A sequent calculus with dependent types for classical arithmetic

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

Abstract

In a recent paper [11], Herbelin developed dPA$^\omega$, a calculus 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 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 $(a_0, a_1, \dots)$) and of lazy evaluation with sharing (for these coinductive objects).

Elaborating 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 that we developed to prove the normalization of classical call-by-need [20]. On the other hand, we benefit from dL$_{\hat{\text{tp}}}$, a classical sequent calculus with dependent types in which type safety is ensured by using delimited continuations together with a syntactic restriction [19]. 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 Curry-Howard, dependent choice, classical arithmetic, side effects, dependent types, classical realizability, sequent calculus

1 Introduction

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

Dependent types are one of the key features of Martin-Löf's type theory [17], allowing formulas to refer to terms. Notably, the existential quantification rule is defined so that a proof term of type $\exists x^A.B$ is a pair (t,p) where t—the *witness*—is of type A, while p—the *proof*—is of type $B[t/x]$. Dually, the theory enjoys two elimination rules: one with a destructor `wit` to extract the witness, the second one with a destructor `prf` to extract the proof. This allows for a simple and constructive proof of the full axiom of choice [17]:

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

This term is nothing more that an implementation of Brouwer-Heyting-Kolomogoroff interpretation of the axiom of choice [12]: given a proof H of $\forall x^A.\exists y^B.P(x,y)$, it constructs a choice function which simply maps any x to the witness of Hx, while the proof that this function is sound w.r.t. P returns the corresponding certificate.

Yet, this approach deeply relies on the constructivity of the theory. We present here a continuation of Herbelin's works [11], who proposed a way of scaling up Martin-Löf's proof to classical logic. The first idea is to restrict the dependent types to the fragment of *negative-elimination-free* proofs (NEF) which somewhat only contains constructive proofs behaving as values. The second idea is to represent a countable universal quantification as an infinite conjunction. This allows us to internalize 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 $H_\infty = (H0, H1, \dots)$ and select its n^{th}-element with some function `nth`. Then, one might wish that:

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

could stand for a proof for $AC_\mathbb{N}$. One problem is that even if we were effectively able to build such a term, H_∞ might still contain some classical proofs. Therefore, two copies of Hn might end up behaving differently according to the contexts in which they are executed, and thus returning 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:

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

In words, the term H_∞ is now shared between all the places which may require some of its components.

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

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

The stream is, at the level of formulas, an inhabitant of a coinductively defined infinite conjunction $\nu^0_{Xn}(\exists y.P(n,y)) \wedge X(n+1)$. Since 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 (CPS), but translations appeared to be hard to obtain for dPA$^\omega$ as such. In addition to the difficulties caused by control operators and co-fixpoints, dPA$^\omega$ reduction system is defined in a natural deduction fashion, with contextual rules where

LICS '18, July 9–12, 2018, Oxford, United Kingdom
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5583-4/18/07...$15.00
https://doi.org/10.1145/3209108.3209199

the contexts involved can be of arbitrary depth. This kind of rules are indeed difficult to faithfully translate through a CPS.

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, sequent calculus presentations of a calculus usually provides good intermediate steps for CPS translations [8, 21, 22] since they enforce a decomposition of the reduction system into finer-grain rules. To this aim, we first handled separately the difficulties peculiar to the definition of such a calculus: on the one hand, we proved with Herbelin the normalization of a calculus with control operators and lazy evaluation [20]; on the other hand, we defined a classical sequent calculus with dependent types [19]. By combining the techniques developed in these frameworks, we finally manage to define dLPA$^\omega$, which we present here and prove to be normalizing.

