c' be two commands of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ such that $c \rightsquigarrow c'$, then $[\![c]\!]_c =_\beta [\![c']\!]_c$*

Proof. By induction on the reduction rules for $\rightsquigarrow$, using Lemma 7.17 for cases involving a term t. □

We can in fact prove a finer result to show that any infinite reduction sequence in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ is responsible for an infinite reduction sequence through the translation. Using the preservation of typing (Proposition 7.26) 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.

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

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

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

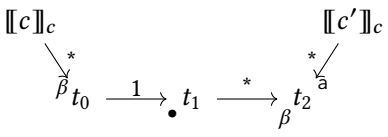
The first point corresponds thereafter to Proposition 7.21, the second one to the Proposition 7.22. As a matter of fact, the most difficult part is somehow anterior to these points. It consists in understanding *how* a reduction step can be reflected through the translation and why it is *enough* to ensure the preservation of normalization (that is the third point). Instead of stating the result directly and give a long and tedious proof of its correctness, we will rather sketch its main steps.

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

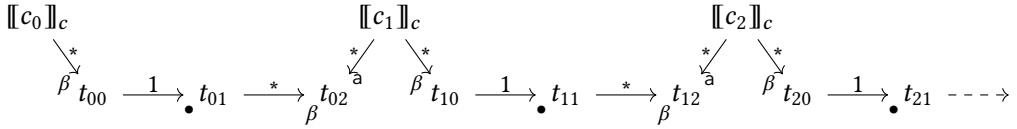
- *administrative reductions*, that we denote by $\longrightarrow_{\mathsf{a}}$, which correspond to continuation-passing and computationally irrelevant (w.r.t. to $\mathrm{dL}_{\hat{\mathfrak{tp}}}$) reduction steps. These are defined as the β-reduction steps of non-annotated λs.
- *distinguished reductions*,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

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

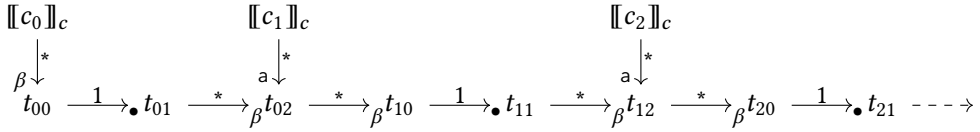
Second, let us assume for a while that we can show for any reduction $c \rightsquigarrow c'$ we obtain:



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



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



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

The first generalization consists to allow distinguished reductions for redexes that are not in head positions. The safety of this generalization follows from this proposition:

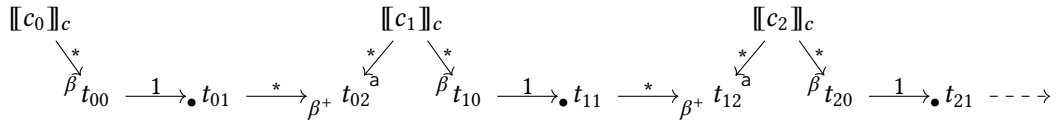
Proposition 7.19. *If $u \longrightarrow_\bullet u'$ and $t[u']$ does not normalize, then neither does $t[u]$.*

Proof. By induction on the structure of t, a very similar proof can be found in [84]. □

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

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

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 reduction relation defined as the union $\longrightarrow_\beta \cup \xrightarrow{?}_\bullet$, which is no longer 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 $[\![c_0]\!]_c$, the previous proposition ensures us that if t_{02} (t_{12}, etc.) does not normalize, it is enough to have an arrow $t_{01} \xrightarrow{*}_{\beta^+} t_{02}$ ($t_{11} \xrightarrow{*}_{\beta^+} t_{12}$, etc.) to deduce that t_{01} does not normalize either. Hence it is enough to prove that we have the following thread scheme, where we took advantage of this observation:

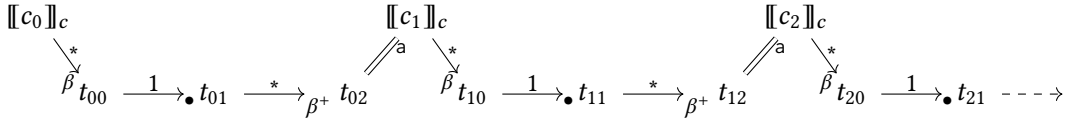


In the same spirit, if we define $=_{\mathsf{a}}$ to be the congruence over terms induced by the reduction $\longrightarrow_{\mathsf{a}}$, we can show that if a term has a redex for the distinguished relation in head position, then so does any (administratively) congruent term.

