8- dLPA$^\omega$: a sequent calculus with dependent types for classical arithmetic

Drawing on the calculi we studied in the last chapters, we shall now present dLPA$^\omega$, a sequent calculus version of Herbelin's dPA$^\omega$. This calculus provides us with dependent types restricted to the NEF fragment, for which dLPA$^\omega$ is an extension of dL$_{\hat{\mathfrak{fp}}}$. Indeed, in addition to the language of dL$_{\hat{\mathfrak{fp}}}$, dLPA$^\omega$ has terms for classical arithmetic in finite types (PA$^\omega$). More importantly, it includes a lazily evaluated co-fixpoint operator. To this end, the calculus uses a shared store, as in the $\overline{\lambda}_{[lv\tau\star]}$-calculus.

We first present the language of dLPA$^\omega$ with its type system and its reduction rules. We prove that the calculus verifies the property of subject reduction and that it is as expressive as dPA$^\omega$. In particular, the proof terms for $AC_{\mathbb{N}}$ and DC of dPA$^\omega$ can be directly defined in dLPA$^\omega$. We then apply once again the methodology of Danvy's semantic artifacts to derive a small-step calculus, from which we deduce a continuation-passing-style translation and a realizability interpretation. Both artifacts are somehow a combination of the corresponding ones that we developed for the $\overline{\lambda}_{[lv\tau\star]}$-calculus and dL$_{\hat{\mathfrak{fp}}}$.

In some sense, there will not be any real novelties in this chapter. In particular, most of the proofs resemble a lot to the corresponding ones in the previous chapters. Yet, as dLPA$^\omega$ gathers all the expressive power and features of the $\overline{\lambda}_{[lv\tau\star]}$-calculus and dL$_{\hat{\mathfrak{fp}}}$, the different proofs also combined all the tools and tricks used in each case. They are therefore very technical and long, in particular proofs by induction require the tedious verification of multiple cases which are very similar to cases of proofs we already did. We will hence sketch them most of the time, trying to highlight the most interesting parts.

Normalization of dLPA$^\omega$

The main result of this chapter consists in the normalization of dLPA$^\omega$, from which it is easy to convince ourself that dPA$^\omega$ normalizes too[1]. We sketch a proof of normalization through a continuation-passing-style translation, which would rely on the normalization of System F_Υ. We then give a detailed proof through the realizability interpretation.

Nonetheless, we should say before starting this chapter that we already have a guardrail for the normalization. Indeed, we already proved the normalization of a simply-typed classical call-by-need calculus and we explained that the proof was scalable to the same calculus with a second-order type system. Yet, co-fixpoints are definable in a second-order calculus[2], for instance a stream for the infinite conjunction $A(0) \wedge A(1) \wedge \dots$ can be obtained through the formula $\exists X.[X(0) \wedge \forall x^{\mathbb{N}}.(X(x) \to A(x) \wedge X(S(x)))]$. Besides, the presence of dependent types does not bring any risk of loosing the normalization, since erasing the dependencies in types yield a system with the exact same computational behavior. Hence the normalization of dL$_{\hat{\mathfrak{fp}}}$ and the one of the second-order $\overline{\lambda}_{[lv\tau\star]}$-calculus should be enough, *a priori*, to guarantee the normalization of dLPA$^\omega$.

[1] As explained in Chapter 5, we will not bother with a formal proof of this statement, neither will we prove any properties on the preservation of dPA$^\omega$ reduction rules through the embedding in dLPA$^\omega$. Indeed, we are already satisfied with the normalization of dLPA$^\omega$, which is as expressive as dPA$^\omega$ and which allows for the same proof terms for dependent and countable choice.

[2] A definition in the framework of dPA$^\omega$ is given in [70].

Another handwavy explanation could consist in arguing that we could authorize infinite stores in the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus without altering its normalization. Indeed, from the point of view of existing programs (which are finite and typed in finite contexts), they are computing with a finite knowledge of the memory (and we proved that all the terms were suitable for a store extension[3]). Note that in the store, we could theoretically replace any co-fixpoint that produces a stream by the (fully developped) stream in question. Due to the presence of backtracks in co-fixpoints, the store would contain all the possible streams (possibly an infinite number of it) produced when reducing co-fixpoints. In this setting, if a term were to perform an infinite number of reductions steps, it would necessarily have to explore an infinite number of cells in the pre-computed memory, independently from its production. This should not be possible either.

The latter argument is actually quite close from Herbelin's original proof sketch, which this thesis is precisely trying to replace with a more formal proof. So that these unprecise explanations should be taken more as spoilers of the final result than as proof sketches. We shall now present formally dLPA$^\omega$ and prove its normalization, which will then not come as a surprise.

8.1 dLPA$^\omega$: a sequent calculus with dependent types for classical arithmetic

8.1.1 Syntax

The language of dLPA$^\omega$ should not be a surprise either. It is based on the syntax of dL$_{\hat{\text{fp}}}$, extended with the expressive power of dPA$^\omega$ and with explicit stores as in the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus. We stick to a stratified presentation of dependent types, which we find very convenient to separate terms and proof terms which are handled differently.

The syntax of terms is extended as in dPA$^\omega$ to include 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$^\omega$ (see Chapter 5). 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. Namely, the syntax is an extension of dL$_{\hat{\text{fp}}}$'s syntax which now includes:

- the usual proofs $\mu\alpha.c$ and contexts $\tilde{\mu}a.c$ of the $\lambda\mu\tilde{\mu}$-calculus;
- pairs (p_1, p_2), which inhabit the conjunction type $A_1 \wedge A_2$;
- co-pairs $\tilde{\mu}(a_1, a_2).c$, which bind the variables a_1 and a_2 in the command c;
- injections $\iota_i(p)$ for the logical disjunction;
- co-injections or pattern-matching $\tilde{\mu}[a_1.c_1|a_2.c_2]$ which bind the variables a_1 in c_1 and a_2 in c_2;
- 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 $\lambda x.p$, which inhabit the dependent product type $\forall x^T.A$;
- dual stacks $t \cdot e$, where t is a term and e a context whose type might be dependent in t;
- functions $\lambda a.p$, which inhabit the dependent product type $\Pi a : A.B$;
- dual stacks $q \cdot e$, where q is a term and e a context whose type might be dependent in q if q is NEF;
- a proof term $\mathtt{refl}$ which is the proof of atomic equalities $t = t$;
- the dual destructor $\tilde{\mu}^{=}.c$ which allows to type the command c modulo an equality of terms;

[3] See Lemma 6.16 for the realizability interpretation and Lemma 6.31 for the CPS translation of the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus.