1.3 Realizability interpretation of classical call-by-need

In the call-by-need evaluation strategy, the substitution of a variable is delayed until knowing whether the argument is needed. To this end, Ariola *et al.* [1] proposed the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, a variant of Curien-Herbelin's $\lambda\mu\tilde{\mu}$-calculus [6] in which substitutions 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 obtained an untyped CPS translation for the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus. By pushing one step further this methodology, we showed with Herbelin how to obtain a realizability interpretation *à la* Krivine for this framework [20]. The main idea, in contrast to usual models of Krivine realizability [14], is that realizers are defined as pairs of a term and a substitution. The adequacy of the interpretation directly provided us with a proof of normalization, and we shall follow here the same methodology 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 language with dependent types compatible with a CPS. 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 [19], 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, we decorated typing derivations with a list of dependencies to ensure subject reduction. Besides, the soundness of the calculus was justified by means of a CPS translation taking the dependencies into account. The very definition of the translation constrained us to use delimited continuations in the calculus when reducing dependently typed terms. At the same time, this unveiled the need for the syntactic restriction of dependencies to the *negative-elimination-free* fragment as in [11]. Additionally, we showed how to relate our calculus to a similar system by Lepigre [16], whose consistency is proved by means of a realizability interpretation. In the present 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 in an abstract machine for which only the term or the context needs to be analyzed in order to decide whether the rule can be applied.

1.5 Contributions of the paper

The main contributions of this paper can be stated as follows. First, we define dLPA$^\omega$ (Section 2), 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 dL$_{\hat{\mathfrak{tp}}}$ [19] and the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus [1, 20] extended with the expressive power of dPA$^\omega$ [11]. Second, we prove the properties of normalization and soundness for dLPA$^\omega$ thanks to a realizability interpretation *à la* Krivine, which we obtain by applying Danvy's methodology of semantic artifacts (Sections 3 and 4). Lastly, dLPA$^\omega$ incidentally provides us with a direct proofs-as-programs interpretation of classical arithmetic with dependent choice, as sketched in [11].

This paper is partially taken from the Chapter 8 of the author's PhD thesis [18]. For more detailed proofs, we refer the reader to the appendices of the version available at: https://hal.inria.fr/hal-01703526.

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$_{\hat{\mathfrak{tp}}}$ [19], extended with the expressive power of dPA$^\omega$ [11] and with explicit stores as in the $\overline{\lambda}_{[l\upsilon\tau\star]}$-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, matching the syntax of 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:

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

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 $\texttt{rec}^t_{xy}[t_0 \mid t_S]$, 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 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 $\lambda x.p$ inhabiting the type $\forall x^T.A$ together with their dual, stacks $t \cdot e$ where e is a context whose type might be dependent in t; functions $\lambda a.p$ which inhabit the dependent product type $\Pi a : A.B$, and, dually, stacks $q \cdot e$, where e is a context whose type might be dependent in q; a proof term $\texttt{refl}$ which is the proof of atomic equalities $t = t$ and a destructor $\tilde{\mu}^=.c$ which allows us to type the command c modulo an equality of terms; operators $\texttt{fix}^t_{ax}[p_0 \mid p_S]$ and $\texttt{cofix}^t_{bx}[p]$, as in dPA$^\omega$, for inductive and coinductive reasoning; delimited continuations through proofs $\mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}}$ and the context $\hat{\mathfrak{tp}}$; a distinguished context $[]$ of type $\bot$, which allows us to reason ex-falso.

Closures $l ::= c\tau$

Commands $c ::= \langle p \| e \rangle$

Proof terms $p, q ::= a \mid \iota_i(p) \mid (p, q) \mid (t, p) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl} \mid \mathtt{fix}^t_{ax}[p_0 \mid p_S] \mid \mathtt{cofix}^t_{bx}[p] \mid \mu\alpha.c \mid \mu\hat{\mathtt{tp}}.c_{\hat{\mathtt{tp}}}$

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

Terms $t, u ::= x \mid 0 \mid S(t) \mid \mathtt{rec}^t_{xy}[t_0 \mid t_S] \mid \lambda x.t \mid t\ u \mid \mathtt{wit}\ p$

