A sequent calculus with dependent types for classical arithmetic

Étienne Miquey

A sequent calculus with dependent types for classical arithmetic

Étienne Miquey
Équipe Galinette
INRIA, LS2N
Nantes, France
emiquey@inria.fr

Abstract

In a recent paper [11], Herbelin developed a calculus dPA$^\omega$ in which constructive proofs for the axioms of countable and dependent choices could be derived via the encoding of a proof of countable universal quantification as a stream of it components. However, the property of normalization (and therefore the one of soundness) was only conjectured. The difficulty for the proof of normalization is due to the simultaneous presence of dependent dependent types (for the constructive part of the choice), of control operators (for classical logic), of coinductive objects (to encode functions of type $\mathbb{N} \to A$ into streams $(a_0, a_1, \ldots)$) and of lazy evaluation with sharing (for these coinductive objects).

Building on previous works, we introduce in this paper a variant of dPA$^\omega$ presented as a sequent calculus. On the one hand, we take advantage of a variant of Krivine classical realizability we developed to prove the normalization of classical call-by-need [19]. On the other hand, we benefit of dL$_{\hat{\text{tp}}}$, a classical sequent calculus with dependent types in which type safety is ensured using delimited continuations together with a syntactic restriction [18]. By combining the techniques developed in these papers, we manage to define a realizability interpretation *à la* Krivine of our calculus that allows us to prove normalization and soundness.

Keywords ???

1 Introduction

1.1 Realizing AC$_\mathbb{N}$ and DC in the presence of classical logic

One the key features of Martin-Löf's type theory is the dependent sum type, which provides a strong existential elimination. In particular, it allows for a simple and constructive proof of the full axiom of choice [16], given by:

$$\begin{array}{rcl} AC_A & := & \lambda H.(\lambda x.\,\mathtt{wit}\,(Hx), \lambda x.\,\mathtt{prf}\,(Hx)) \\ & : & \forall x^A \exists y^B P(x,y) \to \exists f^{A\to B} \forall x^A P(x, f(x)) \end{array}$$

where `wit` and `prf` are the first and second projections of a strong existential quantifier.

We present here a continuation of Herbelin's works [11], who proposed a way of scaling up Martin-Löf proof to classical logic. The first idea is to restrict the dependent sum type to the fragment of *negative-elimination-free* proofs (NEF) in order to make it computationally compatible with classical logic. The second idea is to represent a countable universal quantification as an infinite conjunction. This allows to internalise into a formal system (called dPA$^\omega$) the realizability approach [2, 9] as a direct proofs-as-programs interpretation.

Informally, let us imagine that given a proof $H : \forall x^\mathbb{N} \exists y^B P(x,y)$, we could create the infinite sequence $H_\infty = (H0, H1, \ldots, Hn, \ldots)$ and select its n^{th}-element with some function `nth`. Then, one might wish that:

$$\lambda H.(\lambda n.\,\mathtt{wit}\,(\mathtt{nth}\ n\ H_\infty), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}\ n\ H_\infty))$$

could stand for a proof for $AC_\mathbb{N}$. However, even if we were effectively able to build such a term, H_∞ might still contain some classical proof. Therefore two copies of H_n might end up being different according to the contexts in which they are executed, and thus return two different witnesses (which is known to lead to logical inconsistencies [10]). This problem can be fixed by using a shared version of H_∞, that is to say:

$$\lambda H.\,\mathtt{let}\ a = H_\infty\ \mathtt{in}\ (\lambda n.\,\mathtt{wit}\,(\mathtt{nth}\ n\ a), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}\ n\ a))\,.$$

It only remains to formalize the intuition of H_∞, which is done by means of a stream $\mathtt{cofix}^0_{fn}[(Hn, f(S(n)))]$ iterated on f with parameter n, starting with 0:

$$\begin{array}{rl} AC_\mathbb{N} := \lambda H.\mathtt{let}\ a = \mathtt{cofix}^0_{fn}[(Hn, f(S(n)))] & \\ \qquad \mathtt{in}\ (\lambda n.\,\mathtt{wit}\,(\mathtt{nth}\ n\ a), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}\ n\ a))\,. & \end{array}$$

Whereas the stream is, at level of formulas, an inhabitant of a coinductively defined infinite conjunction $\nu^0_{Xn}(\exists y\, P(0,y) \wedge X(n+1))$, we cannot afford to pre-evaluate each of its components, and we thus have to use a *lazy* call-by-value evaluation discipline. However, it still might be responsible for some non-terminating reductions, all the more as classical proofs may contain backtrack.

1.2 Normalization of dPA$^\omega$

In [11], the property of normalization (on which relies the one of consistency) was only conjectured, and the proof sketch that was given turned out to be hard to formalize properly. Our first attempt to prove the normalization of dPA$^\omega$ was to derive a continuation-passing style translation, but it is very hard to obtain for dPA$^\omega$ as such. In addition to the difficulties caused by control operators and co-fixpoints, the reduction system is defined in a natural-deduction style with contextual rules where the contexts involved can be of arbitrary depth. This kind of rules are, in general and especially in this case, very difficult to translate faithfully through a continuation-passing style translation.

Rather than directly proving the normalization of dPA$^\omega$, we choose to first give an alternative presentation of the system under the form of a sequent calculus, which we call dLPA$^\omega$. Indeed, a sequent calculus presentation of a calculus is usually a good intermediate step for CPS translations [8] in that it enforces a decomposition of the reduction system into finer-grain rules. To this hand, we first handled separately the difficulties to the definition of such a calculus: on the one hand, we proved the normalization of a calculus with control operators and lazy evaluation [19]; on the other hand, we defined a classical sequent calculus with dependent types [18]. By combining the techniques developed in these frameworks, we finally manage to define dLPA$^\omega$, which we present in this paper and prove to be normalizing.

, ,
2018.

$$
\begin{array}{lll}
\textbf{Closures} & l & ::= c\tau \\
\textbf{Commands} & c & ::= \langle p \| e \rangle \\
\textbf{Proof terms} & p,q & ::= a \mid \iota_i(p) \mid (p,q) \mid (t,p) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl} \\
 & & \quad \mid \mathtt{fix}^t_{ax}[p_0 \mid p_S] \mid \mathtt{cofix}^t_{bx}[p] \mid \mu\alpha.c \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}} \\
\textbf{Proof values} & V & ::= a \mid \iota_i(V) \mid (V,V) \mid (V_t,V) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl} \\
\textbf{Terms} & t,u & ::= x \mid 0 \mid S(t) \mid \mathtt{rec}^t_{xy}[t_0 \mid t_S] \mid \lambda x.t \mid t\ u \mid \mathtt{wit}\ p \\
\textbf{Terms values} & V_t & ::= x \mid S^n(0) \mid \lambda x.t
\end{array}
$$

$$
\begin{array}{lll}
\textbf{Stores} & \tau & ::= \varepsilon \mid \tau[a := p_\tau] \mid \tau[\alpha := e] \\
\textbf{Storables} & p_\tau & ::= V \mid \mathtt{fix}^{V_t}_{ax}[p_0 \mid p_S] \mid \mathtt{cofix}^{V_t}_{bx}[p] \\
\textbf{Contexts} & e & ::= f \mid \alpha \mid \tilde{\mu}a.c\tau \\
\textbf{Forcing contexts} & f & ::= [] \mid \tilde{\mu}[a_1.c_1 \mid a_2.c_2] \mid \tilde{\mu}(a_1,a_2).c \\
 & & \quad \mid \tilde{\mu}(x,a).c \mid t \cdot e \mid p \cdot e \mid \tilde{\mu}^{=}.c \\
\textbf{Delimited continuations} & c_{\hat{\mathfrak{tp}}} & ::= \langle p_N \| e_{\hat{\mathfrak{tp}}} \rangle \mid \langle p \| \hat{\mathfrak{tp}} \rangle \\
 & e_{\hat{\mathfrak{tp}}} & ::= \tilde{\mu}a.c_{\hat{\mathfrak{tp}}}\tau \mid \tilde{\mu}[a_1.c_{\hat{\mathfrak{tp}}} \mid a_2.c'_{\hat{\mathfrak{tp}}}] \\
 & & \quad \mid \tilde{\mu}(a_1,a_2).c_{\hat{\mathfrak{tp}}} \mid \tilde{\mu}(x,a).c_{\hat{\mathfrak{tp}}}
\end{array}
$$

$$
\textbf{NEF} \quad c_N ::= \langle p_N \| e_N \rangle \qquad e_N ::= \star \mid \tilde{\mu}[a_1.c_N \mid a_2.c'_N] \mid \tilde{\mu}a.c_N\tau \mid \tilde{\mu}(a_1,a_2).c_N \mid \tilde{\mu}(x,a).c_N
$$
$$
p_N,q_N ::= a \mid \iota_i(p_N) \mid (p_N,q_N) \mid (t,p_N) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl} \mid \mathtt{fix}^t_{ax}[p_N \mid q_N] \mid \mathtt{cofix}^t_{bx}[p_N] \mid \mu\star.c_N \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}}
$$

Figure 1. The language of dLPA$^\omega$

1.3 Realizability interpretation of classical call-by-need

In then call-by-need evaluation strategy, the substitution of variable is delayed until knowing whether the argument is needed. To this end, Ariola *et. al.* [1] proposed the $\overline{\lambda}_{[lv\tau\star]}$-calculus, a variant of Curien-Herbelin's $\lambda\mu\tilde{\mu}$-calculus [6] in which substitution are stored in an explicit environment. Thanks to Danvy's methodology of semantics artifacts [7], which consists in successively refining the reduction system until getting context-free reduction rules[1], they obtain an untyped CPS translation for the $\overline{\lambda}_{[lv\tau\star]}$c-calculus. By pushing one step further this methodology, we showed in [19] how to obtain a realizability interpretation *à la* Krivine for this framework. The main idea, in contrast to usual models of Krivine realizability [13], is that realizers are defined as pairs of a term and a substitution. The adequacy of the interpretation directly provides us with a proof of normalization for the calculus, and we shall follow the same methodology here to prove the normalization of dLPA$^\omega$.

1.4 A sequent calculus with dependent types

While sequent calculi are naturally tailored to smoothly support CPS interpretations, there was no such presentation of a language with dependent types. In addition to the problem of safely combining control operators and dependent types [10], the presentation of a dependently typed language under the form of a sequent calculus is a challenge in itself. In [18], we introduced such a system, called dL$_{\hat{\mathfrak{tp}}}$, which is a call-by-value sequent calculus with classical control and dependent types. In comparison with usual type systems, a list of dependencies decorates typing derivations to ensure subject reduction. Besides, the soundness of the calculus is justified by means of a continuation-passing style translation taking the dependencies into account. The very definition of the translation enforces us to use delimited continuations in the calculus to reduce dependently typed terms. At the same time, this unveils the need for the syntactic restriction of dependencies to the *negative-elimination-free* fragment as in [11]. Additionally, we show how to relate our calculus to a similar system by Lepigre [15], whose consistency is proved by means of a realizability interpretation. In this paper, we use the same techniques, namely a list of dependencies and delimited continuations, to ensure the soundness of dLPA$^\omega$, and we follow Lepigre's interpretation of dependent types for the definition of our realizability model.

[1] That is to say reduction rules for an abstract machine for which only the term or the context needs to be analyzed.

1.5 Contributions of the paper

The main contribution of this paper is the definition of dLPA$^\omega$, a sequent calculus with classical control, dependent types, inductive and coinductive fixpoints and lazy evaluation made available thanks to the presence of stores. This calculus can be seen as a sound combination of the $\overline{\lambda}_{[lv\tau\star]}$-calculus [1, 19] and dL$_{\hat{\mathfrak{tp}}}$ [17] extended with the expressive power of dPA$^\omega$ [11]. We prove the properties of normalization and soundness thanks to a realizability interpretation *à la* Krivine, which we obtain by applying Danvy's methodology of semantic artifacts. Incidentally, dLPA$^\omega$ provides us with a direct proofs-as-programs interpretation of classical arithmetic with dependent choice as in [11].

To save some space, we only give proof sketches, complete proofs are to be found in appendices.

2 A sequent calculus with dependent types for classical arithmetic

2.1 Syntax

The language of dLPA$^\omega$ is based on the syntax of dL$_{\hat{\mathfrak{tp}}}$ [17], extended with the expressive power of dPA$^\omega$ [11] and with explicit stores as in the $\overline{\lambda}_{[lv\tau\star]}$-calculus [1]. We stick to a stratified presentation of dependent types, that is to say that we syntactically distinguish terms—that represent *mathematical objects*—from proof terms—that represent *mathematical proofs*. In particular, types and formulas are separated as well, corresponding to the same syntax as dPA$^\omega$'s formulas. Types are defined as finite types with the set of natural numbers as the sole ground type, while formulas are inductively built on atomic equalities of terms, by means of conjunctions, disjunctions, first-order quantifications, dependent products and co-inductive formulas:

$$
\begin{array}{lll}
\textbf{Types} & T,U & ::= \mathbb{N} \mid T \to U \\
\textbf{Formulas} & A,B & ::= \top \mid \bot \mid t = u \mid A \wedge B \mid A \vee B \\
 & & \quad \mid \Pi a : A.B \mid \forall x^T.A \mid \exists x^T.A \mid \nu^t_{x,f}A
\end{array}
$$

The syntax of terms is identical to the one in dPA$^\omega$, including functions $\lambda x.t$ and applications tu, as well as a recursion operator $\mathtt{rec}^t_{xy}[t_0 \mid t_S]$, so that terms represent objects in arithmetic of finite types. As for proof terms (and contexts, commands), they are now defined with all the expressiveness of dPA$^\omega$. Each constructor in the syntax of formulas is reflected by a constructor in the syntax of proofs and by the dual co-proof (*i.e.* destructor) in the syntax of evaluation contexts. Amongst other things, the syntax includes

Basic rules

$$\langle \lambda x.p \| V_t \cdot e\rangle\tau \to \langle p[V_t/x] \| e\rangle\tau$$
$$(q \in \text{NEF}) \quad \langle \lambda a.p \| q \cdot e\rangle\tau \to \langle \mu\hat{\mathfrak{tp}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{tp}}\rangle\rangle \| e\rangle\tau$$
$$(q \notin \text{NEF}) \quad \langle \lambda a.p \| q \cdot e\rangle\tau \to \langle q \| \tilde{\mu}a.\langle p \| e\rangle\rangle\tau$$
$$(e \neq e_{\hat{\mathfrak{tp}}}) \quad \langle \mu\alpha.c \| e\rangle\tau \to c\tau[\alpha := e]$$
$$\langle V \| \tilde{\mu}a.c\tau'\rangle\tau \to c\tau[a := V]\tau'$$

Elimination rules

$$\langle \iota_i(V) \| \tilde{\mu}[a_1.c_1 \mid a_2.c_2]\rangle\tau \to c_i\tau[a_i := V]$$
$$\langle (V_1, V_2) \| \tilde{\mu}(a_1, a_2).c\rangle\tau \to c\tau[a_1 := V_1][a_2 := V_2]$$
$$\langle (V_t, V) \| \tilde{\mu}(x, a).c\rangle\tau \to (c[t/x])\tau[a := V]$$
$$\langle \mathtt{refl} \| \tilde{\mu}^{=}.c\rangle\tau \to c\tau$$

Delimited continuations

$$(\text{if } c\tau \to c\tau') \quad \langle \mu\hat{\mathfrak{tp}}.c \| e\rangle\tau \to \langle \mu\hat{\mathfrak{tp}}.c \| e\rangle\tau'$$
$$\langle \mu\alpha.c \| e_{\hat{\mathfrak{tp}}}\rangle\tau \to c[e_{\hat{\mathfrak{tp}}}/\alpha]\tau$$
$$\langle \mu\hat{\mathfrak{tp}}.\langle p \| \hat{\mathfrak{tp}}\rangle \| e\rangle\tau \to \langle p \| e\rangle\tau$$

Call-by-value

$$(a \text{ fresh}) \quad \langle \iota_i(p) \| e\rangle\tau \to \langle p \| \tilde{\mu}a.\langle \iota_i(a) \| e\rangle\rangle\tau$$
$$(a_1, a_2 \text{ fresh}) \quad \langle (p_1, p_2) \| e\rangle\tau \to \langle p_1 \| \tilde{\mu}a_1.\langle p_2 \| \tilde{\mu}a_2.\langle (a_1, a_2) \| e\rangle\rangle\rangle\tau$$
$$(a \text{ fresh}) \quad \langle (V_t, p) \| e\rangle\tau \to \langle p \| \tilde{\mu}a.\langle (V_t, a) \| e\rangle\rangle\tau$$

Laziness

$$(a \text{ fresh}) \quad \langle \mathtt{cofix}^{V_t}_{bx}[p] \| e\rangle\tau \to \langle a \| e\rangle\tau[a := \mathtt{cofix}^{V_t}_{bx}[p]]$$
$$(a \text{ fresh}) \langle \mathtt{fix}^{V_t}_{bx}[p_0 \mid p_S] \| e\rangle\tau \to \langle a \| e\rangle\tau[a := \mathtt{fix}^{V_t}_{bx}[p_0 \mid p_S]]$$

Lookup

$$\langle V \| \alpha\rangle\tau[\alpha := e]\tau' \to \langle V \| e\rangle\tau[\alpha := e]\tau'$$
$$\langle a \| f\rangle\tau[a := V]\tau' \to \langle V \| a\rangle\tau[a := V]\tau'$$
$$(b' \text{ fresh}) \quad \langle a \| f\rangle\tau[a := \mathtt{cofix}^{V_t}_{bx}[p]]\tau' \to \langle p[V_t/x][b'/b] \| \tilde{\mu}a.\langle a \| f\rangle\tau'\rangle\tau[b' := \lambda y.\mathtt{cofix}^{y}_{bx}[p]]$$
$$\langle a \| f\rangle\tau[a := \mathtt{fix}^{0}_{bx}[p_0 \mid p_S]]\tau' \to \langle p_0 \| \tilde{\mu}a.\langle a \| f\rangle\tau'\rangle\tau$$
$$(b' \text{ fresh}) \quad \langle a \| f\rangle\tau[a := \mathtt{fix}^{S(t)}_{bx}[p_0 \mid p_S]]\tau' \to \langle p_S[t/x][b'/b] \| \tilde{\mu}a.\langle a \| f\rangle\tau'\rangle\tau[b' := \mathtt{fix}^{t}_{bx}[p_0 \mid p_S]]$$

Terms

$$(\text{if } t \longrightarrow_\beta t') \quad T[t]\tau \to T[t']\tau$$
$$(\forall \alpha, \langle p \| \alpha\rangle\tau \to \langle (t, p') \| \alpha\rangle\tau) \quad T[\mathtt{wit}\ p]\tau \longrightarrow_\beta T[t]$$
$$(\lambda x.t)V_t \longrightarrow_\beta t[V_t/x]$$
$$\mathtt{rec}^{0}_{xy}[t_0 \mid t_S] \longrightarrow_\beta t_0$$
$$\mathtt{rec}^{S(u)}_{xy}[t_0 \mid t_S] \longrightarrow_\beta t_S[u/x][\mathtt{rec}^{u}_{xy}[t_0 \mid t_S]/y]$$

where:

$$C_t[\ ] ::= \langle ([\ ], p) \| e\rangle \mid \langle \mathtt{fix}^{[\,]}_{ax}[p_0 \mid p_S] \| e\rangle \mid \langle \mathtt{cofix}^{[\,]}_{bx}[p] \| e\rangle \mid \langle \lambda x.p \| [\ ] \cdot e\rangle$$
$$T[\ ] ::= C_t[\ ] \mid T[[\ ]u] \mid T[\mathtt{rec}^{[\,]}_{xy}[t_0 \mid t_S]]$$

Figure 2. Reduction rules of dLPA$^\omega$

pairs (t,p) where t is a term and p a proof, which inhabit the dependent sum type $\exists x^T.A$; dual co-pairs $\tilde{\mu}(x,a).c$ which bind the (term and proof) variables x and a in the command c; functions $\lambda x.p$ (and dual stacks $t \cdot e$ where e is a context whose type might be dependent in t) inhabiting the type $\forall x^T.A$; functions $\lambda a.p$ (and dual stacks $q \cdot e$, where e is a context whose type might be dependent in q) which inhabit the dependent product type $\Pi a : A.B$; a proof term `refl` which is the proof of atomic equalities $t = t$ and the dual destructor $\tilde{\mu}^{=}.c$ which allows to type the command c modulo an equality of terms; operators $\mathtt{fix}^t_{ax}[p_0 \mid p_S]$ and $\mathtt{cofix}^t_{bx}[p]$, as in dPA$^\omega$, for inductive and coinductive reasoning; delimited continuations through proofs $\mu\hat{\mathfrak{tp}}..c_{\mathfrak{tp}}$ and the context $\hat{\mathfrak{tp}}$; a distinguished context [] of type $\bot$, which allows us to reason ex-falso.

