A Classical Sequent Calculus with Dependent Types

Étienne Miquey[1,2](✉)

[1] PI.R2 (INRIA), IRIF, Université Paris-Diderot, Paris, France
emiquey@irif.fr

[2] IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

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

We start from the call-by-value version of the $\lambda\mu\tilde{\mu}$ language and design a minimal language with a value restriction and a type system that includes a list of explicit dependencies and maintains type safety. We then show how to relax the value restriction and introduce delimited continuations to directly prove the consistency by means of a continuation-passing-style translation. Finally, we relate our calculus to a similar system by Lepigre [19], and present a methodology to transfer properties from this system to our own.

Keywords: Dependent types · Sequent calculus · Classical logic · Control operators · Call-by-value · Delimited continuations · Continuation-passing style translation · Value restriction

1 Introduction

1.1 Control Operators and Dependent Types

Originally created to deepen the connection between programming and logic, dependent types are now a key feature of numerous functional programing languages. On the programming side, they allow for the expression of very precise specifications, while on the logical side, they permit definitions of proof terms for axioms like the full axiom of choice. This is the case in Coq or Agda, two of the most actively developed proof assistants, which both provide dependent types. However, both of them rely on a constructive type theory (Coquand and Huet's calculus of constructions for Coq [6], and Martin-Löf's type theory [20] for Agda), and lack classical logic.

In 1990, Griffin discovered [12] that the control operator `call/cc` (short for *call with current continuation*) of the Scheme programming language could be typed by Peirce's $((A \to B) \to A) \to A)$, thus extending the formulæ-as-types interpretation [17]. As Peirce's law is known to imply, in an intuitionistic framework, all the other forms of classical reasoning (excluded middle, *reductio ad absurdum*, double negation elimination, etc.), this discovery opened the way for a direct computational interpretation of classical proofs, using control operators and their ability to *backtrack*. Several calculi were born from this idea, such as Parigot's $\lambda\mu$-calculus [22], Barbanera and Berardi's symmetric λ-calculus [3], Krivine's λ_c-calculus [18] or Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [7].

Nevertheless, dependent types are known to misbehave in the presence of control operators, causing a degeneracy of the domain of discourse [14]. Some restrictions on the dependent types are thus necessary to make them compatible with classical logic. Although dependent types and classical logic have been deeply studied separately, the question to know how to design a system compatible with both features does not have yet a general and definitive answer. Recent works from Herbelin [15] and Lepigre [19] proposed some restrictions on the dependent types to tackle the issue in the case of a proof system in natural deduction, while Blot [5] designed a hybrid realizability model where dependent types are restricted to an intuitionistic fragment. Other works by Ahman et al. [1] or Vákár [23] also studied the interplay of dependent types and different computational effects (*e.g.* divergence, I/O, local references, exceptions).

1.2 Call-By-Value and Value Restriction

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

In this paper, we present a language following the call-by-value reduction strategy, which is as much a design choice as a goal in itself. Indeed, when considering a language with control operators (or other kind of side-effects), soundness often turns out to be subtle to preserve in call-by-value. The first issues in call-by-value in the presence of side-effects were related to references [25] and polymorphism [13]. In both cases, a simple and elegant solution (but way too restrictive in practice [11,19]) to solve the inconsistencies consists in a restriction to values for the problematic cases, restoring then a sound type system. Recently, Lepigre presented a proof system providing dependent types and a control operator [19], whose consistency is preserved by means of a semantical value restriction defined for terms that behave as values up to observational equivalence. In the present work, we will rather use a syntactic restriction to a fragment of proofs that allows slightly more than values. This restriction is inspired by the negative-elimination-free fragment of Herbelin's dPAω system [15].

1.3 A Sequent Calculus Presentation

The main achievement of this paper is to give a sequent calculus presentation of a call-by-value language with a control operator and dependent types, and to justify its soundness through a continuation-passing style translation. Our calculus is an extension of the $\lambda\mu\tilde{\mu}$-calculus [7] to dependent types. Amongst other motivations, such a calculus is close to an abstract machine, which makes it particularly suitable to define CPS translations or to be an intermediate language for compilation [8]. In particular, the system we develop might be a first step to allow the adaption of the well-understood continuation-passing style translations for ML in order to design a typed compilation of a system with dependent types such as Coq.