Terms values $V_t ::= x \mid S^n(0) \mid \lambda x.t$

Stores $\tau ::= \varepsilon \mid \tau[a := p_\tau] \mid \tau[\alpha := e]$

Storables $p_\tau ::= V \mid \mathtt{fix}^{V_t}_{ax}[p_0 \mid p_S] \mid \mathtt{cofix}^{V_t}_{bx}[p]$

Contexts $e ::= f \mid \alpha \mid \tilde{\mu}a.c\tau$

Forcing contexts $f ::= [] \mid \tilde{\mu}[a_1.c_1 \mid a_2.c_2] \mid \tilde{\mu}(a_1, a_2).c \mid \tilde{\mu}(x, a).c \mid t \cdot e \mid p \cdot e \mid \tilde{\mu}{=}.c$

Delimited continuations $c_{\hat{\mathtt{tp}}} ::= \langle p_N \| e_{\hat{\mathtt{tp}}} \rangle \mid \langle p \| \hat{\mathtt{tp}} \rangle$

$e_{\hat{\mathtt{tp}}} ::= \tilde{\mu}a.c_{\hat{\mathtt{tp}}}\tau \mid \tilde{\mu}[a_1.c_{\hat{\mathtt{tp}}} \mid a_2.c'_{\hat{\mathtt{tp}}}] \mid \tilde{\mu}(a_1, a_2).c_{\hat{\mathtt{tp}}} \mid \tilde{\mu}(x, a).c_{\hat{\mathtt{tp}}}$

NEF $c_N ::= \langle p_N \| e_N \rangle$ $\quad e_N ::= \star \mid \tilde{\mu}[a_1.c_N \mid a_2.c'_N] \mid \tilde{\mu}a.c_N\tau \mid \tilde{\mu}(a_1, a_2).c_N \mid \tilde{\mu}(x, a).c_N$

$p_N, q_N ::= a \mid \iota_i(p_N) \mid (p_N, q_N) \mid (t, p_N) \mid \lambda x.p \mid \lambda a.p \mid \mathtt{refl} \mid \mathtt{fix}^t_{ax}[p_N \mid q_N] \mid \mathtt{cofix}^t_{bx}[p_N] \mid \mu\star.c_N \mid \mu\hat{\mathtt{tp}}.c_{\hat{\mathtt{tp}}}$

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

As in dL$_{\hat{\mathtt{tp}}}$, 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 $c_{\hat{\mathtt{tp}}}$ within delimited continuations are defined as commands of the shape $\langle p \| \hat{\mathtt{tp}} \rangle$ or formed by a NEF proof and a context of the shape $\tilde{\mu}a.c_{\hat{\mathtt{tp}}}\tau$, $\tilde{\mu}[a_1.c_{\hat{\mathtt{tp}}} | a_2.c'_{\hat{\mathtt{tp}}}]$, $\tilde{\mu}(a_1, a_2).c_{\hat{\mathtt{tp}}}$ or $\tilde{\mu}(x, a).c_{\hat{\mathtt{tp}}}$.

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*[4] in the spirit of the $\overline{\lambda}_{[lv\tau\star]}$-calculus, which are 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, we thus 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* which eagerly require a value to be reduced and trigger the evaluation of lazily stored terms. The resulting language is given in Figure 1.

2.2 Reduction rules

The reduction system of dLPA$^\omega$ is given in Figure 2. The basic rules are those of the call-by-value $\lambda\mu\tilde{\mu}$-calculus and of dL$_{\hat{\mathtt{tp}}}$. The rules for delimited continuations are exactly the same as in dL$_{\hat{\mathtt{tp}}}$, except that we have to prevent $\hat{\mathtt{tp}}$ 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 $e_{\hat{\mathtt{tp}}}$ or not. In the former case, we perform the substitution $[e_{\hat{\mathtt{tp}}}/\alpha]$, 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 or $e_{\hat{\mathtt{tp}}}$ (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 [25]. 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$_{\hat{\mathtt{tp}}}$ 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:

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

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

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

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

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

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[5] in regular mode, that we write $\Gamma \vdash^\sigma J$, and three[6] 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

[2]See [19] for further details.