As in dL$_{\hat{\mathfrak{tp}}}$, the syntax of NEF proofs, contexts and commands is defined as a restriction of the previous syntax. Technically, they are defined (modulo α-conversion) with only one distinguished context variable $\star$ (and consequently only one binder $\mu\star.c$) and without stacks of the shape $t \cdot e$ or $q \cdot e$ (to avoid applications). Intuitively, one can understand NEF proofs as the proofs that cannot drop their continuation[2]. The commands $c_{\hat{\mathfrak{tp}}}$ within delimited continuations are again defined as commands of the shape $\langle p \| \hat{\mathfrak{tp}}\rangle$ or formed by a NEF proof and a context of the shape $\tilde{\mu}a.c_{\hat{\mathfrak{tp}}}\tau$, $\tilde{\mu}[a_1.c_{\hat{\mathfrak{tp}}} | a_2.c'_{\hat{\mathfrak{tp}}}]$, $\tilde{\mu}(a_1,a_2).c_{\hat{\mathfrak{tp}}}$ or $\tilde{\mu}(x,a).c_{\hat{\mathfrak{tp}}}$.

[2]See [18] for further details

We adopt a call-by-value evaluation strategy except for fixpoint operators[3] which are evaluated in a lazy way. To this purpose, we use stores in the spirit of the $\overline{\lambda}_{[lv\tau\star]}$-calculus, which are thus defined as lists of bindings of the shape $[a := p]$ where p is a value or a (co-)fixpoint, and of bindings of the shape $[\alpha := e]$ where e is any context. We assume that each variable occurs at most once in a store τ, therefore we reason up to α-reduction and we assume the capability of generating fresh names. Apart from evaluation contexts of the shape $\tilde{\mu}a.c$ and co-variables α, all the contexts are forcing contexts since they eagerly require a value to be reduced. The resulting language is given in Figure 1.

2.2 Reduction rules

Regarding the reduction system of dLPA$^\omega$, which is given in Figure 2, there is not much to say. The basic rules are those of the call-by-value $\lambda\mu\tilde{\mu}$-calculus and of dL$_{\hat{\mathfrak{tp}}}$. The rules for delimited continuations are exactly the same as in dL$_{\hat{\mathfrak{tp}}}$, except that we have to prevent $\hat{\mathfrak{tp}}$ from being caught and stored by a proof $\mu\alpha.c$. We thus distinguish two rules for commands of the shape $\langle \mu\alpha.c \| e\rangle$, depending on whether e is of the shape $e_{\hat{\mathfrak{tp}}}$ or not. In the former case, we perform the substitution $[e_{\hat{\mathfrak{tp}}}/\alpha]$, which is linear since $\mu\alpha.c$ is necessarily NEF. We should also mention in passing that we abuse the syntax in every other rules, since e should actually refer to e

[3]To highlight the duality between inductive and coinductive fixpoints, we evaluate both in a lazy way. Even though this is not indispensable for inductive fixpoints, we find this approach more natural in that we can treat both in a similar way in the small-step reduction system and thus through the realizability interpretation.

Regular types

$$\frac{\Gamma \vdash^\sigma p : A \quad \Gamma \vdash^\sigma e : B^{\perp\!\!\!\perp} \quad \sigma(A) = \sigma(B)}{\Gamma \vdash^\sigma \langle p \| e \rangle} \text{(Cut)} \qquad \frac{\Gamma, \Gamma' \vdash^{\sigma\sigma'} c \quad \Gamma \vdash^\sigma \tau : (\Gamma'; \sigma')}{\Gamma \vdash c\tau} (l) \qquad \frac{\Gamma \vdash^\sigma \tau : (\Gamma'; \sigma') \quad \Gamma, \Gamma' \vdash^{\sigma\sigma'} p : A}{\Gamma \vdash^\sigma \tau[a := p] : (\Gamma', a : A; \sigma'\{a|p\})} (\tau_p)$$

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash^\sigma a : A} (\text{Ax}_r) \qquad \frac{(\alpha : A^{\perp\!\!\!\perp}) \in \Gamma}{\Gamma \vdash^\sigma \alpha : A^{\perp\!\!\!\perp}} (\text{Ax}_l) \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash^\sigma c}{\Gamma \vdash^\sigma \mu\alpha.c : A} (\mu) \qquad \frac{\Gamma \vdash^\sigma \tau : (\Gamma'; \sigma') \quad \Gamma, \Gamma' \vdash^{\sigma\sigma'} \alpha : A^{\perp\!\!\!\perp}}{\Gamma \vdash^\sigma \tau[\alpha := e] : (\Gamma', \alpha : A^{\perp\!\!\!\perp}; \sigma')} (\tau_e)$$

$$\frac{\Gamma, a : A \vdash^\sigma c\tau}{\Gamma \vdash^\sigma \tilde{\mu}a.c\tau : A^{\perp\!\!\!\perp}} (\tilde{\mu}) \qquad \frac{\Gamma \vdash^\sigma p_1 : A \quad \Gamma \vdash^\sigma p_2 : B}{\Gamma \vdash^\sigma (p_1, p_2) : A \wedge B} (\wedge_r) \qquad \frac{\Gamma, a_1 : A_1, a_2 : A_2 \vdash^\sigma c}{\Gamma \vdash^\sigma \tilde{\mu}(a_1, a_2).c : (A_1 \wedge A_2)^{\perp\!\!\!\perp}} (\wedge_l) \qquad \frac{\Gamma \vdash^\sigma p : A_i}{\Gamma \vdash^\sigma \iota_i(p) : A_1 \vee A_2} (\vee_r)$$

$$\frac{\Gamma, a_1 : A_1 \vdash^\sigma c_1 \quad \Gamma, a_2 : A_2 \vdash^\sigma c_2}{\Gamma \vdash^\sigma \tilde{\mu}[a_1.c_1 \mid a_2.c_2] : (A_1 \vee A_2)^{\perp\!\!\!\perp}} (\vee_l) \qquad \frac{\Gamma \vdash^\sigma p : A[t/x] \quad \Gamma \vdash^\sigma t : T}{\Gamma \vdash^\sigma (t, p) : \exists x^T.A} (\exists_r) \qquad \frac{\Gamma, x : T, a : A \vdash^\sigma c}{\Gamma \vdash^\sigma \tilde{\mu}(x, a).c : (\exists x^T.A)^{\perp\!\!\!\perp}} (\exists_l)$$

$$\frac{\Gamma, x : T \vdash^\sigma p : A}{\Gamma \vdash^\sigma \lambda x.p : \forall x^T.A} (\forall_r) \qquad \frac{\Gamma \vdash^\sigma t : T \quad \Gamma \vdash^\sigma e : A[t/x]^{\perp\!\!\!\perp}}{\Gamma \vdash^\sigma t \cdot e : (\forall x^T.A)^{\perp\!\!\!\perp}} (\forall_l) \qquad \frac{\Gamma \vdash^\sigma t : \mathbb{N}}{\Gamma \vdash^\sigma \texttt{refl} : t = t}\ \texttt{refl} \qquad \frac{\Gamma \vdash^\sigma p : A \quad \Gamma \vdash^\sigma e : A[u/t]}{\Gamma \vdash^\sigma \tilde{\mu}^{=}.\langle p \| e \rangle : (t = u)^{\perp\!\!\!\perp}} (=_l)$$

$$\frac{\Gamma, a : A \vdash^\sigma p : B}{\Gamma \vdash^\sigma \lambda a.p : \Pi a : A.B} (\to_r) \qquad \frac{\Gamma \vdash^\sigma q : A \quad \Gamma \vdash^\sigma e : B[q/a]^{\perp\!\!\!\perp} \quad \text{if } q \notin \text{NEF then } a \notin A}{\Gamma \vdash^\sigma q \cdot e : (\Pi a : A.B)^{\perp\!\!\!\perp}} (\to_l)$$

$$\frac{\Gamma \vdash^\sigma p : A \quad A \equiv B}{\Gamma \vdash^\sigma p : B} (\equiv_r) \qquad \frac{\Gamma \vdash^\sigma e : A^{\perp\!\!\!\perp} \quad A \equiv B}{\Gamma \vdash^\sigma e : B^{\perp\!\!\!\perp}} (\equiv_l) \qquad \frac{\Gamma \vdash^\sigma t : \mathbb{N} \quad \Gamma \vdash^\sigma p_0 : A[0/x] \quad \Gamma, x : T, a : A \vdash^\sigma p_S : A[S(x)/x]}{\Gamma \vdash^\sigma \texttt{fix}^t_{ax}[p_0 \mid p_S] : A[t/x]}\ \texttt{(ind)}$$

$$\frac{}{\Gamma \vdash^\sigma [] : \bot^{\perp\!\!\!\perp}} \bot \qquad \frac{\Gamma \vdash^\sigma t : T \quad \Gamma, f : T \to \mathbb{N}, x : T, b : \forall y^T.f(y) = 0 \vdash^\sigma p : A \quad f \text{ positive in } A}{\Gamma \vdash^\sigma \texttt{cofix}^t_{bx}[p] : \nu^t_{fx}A}\ \texttt{(cofix)}$$

Dependent mode

$$\frac{\Gamma, \Gamma' \vdash_d c_{\hat{\mathfrak{p}}}; \sigma\sigma' \quad \Gamma \vdash^\sigma \tau : (\Gamma'; \sigma')}{\Gamma \vdash_d c_{\hat{\mathfrak{p}}}\tau; \sigma} (l_d) \qquad \frac{\Gamma, \Gamma' \vdash^\sigma p : A \quad \Gamma, \hat{\mathfrak{p}} : B^{\perp\!\!\!\perp}, \Gamma' \vdash_d e : A^{\perp\!\!\!\perp}; \sigma\{\cdot|p\}}{\Gamma, \hat{\mathfrak{p}} : B^{\perp\!\!\!\perp}, \Gamma' \vdash_d \langle p \| e \rangle; \sigma} (\text{Cut}_d)$$

$$\frac{\Gamma, \hat{\mathfrak{p}} : A^{\perp\!\!\!\perp} \vdash_d c_{\hat{\mathfrak{p}}}; \sigma}{\Gamma \vdash^\sigma \mu\hat{\mathfrak{p}}.c_{\hat{\mathfrak{p}}} : A} (\mu\hat{\mathfrak{p}}) \qquad \frac{\sigma(A) = \sigma(B)}{\Gamma, \hat{\mathfrak{p}} : A^{\perp\!\!\!\perp}, \Gamma' \vdash_d \hat{\mathfrak{p}} : B^{\perp\!\!\!\perp}; \sigma\{\cdot|p\}} (\hat{\mathfrak{p}}) \qquad \frac{\Gamma, a_i : A_i \vdash_d c^i_{\hat{\mathfrak{p}}}; \sigma\{\iota_i(a_i)|p_N\}) \quad \forall i \in \{1,2\}}{\Gamma \vdash_d \tilde{\mu}[a_1.c^1_{\hat{\mathfrak{p}}} \mid a_2.c^2_{\hat{\mathfrak{p}}}] : (A_1 \vee A_2)^{\perp\!\!\!\perp}; \sigma\{\cdot|p_N\}} (\vee^d_l)$$

$$\frac{\Gamma, a : A \vdash_d c_{\hat{\mathfrak{p}}}\tau'; \sigma\{a|p_N\}}{\Gamma \vdash_d \tilde{\mu}a.c_{\hat{\mathfrak{p}}}\tau' : A^{\perp\!\!\!\perp}; \sigma\{\cdot|p_N\}} (\tilde{\mu}_d) \qquad \frac{\Gamma, x : T, a : A \vdash_d c_{\hat{\mathfrak{p}}}; \sigma\{(x,a)|p_N\}}{\Gamma \vdash_d \tilde{\mu}(x,a).c_{\hat{\mathfrak{p}}} : (\exists x^T A)^{\perp\!\!\!\perp}; \sigma\{\cdot|p_N\}} (\exists^d_l) \qquad \frac{\Gamma, a_1 : A_1, a_2 : A_2 \vdash_d c_{\hat{\mathfrak{p}}}; \sigma\{(a_1,a_2)|p_N\}}{\Gamma \vdash_d \tilde{\mu}(a_1,a_2).c_{\hat{\mathfrak{p}}} : (A_1 \wedge A_2)^{\perp\!\!\!\perp}; \sigma\{\cdot|p_N\}} (\wedge^d_l)$$

Terms

$$\frac{}{\Gamma \vdash^\sigma 0 : \mathbb{N}} (0) \qquad \frac{\Gamma \vdash^\sigma t : \mathbb{N}}{\Gamma \vdash^\sigma S(t) : \mathbb{N}} (S) \qquad \frac{(x : T) \in \Gamma}{\Gamma \vdash^\sigma x : T} (\text{Ax}_t) \qquad \frac{\Gamma, x : U \vdash^\sigma t : T}{\Gamma \vdash^\sigma \lambda x.t : U \to T} (\lambda)$$

$$\frac{\Gamma \vdash^\sigma t : U \to T \quad \Gamma \vdash^\sigma u : U}{\Gamma \vdash^\sigma t\ u : T} (@) \qquad \frac{\Gamma \vdash^\sigma t : \mathbb{N} \quad \Gamma \vdash^\sigma t_0 : U \quad \Gamma, x : \mathbb{N}, y : U \vdash^\sigma t_S : U}{\Gamma \vdash^\sigma \texttt{rec}^t_{xy}[t_0 \mid t_S] : U}\ \texttt{(rec)} \qquad \frac{\Gamma \vdash^\sigma p : \exists x^T.A \quad p \text{ NEF}}{\Gamma \vdash^\sigma \texttt{wit}\ p : T}\ \texttt{(wit)}$$

Figure 3. Type system for dLPA$^\omega$

or $e_{\hat{\mathfrak{p}}}$ (or the reduction of delimited continuations would be stuck). Elimination rules correspond to commands where the proof is a constructor (say of pairs) applied to values, and where the context is the matching destructor. Call-by-value rules correspond to (ς) rule of Wadler's sequent calculus [22]. The next rules express the fact that (co-)fixpoints are lazily stored, and reduced only if their value is eagerly demanded by a forcing context. Lastly, terms are reduced according to the usual β-reduction, with the operator rec computing with the usual recursion rules. It is worth noting that the stratified presentation allows to define the reduction of terms as external: within proofs and contexts, terms are reduced in place. Consequently, as in dL$_{\hat{\mathfrak{p}}}$ the very same happen for NEF proofs embedded within terms. Computationally speaking, this corresponds indeed to the intuition that terms are reduced on an external device.

2.3 Typing rules

As often in Martin-Löf's intensional type theory, formulas are considered up to equational theory on terms. We denote by $A \equiv B$ the reflexive-transitive-symmetric closure of the relation $\triangleright$ induced by the reduction of terms and NEF proofs as follows:

$$\begin{array}{lcll} A[t] & \triangleright & A[t'] & \text{whenever} \quad t \to_\beta t' \\ A[p] & \triangleright & A[q] & \text{whenever} \quad \forall \alpha\, (\langle p \| \alpha \rangle \to \langle q \| \alpha \rangle) \end{array}$$

in addition to the reduction rules for equality and for coinductive formulas:

$$\begin{array}{ll} 0 = S(t) \triangleright \bot & S(t) = S(u) \triangleright t = u \\ S(t) = 0 \triangleright \bot & \nu^t_{fx}A \triangleright A[t/x][\nu^y_{fx}A/f(y) = 0] \end{array}$$

We work with one-sided sequents[4] where typing contexts are defined by:

$$\Gamma,\Gamma' \quad ::= \quad \varepsilon \mid \Gamma, x : T \mid \Gamma, a : A \mid \Gamma, \alpha : A^{\perp\!\!\!\perp} \mid \Gamma, \hat{\mathfrak{p}} : A^{\perp\!\!\!\perp}.$$

using the notation $\alpha : A^{\perp\!\!\!\perp}$ for an assumption of the refutation of A. This allows us to mix hypotheses over terms, proofs and contexts while keeping track of the order in which they are added (which is necessary because of the dependencies). We assume that a variable occurs at most once in a typing context.

We define nine syntactic kinds of typing judgments: six in regular mode, that we write $\Gamma \vdash^{\sigma} J$, and three more for the dependent mode, that we write $\Gamma \vdash_d J; \sigma$. In each case, σ is a list of dependencies—we explain the presence of a list of dependencies in each case thereafter—, which are defined from the following grammar:

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

The substitution on formulas according to a list of dependencies σ is defined by:

$$\varepsilon(A) \triangleq \{A\} \qquad \sigma\{p|q\}(A) \triangleq \begin{cases} \sigma(A[q/p]) & \text{if } q \in \text{NEF} \\ \sigma(A) & \text{otherwise} \end{cases}$$

Because the language of proof terms now include constructors for pairs, injections, etc, the notation $A[q/p]$ does not refer to usual substitutions properly speaking: p can be a pattern (for instance (a_1, a_2)) and not only a variable.

We shall attract the reader's attention to the fact that, unlike in $\text{dL}_{\hat{\mathfrak{p}}}$, all typing judgments include a list of dependencies. As in the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, when a proof or a context is caught by a binder, say V and $\tilde{\mu}a$, the substitution $[V/a]$ is not performed but rather put in the store: $\tau[a := V]$. This forces us to slightly change the rules from $\text{dL}_{\hat{\mathfrak{p}}}$. Indeed, consider for instance the reduction of a dependent function $\lambda a.p$ (of type $\Pi a : A.B$) applied to a stack $V \cdot e$[5]:

$$\begin{aligned}\langle \lambda a.p \| V \cdot e\rangle\tau &\to \langle \mu\hat{\mathfrak{p}}.\langle V \| \tilde{\mu}a.\langle p \| \hat{\mathfrak{p}}\rangle\rangle \| e\rangle\tau \\ &\to \langle \mu\hat{\mathfrak{p}}.\langle p \| \hat{\mathfrak{p}}\rangle \| e\rangle\tau[a := V] \to \langle p \| e\rangle\tau[a := V]\end{aligned}$$

In $\text{dL}_{\hat{\mathfrak{p}}}$, the reduced command is $\langle p[V/a] \| e\rangle$, which is typed with the (Cut) rule over the formula $B[V/a]$. In the present case, p still contains the variable a, whence his type is still $B[a]$, whereas the type of e is $B[V]$. We thus need to compensate the missing substitution.