However, in addition to the simultaneous presence of control and dependent types, the sequent calculus presentation itself is responsible for another difficulty. As we will see in Sect. 2.5, the usual call-by-value strategy of the $\lambda\mu\tilde{\mu}$-calculus causes subject reduction to fail. The problem can be understood as a desynchronization of the type system with the reduction. It can be solved by the addition of an explicit list of dependencies in the type derivations.

1.4 Delimited Continuations and CPS Translation

Yet, we will show that the compensation within the typing derivations does not completely fix the problem, and in particular that we are unable to derive a continuation-passing style translation. We present a way to solve this issue by introducing delimited continuations, which are used to force the purity needed for dependent types in an otherwise impure language. It also justifies the relaxation of the value restriction and leads to the definition of the negative-elimination-free fragment (Sect. 3). Finally, it permits the design in Sect. 4 of a continuation-passing style translation that preserves dependent types and allows for proving the soundness of our system.

1.5 Contributions of the Paper

Our main contributions in this paper are:

- we soundly combine dependent types and control operators by mean of a syntactic restriction to the negative-elimination-free fragment;
- we give a sequent calculus presentation and solve the type-soundness issues it raises in two different ways;

- our second solution uses delimited continuations to ensure consistency with dependent types and provides us with a CPS translation (carrying dependent types) to a calculus without control operator;
- we relate our system to Lepigre's calculus, which offers an additional way of proving the consistency of our system.

For economy of space, most of our statements only comes with sketches of their proofs, full proofs are given in the appendices of a longer version available at:

https://hal.inria.fr/hal-01375977.

2 A Minimal Classical Language

2.1 A Brief Recap on the $\lambda\mu\tilde{\mu}$-Calculus

We recall here the spirit of the $\lambda\mu\tilde{\mu}$-calculus, for further details and references please refer to the original article [7]. The syntax and reduction rules (parameterized over a sets of proofs $\mathcal{V}$ and a set of contexts $\mathcal{E}$) are given by:

$$\begin{array}{lrl|lcll}
\text{Proofs} & p & ::= V \mid \mu\alpha.c & & & & \\
\text{Values} & V & ::= a \mid \lambda a.p & \langle t \| \tilde{\mu}x.c\rangle & \to & c[x := t] & v \in \mathcal{V} \\
\text{Contexts} & e & ::= E \mid \tilde{\mu}a.c & \langle \mu\alpha.c \| e\rangle & \to & c[\alpha := e] & e \in \mathcal{E} \\
\text{Co-values} & E & ::= \alpha \mid p \cdot e & \langle \lambda x.t \| u \cdot e\rangle & \to & \langle u \| \tilde{\mu}x.\langle t \| e\rangle\rangle & \\
\text{Commands} & c & ::= \langle p \| e\rangle & & & &
\end{array}$$

where $\tilde{\mu}a.c$ can be read as a context **let** $a = [\ ]$ **in** c. A command can be understood as a state of an abstract machine, representing the evaluation of a proof (the program) against a context (the stack). The μ operator comes from Parigot's $\lambda\mu$-calculus [22], $\mu\alpha$ binds an evaluation context to a context variable α in the same way $\tilde{\mu}a$ binds a proof to some proof variable a.

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

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

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

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

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

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

The difference between call-by-name and call-by-value can be characterized by how this critical pair is solved, by defining $\mathcal{V}$ and $\mathcal{E}$ such that the two rules do not overlap. The call-by-name evaluation strategy amounts to the case where $\mathcal{V} \triangleq$ *Proofs* and $\mathcal{E} \triangleq$ *Co-values*, while call-by-value corresponds to $\mathcal{V} \triangleq$ *Values* and $\mathcal{E} \triangleq$ *Contexts*. Both strategies can also been characterized through different CPS translations [7, Sect. 8].

2.2 The Language

As shown by Herbelin [14], it is possible to derive inconsistencies from a minimal classical language with dependent types. Intuitively, the incoherence comes from the fact that if p is a classical proof of the form

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

the seek of a witness by a term $\mathsf{wit}\,p$ is likely to reduce to 0, while the reduction of $\mathsf{prf}\,p$ would have backtracked before giving 1 as a witness and the corresponding certificate. The easiest and usual approach to prevent this is to impose a restriction to values for proofs appearing inside dependent types and operators. In this section we will focus on this solution in the similar minimal framework, and show how it permits to keep the proof system coherent. We shall see further in Sect. 3 how to relax this constraint.