[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.

[4]Our so-called *stores* somewhat behave like lazy explicit substitutions or mutable environments. See [20] for a discussion on this point.

[5]For terms, proofs, contexts, commands, closures and stores.

[6]For contexts, commands and closures.

Basic rules

$$\begin{array}{lrcl}
 & \langle \lambda x.p \| V_t \cdot e\rangle\tau & \to & \langle p[V_t/x]\|e\rangle\tau \\
(q \in \textsc{nef}) & \langle \lambda a.p \| q \cdot e\rangle\tau & \to & \langle \mu\hat{\mathtt{tp}}.\langle q\|\tilde{\mu}a.\langle p\|\hat{\mathtt{tp}}\rangle\rangle\|e\rangle\tau \\
(q \notin \textsc{nef}) & \langle \lambda a.p \| q \cdot e\rangle\tau & \to & \langle q\|\tilde{\mu}a.\langle p\|e\rangle\rangle\tau \\
(e \neq e_{\hat{\mathtt{tp}}}) & \langle \mu\alpha.c\|e\rangle\tau & \to & c\tau[\alpha := e] \\
 & \langle V\|\tilde{\mu}a.c\tau'\rangle\tau & \to & c\tau[a := V]\tau'
\end{array}$$

Elimination rules

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

Delimited continuations

$$\begin{array}{lrcl}
(\text{if } c\tau \to c\tau') & \langle \mu\hat{\mathtt{tp}}.c\|e\rangle\tau & \to & \langle \mu\hat{\mathtt{tp}}.c\|e\rangle\tau' \\
 & \langle \mu\alpha.c\|e_{\hat{\mathtt{tp}}}\rangle\tau & \to & c[e_{\hat{\mathtt{tp}}}/\alpha]\tau \\
 & \langle \mu\hat{\mathtt{tp}}.\langle p\|\hat{\mathtt{tp}}\rangle\|e\rangle\tau & \to & \langle p\|e\rangle\tau
\end{array}$$

Call-by-value

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

Laziness

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

Lookup

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

Terms

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

where:

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

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

grammar:

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

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

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

Because the language of proof terms 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 all typing judgments include a list of dependencies. Indeed, as in the $\overline{\lambda}_{[lv\tau\star]}$-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: $\tau[a := V]$. Now, consider for instance the reduction of a dependent function $\lambda a.p$ (of type $\Pi a : A.B$) applied to a stack $V \cdot e$[7]:

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

Since 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[8].

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

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

$$\begin{array}{l|ll}
\text{where:} & \tau[a := p_N](c) \triangleq \tau(c[p_N/a]) & (p \in \textsc{nef}) \\
\tau[\alpha := e](c) \triangleq \tau(c) & \tau[a := p](c) \triangleq \tau(c) & (p \notin \textsc{nef})
\end{array}$$

Or we type stores in the spirit of the $\overline{\lambda}_{[lv\tau\star]}$-calculus, and we carry along the derivations all the bindings liable 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 favor the other solution, which is closer to the calculus in our opinion. Yet, it has the drawback that it 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 of 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.

[7]We refer the reader to [19] for detailed explanations on this rule.

[8]On the contrary, the reduced command in dL$_{\hat{\mathtt{tp}}}$ would have been $\langle p[V/a]\|e\rangle$, which is typable with the (Cut) rule over the formula $B[V/a]$.

Regular types

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

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

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

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

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

$$\frac{\Gamma, a : A \vdash^\sigma p : B}{\Gamma \vdash^\sigma \lambda a.p : \Pi a : A.B}\ (\rightarrow_r) \qquad \frac{\Gamma \vdash^\sigma q : A \quad \Gamma \vdash^\sigma e : B[q/a]^{\bot\!\!\bot} \quad \text{if } q \notin \textsc{nef} \text{ then } a \notin A}{\Gamma \vdash^\sigma q \cdot e : (\Pi a : A.B)^{\bot\!\!\bot}}\ (\rightarrow_l)$$