We are mostly left with two choices. Either we mimic the substitution in the type system, which would amount to the following typing rule:

$$\frac{\Gamma,\Gamma' \vdash \tau(c) \quad \Gamma \vdash \tau : \Gamma'}{\Gamma \vdash c\tau}$$

where:		
$\tau[\alpha := e](c) \triangleq \tau(c)$	$\tau[a := p_N](c) \triangleq \tau(c[p_N/a])$	$(p \in \text{NEF})$
	$\tau[a := p](c) \triangleq \tau(c)$	$(p \notin \text{NEF})$

Or we type stores in the spirit of the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, and we carry along the derivations all the bindings susceptible to be used in types, which constitutes again a list of dependencies.

The former solution has the advantage of solving the problem before typing the command, but it has the flaw of performing computations which would not occur in the reduction system. For instance, the substitution $\tau(c)$ could duplicate co-fixpoints (and their typing derivations), which would never happen in the calculus. That is the reason why we privilege the other solution, which is closer to the calculus in our opinion. Yet, it has the inconvenient that if forces us to carry a list of dependencies even in regular mode. Since this list is fixed (it does not evolve in the derivation except when stores occur), we differentiate the denotation of regular typing judgments, written $\Gamma \vdash^{\sigma} J$, from the one judgments in dependent mode, which we write $\Gamma \vdash_d J; \sigma$ to highlight that σ grows along derivations. The type system we obtain is given in Figure 3.

2.4 Subject reduction

It only remains to prove that typing is preserved along reduction. As for the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, the proof is simplified by the fact that substitutions are not performed (except for terms), which keeps us from proving the safety of the corresponding substitutions. Yet, we first need to prove some technical lemmas about dependencies. To this aim, we define a relation $\sigma \Rrightarrow \sigma'$ between lists of dependencies, which expresses the fact that any typing derivation obtained with σ could be obtained as well as with σ':

$$\sigma \Rrightarrow \sigma' \triangleq \sigma(A) = \sigma(B) \Rightarrow \sigma'(A) = \sigma'(B) \qquad \text{(for any } A, B)$$

Proposition 2.1 (Dependencies weakening). *If σ, σ' are two dependencies list such that $\sigma \Rrightarrow \sigma'$, then any derivation using σ can be one using σ' instead. In other words, the following rules are admissible:*

$$\frac{\Gamma \vdash^{\sigma} J}{\Gamma \vdash^{\sigma'} J}\ (w) \qquad\qquad \frac{\Gamma \vdash_d J; \sigma}{\Gamma \vdash_d J; \sigma'}\ (w_d)$$

We also need a simple lemma about stores to simplify the proof of subject reduction:

Lemma 2.2. *The following rule is admissible:*

$$\frac{\Gamma \vdash^{\sigma} \tau_0 : (\Gamma_0; \sigma_0) \quad \Gamma, \Gamma_0 \vdash^{\sigma\sigma_0} \tau_1 : (\Gamma_1; \sigma_1)}{\Gamma \vdash^{\sigma} \tau_0\tau_1 : (\Gamma_0, \Gamma_1; \sigma_0, \sigma_1)}\ (\tau\tau')$$

Proof. By induction on the structure of τ_1. □

Lemma 2.3 (Safe term substitution). *If $\Gamma \vdash^{\sigma} t : T$ then for any conclusion J for typing proofs, contexts, terms, etc; the following holds:*

1. *If $\Gamma, x : T, \Gamma' \vdash^{\sigma} J$ then $\Gamma, \Gamma'[t/x] \vdash^{\sigma[t/x]} J[t/x]$.*
2. *If $\Gamma, x : T, \Gamma' \vdash_d J; \sigma$ then $\Gamma, \Gamma'[t/x] \vdash_d J[t/x]; \sigma[t/x]$.*

Proof. By induction on typing rules. □

Theorem 2.4 (Subject reduction). *For any context Γ and any closures $c\tau$ and $c'\tau'$ such that $c\tau \to c'\tau'$, we have:*

1. If $\Gamma \vdash c\tau$ then $\Gamma \vdash c'\tau'$. *2. If $\Gamma \vdash_d c\tau; \varepsilon$ then $\Gamma \vdash_d c'\tau'; \varepsilon$.*

Proof. The proof follows the usual proof of subject reduction, by induction on the reduction $c\tau \to c'\tau'$. The complete proof is given in Appendix A. □

[4] This is essentially an aesthetic choice, which we hope to ease the readability of sequents. On top of that, it avoids us to deal with unified contexts $\Gamma \cup \Delta$ (see ??) as we would have done with two-sided sequents.

[5] We refer the reader to [18] for detailed explanations on this rule.

$$\frac{\Gamma \vdash p : \exists x^T.A \quad \Gamma, x : T, a : A \vdash q : B[(x,a)/\bullet] \quad p \notin \textsc{nef} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q : B[p/\bullet]}\ (\mathtt{dest}) \qquad \frac{\Gamma \vdash p : \bot}{\Gamma \vdash \mathtt{exfalso}\ p : B}\ (\bot) \qquad \frac{\Gamma, a : A \vdash q : B[a/\bullet] \quad p \notin \textsc{nef} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{let}\ a = p\ \mathtt{in}\ q : B[p/\bullet]}\ (\mathtt{let})$$

$$\frac{\Gamma \vdash p : A_1 \wedge A_2 \quad \Gamma, a_1 : A_1, a_2 : A_2 \vdash q : B[(a_1,a_2)/\bullet] \quad p \notin \textsc{nef} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ q : B[p/\bullet]}\ (\mathtt{split}) \qquad \frac{\Gamma \vdash p : A_1 \wedge A_2}{\Gamma \vdash \pi_i(p) : A_i}\ (\wedge_E^i) \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash p : A}{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash \mathtt{throw}\ \alpha\, p : B}\ (\mathtt{throw})$$

$$\frac{\Gamma \vdash p : A_1 \vee A_2 \quad \Gamma, a_i : A_i \vdash q : B[\iota_i(a)_i/\bullet] \quad \text{for } i = 1,2 \quad p \notin \textsc{nef} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2] : B[p/\bullet]}\ (\mathtt{case}) \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash p : A}{\Gamma \vdash \mathtt{catch}_\alpha\ p : A}\ (\mathtt{catch}) \qquad \frac{\Gamma \vdash p : \exists x^T.A(x)}{\Gamma \vdash \mathtt{prf}\ p : A(\mathtt{wit}\ p)}\ (\mathtt{prf})$$

Figure 4. Typing rules of dPA$^\omega$

2.5 Natural deduction as macros

We can recover the usual proof terms for elimination rules in natural deduction systems, and in particular the ones from dPA$^\omega$, by defining them as macros in our language. The definitions are straightforward, using delimited continuations for `let ... in` and the constructors over NEF proofs which might be dependently typed:

$$\begin{array}{rcl}
\mathtt{let}\ a = p\ \mathtt{in}\ q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu} a.\langle q \| \alpha_p \rangle\rangle \\
\mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}(a_1,a_2).\langle q \| \alpha_p \rangle\rangle \\
\mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2] & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}[a_1.\langle p_1 \| \alpha_p\rangle | a_2.\langle p_2 \| \alpha_p\rangle]\rangle \\
\mathtt{dest}\ p\ \mathtt{as}\ (a,x)\ \mathtt{in}\ q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}(x,a).\langle q \| \alpha_p \rangle\rangle \\
\mathtt{prf}\ p & \triangleq & \mu\hat{\mathfrak{tp}}.\langle p \| \tilde{\mu}(x,a).\langle a \| \hat{\mathfrak{tp}} \rangle\rangle
\end{array}$$

$$\begin{array}{rcl|rcl}
\mathtt{subst}\ p\ q & \triangleq & \mu\alpha.\langle p \| \tilde{\mu}^{=}.\langle q \| \alpha\rangle\rangle & \mathtt{catch}_\alpha\ p & \triangleq & \mu\alpha.\langle p \| \alpha\rangle \\
\mathtt{exfalso}\ p & \triangleq & \mu\alpha.\langle p \| [] \rangle & \mathtt{throw}\ \alpha\ p & \triangleq & \mu_.\langle p \| \alpha\rangle
\end{array}$$

where $\alpha_p = \hat{\mathfrak{tp}}$ if p is NEF and $\alpha_p = \alpha$ otherwise.

Proposition 2.5 (Natural deduction). *The typing rules from dPA$^\omega$, given in Figure 4, are admissible.*

Proof. Straightforward derivations, see the Appendix B for the cases `prf` $p\ q$ and `subst` $p\ q$. □

One can even check that the reduction rules in dLPA$^\omega$ for these proofs almost mimic the ones of dPA$^\omega$. To be more precise, the rules of dLPA$^\omega$ do not allow to simulate each rule of dPA$^\omega$, due to the head-reduction strategy, amongst other things. Nonetheless, up to a few details the reduction of a command in dLPA$^\omega$ follows one particular reduction path of the corresponding proof in dPA$^\omega$, or in other word, one reduction strategy.

The main result is that using the macros, the same proof terms are suitable for countable and dependent choice [11]. We do not state it here, but following the approach of [11], we could also extend dLPA$^\omega$ to obtain a proof for the axiom of bar induction.

Theorem 2.6 (Countable choice [11]). *We have:*

$$\begin{array}{rcl}
AC_{\mathbb{N}} & := & \lambda H.\mathtt{let}\ a = \mathtt{cofix}^0_{bn}[(Hn, b(S(n))] \\
& & \quad \mathtt{in}\ (\lambda n.\mathtt{wit}\ (\mathtt{nth}_n\ a), \lambda n.\mathtt{prf}\ (\mathtt{nth}_n\ a)) \\
& : & \forall x^{\mathbb{N}} \exists y^T P(x,y) \to \exists f^{\mathbb{N}\to T} \forall x^{\mathbb{N}} P(x, f(x))
\end{array}$$

where $\mathtt{nth}_n\ a := \pi_1(\mathtt{fix}^n_{x,c}[a \mid \pi_2(c)])$.

Proof. See Figure 6 in the Appendix C. □

Theorem 2.7 (Dependent choice [11]). *We have:*

$$\begin{array}{rcl}
DC & := & \lambda H.\lambda x_0.\mathtt{let}\ a = (x_0, \mathtt{cofix}^0_{bn}[d_n]) fsix \\
& & \quad \mathtt{in}\ (\lambda n.\mathtt{wit}\ (\mathtt{nth}_n\ a), (\mathtt{refl}, \lambda n.\pi_1(\mathtt{prf}\ (\mathtt{prf}\ (\mathtt{nth}_n\ a)))))) \\
& : & \forall x^T.\exists y^T.P(x,y) \to \\
& & \quad \forall x_0^T.\exists f \in T^{\mathbb{N}}.(f(0) = x_0 \wedge \forall n^{\mathbb{N}}.P(f(n), f(s(n))))
\end{array}$$

where $d_n := \mathtt{dest}\ Hn\ \mathtt{as}\ (y,c)\ \mathtt{in}\ (y,(c,b\,y)))$
and $\mathtt{nth}_n\ a := \mathtt{fix}^n_{x,d}[a \mid (\mathtt{wit}\ (\mathtt{prf}\ d), \pi_2(\mathtt{prf}\ (\mathtt{prf}\ (d))))]$.

3 Small-step calculus

As for the $\overline{\lambda}_{[lv\tau\star]}$-calculus [1, 19], we follow here Danvy's methodology of semantic artifacts [1, 7] to obtain a realizability interpretation. We first decompose the reduction system of dLPA$^\omega$ into small-step reduction rules, that we denote by $\rightsquigarrow_s$. This requires a refinement and an extension of the syntax, that we shall now present. To keep us from boring the reader stiff with new (huge) tables for the syntax, typing rules and so forth, we will introduce them step by step. We hope it will help the reader to convince herself of the necessity and of the somewhat naturality of these extensions.

3.1 Values

First of all, we need to refine the syntax to distinguish between strong and weak values in the syntax of proof terms. As in the $\overline{\lambda}_{[lv\tau\star]}$-calculus, this refinement is induced by the computational behavior of the calculus: weak values are the ones which are stored by $\tilde{\mu}$ binders, but which are not values enough to be eliminated in front of a forcing context, that is to say variables. Indeed, if we observe the reduction system, we see that in front of a forcing context f, a variable leads a search through the store for a "stronger" value, which could incidentally provoke the evaluation of some fixpoints. On the other hand, strong values are the ones which can be reduced in front of the matching forcing context, that is to say functions, `refl`, pairs of (weak) values, injections or dependent pairs:

Weak values	$V ::= a \mid v$
Strong values	$v ::= \iota_i(V) \mid (V,V) \mid (V_t,V) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl}$

This allows us to distinguish commands of the shape $\langle v \| f \rangle \tau$, where the forcing context (and next the strong value) are examined to determine whether the command reduces or not; from commands of the shape $\langle a \| f \rangle \tau$ where the focus is put on the variable a, which leads to a lookup for the associated proof in the store.

3.2 Terms

Next, we need to explicit the reduction of terms. To this purpose, we include a machinery to evaluate terms in a way which resemble the evaluation of proofs. In particular, we define new commands which we write $\langle t \| \pi \rangle$ where t is a term and π is a context for terms (or co-term). Co-terms are either of the shape $\tilde{\mu}x.c$ or stacks ot the shape $u \cdot \pi$. These constructions are the usual ones of the $\lambda\mu\tilde{\mu}$-calculus (which are also the ones for proofs). We also extend the definitions of commands with delimited continuations to include

the corresponding commands for terms:

Commands	$c ::= \langle p \| e\rangle \mid \langle t \| \pi\rangle$	$c_{\hat{\mathfrak{tp}}} ::= \cdots \mid \langle t \| \pi_{\hat{\mathfrak{tp}}}\rangle$
Co-terms	$\pi ::= t \cdot \pi \mid \tilde{\mu}x.c$	$\pi_{\hat{\mathfrak{tp}}} ::= t \cdot \pi_{\hat{\mathfrak{tp}}} \mid \tilde{\mu}x.c_{\hat{\mathfrak{tp}}}$

We give typing rules for these new constructions, which are the usual rules for typing contexts in the $\lambda\mu\tilde{\mu}$-calculus:

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash \pi : U^{\perp\!\!\!\perp}}{\Gamma \vdash t \cdot \pi : (T \to U)^{\perp\!\!\!\perp}}\ (\to_l) \qquad \frac{c : (\Gamma, x : T)}{\Gamma \vdash \tilde{\mu}x.c : T^{\perp\!\!\!\perp}}\ (\tilde{\mu}_x)$$

$$\frac{\Gamma \vdash^{\sigma} t : T \quad \Gamma \vdash^{\sigma} \pi : T^{\perp\!\!\!\perp}}{\Gamma \vdash^{\sigma} \langle t \| \pi\rangle}\ (\text{CUT}_t)$$

It is worth noting that the syntax as well as the typing and reduction rules for terms now match exactly the ones for proofs[6]. In other words, with these definitions, we could abandon the stratified presentation without any trouble, since reduction rules for terms will naturally collapse to the ones for proofs.

3.3 Co-delimited continuations

Finally, in order to maintain typability when reducing dependent pairs of the strong existential type, we need to add what we call *co-delimited continuations*. As observed in [18], the CPS translation of pairs (t,p) in $\text{dL}_{\hat{\mathfrak{tp}}}$ is not the expected one, reflecting the need for a special reduction rule. Indeed, consider such a pair of type $\exists x^T.A$, the standard way of reducing it would be a rule like:

$$\langle (t,p) \| e\rangle \tau \rightsquigarrow_s \langle t \| \tilde{\mu}x.\langle p \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle \tau$$

but such a rule does not satisfy subject reduction. Consider indeed a typing derivation for the left-hand side command, when typing the pair (t,p), p is of type $A[t]$. On the command on the right-hand side, the variable a will then also be of type $A[t]$, while it should be of type $A[x]$ for the pair (x,a) to be typed. We thus need to compensate this mismatching of types, by reducing t within a context where a is not linked to p but to a co-reset $\check{\mathfrak{tp}}$ (dually to reset $\hat{\mathfrak{tp}}$), whose type can be changed from $A[x]$ to $A[t]$ thanks to a list of dependencies:

$$\langle (t,p) \| e\rangle_p \tau \rightsquigarrow_s \langle p \| \tilde{\mu}\check{\mathfrak{tp}}.\langle t \| \tilde{\mu}x.\langle \check{\mathfrak{tp}} \| \tilde{\mu}a.\langle (x,a) \| e\rangle\rangle\rangle\rangle_p \tau$$

We thus equip the language with new contexts $\tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}}$, which we call co-shifts, and where $c_{\check{\mathfrak{tp}}}$ is a command whose last cut is of the shape $\langle \check{\mathfrak{tp}} \| e\rangle$. This corresponds formally to the following syntactic sets, which are dual to the ones introduced for delimited continuations:

Contexts	e	::=	$\cdots \mid \tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}}$
Co-delimited continuations	$c_{\check{\mathfrak{tp}}}$	::=	$\langle p_N \| e_{\check{\mathfrak{tp}}}\rangle \mid \langle t \| \pi_{\check{\mathfrak{tp}}}\rangle \mid \langle \check{\mathfrak{tp}} \| e\rangle$
	$e_{\check{\mathfrak{tp}}}$	::=	$\tilde{\mu}a.c_{\check{\mathfrak{tp}}} \mid \tilde{\mu}[a_1.c_{\check{\mathfrak{tp}}} \mid a_2.c'_{\check{\mathfrak{tp}}}]$
		$\mid$	$\tilde{\mu}(a_1,a_2).c_{\check{\mathfrak{tp}}} \mid \tilde{\mu}(x,a).c_{\check{\mathfrak{tp}}}$
	$\pi_{\check{\mathfrak{tp}}}$	::=	$t \cdot \pi_{\check{\mathfrak{tp}}} \mid \tilde{\mu}x.c_{\check{\mathfrak{tp}}}$
NEF	e_N	::=	$\cdots \mid \tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}}$

This might seem to be a heavy addition to the language, but we insist on the fact that these artifacts are merely the dual constructions of delimited continuations that we introduced in $\text{dL}_{\hat{\mathfrak{tp}}}$, with a very similar intuition. In particular, it might be helpful for the reader to think of the fact that we introduced delimited continuations for type safety of the evaluation of dependent products in $\Pi a : A.B$ (which naturally extends to the case $\forall x^T.A$). Therefore, to maintain type safety of dependent sums in $\exists x^T.A$, we need to introduce the dual constructions of co-delimited continuations. We also give typing rules to these constructions, which are dual to the typing rules for delimited-continuations:

$$\frac{\Gamma, \check{\mathfrak{tp}} : A \vdash_d c_{\check{\mathfrak{tp}}} ; \sigma}{\Gamma \vdash^{\sigma} \tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}} : A^{\perp\!\!\!\perp}}\ (\tilde{\mu}\check{\mathfrak{tp}}) \qquad \frac{\Gamma, \Gamma' \vdash^{\sigma} e : A^{\perp\!\!\!\perp} \quad \sigma(A) = \sigma(B)}{\Gamma, \check{\mathfrak{tp}} : B, \Gamma' \vdash_d \langle \check{\mathfrak{tp}} \| e\rangle ; \sigma}\ (\check{\mathfrak{tp}})$$

Note that we also need to extend the definition of list of dependencies so as to include bindings of the shape $\{x|t\}$ for terms, and that we have to give the corresponding typing rules to type commands of terms in dependent mode:

$$\frac{c : (\Gamma, x : T; \sigma\{x|t\})}{\Gamma \vdash_d \tilde{\mu}x.c : T^{\perp\!\!\!\perp}; \sigma\{\cdot|t\}}\ (\tilde{\mu}) \qquad \frac{\Pi_t \quad \Gamma, \check{\mathfrak{tp}} : B, \Gamma' \vdash_d \pi : A^{\perp\!\!\!\perp}; \sigma\{\cdot|t\}}{\Gamma, \check{\mathfrak{tp}} : B, \Gamma' \vdash_d \langle t \| \pi\rangle ; \sigma}\ (\text{CUT})$$

where $\Pi_t \triangleq \Gamma, \Gamma' \vdash^{\sigma} t : T$.