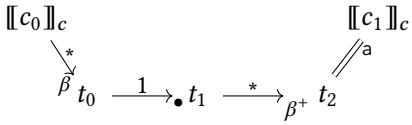
Proposition 7.20. *If $t \xrightarrow{1}_\bullet u$ and $t =_{\mathsf{a}} t'$, then there exists u' such that $t' \xrightarrow{1}_\bullet u'$ and $u =_{\mathsf{a}} u'$.*

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

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



For all the reasons explained above, such a reduction scheme 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:



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

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

which are reflected in the reduction scheme (2).

Proof. The proof is done by induction on the reduction $\rightsquigarrow$ (see Figure 7.4). 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 $\overset{?}{\longrightarrow}_{\mathsf{a}}$ to denote the possibility of performing an administrative reduction not in head position, defined by:

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

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

- **Case** $\langle \mu\alpha.c \| e\rangle \rightsquigarrow c[e/\alpha]$:
  We have:
  $$\begin{array}{rcl} [\![\langle \mu\alpha.c \| e\rangle]\!]_c & = & (\lambda^\bullet \alpha.[\![c]\!]_c)[\![e]\!]_e \\ & \longrightarrow_\bullet & [\![c]\!]_c[[\![e]\!]_e/\alpha] = [\![c[e/\alpha]]\!]_c \end{array}$$

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

- **Case** $\langle \lambda a.p \| q_N \cdot e\rangle \overset{q_N \in \mathrm{NEF}}{\rightsquigarrow} \langle \mu\hat{\mathfrak{tp}}.\langle q_N \| \tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}}\rangle\rangle \| e\rangle$:
  We know by Lemma 7.17 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:

  $$[\![q_N]\!]_p\ k \overset{*}{\longrightarrow}_\beta k\ q_N^+ \longrightarrow_\bullet k'\ q_N^+ \underset{\beta}{\longleftarrow} [\![q_N]\!]_p\ k'$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **Case** $c \rightsquigarrow c' \Rightarrow \langle \mu\hat{\mathfrak{p}}.c \| e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.c' \| e\rangle$:
  By induction hypothesis, we get that $[\![c]\!]_c \xrightarrow{*}_{\beta^+} t =_{\mathsf{a}} [\![c']\!]_c$ for some term t. Therefore we have:
  $$\begin{array}{rcl} \langle \mu\hat{\mathfrak{p}}.c \| e\rangle & = & (\lambda k.[\![c]\!]_c\ k)\ [\![e]\!]_e \\ & \longrightarrow_{\mathsf{a}} & [\![c]\!]_c\ [\![e]\!]_e \\ & \xrightarrow{*}_{\beta^+} & t\ [\![e]\!]_e \\ & =_{\mathsf{a}} & [\![c']\!]_c\ [\![e]\!]_e \\ & {}_{\mathsf{a}}\longleftarrow & (\lambda k.[\![c']\!]_c\ k)\ [\![e]\!]_e = \langle \mu\hat{\mathfrak{p}}.c' \| e\rangle \end{array}$$