We give here a stratified presentation of dependent types, by syntactically distinguishing *terms*—that represent mathematical objects—from *proof terms*–that represent mathematical proofs[1]. We place ourselves in the framework of the $\lambda\mu\tilde{\mu}$-calculus to which we add:

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

For simplicity reasons, we will only consider terms of type $\mathbb{N}$ throughout this paper. We address the question of extending the domain of terms in Sect. 6.2.

[1] This design choice is usually a matter of taste and might seem unusual for some readers. However, it has the advantage of clearly enlighten the different treatments for term and proofs through the CPS in the next sections.

[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.

The syntax of the corresponding system, that we call dL, is given by:

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

The formulas are defined by:

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

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

2.3 Reduction Rules

As explained in the introduction of this section, a backtracking proof might give place to different witnesses and proofs according to the context of reduction, leading to incoherences [14]. 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 below (where $t \to t'$ denotes the reduction of terms and $c \rightsquigarrow c'$ the reduction of commands), follow the call-by-value evaluation principle:

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

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

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 $A \triangleright B$, defined as a congruence over term reduction (*i.e.* if $t \to t'$ then $A[t] \ \triangleright \ A[t']$) and by the rules:

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

2.4 Typing Rules

As we previously explained, in this section we will limit ourselves to the simple case where dependent types are restricted to values, to make them compatible

[3] The reader might recognize the rule (ς) of Wadler's sequent calculus [24].

with classical logic. But even with this restriction, defining the type system in the most naive way leads to a system in which subject reduction will fail. Having a look at the β-reduction rule gives us an insight of what happens. Let us consider a proof $\lambda a.p : \Pi_{a:A}B$ and a context $q \cdot e : \Pi_{a:A}B$ (with q a value). A typing derivation of the corresponding command is of the form:

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

while the command will reduce as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 1. Typing rules

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

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

Furthermore, we introduce the notation $\Gamma \mid e : A \vdash \Delta;\sigma\{\cdot|\dagger\}$ to avoid the definition of a second type of sequent $\Gamma \mid e : A \vdash \Delta;\sigma$ to type contexts when dropping the (open) binding $\{\cdot|p\}$. Alternatively, one can think of $\dagger$ as any proof term not in $\mathcal{D}$, which is the same with respect to the dependencies list. The resulting set of typing rules is given in Fig. 1, where we assume that every variable bound in the typing context is bound only once (proofs and contexts are considered up to α-conversion).

Note that we work with two-sided sequents here to stay as close as possible to the original presentation of the $\lambda\mu\tilde{\mu}$-calculus [7]. In particular it means that a type in Δ might depend on a variable previously introduced in Γ and reciprocally, so that the split into two contexts makes us lose track of the order of introduction of the hypothesis. In the sequel, to be able to properly define a typed CPS translation, we consider that we can unify both contexts into a single one that is coherent with respect to the order in which the hypothesis have been introduced. We denote by $\Gamma \cup \Delta$ this context, where the assumptions of Γ remain unchanged, while the former assumptions $(\alpha : A)$ in Δ are denoted by $(\alpha : A^{\perp\!\!\perp})$.

2.5 Subject Reduction

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

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

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

Lemma 1 (Dependencies weakening). *For any dependencies list σ we have:*

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

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

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

Corollary 2. *If $\sigma \Rrightarrow \sigma'$, then for any p, e, Γ, Δ:*

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

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

Theorem 3 (Subject reduction). *If c, c' are two commands of dL such that $c : (\Gamma \vdash \Delta)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta)$.*

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

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

where the hypothesis $A \equiv B$ is implicit. We only give the key case of β-reduction.

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

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

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

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

using Corollary 2 to weaken the dependencies in Π_e. □

2.6 Soundness

We sketch here a proof of the soundness of dL with a value restriction. A more interesting proof through a continuation-passing translation is presented in Sect. 4. We first show that typed commands of dL normalize by translation to the simply-typed $\lambda\mu\tilde\mu$-calculus with pairs (*i.e.* extended with proofs of the form (p_1, p_2) and contexts of the form $\tilde\mu(a_1, a_2).c$). The translation essentially consists of erasing the dependencies in types, turning the dependent products into arrows and the dependent sum into a pair. The erasure procedure is defined by:

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

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

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

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

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

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

Proposition 4. *If $c : \Gamma \vdash \Delta; \sigma$, then $c^* : \Gamma^* \vdash \Delta^*$. The same holds for proofs and contexts.*