The small-step reduction system is given in Appendix D. The rules are written $c_\iota \tau \rightsquigarrow_s c'_o \tau'$ where the annotation ι, p on commands are indices (*i.e.* $c, p, e, V, f, t, \pi, V_t$) indicating which part of the command is in control. As in the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we observe an alternation of steps descending from p to f for proofs and from t to V_t for terms. The descent for proofs can be divided in two main phases. During the first phase, from p to e we observe the call-by-value process, which extracts values from proofs, opening recursively the constructors and computing values. In the second phase, the core computation takes place from V to f, with the destruction of constructors and the application of function to their arguments. The laziness corresponds precisely to a skip of the first phase, waiting to possibly reach the second phase before actually going through the first one.

We briefly state the important properties of this system.

Proposition 3.1 (Subject reduction)**.** *The small-step reduction rules satisfy subject reduction.*

Proof. The proof is again an induction on $\rightsquigarrow_s$, see Appendix D. □

It is direct to check that the small-step reduction system simulates the big-step one, and in particular that it preserves the normalization :

Proposition 3.2. *If a closure $c\tau$ normalizes for the reduction $\rightsquigarrow_s$, then it normalizes for $\to$.*

Proof. By contraposition, see Appendix D. □

4 Normalization of dLPA$^\omega$

4.1 A realizability interpretation of dLPA$^\omega$

We shall now present the realizability interpretation of dLPA$^\omega$, which will finally give us a proof of its normalization. Here again, the interpretation combines ideas of the interpretations for the $\overline{\lambda}_{[lv\tau\star]}$-calculus [19] and for $\text{dL}_{\hat{\mathfrak{tp}}}$ through the embedding in Lepigre's calculus [15, 18]. Namely, as for the $\overline{\lambda}_{[lv\tau\star]}$-calulus, formulas will be interpreted by sets of proofs-in-store of the shape $(p|\tau)$, and the orthogonality will be defined between proofs-in-store $(p|\tau)$ and contexts-in-store $(e|\tau')$ such that the stores τ an τ' are compatible.

We recall the main definition necessary to the realizability interpretation:

Definition 4.1 (Proofs-in-store)**.** We call *closed proof-in-store* (resp. *closed context-in-store*, *closed term-in-store*, etc) the combination of a proof p (resp. context e, term t, etc) with a closed store τ such that

[6] Except for substitutions of terms, which we could store as well.

$FV(p) \subseteq \mathrm{dom}(\tau)$. We use the notation $(p|\tau)$ to denote such a pair. In addition, we denote by Λ_p (resp. Λ_e, etc.) the set of all proofs and by Λ_p^τ (resp. Λ_e^τ, etc.) the set of all proofs-in-store.

We denote the sets of closed closures by $\mathcal{C}_0$ and we identify $(c|\tau)$ with the closure $c\tau$ when c is closed in τ.

We now recall the notion of compatible stores [19], which allows us to define an orthogonality relation between proofs- and contexts-in-store.

Definition 4.2 (Compatible stores and union)**.** Let τ and τ' be stores, we say that:

- they are *independent* and note $\tau \# \tau'$ if $\mathrm{dom}(\tau) \cap \mathrm{dom}(\tau') = \emptyset$.
- they are *compatible* and note $\tau \diamond \tau'$ if for all variables a (resp. co-variables α) present in both stores: $a \in \mathrm{dom}(\tau) \cap \mathrm{dom}(\tau')$; the corresponding proofs (resp. contexts) in τ and τ' coincide.
- τ' is an *extension* of τ and we write $\tau \lhd \tau'$ whenever $\tau \diamond \tau'$ and $\mathrm{dom}(\tau) \subseteq \mathrm{dom}(\tau')$.
- $\overline{\tau\tau'}$ is *the compatible union* of compatible closed stores τ and τ'. It is defined as $\overline{\tau\tau'} \triangleq \mathrm{join}(\tau, \tau')$, which itself given by:

$$\begin{aligned}
\mathrm{join}(\tau_0[a := p]\tau_1, \tau_0'[a := p]\tau_1') &\triangleq \tau_0\tau_0'[a := p]\mathrm{join}(\tau_1, \tau_1') \\
\mathrm{join}(\tau_0[\alpha := e]\tau_1, \tau_0'[\alpha := e]\tau_1') &\triangleq \tau_0\tau_0'[\alpha := e]\mathrm{join}(\tau_1, \tau_1') \\
\mathrm{join}(\tau_0, \tau_0') &\triangleq \tau_0\tau_0'
\end{aligned}$$

  where $\tau_0 \# \tau_0'$.

The next lemma (which follows from the previous definition) states the main property we will use about union of compatible stores.

Lemma 4.3. *If τ and τ' are two compatible stores, then $\tau \lhd \overline{\tau\tau'}$ and $\tau' \lhd \overline{\tau\tau'}$. Besides, if τ is of the form $\tau_0[x := t]\tau_1$, then $\overline{\tau\tau'}$ is of the form $\overline{\tau_0}[x := t]\overline{\tau_1}$ with $\tau_0 \lhd \overline{\tau_0}$ and $\tau_1 \lhd \overline{\tau_1}$.*

We can now define the notion of pole, which has to satisfy an extra condition due to the presence of delimited continuations

Definition 4.4 (Pole)**.** A subset $\bot\!\!\!\bot \in \mathcal{C}_0$ is said to be *saturated* or *closed by anti-reduction* whenever for all $(c|\tau), (c'|\tau') \in \mathcal{C}_0$, we have

$$(c'\tau' \in \bot\!\!\!\bot) \ \wedge \ (c\tau \to c'\tau') \ \Rightarrow \ (c\tau \in \bot\!\!\!\bot)$$

It is said to be *closed by store extension* if whenever $c\tau$ is in $\bot\!\!\!\bot$, for any store τ' extending τ, $c\tau'$ is also in $\bot\!\!\!\bot$:

$$(c\tau \in \bot\!\!\!\bot) \ \wedge \ (\tau \lhd \tau') \ \Rightarrow \ (c\tau' \in \bot\!\!\!\bot)$$

It is said to be *closed under delimited continuations* if whenever $c[e/\hat{\mathfrak{k}}]\tau$ (resp. $c[V/\check{\mathfrak{k}}]\tau$) is in $\bot\!\!\!\bot$, then $\langle \mu\hat{\mathfrak{k}}.c \| e\rangle\tau$ (resp $.\langle V \| \tilde{\mu}\check{\mathfrak{k}}.c\rangle\tau$) belongs to $\bot\!\!\!\bot$:

$$\begin{aligned}
(c[e/\hat{\mathfrak{k}}]\tau \in \bot\!\!\!\bot) \ &\Rightarrow \ (\langle \mu\hat{\mathfrak{k}}.c \| e\rangle\tau \in \bot\!\!\!\bot) \\
(c[V/\check{\mathfrak{k}}]\tau \in \bot\!\!\!\bot) \ &\Rightarrow \ (\langle V \| \tilde{\mu}\check{\mathfrak{k}}.c\rangle\tau \in \bot\!\!\!\bot)
\end{aligned}$$

A *pole* is defined as any subset of $\mathcal{C}_0$ that is closed by anti-reduction, by store extension and under delimited continuations.

We verify that the set of normalizing command is indeed a pole:

Proposition 4.5. *The set $\bot\!\!\!\bot_{\Downarrow} = \{c\tau \in \mathcal{C}_0 : \ c\tau$ normalizes$\}$ is a pole.*

Proof. See Appendix E. □

We finally recall the definition of the orthogonality relation with respect to a pole, which is identical to the one for the $\overline{\lambda}_{[lv\tau\star]}$-calculus:

Definition 4.6 (Orthogonality)**.** Given a pole $\bot\!\!\!\bot$, we say that a proof-in-store $(p|\tau)$ is *orthogonal* to a context-in-store $(e|\tau')$ and write $(p|\tau) \bot\!\!\!\bot (e|\tau')$ if τ and τ' are compatible and $\langle p \| e\rangle\overline{\tau\tau'} \in \bot\!\!\!\bot$. The orthogonality between terms and coterms is defined identically.

We are now equipped to define the realizability interpretation of dLPA$^\omega$. Firstly, in order to simplify the treatment of coinductive formulas, we extend the language of formulas with second-order variables $X, Y, \ldots$ and we replace $\nu^t_{fx}A$ by $\nu^t_{Xx}A[X(y)/f(y) = 0]$. The typing rule for co-fixpoint operators then becomes:

$$\frac{\Gamma \vdash^\sigma t : T \quad \Gamma, x : T, b : \forall y^T.X(y) \vdash^\sigma p : A \quad X \notin FV(\Gamma)}{\Gamma \vdash^\sigma \mathtt{cofix}^t_{bx}[p] : \nu^t_{Xx}A} \ (\mathtt{cofix})$$

where X has to be positive in A.

Secondly, as in the interpretation of dL$_{\hat{\mathfrak{k}}}$ through Lepigre's calculus, we introduce two new predicates, $p \in A$ for NEF proofs and $t \in T$ for terms. This allows us to decompose the dependent products and sums into:

$$\begin{array}{l|l}
\forall x^T.A \triangleq \forall x.(x \in T \to A) & \Pi a : A.B \triangleq A \to B \qquad (a \notin FV(B)) \\
\exists x^T.A \triangleq \exists x.(x \in T \to A) & \Pi a : A.B \triangleq \forall a.(a \in A \to B) \quad \text{(otw.)}
\end{array}$$

This corresponds to the language of formulas and types defined by:

$$\begin{array}{llll}
\textbf{Types} & T, U & ::= & \mathbb{N} \mid T \to U \mid t \in T \\
\textbf{Formulas} & A, B & ::= & \top \mid \bot \mid X(t) \mid t = u \mid A \wedge B \mid A \vee B \\
 & & \mid & \forall x.A \mid \exists x.A \mid \forall a.A \mid \nu^t_{Xx}A \mid a \in A
\end{array}$$

and to the following inference rules:

$$\frac{\Gamma \vdash^\sigma v : A \quad a \notin FV(\Gamma)}{\Gamma \vdash^\sigma v : \forall a.A} \ (\forall^a_r) \qquad \frac{\Gamma \vdash^\sigma e : A[q/a] \quad q \ \text{NEF}}{\Gamma \vdash^\sigma e : (\forall a.A)^{\bot\!\!\!\bot}} \ (\forall^a_l)$$

$$\frac{\Gamma \vdash^\sigma v : A \quad x \notin FV(\Gamma)}{\Gamma \vdash^\sigma v : \forall x.A} \ (\forall^x_r) \qquad \frac{\Gamma \vdash^\sigma e : A[t/x]}{\Gamma \vdash^\sigma e : (\forall x.A)^{\bot\!\!\!\bot}} \ (\forall^x_l)$$

$$\frac{\Gamma \vdash^\sigma v : A[t/x]}{\Gamma \vdash^\sigma v : \exists x.A} \ (\exists^x_r) \qquad \frac{\Gamma \vdash^\sigma e : A \quad x \notin FV(\Gamma)}{\Gamma \vdash^\sigma e : (\exists x.A)^{\bot\!\!\!\bot}} \ (\exists^x_l)$$

$$\frac{\Gamma \vdash^\sigma p : A \quad p \ \text{NEF}}{\Gamma \vdash^\sigma p : p \in A} \ (\in^p_r) \qquad \frac{\Gamma \vdash^\sigma e : A^{\bot\!\!\!\bot}}{\Gamma \vdash^\sigma e : (q \in A)^{\bot\!\!\!\bot}} \ (\in^p_l)$$

$$\frac{\Gamma \vdash^\sigma t : T}{\Gamma \vdash^\sigma t : t \in T} \ (\in^t_r) \qquad \frac{\Gamma \vdash^\sigma \pi : T^{\bot\!\!\!\bot}}{\Gamma \vdash^\sigma \pi : (t \in T)^{\bot\!\!\!\bot}} \ (\in^t_l)$$

These rules are exactly the same as in Lepigre's calculus [15] up to our stratified presentation in a sequent calculus fashion and modulo our syntactic restriction to NEF proofs instead of his semantical restriction. It is a straightforward verification to check that the typability is maintained through the decomposition of dependent products and sums.

Another similarity with Lepigre's realizability model is that truth/falsity values will be closed under observational equivalence of proofs and terms. To this purpose, for each store τ we introduce the relation $\equiv_\tau$, which we define as the reflexive-transitive-symmetric closure of the relation $\triangleright_\tau$:

$$\begin{array}{llll}
t & \triangleright_\tau & t' & \text{whenever} \quad \exists \tau', \forall \pi, (\langle t \| \pi\rangle\tau \to \langle t' \| \pi\rangle\tau' \\
p & \triangleright_\tau & q & \text{whenever} \quad \exists \tau', \forall f \ (\langle p \| f\rangle\tau \to \langle q \| f\rangle\tau')
\end{array}$$

All this being settled, it only remains to determine how to interpret coinductive formulas. While it would be natural to try to interpret them by fixpoints in the semantics, this poses difficulties for the proof of adequacy. We will discuss this matter in the next section, but as for now, we will give a simpler interpretation. We

$$
\begin{array}{rl|rl}
\|\bot\|_f & \triangleq \Lambda_f^\tau & \|t = u\|_f & \triangleq \begin{cases} \{(\tilde{\mu}=.c|\tau) : c\tau \in \perp\!\!\!\perp\} & \text{if } t \equiv_\tau u \\ \Lambda_f^\tau & \text{otherwise} \end{cases} \\
\|\top\|_f & \triangleq \emptyset & \|p \in A\|_f & \triangleq \{(V|\tau) \in |A|_V : V \equiv_\tau p\}^{\perp\!\!\!\perp f} \\
\|\dot{F}(t)\|_f & \triangleq F(t) & \|T \to B\|_f & \triangleq \{(V_t \cdot e|\tau) : (V_t|\tau) \in |t \in T|_{V_t} \wedge (e|\tau) \in \|B\|_e\} \\
\|\exists x.A\|_f & \triangleq \bigcap_{t\in\Lambda_t} \|A[t/x]\|_f & \|A \to B\|_f & \triangleq \{(V \cdot e|\tau) : (V|\tau) \in |A|_V \wedge (e|\tau) \in \|B\|_e\} \\
\|\forall x.A\|_f & \triangleq (\bigcap_{t\in\Lambda_t} \|A[t/x]\|_f^{\perp\!\!\!\perp v})^{\perp\!\!\!\perp f} & \|T \wedge A\|_f & \triangleq \{(\tilde{\mu}(x,a).c|\tau) : \forall \tau', V_t \in |T|_{V_t}^{\tau'}, V \in |A|_V^{\tau'}, \tau \diamond \tau' \Rightarrow c[V_t/x]\overline{\tau\tau'}[a := V] \in \perp\!\!\!\perp\} \\
\|\forall a.A\|_f & \triangleq (\bigcap_{t\in\Lambda_p} \|A[p/a]\|_f^{\perp\!\!\!\perp v})^{\perp\!\!\!\perp f} & \|A_1 \wedge A_2\|_f & \triangleq \{(\tilde{\mu}(a_1,a_2).c|\tau) : \forall \tau', V_1 \in |A_1|_V^{\tau'}, V_2 \in |A_2|_V^{\tau'}, \tau \diamond \tau' \Rightarrow c\overline{\tau\tau'}[a_1 := V_1][a_2 := V_2] \in \perp\!\!\!\perp\} \\
\|\nu_{fx}^t A\|_f & \triangleq \bigcup_{n\in\mathbb{N}} \|F_{A,t}^n\|_f & \|A_1 \vee A_2\|_f & \triangleq \{(\tilde{\mu}[a_1.c_1|a_2.c_2]|\tau) : \forall \tau', V \in |A_i|_V^{\tau'}, \tau \diamond \tau' \Rightarrow c\overline{\tau\tau'}[a_i := V] \in \perp\!\!\!\perp\} \\
|A|_V & \triangleq \|A\|_f^{\perp\!\!\!\perp V} & |A|_p & \triangleq \|A\|_e^{\perp\!\!\!\perp p} \\
\|A\|_e & \triangleq |A|_V^{\perp\!\!\!\perp e} & & \\
\end{array}
$$

$$
\begin{array}{rl|rl}
|\mathbb{N}|_{V_t} & \triangleq \{(S^n(0)|\tau), n \in \mathbb{N}\} & |T|_\pi & \triangleq |A|_{V_t}^{\perp\!\!\!\perp \pi} \\
|t \in T|_{V_t} & \triangleq \{(V_t|\tau) \in |T|_{V_t} : V_t \equiv_\tau t\} & |T|_t & \triangleq |A|_\pi^{\perp\!\!\!\perp t} \\
|T \to U|_{V_t} & \triangleq \{(\lambda x.t|\tau) : \forall V_t \tau', \tau \diamond \tau' \wedge (V_t|\tau') \in |T|_{V_t} \Rightarrow (t[V_t/x]|\overline{\tau\tau'}) \in |U|_t\} & & \\
\end{array}
$$

where:

- $p \in S^\tau$ (resp. e,V,etc.) denotes $(p|\tau) \in S$ (resp. $(e|\tau)$, $(V|\tau)$, etc.),
- F is a function from Λ_t to $\mathcal{P}(\Lambda_f^\tau)_{/\equiv_\tau}$.

Figure 5. Realizability interpretation for dLPA$^\omega$

stick to the intuition that since `cofix` operators are lazily evaluated, they actually are realizers of every finite approximation of the (possibly infinite) coinductive formulas. Consider for instance the case of a stream

$$\mathtt{str}_\infty^0 p \triangleq \mathtt{cofix}_{bx}^0[(px, b(S(x)))]$$

of type $\nu_{Xx}^0 A(x) \wedge X(S(x))$. Such stream will produce on demand any tuple $(p0,(p1,\ldots(pn,\Box)\ldots)$ where $\Box$ denotes the fact that it could be any term, in particular $\mathtt{str}_\infty^{n+1} p$. So that $\mathtt{str}_\infty^0 p$ should be a successful defender of the formula

$$(A(0) \wedge (A(1) \wedge \ldots(A(n) \wedge \top)\ldots)$$

Since `cofix` operators only reduce when they are bound to a variable in front of a forcing context, it suggests to interpret the coinductive formula $\nu_{Xx}^0 A(x) \wedge X(S(x))$ at level f as the union of all the opponents to a finite approximation.