- **Case** $t \to t' \Rightarrow c[t] \rightsquigarrow c[t']$:
  As such, the translation does not allow an analysis of this case, mainly because we did not give an explicit small-step semantics for terms, and defined terms reduction through a big-step semantics:
  $$\forall \alpha, \langle p \| \alpha\rangle \overset{*}{\rightsquigarrow} \langle (t,q) \| \alpha\rangle \Rightarrow \mathtt{wit}\ p \to t$$
  However, we claim that we could have extended the language of $\mathrm{dL}_{\hat{\mathfrak{p}}}$ with commands for terms:
  $$c_t ::= \langle t \| e_t\rangle \qquad e_t ::= \tilde{\mu}x.c[t] \qquad c[] ::= \langle ([],p) \| e\rangle \mid \langle \lambda x.p \| [] \cdot e\rangle$$
  and adding dual operators for (co-)delimited continuations to allow for a small-step definition of terms reduction:
  $$\begin{array}{l|r} \langle \lambda x.p \| t \cdot e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle & \langle V_t \| \tilde{\mu}x.c_t\rangle \rightsquigarrow c_t[V_t/x] \\ \langle \mathtt{wit}\ p \| e_t\rangle \rightsquigarrow \langle p \| \tilde{\mu}a.\langle \mathtt{wit}\ a \| e_t\rangle\rangle & \langle \mathtt{wit}\ (V_t,V_p) \| e_t\rangle \rightsquigarrow \langle V_t \| e_t\rangle \\ \langle (t,p) \| e\rangle \rightsquigarrow \langle p \| \tilde{\mu}\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \hat{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle\rangle & \langle V_p \| \tilde{\mu}\hat{\mathfrak{p}}.\langle \hat{\mathfrak{p}} \| e\rangle\rangle \rightsquigarrow \langle V_p \| e\rangle \end{array}$$
  $$c \rightsquigarrow c' \Rightarrow \langle p \| \tilde{\mu}\hat{\mathfrak{p}}.c\rangle \rightsquigarrow \langle p \| \tilde{\mu}\hat{\mathfrak{p}}.c'\rangle$$
  It is worth noting that these rules simulate the big-step definitions we had before while preserving the global call-by-value strategy. Defining the translation for terms in the extended syntax:
  $$\begin{array}{ll} [\![\mathtt{wit}\ V_t]\!]_t \triangleq \lambda k.k\,(\mathtt{wit}\,[\![V_t]\!]_{V_t}) & [\![\tilde{\mu}x.c]\!]_t \triangleq \lambda^{\bullet}x.[\![c]\!]_c \\ [\![\mathtt{wit}\ p]\!]_t \triangleq \lambda k.[\![p]\!]_p\,(\lambda^{\bullet}q.k\,(\mathtt{wit}\ q)) & [\![\langle t \| e_t\rangle]\!]_t \triangleq [\![t]\!]_t\ [\![e_t]\!]_t \\ [\![\tilde{\mu}\hat{\mathfrak{p}}.c_t]\!]_t \triangleq [\![c_t]\!]_t & [\![\hat{\mathfrak{p}}]\!]_p \triangleq \lambda^{\bullet}k.k \end{array}$$
  We can then prove that each reduction rule satisfies the expected scheme.
  Case $\langle \lambda x.p \| t \cdot e\rangle \rightsquigarrow \langle \mu\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle$:
  We have:
  $$\begin{array}{rcl} \langle \lambda x.p \| t \cdot e\rangle & = & (\lambda^{\bullet}k.k\ \lambda^{\bullet}x.[\![p]\!]_p)\ (\lambda p.([\![t]\!]_t\ (\lambda^{\bullet}v.p\ v))\ [\![e]\!]_e) \\ & \longrightarrow_{\bullet} & (\lambda p.([\![t]\!]_t\ (\lambda^{\bullet}v.p\ v))\ [\![e]\!]_e)\ \lambda^{\bullet}x.[\![p]\!]_p \\ & \longrightarrow_{\mathsf{a}} & ([\![t]\!]_t\ (\lambda^{\bullet}v.(\lambda^{\bullet}x.[\![p]\!]_p)\ v))\ [\![e]\!]_e \\ & \xrightarrow{?}_{\bullet} & ([\![t]\!]_t\ (\lambda^{\bullet}x.[\![p]\!]))\ [\![e]\!]_e \\ & {}_{\mathsf{a}^+}\longleftarrow & \lambda k.(([\![t]\!]_t\ (\lambda^{\bullet}x.[\![p]\!]))\ k)\ [\![e]\!]_e = [\![\langle \mu\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle]\!]_c \end{array}$$
  Case $\langle (t,p) \| e\rangle \rightsquigarrow \langle p \| \tilde{\mu}\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \hat{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle\rangle$:
  We have:
  $$\begin{array}{rcl} \langle (t,p) \| e\rangle & = & (\lambda^{\bullet}k.[\![p]\!]_p\ ([\![t]\!]_t\ (\lambda x.\lambda^{\bullet}a.k\ (x,a))))\ [\![e]\!]_e \\ & \longrightarrow_{\bullet} & [\![p]\!]_p\ ([\![t]\!]_t\ (\lambda x.\lambda^{\bullet}a.[\![e]\!]_e\ (x,a))) \\ & {}_{\mathsf{a}^+}\longleftarrow & [\![p]\!]_p\ ([\![t]\!]_t\ (\lambda x.(\lambda k.k)\lambda^{\bullet}a.[\![e]\!]_e\ (x,a))) \\ & {}_{\mathsf{a}^+}\longleftarrow & [\![p]\!]_p\ ([\![t]\!]_t\ (\lambda x.(\lambda k.k)\lambda^{\bullet}a.(\lambda k.k\ (x,a))\ [\![e]\!]_e)) \\ & = & [\![\langle p \| \tilde{\mu}\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \hat{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle\rangle]\!]_c \end{array}$$