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

Proposition 5. *If $c : (\Gamma \vdash \Delta; \varepsilon)$, then c normalizes.*

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

Theorem 6 *(Soundness). For any $p \in dL$, we have $\nvdash p : \bot$.*

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

2.7 Toward a Continuation-Passing Style Translation

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

To demonstrate the correctness of our system, we might be tempted to define a translation to a subsystem without dependent types or control operator. We will discuss later in Sect. 5 a solution to handle the dependencies. We will focus here on the possibility of removing the classical part from dL, that is to define a translation that gets rid of the control operator. The use of continuation-passing style translations to address this issue is very common, and it was already studied for the simply-typed $\lambda\mu\tilde{\mu}$-calculus [7]. However, as it is defined to this point, dL is not suitable for the design of a CPS translation.

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

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

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

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

We follow the idea that the correctness is guaranteed by the head-reduction strategy, preventing $\langle p \| e\rangle$ from reducing before the substitution of a was made. We would like to ensure the same thing happens in the target language (that will also be equipped with a head-reduction strategy), namely that $[\![p]\!]$ cannot be applied to $[\![e]\!]$ before $[\![q]\!]$ has furnished a value to substitute for a. This would correspond informally to the term[6]:

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

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

The second observation is that such a term suggests the use of delimited continuations[7] 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 $\tilde{\mu} a.\langle p \| \hat{\mathfrak{tp}}\rangle$. This will also allow us to restrict the use of the dependencies list to the derivation of judgments involving a delimited continuation, and to fully absorb the potential inconsistency in the type of $\hat{\mathfrak{tp}}$.

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

3 Extension of the system

3.1 Limits of the Value Restriction

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

[6] We will see in Sect. 4.3 that such a term could be typed by turning the type $A \to \bot$ of the continuation that $[\![q]\!]$ is waiting for into a (dependent) type $\Pi_{a:A} R[a]$ parameterized by R. This way we could have $[\![q]\!] : \forall R(\Pi_{a:A} R[a] \to R[q])$ instead of $[\![q]\!] : ((A \to \bot) \to \bot)$. For $R[a] := (B(a) \to \bot) \to \bot$, the whole term is well-typed. Readers familiar with realizability will also note that such a term is realizable, since it eventually terminates on a correct position $[\![p[q/a]]\!]\,[\![e]\!]$.

[7] We stick here to the presentations of delimited continuations in [2,16], where $\hat{\mathfrak{tp}}$ is used to denote the top-level delimiter.

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 $\star$ (see Fig. 2), and which do not contain application[8]. Interestingly, this corresponds exactly to the so-called *negative-elimination-free* (NEF) proofs of Herbelin [15]. 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 [19] a classical proof system with dependent types, where the dependencies are restricted to values. However, the type system allows derivations of judgments up to an observational equivalence, and thus any proof computationally equivalent to a value can be used. In particular, any proof in the NEF fragment is observationally equivalent to a value, hence is compatible with the dependencies of Lepigre's calculus.

From now on, we consider $\mathrm{dL}_{\hat{\mathtt{tp}}}$ the system dL of Sect. 2 extended with delimited continuations, and define the fragment of *negative-elimination-free* proof terms (NEF). The syntax of both categories is given by Fig. 2, the proofs in the NEF fragment are considered up to α-conversion for the context variables[9]. The reduction rules, given below, are slightly different from the rules in Sect. 2:

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

where:

$$V_t ::= x \mid n \qquad\qquad V_p ::= a \mid \lambda a.p \mid \lambda x.p \mid (V_t,V_p) \mid \mathsf{refl}.$$

In the case $\langle \lambda a.p \| q\cdot e\rangle$ with $q \in$ NEF (resp. $\langle \mathsf{prf}\, p \| e\rangle$), a delimited continuation is now produced during the reduction of the proof term q (resp. p) that is involved in dependencies. As terms can now contain proofs which are not values, we enforce the call-by-value reduction by asking proof values to only contain term values. We elude the problem of reducing terms, by defining meta-rules for them[10]. We add standard rules for delimited continuations [2,16], expressing

[8] 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.

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

[10] 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

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

(a) Language

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

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

(b) Typing rules

Fig. 2. $\mathrm{dL}_{\hat{\mathfrak{tp}}}$: extension of dL with delimited continuations

the fact that when a proof $\mu\hat{\mathfrak{tp}}.c$ is in active position, the current context is temporarily frozen until c is fully reduced.