Closures	l	::=	$c\tau$
Commands	c	::=	$\langle p \| e \rangle$
Proof terms	p, q	::=	$a \mid \iota_i(p) \mid (p,q) \mid (t,p) \mid \lambda x.p \mid \lambda a.p \mid \texttt{refl}$
		$\mid$	$\texttt{ind}^t_{ax}[p_0 \mid p_S] \mid \texttt{cofix}^t_{bx}[p] \mid \mu\alpha.c \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}}$
Proof values	V	::=	$a \mid \iota_i(V) \mid (V,V) \mid (V_t,V) \mid \lambda x.p \mid \lambda a.p \mid \texttt{refl}$
Contexts	e	::=	$f \mid \alpha \mid \tilde{\mu}a.c\tau$
Forcing contexts	f	::=	$[] \mid \tilde{\mu}[a_1.c_1 \mid a_2.c_2] \mid \tilde{\mu}(a_1,a_2).c \mid \tilde{\mu}(x,a).c$
		$\mid$	$t \cdot e \mid p \cdot e \mid \tilde{\mu}^{=}.c$
Stores	τ	::=	$\varepsilon \mid \tau[a := p_\tau] \mid \tau[\alpha := e]$
Storables	p_τ	::=	$V \mid \texttt{ind}^{V_t}_{ax}[p_0 \mid p_S] \mid \texttt{cofix}^{V_t}_{bx}[p]$
Terms	t, u	::=	$x \mid 0 \mid S(t) \mid \texttt{rec}^t_{xy}[t_0 \mid t_S] \mid \lambda x.t \mid t\ u \mid \texttt{wit}\ p$
Terms values	V_t	::=	$x \mid S^n(0) \mid \lambda x.t$
Delimited continuations	$c_{\hat{\mathfrak{tp}}}$	::=	$\langle p_N \| e_{\hat{\mathfrak{tp}}} \rangle \mid \langle p \| \hat{\mathfrak{tp}} \rangle$
	$e_{\hat{\mathfrak{tp}}}$	::=	$\tilde{\mu}a.c_{\hat{\mathfrak{tp}}}\tau \mid \tilde{\mu}[a_1.c_{\hat{\mathfrak{tp}}} \mid a_2.c'_{\hat{\mathfrak{tp}}}] \mid \tilde{\mu}(a_1,a_2).c_{\hat{\mathfrak{tp}}} \mid \tilde{\mu}(x,a).c_{\hat{\mathfrak{tp}}}$
NEF	c_N	::=	$\langle p_N \| e_N \rangle$
	p_N, q_N	::=	$a \mid \iota_i(p_N) \mid (p_N,q_N) \mid (t,p_N) \mid \lambda x.p \mid \lambda a.p \mid \texttt{refl}$
		$\mid$	$\texttt{ind}^t_{ax}[p_N \mid q_N] \mid \texttt{cofix}^t_{bx}[p_N] \mid \mu\star.c_N \mid \mu\hat{\mathfrak{tp}}.c_{\hat{\mathfrak{tp}}}$
	e_N	::=	$\star \mid \tilde{\mu}[a_1.c_N \mid a_2.c'_N] \mid \tilde{\mu}a.c_N\tau \mid \tilde{\mu}(a_1,a_2).c_N \mid \tilde{\mu}(x,a).c_N$

Figure 8.1: The language of dLPA$^\omega$

- operators $\texttt{ind}^t_{ax}[p_0 \mid p_S]$ and $\texttt{cofix}^t_{bx}[p]$, as in dPA$^\omega$, for inductive and coinductive reasoning;
- delimited continuations through proofs $\mu\hat{\mathfrak{tp}}..c_{\mathfrak{tp}}$ 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. Here again, 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). 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 $\tilde{\mu}(x,a).c_{\hat{\mathfrak{tp}}}$.

We adopt a call-by-value evaluation strategy except for fixpoint operators[4] 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 8.1.

[4]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 CPS translation or the realizability interpretation.

Basic rules

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

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

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

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

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

Lookup

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

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:

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

Figure 8.2: Reduction rules of dLPA$^\omega$

8.1.2 Reduction rules

Concerning the reduction system of dLPA$^\omega$, which is given in Figure 8.2, 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 or $e_{\mathfrak{tp}}$ (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 [161]. 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$_{\hat{\mathfrak{tp}}}$ 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.

8.1.3 Typing rules

The language of types and formulas is the same as for dPA$^\omega$. As explained, terms are simply typed, with the set of natural numbers as the sole ground type. The formulas are inductively built on atomic equalities of terms, by means of conjunctions, disjunctions, first-order quantifications, dependent products and co-inductive formulas. As in dL$_{\hat{\mathfrak{tp}}}$, the dependent product $\Pi a : A.B$ corresponds to the usual implication if a does not occur in the conclusion B. Formulas and types are formally defined by:

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

Formulas are considered up to equational theory on terms, as often in Martin-Löf's intensional type theory. 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:

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

We work with one-sided sequents[5] where typing contexts are defined by:

$$\textbf{Typing contexts} \qquad \Gamma, \Gamma' \quad ::= \quad \varepsilon \mid \Gamma, x : T \mid \Gamma, a : A \mid \Gamma, \alpha : A^{\perp\!\!\!\perp} \mid \Gamma, \hat{\mathfrak{tp}} : A^{\perp\!\!\!\perp}.$$

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.

[5] 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 Section 4.2.3.2) as we would have done with two-sided sequents.

We define nine syntactic kinds of typing judgments:

- six in regular mode, that we write $\Gamma \vdash^{\sigma} J$:
  1. $\Gamma \vdash^{\sigma} t : T$ for typing terms,
  2. $\Gamma \vdash^{\sigma} p : A$ for typing proofs,
  3. $\Gamma \vdash^{\sigma} e : A^{\perp\!\!\!\perp}$ for typing contexts,
  4. $\Gamma \vdash^{\sigma} c$ for typing commands,
  5. $\Gamma \vdash^{\sigma} c\tau$ for typing closures,
  6. $\Gamma \vdash^{\sigma} \tau' : (\Gamma'; \sigma')$ for typing stores;
- three more for the dependent mode, that we write $\Gamma \vdash_d J; \sigma$:
  7. $\Gamma \vdash_d e : A^{\perp\!\!\!\perp}; \sigma$ for typing contexts,
  8. $\Gamma \vdash_d c; \sigma$ for typing commands,
  9. $\Gamma \vdash_d c\tau; \sigma$ for typing closures.

In each case, σ is a list of dependencies—we explain the presence of a list of dependencies in each case thereafter—, which are still 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:

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

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 all typing judgments include a list of dependencies. 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: $\tau[a := V]$. This forces us to slightly change the rules from $\mathrm{dL}_{\hat{\mathfrak{tp}}}$. Indeed, consider for instance the reduction of a dependent function $\lambda a.p$ (of type $\Pi a : A.B$) applied to a stack $V \cdot e$:

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

which we examined in details in the previous chapter (see Section 7.1.3). In $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, the reduced command was $\langle p[V/a] \| e \rangle$, which was 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:

$$\frac{\Gamma, \Gamma' \vdash \tau(c) \quad \Gamma \vdash \tau : \Gamma'}{\Gamma \vdash c\tau} \qquad \begin{array}{l} \text{where:} \\ \tau[\alpha := e](c) \triangleq \tau(c) \end{array} \qquad \left| \qquad \begin{array}{ll} \tau[a := p_N](c) \triangleq \tau(c[p_N/a]) & (p \in \textsc{nef}) \\ \tau[a := p](c) \triangleq \tau(c) & (p \notin \textsc{nef}) \end{array} \right.$$

Or we type stores in the spirit of the $\overline{\lambda}_{[lv\tau\star]}$-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 dependencies list 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 8.3.