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

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

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

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

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

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

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

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

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

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

□

Proposition 7.22. *There is no infinite sequence only made of reductions:*

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

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

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

then the rule (1) makes the quantity (1) decrease while preserving the others, (2) makes the quantity (2) decrease and 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 7.23 (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 7.21). Thus, there is an infinite reduction sequence from $[\![c]\!]_c$ too. □

Theorem 7.24 (Normalization)**.** *If $c : \Gamma \vdash \Delta$, then c normalizes.*

Proof. Using the preservation of typing (Proposition 7.26), 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. □

7.3.4 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 [19]$[\![A]\!]^*$. 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 7.17 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 7.5.2, this will no longer be the case when extending the domain of terms. We naturally extend the translation for types to the translation of contexts, where we consider unified contexts of the form $\Gamma \cup \Delta$:

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

As explained informally in Section 7.1.6 and stated by Lemma 7.17, the translation of a NEF proof term p of type A uses its continuation linearly. In particular, this allows us to refine its type to make it parametric in the return type of the continuation. From a logical point of view, it amounts to replace the double-negation $(A \to \bot) \to \bot$ by Friedman's translation [53]: $\forall R.(A \to R) \to R$. It is worth noticing the correspondences with the continuation monad [46] and the codensity monad. 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 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.

[19] To follow the notations in the previous chapters, we could have written $[\![A]\!]_p$ and $[\![A]\!]_V$ instead of $[\![A]\!]^*$ and $[\![A]\!]^+$. To avoid confusion, we preferred to stick with the notation p^+ for the translation of NEF proofs, which are of type $[\![A]\!]^+$ and not necessarily values.

Lemma 7.25 (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 the typing derivation. We only give the key cases of the proof.

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

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

We want to show that:

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

By induction hypothesis, we have:

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

hence it amounts to showing that for any X we can build the following derivation, where we write Γ_k for the context $[\![\Gamma \cup \Delta]\!], k : \forall x^{\mathbb{N}}X(x)$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma_k \vdash k : \forall x^{\mathbb{N}}X(x)}\ (\mathrm{Ax}_p) \quad \dfrac{\dfrac{}{\Gamma_k, a : \exists x^{\mathbb{N}}.[\![A]\!]^+ \vdash a : \exists x^{\mathbb{N}}.[\![A]\!]^+}\ (\mathrm{Ax}_p)}{\Gamma_k, a : \exists x^{\mathbb{N}}.[\![A]\!]^+ \vdash \texttt{wit}\ a : \mathbb{N}}\ (\texttt{wit})}{\Gamma_k, a : \exists x^{\mathbb{N}}.[\![A]\!]^+ \vdash k\ (\texttt{wit}\ a) : X(\texttt{wit}\ a)}\ (\forall^1_E)}{\Gamma_k \vdash \lambda a.k(\texttt{wit}\ a) : \Pi a : \exists x^{\mathbb{N}}[\![A]\!]^+.X(\texttt{wit}\ a)}\ (\to_I)$$