3.2 Delimiting the Scope of Dependencies

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

$$\mathcal{D} \triangleq \textsc{nef}$$

and we now distinguish two modes. The regular mode corresponds to a derivation without dependency issues whose typing rules are the same as in Fig. 1 without the dependencies list (we do not recall them to save some space); plus the new rule of introduction of a delimited continuation $\hat{\mathfrak{tp}}_I$. The dependent mode is used to type commands and contexts involving $\hat{\mathfrak{tp}}$, and we use the sign $\vdash_d$ to denote the sequents. There are three rules: one to type $\hat{\mathfrak{tp}}$, which is the only one where we use the dependencies to unify dependencies; one to type context of the form $\tilde{\mu}a.c$ (the rule is the same as the former rule for $\tilde{\mu}a.c$ in Sect. 2); and a last one to type commands $\langle p \| e \rangle$, where we observe that the premise for p is typed in regular mode.

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

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

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.

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

Lemma 7. *For any context Γ, Δ, any type A and any $e, \mu\star.c$:*

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

Proof. By definition of the NEF proof terms, $\mu\star.c$ is of the general form $\mu\star.c = \mu\star.\langle p_1 \| \tilde{\mu}a_1.\langle p_2 \| \tilde{\mu}a_2.\langle \ldots \| \tilde{\mu}a_n.\langle p_n \| \star\rangle\rangle\rangle\rangle$. In the case $n = 2$, proving the lemma essantially amounts to showing that for any variable a and any σ:

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

□

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

Theorem 8 (Subject reduction). *If c, c' are two commands of $dL_{\hat{\mathfrak{tp}}}$ such that $c : (\Gamma \vdash \Delta)$ and $c \rightsquigarrow c'$, then $c' : (\Gamma \vdash \Delta)$.*

Proof. Actually, the proof is slightly easier than for Theorem 3, because most of the rules do not involve dependencies. We only present one case here, other key cases are proved in the appendix.

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

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

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

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

□

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

4 A Continuation-Passing Style Translation

We shall now see how to define a continuation-passing style translation from $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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 control operators into purely functional ones [7,12]. From a logical point of view, they generally amount to negative translations that allow to embed classical logic into intuitionistic logic [9]. Yet, we know that removing the control operator (*i.e.* classical logic) of our language leaves us with a sound intuitionistic type theory. We will now see how to design a CPS translation for our language which will allow us to prove its soundness.

4.1 Target Language

We choose the target language an intuitionistic theory in natural deduction that has exactly the same elements as $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ but the control operator: the language makes the difference between terms (of type $\mathbb{N}$) and proofs, it also includes dependent sums and products for type referring to term as well as a dependent product at the level of proofs. As it is common for CPS translations, the evaluation follows a head-reduction strategy. The syntax of the language and its reduction rules are given by Fig. 3.

The type system, presented in Fig. 3, 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:

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

4.2 Translation of the Terms

We can now define the translation of terms, proofs, contexts and commands. The translation for delimited continuation follows the intuition we presented in Sect. 2.7, and the definition for stacks $t \cdot e$ and $q \cdot e$ (with q NEF) inline the reduction producing a command with a delimited continuation. All the other rules are natural, except for the translation of pairs (t, p) in the NEF case:

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

The natural definition is the one given in the non NEF case, but as we observe in the proof of Lemma 11, this definition is incompatible with the expected type for the translation of NEF proofs. This somehow strange definition corresponds to the intuition that we reduce $[\![t]\!]_t$ within a delimited continuation, in order to guarantee that we will not reduce $[\![p]\!]_p$ before $[\![t]\!]_t$ has returned a value to substitute for u. Indeed, the type of $[\![p]\!]_p$ depends on t, while the continuation

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

(a) Language and formulas

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

(b) Reduction rules

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

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

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

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

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

(c) Type system

Fig. 3. Target language

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

Fig. 4. Continuation-passing style translation

$(\lambda q.k\,(u,q))$ depends on u, but both become compatible once u is substituted by the value return by $[\![t]\!]_t$. The complete translation is given in Fig. 4.

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 9. *The translation satisfies the following properties:*

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

Proof. Straightforward mutual induction on the translation, adding similar induction hypothesis for NEF contexts and commands. The terms t^+ and proofs p^+ are given in Fig. 5. We detail the case (t,p) with $p \in$ NEF to give an insight of the proof.