Regular types

$$\frac{\Gamma \vdash^\sigma \tau : (\Gamma'; \sigma') \quad \Gamma, \Gamma' \vdash^{\sigma\sigma'} p : A}{\Gamma \vdash^\sigma \tau[a := p] : (\Gamma', a : A; \sigma'\{a|p\})}\ (\tau_p) \qquad \frac{\Gamma \vdash^\sigma \tau : (\Gamma'; \sigma') \quad \Gamma, \Gamma' \vdash^{\sigma\sigma'} \alpha : A^{\perp\!\!\!\perp}}{\Gamma \vdash^\sigma \tau[\alpha := e] : (\Gamma', \alpha : A^{\perp\!\!\!\perp}; \sigma')}\ (\tau_e)$$

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

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash^\sigma a : A}\ (\text{Ax}_r) \qquad \frac{(\alpha : A^{\perp\!\!\!\perp}) \in \Gamma}{\Gamma \vdash^\sigma \alpha : A^{\perp\!\!\!\perp}}\ (\text{Ax}_l) \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash^\sigma c}{\Gamma \vdash^\sigma \mu\alpha.c : A}\ (\mu)$$

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

$$\frac{\Gamma \vdash^\sigma p : A_i}{\Gamma \vdash^\sigma \iota_i(p) : A_1 \vee A_2}\ (\vee_r) \qquad \frac{\Gamma, a_1 : A_1 \vdash^\sigma c_1 \quad \Gamma, a_2 : A_2 \vdash^\sigma c_2}{\Gamma \vdash^\sigma \tilde{\mu}[a_1.c_1 \mid a_2.c_2] : (A_1 \vee A_2)^{\perp\!\!\!\perp}}\ (\vee_l)$$

$$\frac{\Gamma \vdash^\sigma p : A[t/x] \quad \Gamma \vdash^\sigma t : T}{\Gamma \vdash^\sigma (t, p) : \exists x^T.A}\ (\exists_r) \qquad \frac{\Gamma, x : T, a : A \vdash^\sigma c}{\Gamma \vdash^\sigma \tilde{\mu}(x, a).c : (\exists x^T.A)^{\perp\!\!\!\perp}}\ (\exists_l)$$

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

$$\frac{\Gamma \vdash^\sigma p : A \quad \Gamma \vdash^\sigma e : A[u/t]}{\Gamma \vdash^\sigma \tilde{\mu}{=}.\langle p \| e \rangle : (t = u)^{\perp\!\!\!\perp}}\ (=_l) \qquad \frac{\Gamma \vdash^\sigma p : A \quad A \equiv B}{\Gamma \vdash^\sigma p : B}\ (\equiv_r) \qquad \frac{\Gamma \vdash^\sigma e : A^{\perp\!\!\!\perp} \quad A \equiv B}{\Gamma \vdash^\sigma e : B^{\perp\!\!\!\perp}}\ (\equiv_l)$$

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

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

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

Dependent mode

$$\frac{\Gamma, \hat{\mathfrak{p}} : A^{\perp\!\!\!\perp} \vdash_d c_{\hat{\mathfrak{p}}}; \sigma}{\Gamma \vdash^\sigma \mu\hat{\mathfrak{p}}.c_{\hat{\mathfrak{p}}} : A}\ (\mu\hat{\mathfrak{p}}) \qquad \frac{\sigma(A) = \sigma(B)}{\Gamma, \hat{\mathfrak{p}} : A^{\perp\!\!\!\perp}, \Gamma' \vdash_d \hat{\mathfrak{p}} : B^{\perp\!\!\!\perp}; \sigma\{\cdot|p\}}\ (\hat{\mathfrak{p}})$$

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

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

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

Terms

$$\frac{}{\Gamma \vdash^\sigma 0 : \mathbb{N}}\ (0) \qquad \frac{\Gamma \vdash^\sigma t : \mathbb{N}}{\Gamma \vdash^\sigma S(t) : \mathbb{N}}\ (S) \qquad \frac{\Gamma, x : U \vdash^\sigma t : T}{\Gamma \vdash^\sigma \lambda x.t : U \to T}\ (\lambda) \qquad \frac{\Gamma \vdash^\sigma t : U \to T \quad \Gamma \vdash^\sigma u : U}{\Gamma \vdash^\sigma t\, u : T}\ (@)$$

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

Figure 8.3: Type system for dLPA$^\omega$

8.1.4 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. As in the previous chapter, 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':

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

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

Lemma 8.1 (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 8.2 (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 8.1. □

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

Lemma 8.3. *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 8.4 (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 8.5 (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 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 8.3 from top to bottom.

- The cases for reductions of λ are identical to the cases proven in the previous chapter for dL$_{\hat{\mathfrak{ph}}}$.
- 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:

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

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

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

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 8.1 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 dL$_{\hat{\mathfrak{ph}}}$.

- The cases for the so-called "call-by-value" rules opening constructors are straightforward, using again Lemma 8.1 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:

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

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 $v^t_{fx}A$ when typing the rest of the command):

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

- 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).

□

8.1.5 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:

$$\begin{array}{rcl}
\texttt{let}\, a = p\, \texttt{in}\, q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu} a.\langle q \| \alpha_p\rangle\rangle \\
\texttt{split}\ p\ \texttt{as}\,(a_1,a_2)\ \texttt{in}\ q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}(a_1,a_2).\langle q \| \alpha_p\rangle\rangle \\
\texttt{case}\ p\ \texttt{of}\,[a_1.p_1 \mid a_2.p_2] & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}[a_1.\langle p_1 \| \alpha_p\rangle | a_2.\langle p_2 \| \alpha_p\rangle]\rangle \\
\texttt{dest}\ p\ \texttt{as}\,(a,x)\ \texttt{in}\ q & \triangleq & \mu\alpha_p.\langle p \| \tilde{\mu}(x,a).\langle q \| \alpha_p\rangle\rangle \\
\texttt{prf}\ p & \triangleq & \mu\hat{\mathfrak{p}}.\langle p \| \tilde{\mu}(x,a).\langle a \| \hat{\mathfrak{p}}\rangle\rangle
\end{array}
\qquad
\begin{array}{rcl}
\texttt{subst}\ p\, q & \triangleq & \mu\alpha.\langle p \| \tilde{\mu}_{=}.\langle q \| \alpha\rangle\rangle \\
\texttt{exfalso}\ p & \triangleq & \mu\alpha.\langle p \| []\rangle \\
\texttt{catch}_\alpha\, p & \triangleq & \mu\alpha.\langle p \| \alpha\rangle \\
\texttt{throw}\ \alpha p & \triangleq & \mu_.\langle p \| \alpha\rangle
\end{array}$$