- **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:

$$[\![\Gamma \cup \Delta]\!] \vdash [\![t]\!]_t : \forall X.(\forall x^{\mathbb{N}}X(x) \to X(t^+)) \qquad (IH_t)$$
$$[\![\Gamma \cup \Delta]\!] \vdash [\![p]\!]_p : \forall Y.(\Pi a : A(t^+).Y(a) \to Y(p^+)) \qquad (IH_p)$$

and we want to show that for any Z:

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

So we need to prove that:

$$[\![\Gamma \cup \Delta]\!], k : \Pi q : \exists x^{\mathbb{N}}.A.Z(q) \vdash [\![p]\!]_p([\![t]\!]_t\ (\lambda x a.k\ (x,a))) : Z(t^+,p^+)$$

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 $[\![t]\!]_p$, and using $Y(a) \triangleq Z(t^+,a)$ in the type of $[\![p]\!]_p$:

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

• **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 $\mathrm{dL}_{\hat{\Psi}}$:

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

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

$$\Gamma' \vdash \lambda k.[\![p]\!]_p\,(\lambda a.[\![q]\!]_p\,k) : \Pi b : B.X(b) \to X(q^+[p^+/a]).$$

By induction, we have:

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

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

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

□

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

Proposition 7.26 (Preservation of typing)**.** *The translation is well-typed, i.e. the following holds:*

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

Proof. The proof is done by induction on the typing derivation, distinguishing cases according to the typing rule used in the conclusion. It is clear that for the NEF cases, Lemma 7.25 implies the result by taking $X(a) = \bot$. The rest of the cases are straightforward, except for the delimited continuations that we detail hereafter. We consider a command $\langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}} \rangle \rangle \| e \rangle$ 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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ is of the form:

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

By induction hypothesis for e and p we obtain:

$$\begin{aligned} \Gamma' &\vdash \llbracket e \rrbracket_e : \llbracket B[q^+] \rrbracket^+ \to \bot \\ \Gamma', a : A^+ &\vdash \llbracket p \rrbracket_p : \llbracket B[a] \rrbracket^* \\ \Gamma' &\vdash \lambda a.\llbracket p \rrbracket_p : \Pi a : A^+.\llbracket B[a] \rrbracket^*, \end{aligned}$$

where $\Gamma' = \llbracket \Gamma \cup \Delta \rrbracket$. Applying Lemma 7.25 for $q \in$ NEF we can derive:

$$\dfrac{\Gamma' \vdash \llbracket q \rrbracket_p : \forall X.(\Pi a : A^+.X(a) \to X(q^+))}{\Gamma' \vdash \llbracket q \rrbracket_p : (\Pi a : A^+.\llbracket B[a] \rrbracket^* \to \llbracket B[q^+] \rrbracket^*}\ (\forall_E^2)$$

We can thus derive that:

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

and finally conclude that:

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

□

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

Theorem 7.27 (Soundness)**.** *For any $p \in \mathrm{dL}_{\hat{\mathfrak{tp}}}$, we have: $\not\vdash p : \bot$.*

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. □

7.4 Embedding into Lepigre's calculus

In a recent paper [108], 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 [97]. In his system, the relativization $a \in A$ is restricted to values, so that we can only type $V : V \in A$:

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

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 7.17, 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[20].