To this end, given a coinductive formula $\nu_{Xx}^0 A$ where X is positive in A, we define its finite approximations by:

$$F_{A,t}^0 \triangleq \top \qquad\qquad F_{A,t}^{n+1} \triangleq A[t/x][F_{A,y}^n/X(y)]$$

Since f is positive in A, we have for any integer n and any term t that $\|F_{A,t}^n\|_f \subseteq \|F_{A,t}^{n+1}\|_f$. We can finally define the interpretation of coinductive formulas by:

$$\|\nu_{Xx}^t A\|_f \triangleq \bigcup_{n\in\mathbb{N}} \|F_{A,t}^n\|_f$$

The realizability interpretation of closed formulas and types is defined in Figure 5 by induction on the structure of formulas at level f, and by orthogonality at levels V, e, p. When S is a subset of $\mathcal{P}(\Lambda_p^\tau)$ (resp. $\mathcal{P}(\Lambda_e^\tau), \mathcal{P}(\Lambda_t^\tau), \mathcal{P}(\Lambda_\pi^\tau)$), we use the notation $S^{\perp\!\!\!\perp f}$ (resp. $S^{\perp\!\!\!\perp V}$, etc.) to denote its orthogonal set restricted to Λ_f^τ (resp. Λ_V^τ, etc.):

$$S^{\perp\!\!\!\perp f} \triangleq \{(f|\tau) \in \Lambda_f^\tau : \forall (p|\tau') \in S, \tau \diamond \tau' \Rightarrow \langle p \| f\rangle \overline{\tau\tau'} \in \perp\!\!\!\perp\}$$

At level f, closed formulas are interpreted by sets of strong forcing contexts-in-store $(f|\tau)$. As observed in the previous section, these sets are besides closed under the relation $\equiv_\tau$ along their component τ, we thus denote them by $\mathcal{P}(\Lambda_f^\tau)_{/\equiv_\tau}$. Second-order variables $X, Y, \ldots$ are then interpreted by functions from the set of terms Λ_t to $\mathcal{P}(\Lambda_f^\tau)_{/\equiv_\tau}$ and as usual for each such function F we add a predicate symbol $\dot{F}$ in the language.

We shall now prove the adequacy of the interpretation with respect to the type system. To this end, we need to recall a few definitions and lemmas. Since stores only contain proof terms, we need to define valuations for term variables in order to close formulas[7]. These valuations are defined by the usual grammar:

$$\rho ::= \varepsilon \mid \rho[x \mapsto V_t] \mid \rho[X \mapsto \dot{F}]$$

We denote by $(p|\tau)_\rho$ (resp. p_ρ, A_ρ) the proof-in-store $(p|\tau)$ where all the variables $x \in \mathrm{dom}(\rho)$ (resp. $X \in \mathrm{dom}(\rho)$) have been substituted by the corresponding term $\rho(x)$ (resp. falsity value $\rho(x)$).

Definition 4.7. Given a closed store τ, a valuation ρ and a fixed pole $\perp\!\!\!\perp$, we say that the pair (τ,ρ) *realizes* Γ, which we write[8] $(\tau,\rho) \Vdash \Gamma$, if:

1. for any $(a : A) \in \Gamma$, $(a|\tau)_\rho \in |A_\rho|_V$
2. for any $(\alpha : A_\rho^{\perp\!\!\!\perp}) \in \Gamma$, $(\alpha|\tau)_\rho \in \|A_\rho\|_e$
3. for any $\{a|p\} \in \sigma$, $a \equiv_\tau p$
4. for any $(x : T) \in \Gamma$, $x \in \mathrm{dom}(\rho)$ and $(\rho(x)|\tau) \in |T_\rho|_{V_t}$

We recall two key properties of the interpretation, whose proofs are similar to the proofs for the corresponding statement in the $\overline{\lambda}_{[lv\tau\star]}$-calculus [19]:

Lemma 4.8 (Store weakening). *Let τ and τ' be two stores such that $\tau \lhd \tau'$, let Γ be a typing context, let $\perp\!\!\!\perp$ be a pole and ρ a valuation. The following statements hold:*

1. $\overline{\tau\tau'} = \tau'$
2. *If $(p|\tau)_\rho \in |A_\rho|_p$ for some closed proof-in-store $(p|\tau)_\rho$ and formula A, then $(p|\tau')_\rho \in |A_\rho|_p$. The same holds for each level e, E, V, f, t, π, V_t of the interpretation.*
3. *If $(\tau,\rho) \Vdash \Gamma$ then $(\tau',\rho) \Vdash \Gamma$.*

Proposition 4.9 (Monotonicity). *For any closed formula A, any type T and any given pole $\perp\!\!\!\perp$, we have the following inclusions:*

$$|A|_V \subseteq |A|_p \qquad \|A\|_f \subseteq \|A\|_e \qquad |T|_{V_t} \subseteq |T|_t$$

[7] Alternatively, we could have modified the small-step reduction rules to include substitutions of terms.

[8] Once again, we should formally write $(\tau,\rho) \Vdash_{\perp\!\!\!\perp} \Gamma$ but we will omit the annotation by $\perp\!\!\!\perp$ as often as possible.

Finally we can check that the interpretation is indeed defined up to the relations $\equiv_\tau$:

Lemma 4.10. *For any store τ and any valuation ρ, the component along τ of the truth and falsity values defined in Figure 5 are closed under the relation $\equiv_\tau$:*

1. *if $(f|\tau)_\rho \in \|A_\rho\|_f$ and $A_\rho \equiv_\tau B_\rho$, then $(f|\tau)_\rho \in \|B_\rho\|_f$,*
2. *if $(V_t|\tau)_\rho \in |A_\rho|_{V_t}$ and $A_\rho \equiv_\tau B_\rho$, then $(V_t|\tau)_\rho \in |B_\rho|_v$.*

The same applies with $|A_\rho|_p$, $\|A_\rho\|_e$, etc.

Proof. By induction on the structure of A_ρ and the different levels of interpretation. The different base cases ($p \in A_\rho$, $t \in T$, $t = u$) are direct since their components along τ are defined modulo $\equiv_\tau$, the other cases are trivial inductions. □

Proposition 4.11 (Adequacy)**.** *The typing rules are adequate with respect to the realizability interpretation.*

Proof. The proof is done by induction on the typing derivation such as given in the system extended with the small-step reduction $\rightsquigarrow_s$. See Appendix E. □

We can finally deduce from Propositions 4.5 and 4.11 that dLPA$^\omega$ is normalizing and sound.

Theorem 4.12 (Normalization)**.** *If $\Gamma \vdash^\sigma c$, then c is normalizable.*

Theorem 4.13 (Consistency)**.** $\nvdash_{dLPA^\omega} p : \bot$

Proof. Assume there is such a proof p, by adequacy $(p|\varepsilon)$ is in $|\bot|_p$ for any pole. Yet, the set $\perp\!\!\!\perp \triangleq \emptyset$ is a valid pole, and with this pole, $|\bot|_p = \emptyset$, which is absurd. □

5 Conclusion and perspectives

Conclusion At the end of the day, we met our main objective, namely proving the soundness and the normalization of a language which includes proof terms for dependent and countable choice in a classical setting. This language, which we called dLPA$^\omega$, provides us with the same computational features as dPA$^\omega$ but in a sequent-calculus fashion. Interstingly, in our search for a proof of normalization for dLPA$^\omega$, we developed novel tools to study these side-effects and dependent types in presence of classical logic. On the one hand, we set out in [18] the difficulties related to the definition of a sequent calculus with dependent types. On the other hand, building on [19], we developed a variant of Krivine realizability adapted to a lazy calculus where delayed substitutions are stored in an explicit environment. These computational features allows dLPA$^\omega$ to internalize the realizability approach of [9**?**] as a direct proofs-as-programs interpretation: both proof terms for countable and dependent choices furnish a lazy witness for the ideal choice function which is evaluated on demand. This interpretation is in line with the slogan that with new programing principles—here the lazy evaluation and the co-inductive objects—come new reasoning principles—here the axioms $AC_{\mathbb{N}}$ and DC.

Krivine's interpretations of dependent choice The computational content we give to the axiom of dependent choice is pretty different of Krivine's usual realizer of the same [12]. Indeed, our proof uses dependent types to get witnesses of existential formulas, and represents the choice function through the lazily evaluated stream of its values. In turn, Krivine realizes a statement which is logically equivalent to the axiom of dependent choice thanks to the instruction `quote`, which injectively associates a natural number to each closed λ_c-term. In a more recent work [14], Krivine proposes a realizability model which has a bar-recursor and where the axiom of dependent choice is realized using the bar-recursion. This realizability model satisfies the continuum hypothesis and many more properties, in particular the real numbers have the same properties as in the ground model. However, the very structure of this model, where $\mathbf{\Lambda}$ is of cardinal $\aleph_1$ (in particular infinite streams of integer are terms), makes it incompatible with the instruction `quote`.

It is clear that the three approaches are different in terms of programming languages. Nonetheless, it could be interesting to compare them from the point of view of the realizability models they give rise to. In particular, our analysis of the interpretation of co-inductive formulas[9] may suggest that the interest of lazy co-fixpoints is precisely to approximate the limit situation where Λ has infinite objects.

Reduction of the consistency of classical arithmetic in finite types with dependent choice to the consistency of second-order arithmetic The standard approach to the computational content of classical dependent choice in the classical arithmetic in finite types is via realizability as initiated by Spector [21] in the context of Gödel's functional interpretation, and later adapted to the context of modified realizability by Berardi *et al* [**?**]. In the different settings of second-order arithmetic [13] and classical realizability, Krivine [12] gives a realization of a formulation of dependent choice over sets of numbers using side-effects (a clock or a quote operator).

In all these approaches, the correctness of the realizer, which implies consistency of the system, is itself justified by a use at the meta-level of a principle classically equivalent to dependent choice (dependent choice itself in Krivine, bar induction or update induction [3] in the case of Spector or Berardi *et al*).

Our approach is here different. Not only we directly interpret proofs of dependent choice in classical arithmetic computationally but we propose a path to a computational reduction of the consistency of classical arithmetic in finite types (PA^ω) to the one of the target language F_Υ. This system is an extension of system F, but it is not clear whether its consistency is conservative of not over system F. Ultimately, we would be interested in a computational reduction of the consistency of dPA$^\omega$ or dLPA$^\omega$ to the one of PA2, that is to the consistency of second-order arithmetic. While it is well-known that DC is conservative over second-order arithmetic with full comprehension (see [20, Theorem VII.6.20]), it would nevertheless be very interesting to have such a direct computational reduction. The converse direction has been recently studied by Valentin Blot, who presented in [4] a translation of System F into a simply-typed total language with a variant of bar recursion.

Acknowledgments

The author warmly thanks Hugo Herbelin for numerous discussions and attentive reading of this work during his PhD years.

References

[1] Zena M. Ariola, Paul Downen, Hugo Herbelin, Keiko Nakata, and Alexis Saurin. 2012. Classical Call-by-Need Sequent Calculi: The Unity of Semantic Artifacts. In *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings (Lecture Notes in Computer Science)*, Tom Schrijvers and Peter Thiemann (Eds.). Springer, 32–46. https://doi.org/10.1007/978-3-642-29822-6

[9] See also Appendix F.

[2] Stefano Berardi, Marc Bezem, and Thierry Coquand. 1998. On the Computational Content of the Axiom of Choice. *J. Symb. Log.* 63, 2 (1998), 600–622. https://doi.org/10.2307/2586854

[3] Ulrich Berger. 2004. A Computational Interpretation of Open Induction. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*. IEEE Computer Society, 326.

[4] Valentin Blot. 2017. An interpretation of system F through bar recursion. In *LICS 2017, Reijkavik, Iceland*.

[5] P. Cousot and R. Cousot. 1979. Constructive Versions of Tarski's Fixed Point Theorems. *Pacific J. Math.* 81, 1 (1979), 43–57.

[6] Pierre-Louis Curien and Hugo Herbelin. 2000. The duality of computation. In *Proceedings of ICFP 2000 (SIGPLAN Notices 35(9))*. ACM, 233–243. https://doi.org/10.1145/351240.351262

[7] Olivier Danvy, Kevin Millikin, Johan Munk, and Ian Zerny. 2010. *Defunctionalized Interpreters for Call-by-Need Evaluation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 240–256. https://doi.org/10.1007/978-3-642-12251-4_18

[8] Paul Downen, Luke Maurer, Zena M. Ariola, and Simon Peyte Jones. 2016. Sequent calculus as a compiler intermediate language. In *ICFP 2016*. http://research.microsoft.com/en-us/um/people/simonpj/papers/sequent-core/scfp_ext.pdf

[9] Martín H. Escardó and Paulo Oliva. 2014. Bar Recursion and Products of Selection Functions. *CoRR* abs/1407.7046 (2014). http://arxiv.org/abs/1407.7046

[10] Hugo Herbelin. 2005. On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic. In *Typed Lambda Calculi and Applications, 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings (Lecture Notes in Computer Science)*, Pawel Urzyczyn (Ed.), Vol. 3461. Springer, 209–220. https://doi.org/10.1007/11417170_16

[11] Hugo Herbelin. 2012. A Constructive Proof of Dependent Choice, Compatible with Classical Logic. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 365–374. https://doi.org/10.1109/LICS.2012.47

[12] J.-L. Krivine. 2003. Dependent choice, 'quote' and the clock. *Th. Comp. Sc.* 308 (2003), 259–276.

[13] J.-L. Krivine. 2009. Realizability in classical logic. In Interactive models of computation and program behaviour. *Panoramas et synthèses* 27 (2009).

[14] Jean-Louis Krivine. 2016. Bar Recursion in Classical Realisability: Dependent Choice and Continuum Hypothesis. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Jean-Marc Talbot and Laurent Regnier (Eds.), Vol. 62. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 25:1–25:11. https://doi.org/10.4230/LIPIcs.CSL.2016.25

[15] Rodolphe Lepigre. 2016. A Classical Realizability Model for a Semantical Value Restriction. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings (Lecture Notes in Computer Science)*, Peter Thiemann (Ed.), Vol. 9632. Springer, 476–502.

[16] P. Martin-Löf. 1998. An intuitionistic theory of types. In Twenty-five years of constructive type theory. *Oxford Logic Guides* 36 (1998), 127–172.

[17] Alexandre Miquel. 2017. Implicative algebras. *Private communication* (2017).

[18] Étienne Miquey. 2017. A Classical Sequent Calculus with Dependent Types. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings*, Hongseok Yang (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 777–803. https://doi.org/10.1007/978-3-662-54434-1_29

[19] Étienne Miquey and Hugo Herbelin. 2018. Realizability interpretation and normalization of typed call-by-need λ-calculus with control. *Accepted at FCSD 2018* (2018). https://hal.inria.fr/hal-01624839

[20] Stephen G. Simpson. 2009. *Subsystems of Second Order Arithmetic* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511581007

[21] Clifford Spector. 1962. Provably recursive functionals of analysis: A consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In *Recursive function theory: Proceedings of symposia in pure mathematics*, F. D. E. Dekker (Ed.), Vol. 5. American Mathematical Society, Providence, Rhode Island, 1–27.

[22] Philip Wadler. 2003. Call-by-value is dual to call-by-name. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, Colin Runciman and Olin Shivers (Eds.). ACM, 189–201. https://doi.org/10.1145/944705.944723

Since most of the proofs contain typing derivations, we switch to a one column format to easen their display.

A Subject reduction

.

We first show that the cases which we encounter in the proof of subject reduction satisfy this relation:

Lemma A.1 (Dependencies implication)**.** *The following holds for any $\sigma, \sigma', \sigma''$:*

1. $\sigma\sigma'' \Rrightarrow \sigma\sigma'\sigma'$
2. $\sigma\{(a_1,a_2)|(V_1,V_2)\} \Rrightarrow \sigma\{a_1|V_1\}\{a_2|V_2\}$
3. $\sigma\{\iota_i(a)|\iota_i(V)\} \Rrightarrow \sigma\{a|V\}$
4. $\sigma\{(x,a)|(t,V)\} \Rrightarrow \sigma\{a|V\}\{x|t\}$
5. $\sigma\{\cdot|(p_1,p_2)\} \Rrightarrow \sigma\{a_1|p_1\}\{a_2|p_2\}\{\cdot|(a_1,a_2)\}$
6. $\sigma\{\cdot|\iota_i(p)\} \Rrightarrow \sigma\{a|p\}\{\cdot|\iota_i(a)\}$
7. $\sigma\{\cdot|(t,p)\} \Rrightarrow \sigma\{a|p\}\{\cdot|(t,a)\}$

where the fourth item abuse the definition of list of dependencies to include a substitution of terms.

Proof. All the properties are trivial from the definition of the substitution $\sigma(A)$. □

Proposition 2.1 (Dependencies weakening)**.** *If σ, σ' are two dependencies list such that $\sigma \Rrightarrow \sigma'$, then any derivation using σ can be one using σ' instead. In other words, the following rules are admissible:*

$$\frac{\Gamma \vdash^{\sigma} J}{\Gamma \vdash^{\sigma'} J}\ (w) \qquad\qquad \frac{\Gamma \vdash_d J;\sigma}{\Gamma \vdash_d J;\sigma'}\ (w_d)$$

Proof. Simple induction on the typing derivations. The rules ($\hat{\mathfrak{tp}}$) and (Cut) where the list of dependencies is used exactly match the definition of $\Rrightarrow$. Every other case is direct using the first item of Lemma A.1. □

Theorem 2.4. *For any context Γ and any closures $c\tau$ and $c'\tau'$ such that $c\tau \to c'\tau'$, we have:*

1. *If $\Gamma \vdash c\tau$ then $\Gamma \vdash c'\tau'$.*
2. *If $\Gamma \vdash_d c\tau;\varepsilon$ then $\Gamma \vdash_d c'\tau';\varepsilon$.*

Proof. The proof follows the usual proof of subject reduction, by induction on the typing derivation and the reduction $c\tau \to c'\tau'$. Since there is no substitution but for terms (proof terms and contexts being stored), there is no need for auxiliary lemmas about the safety of substitution. We sketch it by examining all the rules from Figure 3 from top to bottom.