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

Proposition 8.6 (Natural deduction)**.** *The typing rules from dPA$^\omega$, given in Section 8.1.5), are admissible*

Proof. Straightforward derivations, the cases for `prf` $p\ q$ and `subst` $p\, q$ are given in Section 7.5.4. □

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 [70]. We do not state it here, but following the approach of [70], we could also extend dLPA$^\omega$ to obtain a proof for the axiom of bar induction.

$$\frac{\Gamma \vdash p : \exists x^T.A \quad \Gamma, x : T, a : A \vdash q : B[(x,a)/\bullet] \quad p \notin \text{NEF} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q : B[p/\bullet]}\ (\mathtt{dest}) \qquad \frac{\Gamma \vdash p : \exists x^T.A(x)}{\Gamma \vdash \mathtt{prf}\ p : A(\mathtt{wit}\ p)}\ (\mathtt{prf})$$

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

$$\frac{\Gamma \vdash p : A_1 \vee A_2 \quad \Gamma, a_i : A_i \vdash q : B[\iota_i(a)_i/\bullet] \quad \text{for } i = 1,2 \quad p \notin \text{NEF} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2] : B[p/\bullet]}\ (\mathtt{case}) \qquad \frac{\Gamma \vdash p : \bot}{\Gamma \vdash \mathtt{exfalso}\ p : B}\ (\bot)$$

$$\frac{\Gamma, a : A \vdash q : B[a/\bullet] \quad p \notin \text{NEF} \Rightarrow \bullet \notin B}{\Gamma \vdash \mathtt{let}\ a = p\ \mathtt{in}\ q : B[p/\bullet]}\ (\mathtt{let}) \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash p : A}{\Gamma \vdash \mathtt{catch}_\alpha\ p : A} \qquad \frac{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash p : A}{\Gamma, \alpha : A^{\perp\!\!\!\perp} \vdash \mathtt{throw}\ \alpha\ p : B}$$

Figure 8.4: Typing rules of dPA$^\omega$

Theorem 8.7 (Countable choice [70])**.** *We have:*

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

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

Proof. See Figure 8.5. □

Theorem 8.8 (Dependent choice [70])**.** *We have:*

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

where $\mathtt{nth}_n\ a := \mathtt{ind}^n_{x,d}[a \mid (\mathtt{wit}\ (\mathtt{prf}\ d), \pi_2(\mathtt{prf}\ (\mathtt{prf}\ (d))))]$.

Proof. Left to the reader. □

8.2 Small-step calculus

Once more, we follow Danvy's methodology of semantic artifacts to obtain a continuation-passing style translation and 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.

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 $\overline{\lambda}_{[l\upsilon\tau\star]}$-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

Notations:

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

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

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

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

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

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

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

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 8.5: Proof of the axiom of countable choice in dLPA$^\omega$

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

This allows 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.

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

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

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

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.

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*. We saw in the previous chapter that the CPS translation of pairs (t,p) was not the expected one, and we mentioned the fact that it reflected 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. Indeed, consider 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 $\check{\mathfrak{p}}$ (dually to reset $\hat{\mathfrak{p}}$), whose type can be changed from $A[x]$ to $A[t]$ thanks to a list of dependencies:

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

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

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

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

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

The small-step reduction system is given in Figure 8.6. 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 $\overline{\lambda}_{[l v \tau \star]}$-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 8.9 (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:

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

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):

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

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) $\quad \langle \mu \hat{\mathfrak{tp}}.c\tau'' \| e \rangle_p \tau \rightsquigarrow_s \langle \mu \hat{\mathfrak{tp}}.c'\tau'' \| e \rangle_p \tau' \quad$ (if $c_\iota \tau \rightsquigarrow_s c'_o \tau'$)

$$\langle \mu \hat{\mathfrak{tp}}.\langle p \| \hat{\mathfrak{tp}} \rangle \| e \rangle_p \tau \rightsquigarrow_s \langle p \| e \rangle_p \tau$$

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

$$\langle V \| \tilde{\mu} \check{\mathfrak{tp}}.\langle \check{\mathfrak{tp}} \| e \rangle \rangle_e \tau \rightsquigarrow_s \langle V \| e \rangle_e \tau$$

Proofs

($e \neq e_{\hat{\mathfrak{tp}}}$) $\quad \langle \mu\alpha.c \| e \rangle_p \tau \rightsquigarrow_s c_c\, \tau[\alpha := e]$

$$\langle \mu\alpha.c \| e_{\hat{\mathfrak{tp}}} \rangle_p \tau \rightsquigarrow_s c_c[e_{\hat{\mathfrak{tp}}}/\alpha]\, \tau$$

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

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

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

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

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

$$\langle V \| e \rangle_p \tau \rightsquigarrow_s \langle V \| e \rangle_e$$

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) $\quad \langle a \| f \rangle_V \tau[a = \mathtt{cofix}^t_{bx}[p]] \tau' \rightsquigarrow_s \langle p[t/x][b'/b] \| \tilde{\mu} a.\langle a \| f \rangle \tau' \rangle_p \tau[b' := \lambda y.\mathtt{cofix}^y_{bx}[p]]$

$$\langle a \| f \rangle_V \tau[a = \mathtt{ind}^0_{bx}[p_0 \mid p_S]] \tau' \rightsquigarrow_s \langle p_0 \| \tilde{\mu} a.\langle a \| f \rangle \tau' \rangle_p \tau$$

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

Forcing contexts

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

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

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

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

Terms

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

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

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

($t \notin V_t$) $\quad \langle \mathtt{rec}^t_{xy}[t_0 \mid t_S] \| \pi \rangle_t \tau \rightsquigarrow_s \langle t \| \tilde{\mu} z.\langle \mathtt{rec}^z_{xy}[t_0 \mid t_S] \| \pi \rangle \rangle_t \tau$