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

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

Dependent mode

$$\frac{\Gamma, \Gamma' \vdash_d c_{\hat{\mathfrak{tp}}}; \sigma\sigma' \quad \Gamma \vdash^\sigma \tau : (\Gamma'; \sigma')}{\Gamma \vdash_d c_{\hat{\mathfrak{tp}}}\tau; \sigma}\ (l^d) \qquad \frac{\Gamma, \Gamma' \vdash^\sigma p : A \quad \Gamma, \hat{\mathfrak{tp}} : B^{\bot\!\!\bot}, \Gamma' \vdash_d e : A^{\bot\!\!\bot}; \sigma\{\cdot|p\}}{\Gamma, \hat{\mathfrak{tp}} : B^{\bot\!\!\bot}, \Gamma' \vdash_d \langle p \| e \rangle; \sigma}\ (\textsc{Cut}^d)$$

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

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

Terms

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

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

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

2.4 Subject reduction

We shall now prove that typing is preserved along reduction. As for the $\overline{\lambda}_{[lv\tau\star]}$-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':

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

Proposition 2.1 (Dependencies weakening)**.** *If σ, σ' are two lists of dependencies such that $\sigma \Rrightarrow \sigma'$, then any derivation using σ can be done 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 can prove the safety of reduction with respect to typing:

Theorem 2.2 (Subject reduction)**.** *For any context Γ and any closures $c\tau$ and $c'\tau'$ such that $c\tau \longrightarrow c'\tau'$, 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 $c\tau \longrightarrow c'\tau'$. ◻

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

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

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

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:

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

$$\begin{array}{r|l}
\texttt{subst}\ p\, q \triangleq \mu\alpha.\langle p \| \tilde{\mu}{=}.\langle q \| \alpha\rangle\rangle & \texttt{catch}_\alpha\ p \triangleq \mu\alpha.\langle p \| \alpha\rangle\\
\texttt{exfalso}\ p \triangleq \mu\alpha.\langle p \| [\,]\rangle & \texttt{throw}\ \alpha\ p \triangleq \mu\_.\langle p \| \alpha\rangle
\end{array}$$

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

It is then straightforward to check that the macros match the expected typing rules:

Proposition 2.3 (Natural deduction). *The typing rules from dPA$^\omega$, given in Figure 4, are admissible.*

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 words, 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 2.4 (Countable choice [11]). *We have:*

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

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

Theorem 2.5 (Dependent choice [11]). *We have:*

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

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

3 Small-step calculus

As for the $\overline{\lambda}_{[lv\tau\star]}$-calculus [1, 20], 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 values, injections or dependent pairs:

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

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 of 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:

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

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

It is worth noting that the syntax as well as the typing and reduction rules for terms now match exactly the ones for proofs[9]. 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 [19], the CPS translation of pairs (t, p) in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ 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:

$$\begin{array}{llcl}
\textbf{Contexts} & e & ::= & \cdots \mid \tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}} \\[1ex]
\textbf{Co-delimited} & c_{\check{\mathfrak{tp}}} & ::= & \langle p_N \| e_{\check{\mathfrak{tp}}} \rangle \mid \langle t \| \pi_{\check{\mathfrak{tp}}} \rangle \mid \langle \check{\mathfrak{tp}} \| e \rangle \\
\textbf{continuations} & e_{\check{\mathfrak{tp}}} & ::= & \tilde{\mu}a.c_{\check{\mathfrak{tp}}} \mid \tilde{\mu}[a_1.c_{\check{\mathfrak{tp}}} \mid a_2.c'_{\check{\mathfrak{tp}}}] \\
 & & \mid & \tilde{\mu}(a_1, a_2).c_{\check{\mathfrak{tp}}} \mid \tilde{\mu}(x, a).c_{\check{\mathfrak{tp}}} \\
 & \pi_{\check{\mathfrak{tp}}} & ::= & t \cdot \pi_{\check{\mathfrak{tp}}} \mid \tilde{\mu}x.c_{\check{\mathfrak{tp}}} \\[1ex]
\textbf{NEF} & e_N & ::= & \cdots \mid \tilde{\mu}\check{\mathfrak{tp}}.c_{\check{\mathfrak{tp}}}
\end{array}$$

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 introduced in $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, 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 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:

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

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

Finally, small-step reduction 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.