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

□

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

Fig. 5. Linearity of the translation for NEF proofs

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

Proposition 10 (Preservation of reduction). *Let c, c' be two commands of $\mathrm{dL}_{\hat{\mathtt{tp}}}$. If $c \rightsquigarrow c'$, then $[\![c]\!]_c =_\beta [\![c']\!]_c$*

We could actually prove a finer result to show that any reduction step in $\mathrm{dL}_{\hat{\mathtt{tp}}}$ is responsible for at least one step of reduction through the translation, and use the preservation of typing (Proposition 12) together with the normalization of the target language to prove the normalization of $\mathrm{dL}_{\hat{\mathtt{tp}}}$ for typed proof terms.

Claim 1 *If $c : \Gamma \vdash \Delta$, then c normalizes.*

4.3 Translation of Types

We can now define the translation of types, in order to show further that the translation $[\![p]\!]_p$ of a proof p of type A is of type $[\![A]\!]^*$, where $[\![A]\!]^*$ is the double-negation of a type $[\![A]\!]^+$ that depends on the structure of A. Thanks to the restriction of dependent types to NEF proof terms, we can interpret a dependency in p (resp. t) in $\mathrm{dL}_{\hat{tp}}$ by a dependency in p^+ (resp. t^+) in the target language. Lemma 9 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:

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

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 Sect. 6.2, 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 $\Gamma \cup \Delta$:

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

As explained informally in Sect. 2.7 and stated by Lemma 9, 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 [10]: $\forall R.(A \to R) \to R$. Moreover, we can even make the return type of the continuation dependent on its argument $\Pi_{a:A} \to R(a)$, so that the type of $[\![p]\!]_p$ will correspond to the elimination rule:

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

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

Lemma 11. *The following holds:*

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

Proof. The proof is done by mutual induction on the typing rule of $\mathrm{dL}_{\hat{tp}}$ for terms and NEF proofs. We only give one case here to give an insight of the proof.

Case (t,p): 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, using the same notation Γ' for $[\![\Gamma \cup \Delta]\!]$:

$$\begin{aligned}\Gamma' &\vdash [\![t]\!]_t : \forall X.(\forall x^{T^+} X(x) \to X(t^+))\\ \Gamma' &\vdash [\![p]\!]_p : \forall Y.(\Pi_{a:A(t^+)} Y(a) \to Y(p^+))\end{aligned}$$

and we want to show that for any Z:

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

By definition, we have:

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

so we need to prove that:

$$\Gamma_k \vdash ([\![t]\!]_t\,(\lambda ur.r\,(\lambda q.k\,(u,q))))[\![p]\!]_p : Z(t^+,p^+)$$

where $\Gamma_k = \Gamma', k : \Pi_{a:\exists x^{T^+} A} Z(a)$. We let the reader check that such a type is derivable by using $X(x) \triangleq P(x) \to Z(x,a)$ in the type of $[\![t]\!]_p$ where $P(t^+)$ is the type of $[\![p]\!]_p$, and using $Y(a) \triangleq Z(t^+,a)$ in the type of $[\![p]\!]_p$. The crucial point is to see that the bounded variable r is abstracted with type $P(x)$ in the derivation, which would not have been possible in the definition of $[\![(t,p)]\!]_p$ with $p \notin$ NEF. □

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

Proposition 12 (Preservation of typing). *The translation is well-typed, i.e. the following holds:*

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

Proof. The proof is done by induction on the typing rules of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. It is clear that for the NEF cases, Lemma 11 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 $([\![q]\!]_p\,(\lambda a.[\![p]\!]_p))\,[\![e]\!]_e$. The corresponding typing derivation in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ is of the form:

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

By induction hypothesis for e and p we obtain:

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

where $\Gamma' = [\![\Gamma \cup \Delta]\!]$. Applying Lemma 11 for $q \in$ NEF we can derive:

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

We can thus derive that:

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

and finally conclude that:

$$\Gamma' \vdash ([\![q]\!]_p\,(\lambda a.[\![p]\!]_p))\,[\![e]\!]_e : \bot.$$

□

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

Theorem 13 (Soundness). *For any $p \in dL_{\hat{\mathfrak{tp}}}$, we have: $\nvdash p : \bot$.*

5 Embedding in Lepigre's Calculus

In a recent paper [19], 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 [18]. In his system, the relativization $a \in A$ is restricted to values, so that we can only type $V : V \in A$, but the typing judgments are defined up to observational equivalence, so that if t is observationally equivalent to V, one can derive the judgment $t : V \in A$.