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

Figure 8.6: Small-step reduction rules

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

$$\dfrac{\dfrac{\Pi_p}{\Gamma \vdash^\sigma p : A[t]} \qquad \dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma, \check{\mathfrak{p}} : A[t] \vdash t : T} \qquad \dfrac{\dfrac{\dfrac{\Pi_{(x,a)}}{\Gamma, x : T, a : A[x] \vdash^\sigma \langle (x,a) \| e \rangle : A[x]^{\perp\!\!\!\perp}}\,(\text{Cut})}{\Gamma, x : T \vdash^\sigma \tilde{\mu} a.\langle (x,a) \| e \rangle : A[x]^{\perp\!\!\!\perp}}\,(\tilde{\mu}) \qquad A[t] = (\{x|t\})(A[x])}{\Gamma, \check{\mathfrak{p}} : A[t], x : T \vdash_d \langle \check{\mathfrak{p}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle ; \sigma\{x|t\}}\,(\text{Cut}_d)}{\Gamma, \check{\mathfrak{p}} : A[t/x] \vdash_d \tilde{\mu} x.\langle \check{\mathfrak{p}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle : T ; \sigma\{\cdot|t\}}\,(\tilde{\mu}_x)}{\Gamma, \check{\mathfrak{p}} : A[t] \vdash \langle t \| \tilde{\mu} x.\langle \check{\mathfrak{p}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle ; \sigma}\,(\text{Cut}_d)}{\Gamma \vdash^\sigma \tilde{\mu}\check{\mathfrak{p}}.\langle t \| \tilde{\mu} x.\langle \check{\mathfrak{p}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle : A[t]^{\perp\!\!\!\perp}}\,(\tilde{\mu}\check{\mathfrak{p}})}{\Gamma \vdash^\sigma \langle p \| \tilde{\mu}\check{\mathfrak{p}}.\langle t \| \tilde{\mu} x.\langle \check{\mathfrak{p}} \| \tilde{\mu} a.\langle (x,a) \| e \rangle \rangle \rangle \rangle_p}\,(\text{Cut})$$

where $\Pi_{(x,a)}$ is as expected. □

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 8.10. *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. □

8.3 A continuation-passing style translation

We present in this section the continuation-passing style translation[7] which arises from the small-step reduction system we defined. In practice, we will not give here a formal proof of normalization for dLPA$^\omega$(we will give one using a realizability interpretation in the next section), so that we will deliberately omit some proofs and details. In particular, we have *a priori* two choices for the target language of this translation.

Either our interest in the translation is only to prove the normalization of dLPA$^\omega$, in which case we can erase the dependencies and use a non-dependently typed target language. Starting from dLPA$^\omega$, embedding terms and proofs in a single syntactic set then removing dependent types would roughly leave us with a first-order language similar to the $\overline{\lambda}_{[lv\tau\star]}$-calculus (but more expressive). A good candidate as a target language for a CPS translation erasing dependencies is hence System F_Υ, possibly enriched[8] with conjunction, disjunction, etc... to recover the same expressiveness as dLPA$^\omega$. In this case, the typability of the translation would be greatly simplified and it would mostly amount to the typability of the CPS translation for the $\overline{\lambda}_{[lv\tau\star]}$-calculus in Chapter 6.

On the other hand, we could be interested in a translation carrying the dependencies, and choose a target language compatible with that. In which case, the proof of typability would concentrate both the difficulties for typing the store-passing part of the translation, and the difficulties related to type

[7] As in for the $\overline{\lambda}_{[lv\tau\star]}$-calculus, it is in fact a continuation-and-store passing style translation, but we refer to it as continuation-passing style for conciseness.

[8] It is folklore that conjunctions, disjunctions and even co-inductive types can be encoded in System F, and thus in System F_Υ. Adding primitive constructions both in the syntax of types and programs is thus just a matter of convenience to simplify the translation. We can thus consider without lost of generality that the language includes these constructions, since alternatively, one could combine the CPS translation with the encoding to obtain a translation to "pure" System F_Υ.

dependencies as for the translation of $\mathrm{dL}_{\hat{\mathfrak{tb}}}$. For instance, we could pick the calculus of constructions [28] as a very general target language, in which we would dispose of dependent types and of the expressive power to encode the type of second-order vectors from F_Υ.

We choose to leave the choice of our target language ambiguous, and give the most general translation possible. We thus assume that the proof terms of the target language contain at least constructors for pairs, injections, equality and the same destructors as in dPA^ω (*i.e.* `split`, `case`, `dest`, `subst`, `exfalso`), as well as a way to encode vectors. We do not add substitutions to rename variables, but a thorough definition of the translation should also include an explicit renaming procedure, for the reasons invoked in Section 6.4.1.

This being said, the translation is derived directly from the small-step reduction rules. As for the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, the different levels p, e, V, f, υ and t, π, V_t are reflected in a translation $[\![\cdot]\!]_\iota$ for each level ι. The main subtlety concerns the way we handle inductive and co-inductive fixpoints, and more generally the store. Observe that in dLPA^ω we managed to delimit the unfolding of fixpoints to the store, everything happening as if they were special cells producing computations. In other words, we could have been one step further to remove fixpoints from the syntax of proofs, limiting their occurrences strictly to the store. This is actually what is done through the translation, where we mark some cells with `IND` and `COFIX`. The computational content of the fixpoint is thus decomposed step by step, each step being produced by the `lookup` function, that is defined (in pseudo-code) as follows:

$$
\begin{array}{lll}
\texttt{lookup}\,\kappa\,\tau_1[\kappa := p]\tau_2\,k & := & \texttt{match}\,(\kappa, p)\,\texttt{with} \\
\quad | \quad \alpha, e & \mapsto & e\,\tau_1[a := V]\tau_2\,k \\
\quad | \quad a, V & \mapsto & V\,\tau_1[a := V]\tau_2\,k \\
\quad | \quad a, \texttt{COFIX}^t_{bx} p & \mapsto & (p[t/x][b'/b])\,\tau_1[b' := [\![\lambda y.\texttt{cofix}^y_{bx}[p]]\!]_\upsilon]\,(\lambda\tau q.q\,\tau[a := q]\tau_2\,k) \\
\quad | \quad a, \texttt{IND}^0_{bx}[p_0 \mid p_S] & \mapsto & p_0\,\tau_1\,(\lambda\tau q.q\,\tau[a := q]\tau_2\,k) \\
\quad | \quad a, \texttt{IND}^{S(t)}_{bx}[p_0 \mid p_S] & \mapsto & (p_S[t/x][b'/b])\,\tau_1[b' := \texttt{IND}^t_{bx}[p_0 \mid p_S]]\,(\lambda\tau q.q\,\tau[a := q]\tau_2\,k)
\end{array}
$$

where in each case b' is fresh. In practice, this simply corresponds to a store where cells include a flag so that the `lookup` function given above could be implemented in the target language by means of pattern-matching using injections and `case`. The lookup function is now the only piece of the whole translation which actually has the computational content of a fixpoint.