- The cases for reductions of λ are identical to the cases proven in the previous chapter for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$.
- The rules for reducing μ and $\tilde{\mu}$ are almost the same except that elements are stored, which makes it even easier. For instance in the case of $\tilde{\mu}$, the reduction rule is:

$$\langle V \| \tilde{\mu}a.c\tau_1 \rangle \tau_0 \to c\tau_0[a := V]\tau_1$$

A typing derivation in regular mode for the command on the left-hand side is of the shape:

$$\dfrac{\dfrac{\Pi_V}{\Gamma,\Gamma_0 \vdash^{\sigma\sigma_0} V : A} \quad \dfrac{\dfrac{\dfrac{\Pi_c}{\Gamma,\Gamma_0,a:A,\Gamma_1 \vdash^{\sigma\sigma_0\sigma_1} c} \quad \dfrac{\Pi_{\tau_1}}{\Gamma,\Gamma_0,a:A \vdash^{\sigma\sigma_0} \tau_1 : (\Gamma_1;\sigma_1)}\ (l)}{\Gamma,\Gamma_0,a:A \vdash^{\sigma\sigma_0} c\tau_1}\ (\tilde{\mu})}{\Gamma,\Gamma_0 \vdash^{\sigma\sigma_0} \tilde{\mu}a.c\tau_1 : A^{\perp\!\!\!\perp}}\ (\text{Cut}) \quad \dfrac{\Pi_{\tau_0}}{\Gamma \vdash^{\sigma} \tau_0 : (\Gamma_0;\sigma_0)}}{\Gamma,\Gamma_0 \vdash^{\sigma\sigma_0} \langle V \| \tilde{\mu}a.c\tau_1 \rangle \qquad\qquad \Gamma \vdash^{\sigma} \langle V \| \tilde{\mu}a.c\tau_1 \rangle \tau_0}\ (l)$$

Thus we can type the command on the right-hand side:

$$\dfrac{\dfrac{\Pi_c}{\Gamma,\Gamma_0,a:A,\Gamma_1 \vdash^{\sigma\sigma_0\{a|V\}\sigma_1} c}\ (w)}{\Gamma,\Gamma_0,a:A,\Gamma_1 \vdash^{\sigma\sigma_0\{a|V\}\sigma_1} c} \quad \dfrac{\dfrac{\dfrac{\Pi_{\tau_0}}{\Gamma \vdash^{\sigma} \tau_0 : (\Gamma_0;\sigma_0)} \quad \dfrac{\Pi_V}{\Gamma,\Gamma_0 \vdash^{\sigma\sigma_0} V : A}}{\Gamma \vdash^{\sigma} \tau_0[a := V] : (\Gamma_0, a:A; \sigma_0, \{a|V\})}\ (\tau_p) \quad \dfrac{\Pi_{\tau_1}}{\Gamma,\Gamma_0,a:A \vdash^{\sigma\sigma_0} \tau_1 : (\Gamma_1;\sigma_1)}}{\Gamma \vdash^{\sigma} \tau_0[a := V]\tau_1 : (\Gamma_0, a:A, \Gamma_1; \sigma_0\{a|V\}\sigma_1)}\ (\tau\tau')}{\Gamma \vdash^{\sigma} c\tau_0[a := V]\tau_1}\ (l)$$

As for the dependent mode, the binding $\{a|p\}$ within the list of dependencies is compensated when typing the store as shown in the last derivation.

- Similarly, elimination rules for contexts $\tilde{\mu}[a_1.c_1|a_2.c_2]$, $\tilde{\mu}(a_1,a_2).c$, $\tilde{\mu}(x,a).c$ or $\tilde{\mu}^{=}.c$ are easy to check, using Lemma A.1 and the rule (τ_p) in dependent mode to prove the safety with respect to dependencies.

- The cases for delimited continuations are identical to the corresponding cases for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$.

- The cases for the so-called "call-by-value" rules opening constructors are straightforward, using again Lemma A.1 in dependent mode to prove the consistency with respect to the list of dependencies.

- The cases for the lazy rules are trivial.

- The first case in the "lookup" section is trivial. The three lefts correspond to the usual unfolding of inductive and co-inductive fixpoints. We only sketch the latter in regular mode. The reduction rule is:

$$\langle a \| f \rangle \tau_0[a := \mathtt{cofix}^t_{bx}[p]]\tau_1 \;\to\; \langle p[t/x][b'/b] \| \tilde{\mu}a.\langle a \| f \rangle \tau_1 \rangle \tau_0[b' := \lambda y.\mathtt{cofix}^y_{bx}[p]]$$

The crucial part of the derivation for the left-hand side command is the derivation for the cofix in the store:

$$\dfrac{\dfrac{\Pi_{\tau_0}}{\Gamma \vdash^{\sigma} \tau_0 : (\Gamma_0; \sigma_0)} \quad \dfrac{\dfrac{\Pi_t}{\Gamma \vdash^{\sigma\sigma_0} t : T} \quad \dfrac{\Pi_p}{\Gamma, \Gamma_0, f : T \to \mathbb{N}, x : T, b : \forall y^T.f(y) = 0 \vdash^{\sigma\sigma_0} p : A}}{\Gamma, \Gamma_0 \vdash^{\sigma\sigma_0} \mathtt{cofix}^t_{bx}[p] : \nu^t_{fx} A}\,(\mathtt{cofix})}{\Gamma \vdash^{\sigma} \tau_0[a := \mathtt{cofix}^t_{bx}[p]] : (\Gamma_0, a : \nu^t_{fx} A; \sigma_0)}\,(\tau_p)$$

Then, using this derivation, we can type the store of the right-hand side command:

$$\dfrac{\dfrac{\Pi_{\tau_0}}{\Gamma \vdash^{\sigma} \tau_0 : (\Gamma_0; \sigma_0)} \quad \dfrac{\dfrac{\dfrac{\Pi_p}{\Gamma, \Gamma_0, y : T \vdash^{\sigma\sigma_0} y : T} \quad \dfrac{\Pi_p}{\Gamma, \Gamma_0, f : T \to \mathbb{N}, x : T, b : \forall y^T.f(y) = 0 \vdash^{\sigma\sigma_0} p : A}}{\Gamma, \Gamma_0, y : T \vdash^{\sigma\sigma_0} \mathtt{cofix}^y_{bx}[p] : \nu^y_{fx} A}\,(\mathtt{cofix})}{\Gamma, \Gamma_0 \vdash^{\sigma\sigma_0} \lambda y.\mathtt{cofix}^y_{bx}[p] : \forall y.\nu^t_{fx} A}\,(\forall_r)}{\Gamma \vdash^{\sigma} \tau_0[b' := \lambda y.\mathtt{cofix}^y_{bx}[p]] : \Gamma_0, b' : -\forall y.\nu^y_{fx} A}\,(\tau_p)$$

It only remains to type (we avoid the rest of the derivation, which is less interesting) the proof $p[t/x]$ with this new store to ensure us that the reduction is safe (since the variable a will still be of type $\nu^t_{fx} A$ when typing the rest of the command):

$$\dfrac{\dfrac{\Pi_p}{\Gamma, \Gamma_0, b : \forall y.\nu^y_{fx} A \vdash^{\sigma} p[t/x] : A[t/x][\nu^y_{fx} A / f(y) = 0]} \quad \nu^t_{fx} A \equiv A[t/x][\nu^y_{fx} A / f(y) = 0]}{\Gamma, \Gamma_0, b : \forall y.\nu^y_{fx} A \vdash^{\sigma} p[t/x] : \nu^t_{fx} A}\,(\equiv_r)$$

- The cases for reductions of terms are easy. Since terms are reduced in place within proofs, the only things to check is that the reduction of `wit` preserves types (which is trivial) and that the β-reduction verifies the subject reduction (which is a well-known fact).

□

B Natural deduction as macros

We give here two examples of typing rules for the macros `subst` $p\,q$ and `prf` pq (in natural deduction) that are admissible in dLPA$^{\omega}$. Recall that we have the following typing rules in dLPA$^{\omega}$:

$$\dfrac{\Gamma, x : T, a : A \vdash^{\sigma} c}{\Gamma \vdash^{\sigma} \tilde{\mu}(x,a).c : (\exists x^T.A)^{\perp\!\!\!\perp}}\,(\exists_l) \qquad \dfrac{\Gamma \vdash^{\sigma} p : A \quad \Gamma \vdash^{\sigma} e : A[u/t]}{\Gamma \vdash^{\sigma} \tilde{\mu}^{=}.\langle p \| e \rangle : (t = u)^{\perp\!\!\!\perp}}\,(=_l)$$

and that we defined `prf` p and `subst` $p\,q$ as syntactic sugar:

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

Observe that `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:

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

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

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

.

C Proof of $AC_{\mathbb{N}}$

We give in Figure 6 a complete typing derivation of the proof term for $AC_{\mathbb{N}}$ from Herbelin's paper [11]. The typing derivation for the proof term DC is similar and left to the reader.

Notations:

- $\mathtt{nth}_t\ p \triangleq \pi_1(\mathtt{fix}^t_{sx}[p \mid \pi_2(s)])$
- $A^n_\infty \triangleq \nu^n_{fx}[A(x) \wedge f(S(x)) = 0]$
- $\mathtt{str}^t_\infty\ H \triangleq \mathtt{cofix}^t_{bn}[(Hn, b(S(n))]$
- $A(x) \triangleq \exists y^T.P(x,y)$

Typing derivation for $\mathtt{nth}$ ($\Pi_{\mathtt{nth}}$):

$$\dfrac{\dfrac{}{n:\mathbb{N}\vdash n:\mathbb{N}}\,(\mathrm{Ax}_n) \quad \dfrac{}{a:A^0_\infty\vdash a:A^0_\infty}\,(\mathrm{Ax}_r) \quad \dfrac{\dfrac{\dfrac{}{s:A^m_\infty\vdash s:A^{S(m)}_\infty}\,(\mathrm{Ax}_r) \qquad A^m_\infty\equiv A(m)\wedge A^{S(m)}_\infty}{m:\mathbb{N}, s:A^m_\infty\vdash s:A(m)\wedge A^{S(m)}_\infty}\,(\equiv_r)}{m:\mathbb{N}, s:A^m_\infty\vdash \pi_2(s):A^{S(m)}_\infty}\,(\wedge^2_E)}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{fix}^t_{sx}[a\mid\pi_2(s)]:A^n_\infty}\,(\mathtt{ind}) \qquad A^n_\infty\equiv A(n)\wedge A^{S(n)}_\infty}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{fix}^t_{sx}[a\mid\pi_2(s)]:A(n)\wedge A^{S(n)}_\infty}\,(\equiv_r)}{a:A^0_\infty, n:\mathbb{N}\vdash \pi_1(\mathtt{fix}^t_{sx}[a\mid\pi_2(s)]):A(n)}\,(\wedge^1_E)}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{nth}_n\ a:A(n)}\,(\mathtt{def})$$

Typing derivation for $\mathtt{str}^0_\infty$ ($\Pi_{\mathtt{str}_\infty}$):

$$\dfrac{\dfrac{}{\vdash 0:\mathbb{N}} \quad \dfrac{\dfrac{\dfrac{}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y)\vdash H:\forall x^{\mathbb{N}}\exists y^T P(x,y)}\,(\mathrm{Ax}_r) \quad \dfrac{}{n:\mathbb{N}\vdash n:\mathbb{N}}\,(\mathrm{Ax}_r)}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y), n:\mathbb{N}\vdash Hn:\exists y^T.P(n,y)}\,(\forall_r)}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y), n:\mathbb{N}, b:\forall z^N.f(z)=0\vdash (Hn, b(S(n))):\exists y^T.P(n,y)\wedge f(S(n))=0}}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y)\vdash \mathtt{cofix}^0_{bn}[(Hn,b(S(n))]:\nu^0_{fx}\exists y^T.P(x,y)\wedge f(S(x))=0}\,(\mathtt{def})}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y)\vdash \mathtt{str}^0_\infty\ H:A^0_\infty}$$

Typing derivation for $AC_{\mathbb{N}}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_{\mathtt{nth}}}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{nth}_n\ a:A(n)}}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{nth}_n\ a:\exists y^T.P(n,y)}\,(\mathtt{def})}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{wit}\,(\mathtt{nth}_n\ a):T}\,(\mathtt{wit})}{a:A^0_\infty\vdash \lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a):\mathbb{N}\to T} \quad \dfrac{\dfrac{\dfrac{\dfrac{\Pi_{\mathtt{nth}}}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{nth}_n\ a:A(n)}}{a:A^0_\infty, n:\mathbb{N}\vdash \mathtt{nth}_n\ a:\exists y^T.P(n,y)}\,(\mathtt{def})}{a:A^0_\infty, x:\mathbb{N}\vdash \mathtt{prf}\,(\mathtt{nth}_n\ a):P(x,\mathtt{wit}\,(\mathtt{nth}_x\ a))}\,(\equiv_r)}{a:A^0_\infty, x:\mathbb{N}\vdash \mathtt{prf}\,(\mathtt{nth}_n\ a):P(x,\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a)x)}\,(\forall_r)}{a:A^0_\infty\vdash \lambda n.\,\mathtt{prf}\,(\mathtt{nth}_n\ a):\forall x^{\mathbb{N}}.P(x,(\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a))x)}\,(\exists_r)}{a:A^0_\infty\vdash (\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}_n\ a):\exists f^{\mathbb{N}\to T}.\forall x^{\mathbb{N}}.P(x,f(x))}\,(\mathtt{let})}{H:\forall x^{\mathbb{N}}\exists y^T P(x,y)\vdash \mathtt{let}\ a=\mathtt{str}^0_\infty\ H\ \mathtt{in}\ (\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}_n\ a):\exists f^{\mathbb{N}\to T}.\forall x^{\mathbb{N}}.P(x,f(x))}\,(\to_r)}{\vdash \lambda H.\,\mathtt{let}\ a=\mathtt{str}^0_\infty\ H\ \mathtt{in}\ (\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a), \lambda n.\,\mathtt{prf}\,(\mathtt{nth}_n\ a):\forall x^{\mathbb{N}}.\exists y^T.P(x,y)\to\exists f^{\mathbb{N}\to T}.\forall x^{\mathbb{N}}.P(x,f(x))}$$

where we omit the conversion $P(x,(\lambda n.\,\mathtt{wit}\,(\mathtt{nth}_n\ a))x) \equiv P(x,\mathtt{wit}\,(\mathtt{nth}_x\ a))$ on the right-hand side derivation.

Figure 6. Proof of the axiom of countable choice in dLPA$^\omega$

D Small-step reduction rules

. We give in Figure 7 the full reduction system based on small-step reduction rules which are described in Section 3

Commands

$$\langle p \| e \rangle_c \tau \rightsquigarrow_s \langle p \| e \rangle_p$$
$$\langle t \| \pi \rangle_c \tau \rightsquigarrow_s \langle t \| \pi \rangle_t$$

Delimited continuations

(for any ι, o)
$$\langle \mu\hat{\tilde{\mathrm{p}}}.c\tau'' \| e \rangle_p \tau \rightsquigarrow_s \langle \mu\hat{\tilde{\mathrm{p}}}.c'\tau'' \| e \rangle_p \tau' \quad (\text{if } c_\iota \tau \rightsquigarrow_s c'_o \tau')$$
$$\langle \mu\hat{\tilde{\mathrm{p}}}.\langle p \| \hat{\tilde{\mathrm{p}}} \rangle \| e \rangle_p \tau \rightsquigarrow_s \langle p \| e \rangle_p \tau$$

(for any ι, o)
$$\langle V \| \tilde{\mu}\check{\tilde{\mathrm{p}}}.c \rangle_e \tau \rightsquigarrow_s \langle V \| \tilde{\mu}\check{\tilde{\mathrm{p}}}.c' \rangle_e \tau' \quad (\text{if } c_\iota \tau \rightsquigarrow_s c'_o \tau')$$
$$\langle V \| \tilde{\mu}\check{\tilde{\mathrm{p}}}.\langle \check{\tilde{\mathrm{p}}} \| e \rangle \rangle_e \tau \rightsquigarrow_s \langle V \| e \rangle_e \tau$$

Proofs

$(e \neq e_{\hat{\tilde{\mathrm{p}}}})$
$$\langle \mu\alpha.c \| e \rangle_p \tau \rightsquigarrow_s c_c\, \tau[\alpha := e]$$
$$\langle \mu\alpha.c \| e_{\hat{\tilde{\mathrm{p}}}} \rangle_p \tau \rightsquigarrow_s c_c[e_{\hat{\tilde{\mathrm{p}}}}/\alpha]\, \tau$$

(a fresh)
$$\langle (p_1, p_2) \| e \rangle_p \tau \rightsquigarrow_s \langle p_1 \| \tilde{\mu}a_1.\langle p_2 \| \tilde{\mu}a_2.\langle (a_1, a_2) \| e \rangle \rangle \rangle_p \tau$$

(a fresh)
$$\langle \iota_i(p) \| e \rangle_p \tau \rightsquigarrow_s \langle p \| \tilde{\mu}a.\langle \iota_i(a) \| e \rangle \rangle_p \tau$$

(a fresh)
$$\langle (t, p) \| e \rangle_p \tau \rightsquigarrow_s \langle p \| \tilde{\mu}\check{\tilde{\mathrm{p}}}.\langle t \| \tilde{\mu}x.\langle \check{\tilde{\mathrm{p}}} \| \tilde{\mu}a.\langle (x, a) \| e \rangle \rangle \rangle \rangle_p \tau$$

(y, a fresh)
$$\langle \mathtt{fix}^t_{bx}[p \mid q] \| e \rangle_p \tau \rightsquigarrow_s \langle \mu\hat{\tilde{\mathrm{p}}}.\langle t \| \tilde{\mu}y.\langle a \| \hat{\tilde{\mathrm{p}}} \rangle [a := \mathtt{fix}^y_{bx}[p \mid q]] \rangle \| e \rangle_p \tau$$

(y, a fresh)
$$\langle \mathtt{cofix}^t_{bx}[p] \| e \rangle_p \tau \rightsquigarrow_s \langle \mu\hat{\tilde{\mathrm{p}}}.\langle t \| \tilde{\mu}y.\langle a \| \hat{\tilde{\mathrm{p}}} \rangle \rangle [a := \mathtt{cofix}^y_{bx}[p]] \| e \rangle_p \tau$$
$$\langle V \| e \rangle_p \tau \rightsquigarrow_s \langle V \| e \rangle_e$$

Contexts

$$\langle V \| \alpha \rangle_e \tau[\alpha := e] \tau' \rightsquigarrow_s \langle V \| e \rangle_e \tau[\alpha := e] \tau'$$
$$\langle V \| \tilde{\mu}a.c\tau' \rangle_e \tau \rightsquigarrow_s c_c \tau[a := V] \tau'$$
$$\langle V \| f \rangle_e \tau \rightsquigarrow_s \langle V \| f \rangle_V \tau$$

Values

$$\langle a \| f \rangle_V \tau[a := V] \tau' \rightsquigarrow_s \langle V \| f \rangle_V \tau[a := V] \tau'$$
$$\langle v \| f \rangle_V \tau \rightsquigarrow_s \langle v \| f \rangle_f \tau$$

(b' fresh)
$$\langle a \| f \rangle_V \tau[a = \mathtt{cofix}^t_{bx}[p]] \tau' \rightsquigarrow_s \langle p[t/x][b'/b] \| \tilde{\mu}a.\langle a \| f \rangle \tau' \rangle_p \tau[b' := \lambda y.\mathtt{cofix}^y_{bx}[p]]$$
$$\langle a \| f \rangle_V \tau[a = \mathtt{fix}^0_{bx}[p_0 \mid p_S]] \tau' \rightsquigarrow_s \langle p_0 \| \tilde{\mu}a.\langle a \| f \rangle \tau' \rangle_p \tau$$

(b' fresh)
$$\langle a \| f \rangle_v \tau[a = \mathtt{fix}^{S(t)}_{bx}[p_0 \mid p_S]] \tau' \rightsquigarrow_s \langle p_S[t/x][b'/b] \| \tilde{\mu}a.\langle a \| f \rangle \tau' \rangle_p \tau[b' := \mathtt{fix}^t_{bx}[p_0 \mid p_S]]]$$