We shall now detail an embedding of $\mathrm{dL}_{\hat{\Phi}}$ into Lepigre's calculus, and explain how to transfer normalization and correctness properties along this translation. Actually, his language is more expressive than ours, since it contains records and pattern-matching (we will only use pairs, *i.e.* records with two fields), but it is not stratified: no distinction is made between a language of terms and a language of proofs. We only recall here the syntax for the fragment of Lepigre's calculus we use, for the reduction rules and the type system the reader should refer to [108]:

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

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

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

Similarly, only values can be pushed on stacks, but we can define processes[21] 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 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 7.9:

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

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

The translation for terms, proofs, contexts and commands of $\mathrm{dL}_{\hat{\Phi}}$, given in Figure 7.9 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 $\top^*$,

[20] 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 $\mathrm{dL}_{\hat{\Phi}}$ is in NEF.

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

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

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

Figure 7.9: Translation of proof terms into Lepigre's calculus

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

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

Figure 7.10: Extension of Lepigre's typing rules for stacks

- the strong existential is encoded as a pair, hence `wit` (resp. `prf`) is mapped to the projection π_1 (resp. π_2).

In [108], 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 also directly prove the adequacy of the realizability model (through the translation) with respect to the typing rules of $\mathrm{dL}_{\hat{\mathfrak{p}}}$. We will detail here a proof of adequacy using the former method in the following. We then need to extend Lepigre's system to be able to type stacks. In fact, the proof of adequacy [108, Theorem 6] suggests a way to do so, since any typing rule for typing stacks is valid as long as it is adequate with the realizability model.

We denote by $A^{\perp\!\!\!\perp}$ the type A when typing a stack, in the same fashion we use to go from a type A in a left rule of two-sided sequent to the type $A^{\perp\!\!\!\perp}$ in a one-sided sequent (see the remark at the end of Section 7.1.3). We also add a distinguished bottom stack $\bullet$ to the syntax, which is given the most general type $\bot^{\perp\!\!\!\perp}$. We change the rules $(*)$ and (μ) of the original type system in [108] and add rules for stacks, whose definitions are guided by the proof of the adequacy [108, Theorem 6] in particular by the $(\Rightarrow_e)$-case. These rules are given in Figure 7.10.

We shall now show that these rules are adequate with respect to the realizability model defined in [108, Section 2].

Proposition 7.28 (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 the typing derivation, we only need to do the proof for the rules we define above (all the other cases correspond to the proof of [108, Theorem 6]).

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

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

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

(μ) The proof is the very same as in [108, Theorem 6].

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

($\Rightarrow_l$) If t is a value v, by induction hypothesis, we have that $v\sigma \in [\![A]\!]_\sigma$ and $\pi\sigma \in [\![B]\!]_\sigma^\bot$ and we need to show that $v\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^\bot$. The proof is already done in the case ($\Rightarrow_e$) (see [108, Theorem 6]). Otherwise, by induction hypothesis, we have that $t\sigma \in [\![A]\!]_\sigma^{\bot\bot}$ and $\pi\sigma \in [\![B]\!]_\sigma^\bot$ and we need to show that $t\sigma \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^\bot$. 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 $[\lambda x.u]\pi \in [\![A]\!]_\sigma^\bot$, 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 $t\sigma \in [\![A \Rightarrow B]\!]_\sigma^{\bot\bot}$ and $\pi\sigma \in [\![B]\!]_\sigma^\bot$, thus $v \cdot \pi\sigma \in [\![A \Rightarrow B]\!]_\sigma^\bot$.

□

It only remains to show that the translation we defined in Figure 7.9 preserves typing to conclude the proof of Proposition 7.30.

Lemma 7.29. *If* $\Gamma \vdash p : A \mid \Delta$ *(in* $\mathrm{dL}_{\hat{\mathrm{tp}}}$*), 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* $(\Gamma \cup \Delta)^* \vdash [\![c]\!]_c : \bot$.

Proof. The proof is an 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 $\mathrm{dL}_{\hat{\mathrm{tp}}}$), 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:

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

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

□

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

Proposition 7.30 (Adequacy)**.** *If* $\Gamma \vdash p : A \mid \Delta$ *and* σ *is a substitution realizing* $(\Gamma \cup \Delta)^*$*, then* $[\![p]\!]_p\sigma \in [\![A^*]\!]_\sigma^{\perp\perp}$.