The full translation is given by Figure 8.7, and is by construction correct with respect to reduction. In particular, we could again prove by a tedious induction on the reduction $\rightsquigarrow_s$ that the normalization is preserved:

Proposition 8.11. *If $[\![c\tau]\!]_l$ normalizes, then so does $c\tau$ for $\rightsquigarrow_s$.*

In what concerns the typing of the translation, in the case where we erase the dependencies, it would simply amount to the typing of the translation for the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus, that is to say that the translation of typing judgments for proofs (resp. contexts, etc) will be of the shape:

$$[\![\Gamma \vdash^\sigma p : A]\!] \quad \triangleq \quad (\vdash \ [\![p]\!]_p : [\![\Gamma]\!]_\Gamma \triangleright_p \iota(A))$$

where again:

$$\Upsilon \triangleright_p A \quad \triangleq \quad \forall Y <: \Upsilon.\, Y \to (Y \triangleright_e A) \to \bot\,.$$

The only novelty with respect to the CPS translation for the $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus sites in the `lookup` function in the cases of `IND` and `COFIX`. However, it is easy to check that in both cases, through the translation (and already in the small-step reduction system) these elements take a continuation at level f and put in active position a proof at level p in front of a continuation which is built to be at level e. In particular, types are respected in the sense that $\texttt{lookup}\,a\,(\tau[a := \texttt{COFIX}^t_{bx}[\![p]\!]_p])\,k_f$ is indeed of type $\bot$. We claim

Commands

$$[\![\langle p \| e \rangle]\!]_c\,\tau = [\![p]\!]_p\,\tau\,[\![e]\!]_e$$
$$[\![\langle p \| \hat{\mathfrak{p}} \rangle]\!]_c\,\tau = [\![p]\!]_p\,\tau \qquad [\![\langle t \| \pi \rangle]\!]_c\,\tau = [\![t]\!]_t\,\tau\,\pi$$
$$[\![\langle \check{\mathfrak{p}} \| e \rangle]\!]_c\,\tau = [\![e]\!]_e\,\tau$$

Proofs

$$[\![\mu\hat{\mathfrak{p}}.c]\!]_p\,\tau\,k = ([\![c]\!]_c\,\tau)\,k$$
$$[\![\mu\alpha.c]\!]_p\,\tau\,k = [\![c]\!]_c\,\tau[\alpha := k]$$
$$[\![(p_1,p_2)]\!]_p\,\tau\,k = [\![p_1]\!]_p\,\tau\,(\lambda\tau_1 q_1.[\![p_2]\!]_p\,\tau_1[a_1 := q_1]\,(\lambda\tau_2 q_2.k\,\tau_2[a_2 := q_2]\,[\![(a_1,a_2)]\!]_V))$$
$$[\![\iota_i(p)]\!]_p\,\tau\,k = [\![p]\!]_p\,\tau\,(\lambda\tau q.k\,\tau[a := q]\,[\![\iota_i(a)]\!]_V)$$
$$[\![(t,p)]\!]_p\,\tau\,k = [\![p]\!]_p\,\tau\,(\lambda\tau.[\![t]\!]_t\,\tau\,(\lambda\tau x a.k\,\tau\,[\![(x,a)]\!]_V))$$
(a fresh) $$[\![\mathtt{cofix}^t_{bx}[p]]\!]_p\,\tau\,k = ([\![t]\!]_t\,\tau\,(\lambda\tau y.[\![a]\!]_p\,\tau[a := \mathtt{COFIX}^y_{bx}[\![p]\!]_p]))\,k$$
(a fresh) $$[\![\mathtt{ind}^t_{bx}[p \mid p_s]]\!]_p\,\tau\,k = ([\![t]\!]_t\,\tau\,(\lambda\tau y.[\![a]\!]_p\,\tau[a := \mathtt{IND}^y_{bx}[[\![p]\!]_p \mid [\![q]\!]_p]]))\,k$$
$$[\![V]\!]_p\,\tau\,k = k\,\tau\,[\![V]\!]_V$$

Contexts

$$[\![\tilde{\mu}a.c\tau']\!]_e\,\tau\,V = [\![c]\!]_c\,(\tau[a := V][\![\tau']\!]_\tau) \qquad [\![\alpha]\!]_e\,\tau\,V = \mathtt{lookup}\ \alpha\,\tau\,V$$
$$[\![\tilde{\mu}\check{\mathfrak{p}}.c]\!]_e\,\tau\,V = ([\![c]\!]_c\,\tau)\,V \qquad [\![f]\!]_e\,\tau\,V = V\,\tau\,[\![f]\!]_f$$

Weak values

$$[\![a]\!]_V\,\tau\,k_v = \mathtt{lookup}\ a\,\tau\,k_v \qquad [\![v]\!]_V\,\tau\,k_v = k_v\,\tau\,[\![V]\!]_v$$

Forcing contexts

$$[\![t \cdot e]\!]_f\,\tau\,v = ([\![t]\!]_t\,\tau\,(\lambda\tau x.v\,\tau\,x))\,[\![e]\!]_e$$
($q_N \in$ NEF) $$[\![q_N \cdot e]\!]_f\,\tau\,v = ([\![q_N]\!]_p\,\tau\,(\lambda\tau q'.v\,\tau\,q'))\,[\![e]\!]_e$$
($q \notin$ NEF) $$[\![q \cdot e]\!]_f\,\tau\,v = [\![q]\!]_p\,\tau\,(\lambda\tau q'.v\,\tau\,q'\,[\![e]\!]_e)$$
(b_i fresh) $$[\![\tilde{\mu}(a_1,a_2).c]\!]_f\,\tau\,v = \mathtt{split}\ v\ \mathtt{as}\ (b_1,b_2)\ \mathtt{in}\ ([\![c]\!]_c\,\tau[a_1 := b_1][a_2 := b_2])$$
(b_i fresh) $$[\![\tilde{\mu}[a_1.c_1 \mid a_2.c_2]]\!]_f\,\tau\,v = \mathtt{case}\ v\ \mathtt{of}\ [b_1.[\![c_1]\!]_c\,\tau[a_1 := b_1] \mid b_2.[\![c_2]\!]_c\,\tau[a_2 := b_2,]]$$
(b fresh) $$[\![\tilde{\mu}(x,a).c]\!]_f\,\tau\,v = \mathtt{dest}\ v\ \mathtt{as}\ (y,b)\ \mathtt{in}\ (\lambda x.[\![c]\!]_c)y\,\tau[a := b]$$
$$[\![\tilde{\mu}{=}.c]\!]_f\,\tau\,v = \mathtt{subst}\ v\,[\![c]\!]_c\,\tau$$
$$[\![]]\!]_f\,\tau\,v = \mathtt{exfalso}\ v$$

Strong values

$$[\![\lambda x.p]\!]_v\,\tau\,V_t\,e = [\![p]\!]_p[V_t/x]\,\tau\,e \qquad [\![(a_1,a_2)]\!]_v = ([\![a_1]\!]_V,[\![a_2]\!]_V)$$
$$[\![\lambda a.p]\!]_v\,\tau\,V\,e = [\![p]\!]_p\,\tau[a := V]\,e \qquad [\![\iota_i(a)]\!]_v = \iota_i([\![a]\!]_V)$$
$$[\![\mathtt{refl}]\!]_v = \mathtt{refl} \qquad [\![(V_t,a)]\!]_v = ([\![V_t]\!]_{V_t},[\![a]\!]_V)$$