Interestingly, as highlighted through the CPS translation by Lemma 9, any NEF proof $p : A$ is observationally equivalent to some value p^+, so that we could derive $p : (p \in A)$ from $p^+ : (p^+ \in A)$. The NEF fragment is thus compatible with the semantical value restriction. The converse is obviously false, observational equivalence allowing us to type realizers that would otherwise be untyped[11].

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

We sketch here an embedding of $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ into Lepigre's calculus, and explain how to transfer normalization and correctness properties along this translation. Actually, his language is more expressive than ours, since it contains records and pattern-matching (we will only use pairs, *i.e.* records with two fields), but it is not stratified: no distinction is made between a language of terms and a language of proofs. We only recall here the syntax for the fragment of Lepigre's calculus we use, for the reduction rules and the type system the reader should refer to [19]:

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

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

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

We first define the translation for types (extended for typing contexts) where the predicate $\mathsf{Nat}(x)$ 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 Fig. 6:

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

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 $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, given in Fig. 6 is almost straightforward. We only want to draw the reader's attention on a few points:

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

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

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

Fig. 6. Translation of proof terms to Lepigre's calculus

In [19], the coherence of the system is justified by a realizability model, and the type system does not allow us to type stacks. Thus we cannot formally prove that the translation preserves the typing, unless we extend the type system in which case this would imply the adequacy. We might also directly prove the adequacy of the realizability model (through the translation) with respect to the typing rules of $\mathrm{dL}_{\hat{\mathrm{tp}}}$. We detail a proof of adequacy using the former method in the appendix.

Proposition 14 (Adequacy). *If $\Gamma \vdash p : A \mid \Delta$ and σ is a substitution realizing $(\Gamma \cup \Delta)^*$, then $[\![p]\!]_p \sigma \in [\![A^*]\!]_\sigma^{\perp\perp}$.*

This immediately implies the soundness of $\mathrm{dL}_{\hat{\mathrm{tp}}}$, 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 $\mathrm{dL}_{\hat{\mathrm{tp}}}$ is a consequence of adequacy.

It is worth noting that without delimited continuations, we would not have been able to define an adequate translation, since we would have encountered the same problem as for the CPS translation (see Sect. 2.7).

6 Further Extensions

As we explained in the preamble of Sect. 2, we defined dL and $\mathrm{dL}_{\hat{\mathrm{tp}}}$ as minimal languages containing all the potential sources of inconsistency we wanted to mix: a control operator, dependent types, and a sequent calculus presentation. It had the benefit to focus our attention on the difficulties inherent to the issue, but on the other hand, the language we obtain is far from being as expressive as other usual proof systems. We claimed our system to be extensible, thus we shall now discuss this matter.

6.1 Adding Expressiveness

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

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

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

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

6.2 Extending the Domain of Terms

Throughout the article, we only worked with terms of a unique type $\mathbb{N}$, hence it is natural to wonder whether it is possible to extend the domain of terms in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, 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 see that a *term* of type $T = \mathbb{N}$ is translated into a *proof* of type $\neg\neg T^+ = \neg\neg\mathbb{N}$, from which we can extract a *term* of type $\mathbb{N}$. However, if T was for instance the function type $\mathbb{N} \to \mathbb{N}$, we would only be able to extract a *proof* of type $T^+ = \mathbb{N} \to \neg\neg\mathbb{N}$. In particular, such a proof would be of the form $\lambda x.p$, where p might backtrack to a former position, for instance before it was extracted, and furnish another proof. This accounts for a well-know phenomenon in classical logic, where witness extraction is limited to formulas in the Σ_0^1-fragment [21]. It also corresponds to the type we obtain for the image of a dependent product $\Pi_{a:A} B$, that is translated to a type $\neg\neg\Pi_{a:A^+} B^*$ where the dependence is in a proof of type A^+. This phenomenon is not surprising and was already observed for other CPS translations for type theories with dependent types [4].