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

Theorem 7.31 (Soundness)**.** *For any proof* p *in* dL$_{\hat{\mathfrak{p}}}$*, we have:* $\nvdash p : \perp$.

It is worth noting that without delimited continuations, we would not have been able to define an adequate translation, since we would have encountered the same problem as for the CPS translation (see Section 7.1.6).

7.5 Toward dLPA$^\omega$: further extensions

As we explained in the preamble of Section 7.1, we defined dL and dL$_{\hat{\mathfrak{p}}}$ as minimal 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. We will then be ready to define dLPA$^\omega$ in the next chapter, which is the sequent calculus presentation of dPA$^\omega$ using the techniques developed in this chapter.

7.5.1 Intuitionistic sequent calculus

There is not much to say on this topic, but it is worth mentioning that dL and dL$_{\hat{\mathfrak{p}}}$ could be easily restricted to obtain an intuitionistic framework. Indeed, just like for the passage from LK to LJ, we

pretend that 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.

7.5.2 Extending the domain of terms

Throughout the chapter, we only worked with terms of a unique type $\mathbb{N}$, hence it is natural to wonder whether it is possible to extend the domain of terms in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, 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 [119]. 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 [13].

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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. In particular, the obtained proofs cannot drop continuations, which was the whole point of the restriction to the NEF fragment. Hence we could again refine the translation of types, similarly to what we did in Lemma 7.25. Once more, this refinement would also coincide with a computational property similar to Lemma 7.17, expressing the fact that the extraction can be done simply by passing the identity as a continuation[22]. This witnesses the fact that for any function t in the source language, there exists a term t^+ in the target language which represents the same function, even tough 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 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 presentation of type theories are in this case), then it becomes possible to force the extraction of terms of the same type through the translation.

Another solution would consist to define a separate translation for terms. Indeed, as it was reflected by Lemma 7.17, since neither terms nor the NEFproofs they may contain need continuations, they can be directly translated. The corresponding translation is actually an embedding which maps every pure term (without `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 is maybe a good way to take advantage of the stratified presentation.

[22] 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 $\forall R.(A \to R) \to R$), the continuation need 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.

7.5.3 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 subsume the non-dependent implication). If this is obviously enough to encode the usual constructors for pairs (p_1, p_2) (of type $A_1 \wedge A_2$), injections $\iota_i(p)$ (of type $A_1 \vee A_2$), etc..., it seems reasonable to wonder whether such constructors can be directly defined in the language of proofs. 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 it 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:

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

and the reduction rules:

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

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

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

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

We claim that this methodology furnishes in first approximation an approach to the question "*Can I extend this with ... ?*". In particular, it should be enough to get closer to a realistic programming language and extend the language with inductive fix-point operators. We make plain use of these ideas in the next chapter.

7.5.4 A fully sequent-style dependent calculus

While the aim of this chapter 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 else that it was possible to define a sound sequent-calculus with the key ingredients of dependent types, even if these

were presented in a natural deduction spirit. Rather than having left-rules, we presented the existential type and the equality type with the following elimination rules:

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

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

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

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

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

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

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

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

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

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

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

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

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

7.6 Conclusion

In this chapter, we presented dL, a sequent calculus that combines dependent types and classical control by means of a syntactic restriction to values. We proved in Section 7.1 the normalization of dL for typed terms, as well as its soundness. This calculus can be extended with delimited continuations, which permits us to extend the syntactic restriction for dependent types to the fragment of negative-elimination-free proofs. The resulting calculus $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, that we presented in Section 7.2, is suitable for the definition of a dependently typed translation to an intuitionistic type theory. As shown in Section 8.3, this translation guarantees both the normalization and the soundness of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. Furthermore, a similar translation can be designed to embedded $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ into Lepigre's calculus. As explained in Section 7.4, this provides an alternative way of proving the soundness of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$.

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 with Fürhmann's *thunkability* [54] or Munch-Maccagnoni's *linearity* [127]. Furthermore, it might be of interest to check whether this restriction could make dependent types compatible with other side-effects, in presence of classical logic or not. More generally, we would like to better understand the possible connections between our calculus 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.