Environments

$$[\![\varepsilon]\!]_\tau = \varepsilon$$
$$[\![\tau[a := \mathtt{cofix}^{V_t}_{bx}[p]]]\!]_\tau = [\![\tau]\!]_\tau[a := \mathtt{COFIX}^{[\![V_t]\!]_{V_t}}_{bx}[\![p]\!]_p] \qquad [\![\tau[a := V]]\!]_\tau = [\![\tau]\!]_\tau[a := [\![V]\!]_v]$$
$$[\![\tau[a := \mathtt{ind}^{V_t}_{bx}[p \mid q]]]\!]_\tau = [\![\tau]\!]_\tau[a := \mathtt{IND}^{[\![V_t]\!]_{V_t}}_{bx}[[\![p]\!]_p \mid [\![q]\!]_p]] \qquad [\![\tau[\alpha := e]]\!]_\tau = [\![\tau]\!]_\tau[\alpha := [\![e]\!]_e]$$

Terms

$$[\![V_t]\!]_t\,\tau\,k_t = k_t\,\tau\,[\![V_t]\!]_{V_t}$$
$$[\![S(u)]\!]_t\,\tau\,k_t = [\![u]\!]_t\,\tau\,(\lambda\tau x.k_t\,\tau\,[\![S(x)]\!]_{V_t}))$$
$$[\![t\,u]\!]_t\,\tau\,k_t = [\![t]\!]_t\,\tau\,\lambda\tau v.[\![u]\!]_t\,\tau\,(\lambda\tau w.v\,w\,\tau\,k_t))$$
$$[\![\mathtt{wit}\,(p)]\!]_t\,\tau\,k_t = [\![p]\!]_p\,\tau\,(\lambda\tau q.q\,\tau\,(\lambda\alpha.([\![\tilde{\mu}(x,a).\langle x \| \alpha \rangle]\!]_f)\,k_t))$$
$$[\![\mathtt{rec}^t_{xy}[u_0 \mid u_S]]\!]_t\,\tau\,k_t = [\![t]\!]_t\,\tau\,(\lambda\tau z.\mathtt{rec}^z_{xy}[[\![u_0]\!]_t \mid [\![u_S]\!]_t]\,\tau\,k_t)$$
$$[\![u \cdot \pi]\!]_\pi\,\tau\,v = [\![u]\!]_t\,\tau\,(\lambda\tau w.v\,\tau\,w\,[\![\pi]\!]_\pi) \qquad [\![x]\!]_{V_t} = x$$
$$[\![\tilde{\mu}x.c]\!]_\pi\,\tau\,v = ([\![c]\!]_c\,\tau)[v/x] \qquad [\![S(V_t)]\!]_{V_t} = S([\![V_t]\!]_{V_t})$$
$$[\![0]\!]_{V_t} = 0 \qquad [\![\lambda x.t]\!]_{V_t} = \lambda\tau x k.[\![t]\!]_t\,\tau\,k$$

Figure 8.7: Continuation-and-store-passing style translation

than once we understood how the translation of the $\overline{\lambda}_{[lv\tau\star]}$-calculus was typed, this setting is more or less the same and should not give us a hard time.

However, in the case where we would like to obtain a translation compatible with dependent types, we know that we need to refine the typing of terms and NEF proof terms, as we did in dL$_{\hat{\mathfrak{tp}}}$. This is certainly possible, in particular given a NEF proof term p, it is still possible to pass the continuation $\lambda\tau a.a$ to $[\![p_N]\!]_p$ to force the extraction of a proof p_N^+. This should allow us to refine the type of $[\![p_N]\!]_p$ to obtain something like:

$$\Upsilon \triangleright_p A \quad \triangleq \quad \forall Y <: \Upsilon.\, Y \to \forall R.'\Pi a : (Y \triangleright_e A).R(a) \to R(p_N^+)) \,.$$

However, due to the laziness and the two layers of alternation between proof and contexts, we should probably process to a second extraction to obtain a strong value, and cleverly handle the store while doing so. In the absence of a real motivation for such a translation, we did not take the time to study the question more in depth. However we are confident in the fact that the main difficulties has been studied in the previous chapters, so that if it was worthwhile, with time and rigor, it should be possible to methodically obtain a translation of types compatible with the dependencies.

8.4 Normalization of dLPA$^\omega$

8.4.1 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 (Chapter 6) and for dL$_{\hat{\mathfrak{tp}}}$ through the embedding in Lepigre's calculus (Chapter 7). 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 8.12 (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 C_0, and we identify again $(c|\tau)$ with the closure $c\tau$ when c is closed in τ. We can now define the notion of pole, which has to satisfy an extra condition due to the presence of delimited continuations

Definition 8.13 (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 $c[e/\hat{\mathfrak{tp}}]\tau$ (resp. $c[V/\check{\mathfrak{tp}}]\tau$) is in $\perp\!\!\!\perp$, then $\langle \mu\hat{\mathfrak{tp}}.c \| e\rangle\tau$ (resp $.\langle V \| \tilde{\mu}\check{\mathfrak{tp}}.c\rangle\tau$) belongs to $\perp\!\!\!\perp$:

$$(c[e/\hat{\mathfrak{tp}}]\tau \in \perp\!\!\!\perp) \ \Rightarrow \ (\langle \mu\hat{\mathfrak{tp}}.c \| e\rangle\tau \in \perp\!\!\!\perp) \qquad\qquad (c[V/\check{\mathfrak{tp}}]\tau \in \perp\!\!\!\perp) \ \Rightarrow \ (\langle V \| \tilde{\mu}\check{\mathfrak{tp}}.c\rangle\tau \in \perp\!\!\!\perp)$$

A *pole* is defined as any subset of C_0 that is closed by anti-reduction, by store extension and under delimited continuations. ⌟

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

Proposition 8.14. *The set* $\perp\!\!\!\perp_{\Downarrow} = \{c\tau \in \mathcal{C}_0 : \ c\tau \text{ normalizes } \}$ *is a pole.*

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

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

Definition 8.15 (Compatible stores and union)**.** Let τ and τ' be stores, we say that:

- they are *independent* and note $\tau \# \tau'$ when $\mathrm{dom}(\tau) \cap \mathrm{dom}(\tau') = \emptyset$.
- they are *compatible* and note $\tau \diamond \tau'$ whenever 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 $\mathrm{dom}(\tau) \subseteq \mathrm{dom}(\tau')$ and $\tau \diamond \tau'$.
- the compatible union $\overline{\tau\tau'}$ of compatible closed stores τ and τ', is defined as $\mathsf{join}(\tau, \tau')$, which itself given by:

$$\begin{array}{rcll} \mathsf{join}(\tau_0[a := p]\tau_1, \tau_0'[a := p]\tau_1') & \triangleq & \tau_0\tau_0'[a := p]\mathsf{join}(\tau_1, \tau_1') & (\text{if } \tau_0 \# \tau_0') \\ \mathsf{join}(\tau_0[\alpha := e]\tau_1, \tau_0'[\alpha := e]\tau_1') & \triangleq & \tau_0\tau_0'[\alpha := e]\mathsf{join}(\tau_1, \tau_1') & (\text{if } \tau_0 \# \tau_0') \\ \mathsf{join}(\tau, \tau') & \triangleq & \tau\tau' & (\text{if } \tau \# \tau') \end{array}$$ ⌟