Forcing contexts

$$\langle \lambda x.p \| t \cdot e \rangle_f \tau \rightsquigarrow_s \langle \mu\hat{\tilde{\mathrm{p}}}.\langle t \| \tilde{\mu}x.\langle p \| \hat{\tilde{\mathrm{p}}} \rangle \rangle \| e \rangle_p \tau$$

($q \in$ NEF)
$$\langle \lambda a.p \| q \cdot e \rangle_f \tau \rightsquigarrow_s \langle \mu\hat{\tilde{\mathrm{p}}}.\langle q \| \tilde{\mu}a.\langle p \| \hat{\tilde{\mathrm{p}}} \rangle \rangle \| e \rangle_p \tau$$

($q \notin$ NEF)
$$\langle \lambda a.p \| q \cdot e \rangle_f \tau \rightsquigarrow_s \langle q \| \tilde{\mu}a.\langle p \| e \rangle \rangle_p \tau$$
$$\langle \iota_i(V) \| \tilde{\mu}[a_1.c^1 \mid a_2.c^2] \rangle_f \tau \rightsquigarrow_s c^i_c \tau[a_i := V]$$
$$\langle (V_1, V_2) \| \tilde{\mu}(a_1, a_2).c \rangle_f \tau \rightsquigarrow_s c_c \tau[a_1 := V_1][a_2 := V_2]$$
$$\langle (V_t, V) \| \tilde{\mu}(x, a).c \rangle_f \tau \rightsquigarrow_s (c[V_t/x])_c \tau[a := V]$$
$$\langle \mathtt{refl} \| \tilde{\mu}_{=}.c \rangle_f \tau \rightsquigarrow_s c_c \tau$$

Terms

$$\langle tu \| \pi \rangle_t \tau \rightsquigarrow_s \langle t \| u \cdot \pi \rangle_t \tau$$

(x fresh)
$$\langle S(t) \| \pi \rangle_t \tau \rightsquigarrow_s \langle t \| \tilde{\mu}x.\langle S(x) \| \pi \rangle \rangle_t$$

(x, a fresh)
$$\langle \mathtt{wit}\ p \| \pi \rangle_t \tau \rightsquigarrow_s \langle p \| \tilde{\mu}(x, a).\langle x \| \pi \rangle \rangle_p \tau$$

($t \notin V_t$)
$$\langle \mathtt{rec}^t_{xy}[t_0 \mid t_S] \| \pi \rangle_t \tau \rightsquigarrow_s \langle t \| \tilde{\mu}z.\langle \mathtt{rec}^z_{xy}[t_0 \mid t_S] \| \pi \rangle \rangle_t \tau$$
$$\langle \mathtt{rec}^0_{xy}[t_0 \mid t_S] \| \pi \rangle_t \tau \rightsquigarrow_s \langle t_0 \| \pi \rangle_t \tau$$
$$\langle \mathtt{rec}^{S(V_t)}_{xy}[t_0 \mid t_S] \| \pi \rangle_t \tau \rightsquigarrow_s \langle t_S[V_t/x][\mathtt{rec}^{V_t}_{xy}[t_0 \mid t_S]/y] \| \pi \rangle_t \tau$$
$$\langle V_t \| \pi \rangle_t \tau \rightsquigarrow_s \langle V_t \| \pi \rangle_\pi \tau$$
$$\langle \lambda x.t \| u \cdot \pi \rangle_\pi \tau \rightsquigarrow_s \langle u \| \tilde{\mu}x.\langle t \| \pi \rangle \rangle_t \tau$$
$$\langle V_t \| \tilde{\mu}x.c_t \rangle_\pi \tau \rightsquigarrow_s (c_t \tau)[V_t/x]$$
$$\langle V_t \| \tilde{\mu}x.c \rangle_\pi \tau \rightsquigarrow_s (c_p \tau)[V_t/x]$$

Figure 7. Small-step reduction rules

Proposition 3.1 (Subject Reduction). *The small-step reduction rules satisfy subject reduction.*

Proof. The proof is again a tedious induction on the reduction $\rightsquigarrow_s$. There is almost nothing new in comparison with the cases for the big-step reduction rules: the cases for reduction of terms are straightforward, as well as the administrative reductions changing the focus on a command. We only give the case for the reduction of pairs (t,p). The reduction rule is:

$$\langle (t,p) \| e \rangle_p \tau \rightsquigarrow_s \langle p \| \tilde{\mu} \check{\mathfrak{p}}. \langle t \| \tilde{\mu} x. \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle \rangle \rangle_p \tau$$

Consider a typing derivation for the command on the left-hand side, which is of the shape (we omit the rule (l) and the store for conciseness):

$$\dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma \vdash^\sigma t : T} \quad \dfrac{\Pi_p}{\Gamma \vdash^\sigma p : A[t/x]}}{\Gamma \vdash^\sigma (t,p) : \exists x^T.A}\,(\exists_r) \quad \dfrac{\Pi_e}{\Gamma \vdash^\sigma e : (\exists x^T.A)^{\perp\!\!\!\perp}}}{\Gamma \vdash^\sigma \langle (t,p) \| e \rangle}\,(\textsc{Cut})$$

Then we can type the command on the right-hand side with the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_{(x,a)}}{\Gamma, x : T, a : A[x] \vdash^\sigma \langle (x,a) \| e \rangle : A[x]^{\perp\!\!\!\perp}}\,(\textsc{Cut}) \quad \Pi_e}{\Gamma, x : T \vdash^\sigma \tilde{\mu} a. \langle (x,a) \| e \rangle : A[x]^{\perp\!\!\!\perp}}\,(\tilde{\mu}) \quad A[t] = (\{x|t\})(A[x])}{\Gamma, \check{\mathfrak{p}} : A[t], x : T \vdash_d \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle ; \sigma\{x|t\}}\,(\textsc{Cut}_d) \quad \dfrac{}{\Gamma, \check{\mathfrak{p}} : A[t/x] \vdash_d \tilde{\mu} x. \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle : T ; \sigma\{\cdot|t\}}\,(\tilde{\mu}_x)}{\Pi'_t \quad \Gamma, \check{\mathfrak{p}} : A[t] \vdash \langle t \| \tilde{\mu} x. \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle \rangle ; \sigma}\,(\textsc{Cut}_d) \quad \dfrac{}{\Gamma \vdash^\sigma \tilde{\mu} \check{\mathfrak{p}}. \langle t \| \tilde{\mu} x. \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle \rangle : A[t]^{\perp\!\!\!\perp}}\,(\tilde{\mu}\check{\mathfrak{p}})}{\Pi_p \quad \Gamma \vdash^\sigma \langle p \| \tilde{\mu} \check{\mathfrak{p}}. \langle t \| \tilde{\mu} x. \langle \check{\mathfrak{p}} \| \tilde{\mu} a. \langle (x,a) \| e \rangle \rangle \rangle \rangle_p}\,(\textsc{Cut})$$

where $\Pi_{(x,a)}$ is as expected. □

Proposition 3.2. *If a closure $c\tau$ normalizes for the reduction $\rightsquigarrow_s$, then it normalizes for $\rightarrow$.*

Proof. By contraposition, one proves that if a command $c\tau$ produces an infinite number of steps for the reduction $\rightarrow$, then it does not normalize for $\rightsquigarrow_s$ either. This is proved by showing by induction on the reduction $\rightarrow$ that each step, except for the contextual reduction of terms, is reflected in at least on for the reduction $\rightsquigarrow_s$. The rules for term reductions require a separate treatment, which is really not interesting at this point. We claim that the reduction of terms, which are usual simply-typed λ-terms, is known to be normalizing anyway and does not deserve that we spend another page proving it in this particular setting. □

E Adequacy

Proposition 4.5. *The set $\perp\!\!\!\perp_{\Downarrow} = \{ c\tau \in \mathcal{C}_0 : \ c\tau \text{ normalizes} \}$ is a pole.*

Proof. The first two conditions are already verified for the $\overline{\lambda}_{[l v \tau \star]}$-calculus [19]. The third one is straightforward, since if a closure $\langle \mu \hat{\mathfrak{p}}. c \| e \rangle \tau$ is not normalizing, it is easy to verify that $c[e/\hat{\mathfrak{p}}]$ is not normalizing either. Roughly, there is only two possible reduction steps for a command $\langle \mu \hat{\mathfrak{p}}. c \| e \rangle \tau$: either it reduces to $\langle \mu \hat{\mathfrak{p}}. c' \| e \rangle \tau'$, in which case $c[e/\hat{\mathfrak{p}}]\tau$ also reduces to a closure which is almost $(c'\tau')[e/\hat{\mathfrak{p}}]$; or c is of the shape $\langle p \| \hat{\mathfrak{p}} \rangle$ and it reduces to $c[e/\hat{\mathfrak{p}}]\tau$. In both cases, if $\langle \mu \hat{\mathfrak{p}}. c \| e \rangle \tau$ can reduce, so can $c[e/\hat{\mathfrak{p}}]\tau$. The same reasoning allows us to show that if $c[V/\check{\mathfrak{p}}]\tau$ normalizes, then so does $\langle V \| \tilde{\mu} \check{\mathfrak{p}}. c \rangle \tau$ for any value V. □

We give here the complete proof of adequacy of the typing rules with respect to the realizability interpretation, defined in Figure 5.

Proposition 4.11. *The typing rules are adequate with respect to the realizability interpretation. In other words, if Γ is a typing context, $\perp\!\!\!\perp$ a pole, ρ a valuation and τ a store such that $(\tau, \rho) \Vdash \Gamma ; \sigma$, then the following hold:*

1. *If v is a strong value such that $\Gamma \vdash^\sigma v : A$ or $\Gamma \vdash_d v : A ; \sigma$, then $(v|\tau)_\rho \in |A_\rho|_V$.*
2. *If f is a forcing context such that $\Gamma \vdash^\sigma f : A^{\perp\!\!\!\perp}$ or $\Gamma \vdash_d f : A^{\perp\!\!\!\perp} ; \sigma$, then $(f|\tau)_\rho \in \|A_\rho\|_f$.*
3. *If V is a weak value such that $\Gamma \vdash^\sigma V : A$ or $\Gamma \vdash_d V : A ; \sigma$, then $(V|\tau)_\rho \in |A_\rho|_V$.*
4. *If e is a context such that $\Gamma \vdash^\sigma e : A^{\perp\!\!\!\perp}$ or $\Gamma \vdash_d e : A^{\perp\!\!\!\perp} ; \sigma$, then $(e|\tau)_\rho \in \|A_\rho\|_e$.*
5. *If p is a proof term such that $\Gamma \vdash^\sigma p : A$ or $\Gamma \vdash_d p : A ; \sigma$, then $(p|\tau)_\rho \in |A_\rho|_p$.*
6. *If V_t is a term value such that $\Gamma \vdash^\sigma V_t : T$, then $(V_t|\tau)_\rho \in |T_\rho|_{V_t}$.*
7. *If π is a term context such that $\Gamma \vdash^\sigma \pi : T$, then $(\pi|\tau)_\rho \in |T_\rho|_\pi$.*
8. *If t is a term such that $\Gamma \vdash^\sigma t : T$, then $(t|\tau)_\rho \in |T_\rho|_t$.*
9. *If τ' is a store such that $\Gamma \vdash^\sigma \tau' : (\Gamma' ;) \sigma'$, then $(\tau\tau', \rho) \Vdash (\Gamma, \Gamma' ; \sigma\sigma')$.*
10. *If c is a command such that $\Gamma \vdash^\sigma c$ or $\Gamma \vdash_d c ; \sigma$, then $(c\tau)_\rho \in \perp\!\!\!\perp$.*
11. *If $c\tau'$ is a closure such that $\Gamma \vdash^\sigma c\tau'$ or $\Gamma \vdash_d c\tau' ; \sigma$, then $(c\tau\tau')_\rho \in \perp\!\!\!\perp$.*

Proof. The proof is done by induction on the typing derivation such as given in the system extended with the small-step reduction $\rightsquigarrow_s$. Most of the cases correspond to the proof of adequacy for the interpretation of the $\overline{\lambda}_{[l v \tau \star]}$-calculus, so that we only give the most interesting cases. To lighten the notations, we omit the annotation by the valuation ρ whenever it is possible.

• **Case** ($\exists_r$). We recall the typing rule through the decomposition of dependent sums:

$$\frac{\Gamma \vdash^\sigma t : u \in T \quad \Gamma \vdash^\sigma p : A[u/x]}{\Gamma \vdash^\sigma (t,p) : (u \in T \wedge A[u])}$$

By induction hypothesis, we obtain that $(t|\tau) \in |u \in T|_t$ and $(p|\tau) \in |A[u]|_p$. Consider thus any context-in-store $(e|\tau') \in \|u \in T \wedge A[u]\|_e$ such that τ and τ' are compatible, and let us denote by τ_0 the union $\overline{\tau\tau'}$. We have:

$$\langle (t,p) \| e \rangle_p \tau_0 \rightsquigarrow_s \langle p \| \tilde{\mu}\check{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \check{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle \rangle \rangle_p \tau_0$$

so that by anti-reduction, we need to show that $\tilde{\mu}\check{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \check{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle \rangle \in \|A[u]\|_e$. Let us then consider a value-in-store $(V|\tau_0') \in |A[u]|_V$ such that τ_0 and τ_0' are compatible, and let us denote by τ_1 the union $\overline{\tau_0\tau_0'}$. By closure under delimited continuations, to show that $\langle V \| \tilde{\mu}\check{\mathfrak{p}}.\langle t \| \tilde{\mu}x.\langle \check{\mathfrak{p}} \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle \rangle \rangle_p \tau_1$ is in the pole it is enough to show that the closure $\langle t \| \tilde{\mu}x.\langle V \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle \rangle \tau_1$ is in $\perp\!\!\!\perp_,$. Thus it suffices to show that the coterm-in-store $(\tilde{\mu}x.\langle V \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle | \tau_1)$ is in $|u \in T|_\pi$.

Consider a term value-in-store $(V_t|\tau_1') \in |u \in T|_{V_t}$, such that τ_1 and τ_1' are compatible, and let us denote by τ_2 the union $\overline{\tau_1\tau_1'}$. We have:

$$\langle V_t \| \tilde{\mu}x.\langle V \| \tilde{\mu}a.\langle (x,a) \| e \rangle \rangle \rangle \tau_2 \rightsquigarrow_s \langle V \| \tilde{\mu}a.\langle (V_t,a) \| e \rangle \rangle \tau_2 \rightsquigarrow_s \langle (V_t,a) \| e \rangle \tau_2[a := V]$$

It is now easy to check that $((V_t,a)|\tau_2[a := V]) \in |u \in T \wedge A[u]|_V$ and to conclude, using Lemma 4.8 to get $(e|\tau_2[a := V]) \in \|u \in T \wedge A[u]\|_e$, that this closure is finally in the pole.

• **Case** ($\equiv_r$),($\equiv_l$). These cases are direct consequences of Lemma 4.10 since if A, B are two formulas such that $A \equiv B$, in particular $A \equiv_\tau B$ and thus $|A|_\upsilon = |B|_\upsilon$.

• **Case** (`refl`),($=_l$). The case for `refl` is trivial, while it is trivial to show that $(\tilde{\mu}_{=}.\langle p \| e \rangle | \tau)$ is in $\|t = u\|_f$ if $(p|\tau) \in |A[t]|_p$ and $(e|\tau) \in \|A[u]\|_e$. Indeed, either $t \equiv_\tau u$ and thus $A[t] \equiv_\tau A[u]$ (Lemma 4.10, or $t \not\equiv_\tau u$ and $\|t = u\|_f = \Lambda_f^\tau$.

• **Case** ($\forall_r^x$). This case is standard in a call-by-value language with value restriction. We recall the typing rule:

$$\frac{\Gamma \vdash^\sigma \upsilon : A \quad x \notin FV(\Gamma)}{\Gamma \vdash^\sigma \upsilon : \forall x.A} \ (\forall_r^x)$$

The induction hypothesis gives us that $(\upsilon|\tau)_\rho$ is in $|A_\rho|_V$ for any valuation $\rho[x \mapsto t]$. Then for any t, we have $(\upsilon|\tau)_\rho \in \|A_\rho[t/x]\|_f^{\perp\!\!\!\perp\upsilon}$ so that $(\upsilon|\tau)_\rho \in (\bigcap_{t \in \Lambda_t} \|A[t/x]\|_f^{\perp\!\!\!\perp\upsilon})$. Therefore if $(f|\tau')_\rho$ belongs to $\|\forall x.A_\rho\|_f = (\bigcap_{t \in \Lambda_t} \|A[t/x]\|_f^{\perp\!\!\!\perp\upsilon})^{\perp\!\!\!\perp f}$, we have by definition that $(\upsilon|\tau)_\rho \perp\!\!\!\perp (f|\tau')_\rho$.

• **Case** (`ind`). We recall the typing rule:

$$\frac{\Gamma \vdash^\sigma t : \mathbb{N} \quad \Gamma \vdash^\sigma p_0 : A[0/x] \quad \Gamma, x : T, a : A \vdash^\sigma p_S : A[S(x)/x]}{\Gamma \vdash^\sigma \mathtt{fix}_{ax}^t[p_0 \,|\, p_S] : A[t/x]} \ (\mathtt{ind})$$

We want to show that $(\mathtt{fix}_{ax}^t[p_0 \,|\, p_S]|\tau) \in |A[t]|_p$, let us then consider $(e|\tau') \in \|A[t]\|_e$ such that τ and τ' are compatible, and let us denote by τ_0 the union $\overline{\tau\tau'}$. By induction hypothesis, we have[10] $t \in |t \in \mathbb{N}|_t$ and we have:

$$\langle \mathtt{fix}_{bx}^t[p_0 \,|\, p_S] \| e \rangle_p \tau_0 \rightsquigarrow_s \langle \mu\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}y.\langle a \| \hat{\mathfrak{p}} \rangle [a := \mathtt{fix}_{bx}^y[p_0 \,|\, p_S]] \rangle \| e \rangle_p \tau_0$$

so that by anti-reduction and closure under delimited continuations, it is enough to show that the coterm-in-store $(\tilde{\mu}y.\langle a \| e \rangle [a := \mathtt{fix}_{bx}^y[p_0 \,|\, p_S]] | \tau_0)$ is in $|t \in \mathbb{N}|_\pi$. Let us then consider $(V_t|\tau_0') \in |t \in \mathbb{N}|_{V_t}$ such that τ_0 and τ_0' are compatible, and let us denote by τ_1 the union $\overline{\tau_0\tau_0'}$. By definition, $V_t = S^n(0)$ for some $n \in \mathbb{N}$ and $t \equiv_{\tau_1} S^n(0)$, and we have:

$$\langle S^n(0) \| \tilde{\mu}y.\langle a \| e \rangle [a := \mathtt{fix}_{bx}^y[p_0 \,|\, p_S]] \rangle \tau_1 \rightsquigarrow_s \langle a \| e \rangle \tau_1 [a := \mathtt{fix}_{bx}^{S^n(0)}[p_0 \,|\, p_S]]$$

We conclude by showing by induction on the natural numbers that for any $n \in N$, the value-in-store $(a|\tau_1[a := \mathtt{fix}_{bx}^{S^n(0)}[p_0 \,|\, p_S]])$ is in $|A[S^n(0)]|_V$. Let us consider $(f|\tau_1') \in \|A[S^n(0)]\|_f$ such that the store $\tau_1[a := \mathtt{fix}_{bx}^{S^n(0)}[p_0 \,|\, p_S]]$ and τ_1' are compatible, and let us denote by $\tau_2[a := \mathtt{fix}_{bx}^{S^n(0)}[p_0 \,|\, p_S]]\tau_2'$ their union.