Here again, reduction is safe with respect to the type system:

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

It is also 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 $\to$.*

4 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 [20] and for $\mathrm{dL}_{\hat{\mathfrak{tp}}}$ through the embedding in Lepigre's calculus [16, 19]. Namely, as for the $\overline{\lambda}_{[lv\tau\star]}$-calculus, 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 and τ' are compatible.

We recall the main definitions 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 $FV(p) \subseteq \mathrm{dom}(\tau)$. 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 [20], which allows us to define an orthogonality relation between proofs- and contexts-in-store.

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

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 $\perp\!\!\!\perp \in C_0$ is said to be *saturated* or *closed by anti-reduction* whenever for all $(c|\tau), (c'|\tau') \in C_0$, we have:

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

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

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

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

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

A *pole* is defined as any subset of 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 $\perp\!\!\!\perp_{\Downarrow} = \{c\tau \in C_0 : c\tau$ normalizes $\}$ is a pole.*

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

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

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

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:

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

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

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

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

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

These rules are exactly the same as in Lepigre's calculus [16] 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[10]. We stick to the intuition that since `cofix` operators are lazily evaluated, they actually are realizers of every finite approximation of the (possibly infinite) coinductive formula. Consider for instance the case of a stream:

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

of type $\nu^0_{Xx}A(x) \wedge X(S(x))$. 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 $\mathtt{str}^{n+1}_\infty p$. Therefore, $\mathtt{str}^0_\infty p$ should be a successful defender of the formula

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

[10]See [18, Sec. 8.4.2] for a discussion on this matter.

$$
\begin{aligned}
\|\bot\|_f &\triangleq \Lambda_f^\tau \\
\|\top\|_f &\triangleq \emptyset \\
\|\dot{F}(t)\|_f &\triangleq F(t) \\
\|\exists x.A\|_f &\triangleq \textstyle\bigcap_{t\in\Lambda_t} \|A[t/x]\|_f \\
\|\forall x.A\|_f &\triangleq (\textstyle\bigcap_{t\in\Lambda_t} \|A[t/x]\|_f^{\perp\!\!\!\perp_v})^{\perp\!\!\!\perp_f} \\
\|\forall a.A\|_f &\triangleq (\textstyle\bigcap_{t\in\Lambda_p} \|A[p/a]\|_f^{\perp\!\!\!\perp_v})^{\perp\!\!\!\perp_f} \\
\|\nu_{fx}^t A\|_f &\triangleq \textstyle\bigcup_{n\in\mathbb{N}} \|F_{A,t}^n\|_f \\
|A|_V &\triangleq \|A\|_f^{\perp\!\!\!\perp_V} \\
\|A\|_e &\triangleq |A|_V^{\perp\!\!\!\perp_e}
\end{aligned}
$$