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

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

6.3 A Fully Sequent-Style Dependent Calculus

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

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

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

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

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

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

For any $p \in$ NEF and any variables a, α, $A(\mathsf{wit}\, p)$ is in $A(\mathsf{wit}\,(x, a))_{\{(x,a)|p\}}$ which allows us to derive (using this in the (cut)-rule) the admissibility of the former (prf)-rule (we let the reader check the case of the (subst)-rule):

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

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

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

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

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

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

References

1. Ahman, D., Ghani, N., Plotkin, G.D.: Dependent types and fibred computational effects. In: Jacobs, B., Löding, C. (eds.) FoSSaCS 2016. LNCS, vol. 9634, pp. 36–54. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49630-5_3
2. Ariola, Z.M., Herbelin, H., Sabry, A.: A type-theoretic foundation of delimited continuations. High.-Order Symbolic Comput. **22**(3), 233–273 (2009)
3. Barbanera, F., Berardi, S.: A symmetric lambda calculus for classical program extraction. Inf. Comput. **125**(2), 103–117 (1996)
4. Barthe, G., Hatcliff, J., Sørensen, M.H.B.: CPS translations and applications: the cube and beyond. High.-Order Symbolic Comput. **12**(2), 125–170 (1999)
5. Blot, V.: Hybrid realizability for intuitionistic and classical choice. In: LICS 2016, New York, USA, 5–8 July 2016
6. Coquand, T., Huet, G.: The calculus of constructions. Inf. Comput. **76**(2), 95–120 (1988)
7. Curien, P.-L., Herbelin, H.: The duality of computation. In: Proceedings of ICFP 2000, SIGPLAN Notices, vol. 35, no. 9, pp. 233–243. ACM (2000)
8. Downen, P., Maurer, L., Ariola, Z.M., Jones, S.P.: Sequent calculus as a compiler intermediate language. In: ICFP 2016 (2016)
9. Ferreira, G., Oliva, P.: On various negative translations. In: van Bakel, S., Berardi, S., Berger, U. (eds.) Proceedings Third International Workshop on Classical Logic and Computation, CL&C 2010. EPTCS, Brno, Czech Republic, 21–22 August 2010, vol. 47, pp. 21–33 (2010)
10. Friedman, H.: Classically and intuitionistically provably recursive functions. In: Müller, G.H., Scott, D.S. (eds.) Higher Set Theory. LNM, vol. 669, pp. 21–27. Springer, Heidelberg (1978). doi:10.1007/BFb0103100
11. Garrigue, J.: Relaxing the value restriction. In: Kameyama, Y., Stuckey, P.J. (eds.) FLOPS 2004. LNCS, vol. 2998, pp. 196–213. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24754-8_15
12. Griffin, T.G.: A formulae-as-type notion of control. In: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1990, pp. 47–58. ACM, New York (1990)
13. Harper, R., Lillibridge, M.: Polymorphic type assignment and CPS conversion. LISP Symbolic Comput. **6**(3), 361–379 (1993)
14. Herbelin, H.: On the degeneracy of Σ-types in presence of computational classical logic. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 209–220. Springer, Heidelberg (2005). doi:10.1007/11417170_16

15. Herbelin, H.: A constructive proof of dependent choice, compatible with classical logic. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, 25–28 June 2012, pp. 365–374. IEEE Computer Society (2012)
16. Herbelin, H., Ghilezan, S.: An approach to call-by-name delimited continuations. In: Necula, G.C.,. Wadler, P. (eds.) Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, 7–12 January 2008, pp. 383–394. ACM, January 2008
17. Howard, W.A.: The formulae-as-types notion of construction. Privately circulated notes (1969)
18. Krivine, J.-L.: Realizability in classical logic. Panoramas et synthèses **27**, 197–229 (2009). In: Interactive Models of Computation and Program Behaviour
19. Lepigre, R.: A classical realizability model for a semantical value restriction. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 476–502. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49498-1_19
20. Martin-Löf, P.: Constructive mathematics and computer programming. In: Proceedings of a Discussion Meeting of the Royal Society of London on Mathematical Logic and Programming Languages, pp. 167–184. Prentice-Hall, Inc., Upper Saddle River (1985)
21. Miquel, A.: Existential witness extraction in classical realizability and via a negative translation. Log. Methods Comput. Sci. **7**(2), 1–47 (2011)
22. Parigot, M.: Proofs of strong normalisation for second order classical natural deduction. J. Symb. Log. **62**(4), 1461–1479 (1997)
23. Vákár, M.: A framework for dependent types and effects. CoRR, abs/1512.08009 (2015)
24. Wadler, P.: Call-by-value is dual to call-by-name. In: Runciman, C., Shivers, C. (eds.) Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, 25–29 August 2003, pp. 189–201. ACM (2003)
25. Wright, A.: Simple imperative polymorphism. LISP Symbolic Comput. **8**(4), 343–356 (1995)