- If $n = 0$, we have:

$$\langle a \| f \rangle \tau_2 [a := \mathtt{fix}_{bx}^0[p_0 \,|\, p_S]] \tau_2' \rightsquigarrow_s \langle p_0 \| \tilde{\mu}a.\langle a \| f \rangle \tau_2' \rangle \tau_2$$

  We conclude by anti-reduction and the induction hypothesis for p_0, since it is easy to show that $(\tilde{\mu}a.\langle a \| f \rangle \tau_2' | \tau_2) \in \|A[0]\|_e$.
- If $n = S(m)$, we have:

$$\langle a \| f \rangle \tau_2 [a := \mathtt{fix}_{bx}^{S(S^m(0))}[p_0 \,|\, p_S]] \tau_2' \rightsquigarrow_s \langle p_S[S^m(0)/x][b'/b] \| \tilde{\mu}a.\langle a \| f \rangle \tau_2' \rangle_p \tau_2 [b' := \mathtt{fix}_{bx}^{S^m(0)}[p_0 \,|\, p_S]]$$

  Since we have by induction that $(b'|\tau_2[b' := \mathtt{fix}_{bx}^{S^m(0)}[p_0 \,|\, p_S]])$ is in $|A[S^m(0)]|_V$, we can conclude by anti-reduction, using the induction hypothesis for p_S and the fact that $(\tilde{\mu}a.\langle a \| f \rangle \tau_2' | \tau_2)$ belongs to $\|A[S(S^m(0))]\|_e$.

[10] Recall that any term t of type T can be given the type $t \in T$.

- **Case** (cofix). We recall the typing rule:

$$\frac{\Gamma \vdash^{\sigma} t : T \quad \Gamma, x : T, b : \forall y^{T}.X(y) \vdash^{\sigma} p : A \quad X \text{ positive in } A \quad X \notin FV(\Gamma)}{\Gamma \vdash^{\sigma} \mathtt{cofix}^{t}_{bx}[p] : \nu^{t}_{Xx}A} \ (\mathtt{cofix})$$

We want to show that $(\mathtt{cofix}^{t}_{bx}[p]|\tau) \in |\nu^{t}_{Xx}A|_{p}$, let us then consider $(e|\tau') \in \|\nu^{t}_{Xx}A\|_{e}$ such that τ and τ' are compatible, and let us denote by τ_0 the union $\overline{\tau\tau'}$. By induction hypothesis, we have $t \in |t \in T|_{t}$ and we have:

$$\langle \mathtt{cofix}^{t}_{bx}[p] \| e \rangle_{p} \tau_0 \rightsquigarrow_{s} \langle \mu\hat{\mathfrak{p}}.\langle t \| \tilde{\mu}y.\langle a \| \hat{\mathfrak{p}} \rangle [a := \mathtt{cofix}^{y}_{bx}[p]] \rangle \| e \rangle_{p} \tau_0$$

so that by anti-reduction and closure under delimited continuations, it is enough to show that the coterm-in-store $(\tilde{\mu}y.\langle a \| e \rangle [a := \mathtt{cofix}^{y}_{bx}[p]] | \tau_0)$ is in $|t \in \mathbb{N}|_{\pi}$. Let us then consider $(V_t|\tau'_0) \in |t \in T|_{V_t}$ such that τ_0 and τ'_0 are compatible, and let us denote by τ_2 the union $\overline{\tau_0\tau'_0}$. We have:

$$\langle V_t \| \tilde{\mu}y.\langle a \| e \rangle [a := \mathtt{cofix}^{y}_{bx}[p]] \rangle \tau_1 \rightsquigarrow_{s} \langle a \| e \rangle \tau_1 [a := \mathtt{cofix}^{V_t}_{bx}[p]]$$

It suffices to show now that the value-in store $(a|\tau_1[a := \mathtt{cofix}^{V_t}_{bx}[p]])$ is in $|\nu^{V_t}_{Xx}A|_{V}$. By definition, we have:

$$|\nu^{V_t}_{Xx}A|_{V} = (\bigcup_{n\in\mathbb{N}} \|F^{n}_{A,V_t}\|_{f})^{\perp\!\!\!\perp V} = \bigcap_{n\in\mathbb{N}} \|F^{n}_{A,V_t}\|^{\perp\!\!\!\perp V}_{f} = \bigcap_{n\in\mathbb{N}} |F^{n}_{A,V_t}|_{V}$$

We conclude by showing by induction on the natural numbers that for any $n \in N$ and any V_t, the value-in-store $(a|\tau_1[a := \mathtt{cofix}^{V_t}_{bx}[p]])$ is in $|F^{n}_{A,V_t}|_{V}$.

The case $n = 0$ is trivial since $|F^{0}_{A,V_t}|_{V} = |\top|_{V} = \Lambda^{\tau}_{V}$. Let then n be an integer and any V_t be a term value. Let us consider $(f|\tau'_1) \in \|F^{n+1}_{A,V_t}A\|_{f}$ such that $\tau_1[a := \mathtt{cofix}^{V_t}_{bx}[p]]$ and τ'_1 are compatible, and let us denote by $\tau_2[a := \mathtt{cofix}^{V_t}_{bx}[p]]\tau'_2$ their union. By definition, we have:

$$\langle a \| f \rangle \tau_2 [a := \mathtt{cofix}^{V_t}_{bx}[p]] \tau'_2 \rightsquigarrow_{s} \langle p[V_t/x][b'/b] \| \tilde{\mu}a.\langle a \| f \rangle \tau'_2 \rangle \tau_2 [b' := \lambda y.\mathtt{cofix}^{y}_{bx}[p]]$$

It is straightforward to check, using the induction hypothesis for n, that $(b'|\tau_2[b' := \lambda y.\mathtt{cofix}^{y}_{bx}[p]])$ is in $|\forall y.y \in T \to F^{n}_{A,y}|_{V}$. Thus we deduce by induction hypothesis for p, denoting by S the function $t \mapsto \|F^{n}_{A,t}\|_{f}$, that:

$$(p[V_t/x][b'/b]|\tau_2[b' := \lambda y.\mathtt{cofix}^{y}_{bx}[p]]) \in |A[V_t/x][\dot{S}/X]|_{p} = |A[V_t/x][F^{n}_{A,y}/X(y)]|_{p} = |F^{n+1}_{A,V_t}|_{p}$$

It only remains to show that $(\tilde{\mu}a.\langle a \| f \rangle \tau'_2 | \tau_2) \in \|F^{n+1}_{A,V_t}\|_{e}$, which is trival from the hypothesis for f. □

F About the interpretation of coinductive formulas

While our realizability interpretation finally gave us a proof of normalization and soundness for dLPA$^\omega$, it has two aspects that we could find unsatisfactory. First, regarding the small-step reduction system, one could have expected the lowest level of interpretation to be v instead of f. Moreover, if we observe our definition, we notice that most of the cases of $\|\cdot\|_f$ are in fact defined by orthogonality to a subset of strong values. Indeed, except for coinductive formulas, we could indeed have defined instead an interpretation $|\cdot|_v$ of formulas at level v and then the interpretation $\|\cdot\|_f$ by orthogonality:

$$
\begin{array}{rcl}
|\bot|_v & \triangleq & \emptyset \\
|t = u|_v & \triangleq & \begin{cases} \mathtt{refl} & \text{if } t \equiv u \\ \emptyset & \text{otherwise} \end{cases} \\
|p \in A|_v & \triangleq & \{(v|\tau) \in |A|_v : v \equiv_\tau p\} \\
|T \to B|_v & \triangleq & \{(\lambda x.p|\tau) : \forall V_t \tau', \tau \diamond \tau' \wedge (V_t|\tau') \in |T|_V \Rightarrow (p[V_t/x]|\overline{\tau\tau'}) \in |B|_p\} \\
|A \to B|_v & \triangleq & \{(\lambda a.p|\tau) : \forall V \tau', \tau \diamond \tau' \wedge (V|\tau') \in |A|_V \Rightarrow (p|\overline{\tau\tau'}[a := V]) \in |B|_p\} \\
|T \wedge A|_v & \triangleq & \{((V_t, V)|\tau) : (V_t|\tau) \in |T|_{V_t} \wedge (V|\tau) \in |A_2|_V\} \\
|A_1 \wedge A_2|_v & \triangleq & \{((V_1, V_2)|\tau) : (V_1|\tau) \in |A_1|_V \wedge (V_2|\tau) \in |A_2|_V\} \\
|A_1 \vee A_2|_v & \triangleq & \{(\iota_i(V)|\tau) : (V|\tau) \in |A_i|_V\} \\
|\exists x.A|_v & \triangleq & \bigcup_{t \in \Lambda_t} |A[t/x]|_v \\
|\forall x.A|_v & \triangleq & \bigcap_{t \in \Lambda_t} |A[t/x]|_v \\
|\forall a.A|_v & \triangleq & \bigcap_{p \in \Lambda_p} |A[p/x]|_v \\
\|A\|_f & \triangleq & \{(f|\tau) : \forall v \tau', \tau \diamond \tau' \wedge (v|\tau') \in |A|_v \Rightarrow (v|\tau') \perp\!\!\!\perp (F|\tau)\}
\end{array}
$$

If this definition is somewhat more natural, it poses a problem for the definition of coinductive formulas. Indeed, there is a priori no strong value in the orthogonal of $\|\nu^t_{fv} A\|_f$, which is:

$$(\|\nu^t_{fv} A\|_f)^{\perp\!\!\!\perp_v} = (\bigcup_{n \in \mathbb{N}} \|F^n_{A,t}\|_f)^{\perp\!\!\!\perp_v} = \bigcap_{n \in \mathbb{N}} (\|F^n_{A,t}\|_f)^{\perp\!\!\!\perp_v})$$

For instance, consider again the case of a stream of type $\nu^0_{fx} A(x) \wedge f(S(x)) = 0$, a strong value in the intersection should be in every $|A(0) \wedge (A(1) \wedge \ldots (A(n) \wedge \top) \ldots)|_v$, which is not possible due to the finiteness of terms[11] Thus the definition $|\nu^t_{fv} A|_v \triangleq \bigcap_{n \in \mathbb{N}} |F^n_{A,t}|_v$ would give $|\nu^t_{fx} A|_v = \emptyset = |\bot|_v$.

Interestingly, and this is the second aspect that we do not find completely satisfactory, we could have define instead the truth value of coinductive formulas directly by :

$$|\nu^t_{fx} A|_v \triangleq |A[t/x][\nu^y_{fx} A / f(y) = 0]|_v$$

Let us sketch the proof that such a definition is well-founded. We consider the language of formulas without coinductive formulas and extended with formulas of the shape $X(t)$ where $X, Y, \ldots$ are parameters. At level v, closed formulas are interpreted by sets of strong values-in-store $(v|\tau)$, and as we already observed, these sets are besides closed under the relation $\equiv_\tau$ along their component τ. If $A(x)$ is a formula whose only free variable is x, the function which associates to each term t the set $|A(t)|_v$ is thus a function from Λ_t to $\mathcal{P}(\Lambda^\tau_v)_{\equiv_\tau}$, let us denote the set of these functions by $\mathscr{L}$.

Proposition F.1. *The set $\mathscr{L}$ is a complete lattice with respect to the order $\leq_{\mathscr{L}}$ defined by:*

$$F \leq_{\mathscr{L}} G \triangleq \forall t \in \Lambda_t . F(t) \subseteq G(t)$$

Proof. Trivial since the order on functions is defined pointwise and the co-domain $\mathcal{P}(\Lambda^\tau_v)$ is itself a complete lattice. □

We define valuations, which we write ρ, as functions mapping each parameter X to a function $\rho(X) \in \mathscr{L}$. We then define the interpretations $|A|^\rho_v, \|A\|^\rho_f, \ldots$ of formulas with parameters exactly as above with the additional rule[12]:

$$|X(t)|^\rho_v \triangleq \{(v|\tau) \in \rho(X)(t)\}$$

Let us fix a formula A which has one free variable x and a parameter X such that sub-formulas of the shape $X\ t$ only occur in positive positions in A.

Lemma F.2. *Let $B(x)$ is a formula without parameters whose only free variable is x, and let ρ be a valuation which maps X to the function $t \mapsto |B(t)|_v$. Then $|A|^\rho_v = |A[B(t)/X(t)]|_v$*

[11] Yet, it might possible to consider interpretation with infinite proof terms, the proof of adequacy for proofs and contexts (which are finite) will still work exactly the same. However, another problem will arise for the adequacy of the `cofix` operator. Indeed, with the interpretation above, we would obtain the inclusion $\bigcup_{n \in \mathbb{N}} (\|F^n_{A,t}\|_f) \subset (\bigcap_{n \in \mathbb{N}} |F^n_{A,t}|_t)^{\perp\!\!\!\perp_f} = \|\nu^t_{fx} A\|_f$ which is strict in general. By orthogonality, this gives us that $|\nu^t_{fx} A|_V \subseteq \bigcup_{n \in \mathbb{N}} (\|F^n_{A,t}\|_f))^{\perp\!\!\!\perp_V}$, while the proof of adequacy only proves that $(a|\tau[a := \mathtt{cofix}^t_b[x]p])$ belongs to the latter set.

[12] Observe that this rule is exactly the same as in the previous section (see Figure 5).

Proof. By induction on the structure of A, all cases are trivial, and this is true for the basic case $A \equiv X(t)$:

$$|X(t)|_v^\rho = \rho(X)(t) = |B(t)|_v$$

□

Let us now define φ_A as the following function:

$$\varphi_A : \left\{ \begin{array}{ccc} \mathscr{L} & \to & \mathscr{L} \\ F & \mapsto & t \mapsto |A[t/x]|_v^{[X \mapsto F]} \end{array} \right.$$

Proposition F.3. *The function φ_A is monotone.*

Proof. By induction on the structure of A, where X can only occur in positive positions. The case $|X(t)|_v$ is trivial, and it is easy to check that truth values are monotonic with respect to the interpretation of formulas in positive positions, while falsity values are anti-monotonic. □

We can thus apply Knaster-Tarski theorem to φ_A, and we denote by $\mathtt{gfp}(\varphi_A)$ its greatest fixpoint. We can now define:

$$|\nu^t_{Xx} A|_v \triangleq \mathtt{gfp}(\varphi_A)(t)$$

This definition satisfies the expected equality:

Proposition F.4. *We have:*

$$|\nu^t_{Xx} A|_v = |A[t/x][\nu^y_{Xx} A/X(y)]|_v$$

Proof. Observe first that by definition, the formula $B(z) = |\nu^z_{Xx} A|_v$ satisfies the hypotheses of Lemma F.2 and that $\mathtt{gfp}(\varphi_A) = t \mapsto B(t)$. Then we can deduce :

$$|\nu^t_{Xx} A|_v = \mathtt{gfp}(\varphi_A)(t) = \varphi_A(\mathtt{gfp}(\varphi_A))(t) = |A[t/x]|_v^{[X \mapsto \mathtt{gfp}(\varphi_A)]} = |A[t/x][\nu^y_{Xx} A/X(y)]|_v$$

□

Back to the original language, it only remains to define $|\nu^t_{fx} A|_v$ as the set $|\nu^t_{Xx} A[X(y)/f(y) = 0]|_v$ that we just defined. This concludes our proof that the interpretation of coinductive formulas through the equation in Proposition F.4 is well-founded.

We could also have done the same reasoning with the interpretation from the previous section, by defining $\mathscr{L}$ as the set of functions from Λ_t to $\mathcal{P}(\Lambda_f^\tau)_{\equiv_\tau}$. The function φ_A, which is again monotonic, is then:

$$\varphi_A : \left\{ \begin{array}{ccc} \mathscr{L} & \to & \mathscr{L} \\ F & \mapsto & t \mapsto |A[t/x]|_v^{[X \mapsto F]} \end{array} \right.$$

We recognize here the definition of the formula $F^n_{A,t}$. Defining f^0 as the function $t \mapsto \|\top\|_f$ and $f^{n+1} \triangleq \varphi_A(f^n)$ we have:

$$\forall n \in \mathbb{N}, \|F^n_{A,t}\|_f = f^n(t) = \varphi^n_A(f^0)(t)$$

However, in both cases (defining primitively the interpretation at level v or f), this definition does not allow us to prove[13] the adequacy of the (`cofix`) rule. In the case of an interpretation defined at level f, the best that we can do is to show that for any $n \in \mathbb{N}$, f^n is a post-fixpoint since for any term t, we have:

$$f^n(t) = \|F^n_{A,t}\|_f \subseteq \|F^{n+1}_{A,t}\|_f = f^{n+1}(t) = \varphi_A(f^n)(t)$$

With $\|\nu^t_{fx} A\|_f$ defined as the greatest fixpoint of φ_A, for any term t and any $n \in \mathbb{N}$ we have the inclusion $f^n(t) \subseteq \mathtt{gfp}(\varphi_A)(t) = \|\nu^t_{fx} A\|_f$ and thus:

$$\bigcup_{n \in \mathbb{N}} \|F^n_{A,t}\|_f = \bigcup_{n \in \mathbb{N}} f^n(t) \subseteq \|\nu^t_{fx} A\|_f$$

By orthogonality, we get:

$$|\nu^t_{fx} A|_V \subseteq \bigcap_{n \in \mathbb{N}} |F^n_{A,t}|_V$$

and thus our proof of adequacy from the last section is not enough to conclude that $\mathtt{cofix}^t_{bx}[p] \in |\nu^t_{fx} A|_p$. For this, we would need to prove that the inclusion is an equality. An alternative to this would be to show that the function $t \mapsto \bigcup_{n \in \mathbb{N}} \|F^n_{A,t}\|_f$ is a fixpoint for φ_A. In that case, we could stick to this definition and happily conclude that it satisfies the equation:

$$\|\nu^t_{Xx} A\|_f = \|A[t/x][\nu^y_{Xx} A/X(y)]\|_f$$

This would be the case if the function φ_A was Scott-continuous on $\mathscr{L}$ (which is a dcpo), since we could then apply Kleene fixed-point theorem[14] to prove that $t \mapsto \bigcup_{n \in \mathbb{N}} \|F^n_{A,t}\|_f$ is the stationary limit of $\varphi^n_A(f_0)$. However, φ_A is not Scott-continuous[15] (the definition of falsity values involves double-orthogonal sets which do not preserve supremums), and this does not apply.

[13] To be honest, we should rather say that we could not manage to find a proof, and that we would welcome any suggestion from insightful readers.

[14] In fact, Cousot and Cousot proved a constructive version of Kleene fixed-point theorem which states that without any continuity requirement, the transfinite sequence $(\varphi^\alpha_A(f^0))_{\alpha \in O_n}$ is stationary [5]. Yet, we doubt that the gain of the desired equality is worth a transfinite definition of the realizability interpretation.

[15] In fact, this is nonetheless a good news about our interpretation. Indeed, it is well-know that the more "regular" a model is, the less interesting it is. For instance, Streicher showed that the realizability model induced by Scott domains (using it as a realizability structure) was not only a forcing model by also equivalent to the ground model.