$$
\begin{aligned}
\|t=u\|_f &\triangleq \begin{cases} \{(\tilde{\mu}^=.c|\tau) : c\tau \in \perp\!\!\!\perp\} & \text{if } t \equiv_\tau u \\ \Lambda_f^\tau & \text{otherwise} \end{cases} \\
\|p\in A\|_f &\triangleq \{(V|\tau)\in|A|_V : V\equiv_\tau p\}^{\perp\!\!\!\perp_f} \\
\|T\to B\|_f &\triangleq \{(V_t\cdot e|\tau) : (V_t|\tau)\in|t\in T|_{V_t} \wedge (e|\tau)\in\|B\|_e\} \\
\|A\to B\|_f &\triangleq \{(V\cdot e|\tau) : (V|\tau)\in|A|_V \wedge (e|\tau)\in\|B\|_e\} \\
\|T\wedge A\|_f &\triangleq \{(\tilde{\mu}(x,a).c|\tau) : \forall\tau', V_t\in|T|_{V_t}^{\tau'}, V\in|A|_V^{\tau'}, \tau\diamond\tau' \Rightarrow c[V_t/x]\overline{\tau\tau'}[a:=V]\in\perp\!\!\!\perp\} \\
\|A_1\wedge A_2\|_f &\triangleq \{(\tilde{\mu}(a_1,a_2).c|\tau) : \forall\tau', V_1\in|A_1|_V^{\tau'}, V_2\in|A_2|_V^{\tau'}, \tau\diamond\tau' \Rightarrow c\overline{\tau\tau'}[a_1:=V_1][a_2:=V_2]\in\perp\!\!\!\perp\} \\
\|A_1\vee A_2\|_f &\triangleq \{(\tilde{\mu}[a_1.c_1|a_2.c_2]|\tau) : \forall\tau', V\in|A_i|_V^{\tau'}, \tau\diamond\tau' \Rightarrow c\overline{\tau\tau'}[a_i:=V]\in\perp\!\!\!\perp\} \\
|A|_p &\triangleq \|A\|_e^{\perp\!\!\!\perp_p}
\end{aligned}
$$

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

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$^\omega$

Since `cofix` operators only reduce when they are bound to a variable in front of a forcing context, it suggests interpreting 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 X 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:

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

At level f, closed formulas are interpreted by sets of strong forcing contexts-in-store $(f|\tau)$. As explained earlier, 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 is usual in Krivine realizability [14], 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[11]. 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 $\rho(x)$).

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

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[12] $(\tau,\rho)\Vdash\Gamma$, 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 can check that the interpretation is indeed defined up to the relations $\equiv_\tau$:

Proposition 4.8. *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.

We can now prove the main property of our interpretation:

Proposition 4.9 (Adequacy). *The typing rules are adequate with respect to the realizability interpretation,* i.e. *typed proofs (resp. values, terms, contexts, etc.) belong to the corresponding truth values.*

Proof. By induction on typing derivations such as given in the system extended for the small-step reduction. □

We can finally deduce that dLPA$^\omega$ is normalizing and sound.

Theorem 4.10 (Normalization). *If $\Gamma\vdash^\sigma c$, then c is normalizable.*

Proof. Direct consequence of Propositions 4.5 and 4.9. □

Theorem 4.11 (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. □

[12]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.

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. These computational features allow dLPA$^\omega$ to internalize the realizability approach of [2, 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.

Interestingly, 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 [19] the difficulties related to the definition of a sequent calculus with dependent types. On the other hand, building on [20], we developed a variant of Krivine realizability adapted to a lazy calculus where delayed substitutions are stored in an explicit environment. The sound combination of both frameworks led us to the definition of dLPA$^\omega$ together with its realizability interpretation.

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 [13]. Indeed, our proof uses dependent types to get witnesses of existential formulas, and we represent choice functions through the lazily evaluated stream of their values [13]. 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 [15], 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 Λ is of cardinal $\aleph_1$ (in particular infinite streams of integer are terms), makes it incompatible with `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 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 [24] in the context of Gödel's functional interpretation, and later adapted to the context of modified realizability by Berardi *et al* [2]. The aforementioned works of Krivine [13, 15] in the different settings of PA2 and ZF$_\varepsilon$ also give realizers of dependent choice. 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 [13], bar induction or update induction [3] in the case of [2, 24].).

Our approach is here different, since we directly interpret proofs of dependent choice in classical arithmetic computationally. Besides, the structure of our realizability interpretation for dLPA$^\omega$ suggests the definition of a typed CPS to an extension of system F [18], but it is not clear whether its consistency is itself conservative or 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 [23, 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.

[13]A similar idea can be found in NuPrl BITT type theory, where choice sequences are used in place of functions [5].

Acknowledgments

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

References

[1] Z. M. Ariola, P. Downen, H. Herbelin, K. Nakata, and A. Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In *FLOPS 2012, Proceedings*.
[2] S. Berardi, M. Bezem, and T. Coquand. On the computational content of the axiom of choice. *J. Symb. Log.*, 63(2):600–622, 1998. doi:10.2307/2586854.
[3] U. Berger. A computational interpretation of open induction. In *LICS 2004, Proceedings*, page 326, 2004. doi:10.1109/LICS.2004.1319627.
[4] V. Blot. An interpretation of system F through bar recursion. In *LICS 2017, Proceedings*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005066.
[5] L. Cohen, V. Rahli, M. Bickford, and R. L. Constable. Computability beyond church-turing using choice sequences. In *LICS 2018, Proceedings*, 2018.
[6] P.-L. Curien and H. Herbelin. The duality of computation. In *ICFP 2000, Proceedings*, SIGPLAN Notices 35(9), 2000. doi:10.1145/351240.351262.
[7] O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defunctionalized interpreters for call-by-need evaluation. In *FLOPS 2010, Proceedings*, 2010.
[8] P. Downen, L. Maurer, Z. M. Ariola, and S. P. Jones. Sequent calculus as a compiler intermediate language. In *ICFP 2016, Proceedings*, 2016.
[9] M. H. Escardó and P. Oliva. Bar recursion and products of selection functions. *CoRR*, abs/1407.7046, 2014. URL: http://arxiv.org/abs/1407.7046.
[10] H. Herbelin. On the degeneracy of sigma-types in presence of computational classical logic. In *TLCA 2005, Proceedings*, 2005. doi:10.1007/11417170_16.
[11] H. Herbelin. A constructive proof of dependent choice, compatible with classical logic. In *LICS 2012, Proceedings*, 2012. doi:10.1109/LICS.2012.47.
[12] A. Kolmogoroff. Zur deutung der intuitionistischen logik. *Mathematische Zeitschrift*, 35(1):58–65, Dec 1932. doi:10.1007/BF01186549.
[13] J.-L. Krivine. Dependent choice, 'quote' and the clock. *Th. Comp. Sc.*, 308, 2003.
[14] J.-L. Krivine. Realizability in classical logic. In interactive models of computation and program behaviour. *Panoramas et synthèses*, 27, 2009.
[15] J.-L. Krivine. Bar Recursion in Classical Realisability: Dependent Choice and Continuum Hypothesis. In *CSL 2016, Proceedings*, 2016.
[16] R. Lepigre. A classical realizability model for a semantical value restriction. In *ESOP 2016, Proceedings*, 2016. doi:10.1007/978-3-662-49498-1_19.
[17] P. Martin-Löf. An intuitionistic theory of types. In twenty-five years of constructive type theory. *Oxford Logic Guides*, 36:127–172, 1998.
[18] É. Miquey. *Classical realizability and side-effects*. Theses, Univ.é Paris Diderot ; Univ. de la República, Uruguay, Nov. 2017. URL: https://hal.inria.fr/tel-01653733.
[19] É. Miquey. A classical sequent calculus with dependent types. In H. Yang, editor, *ESOP 2017, Proceedings*, 2017. doi:10.1007/978-3-662-54434-1_29.
[20] É. Miquey and H. Herbelin. Realizability interpretation and normalization of typed call-by-need λ-calculus with control. In *FoSSaCS, Proceedings*, 2018.
[21] G. Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs*. PhD thesis, Univ. Paris Diderot, 2013.
[22] G. Munch-Maccagnoni and G. Scherer. Polarised Intermediate Representation of Lambda Calculus with Sums. In *LICS 2015, Proceedings*, 2015.
[23] S. G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic. Cambridge University Press, 2 edition, 2009. doi:10.1017/CBO9780511581007.
[24] C. Spector. Provably recursive functionals of analysis: A consistency proof of analysis by an extension of principles in current intuitionistic mathematics. 1962.
[25] P. Wadler. Call-by-value is dual to call-by-name. In *ICFP 2003, Proceedings*, 2003.