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

Lemma 8.16. *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 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 8.17 (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 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:

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

Secondly, as in the interpretation of dL$_{\hat{\mathfrak{tp}}}$ 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:

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

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

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

and to the following inference rules:

$$\frac{\Gamma \vdash^\sigma v : A \quad a \notin FV(\Gamma)}{\Gamma \vdash^\sigma v : \forall a.A}\ (\forall^a_r) \qquad \frac{\Gamma \vdash^\sigma v : A \quad x \notin FV(\Gamma)}{\Gamma \vdash^\sigma v : \forall x.A}\ (\forall^x_r) \qquad \frac{\Gamma \vdash^\sigma v : A[t/x]}{\Gamma \vdash^\sigma v : \exists x.A}\ (\exists^x_r)$$

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

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

These rules are exactly the same as in Lepigre's calculus [108] 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 $\triangleright_\tau$:

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

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 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

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

of type $\nu^0_{fx} A(x) \wedge f(S(x)) = 0$. 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 $\mathtt{str}^{n+1}_\infty p$. So that $\mathtt{str}^0_\infty p$ should be a successful defender of the formula

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

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 $\nu^0_{fx} A(x) \wedge X(S(x))$ at level f as the union of all the opponents to a finite approximation.

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

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

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

$$\|\nu^t_{Xx} A\|_f \triangleq \bigcup_{n \in \mathbb{N}} \|F^n_{A,t}\|_f$$

$$
\begin{array}{rcl}
\|\bot\|_f & \triangleq & \Lambda_f^\tau \\
\|\top\|_f & \triangleq & \emptyset \\
\|\dot{F}(t)\|_f & \triangleq & F(t) \\
\|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\} \\
\|\exists x.A\|_f & \triangleq & \bigcap_{t\in\Lambda_t} \|A[t/x]\|_f \\
\|\forall x.A\|_f & \triangleq & (\bigcap_{t\in\Lambda_t} \|A[t/x]\|_f^{\perp\!\!\!\perp_v})^{\perp\!\!\!\perp_f} \\
\|\forall a.A\|_f & \triangleq & (\bigcap_{t\in\Lambda_p} \|A[p/a]\|_f^{\perp\!\!\!\perp_v})^{\perp\!\!\!\perp_f} \\
\|\nu^t_{fx} A\|_f & \triangleq & \bigcup_{n\in\mathbb{N}} \|F^n_{A,t}\|_f \\
|A|_V & \triangleq & \|A\|_f^{\perp\!\!\!\perp_V} = \{(V|\tau) : \forall f\tau', \tau \diamond \tau' \wedge (f|\tau') \in \|A\|_f \Rightarrow (V|\tau)\perp\!\!\!\perp(F|\tau')\} \\
\|A\|_e & \triangleq & |A|_V^{\perp\!\!\!\perp_e} = \{(E|\tau) : \forall V\tau', \tau \diamond \tau' \wedge (V|\tau') \in |A|_V \Rightarrow (V|\tau')\perp\!\!\!\perp(E|\tau)\} \\
|A|_p & \triangleq & \|A\|_e^{\perp\!\!\!\perp_p} = \{(p|\tau) : \forall E\tau', \tau \diamond \tau' \wedge (E|\tau') \in \|A\|_E \Rightarrow (t|\tau)\perp\!\!\!\perp(E|\tau')\} \\
 & & \\
|\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\} \\
|T|_\pi & \triangleq & |A|_{V_t}^{\perp\!\!\!\perp_\pi} = \{(F|\tau) : \forall v\tau', \tau \diamond \tau' \wedge (v|\tau') \in |A|_v \Rightarrow (v|\tau')\perp\!\!\!\perp(F|\tau)\} \\
|T|_t & \triangleq & |A|_\pi^{\perp\!\!\!\perp_t} = \{(V|\tau) : \forall F\tau', \tau \diamond \tau' \wedge (F|\tau') \in \|A\|_F \Rightarrow (V|\tau)\perp\!\!\!\perp(F|\tau')\}
\end{array}
$$

where:

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

Figure 8.8: Realizability interpretation for dLPA$^\omega$

The realizability interpretation of closed formulas and types is defined in Figure 8.8 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[9]. 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 8.18. 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[10] $(\tau, \rho) \Vdash \Gamma$, if:

1. for any $(a : A) \in \Gamma$, $(a|\tau)_\rho \in |A_\rho|_V$
2. for any $(\alpha : A^{\perp\!\!\!\perp}_\rho) \in \Gamma$, $(\alpha|\tau)_\rho \in \|A_\rho\|_e$
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 $\overline{\lambda}_{[l\upsilon\tau\star]}$-calculus (Lemma 6.16 and Proposition 6.14):

Lemma 8.19 (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 8.20 (Monotonicity)**.** *For any closed formula A, any type T and any given pole $\perp\!\!\!\perp$, we have the following inclusions:*

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

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

Lemma 8.21. *For any store τ and any valuation ρ, the component along τ of the truth and falsity values defined in Figure 8.8 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 $(V_t|\tau)_\rho \in |B_\rho|_\upsilon$.*

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 8.22 (Adequacy)**.** *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 υ is a strong value such that $\Gamma \vdash^\sigma \upsilon : A$ or $\Gamma \vdash_d \upsilon : A; \sigma$, then $(\upsilon|\tau)_\rho \in |A_\rho|_V$.*

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

[10] 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.

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 $\overline{\lambda}_{[lv\tau\star]}$-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 8.19 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 8.21 since if A, B are two formulas such that $A \equiv B$, in particular $A \equiv_\tau B$ and thus $|A|_v = |B|_v$.

• **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 8.21, 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:

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

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

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

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

We want to show that $(\mathtt{ind}_{ax}^{t}[p_0 \,|\, p_S]|\tau) \in |A[t]|_p$, 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[11] $t \in |t \in \mathbb{N}|_t$ and we have:

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

so that by anti-reduction and closure under delimited continuations, it is enough to show that the coterm-in-store $(\tilde{\mu}y.\langle a \| e \rangle [a := \mathtt{ind}_{bx}^{y}[p_0 \,|\, p_S]] | \tau_0)$ 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:

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

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

- If $n = 0$, we have:

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

  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:

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

  Since we have by induction that $(b'|\tau_2[b' := \mathtt{ind}_{bx}^{S^m(0)}[p_0 \,|\, p_S]])$ 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 $\|A[S(S^m(0))]\|_e$.

[11] 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:

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

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:

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

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. □

We can finally deduce from Propositions 8.14 and 8.22 that dLPA$^{\omega}$ is normalizable and sound.

Theorem 8.23 (Normalization). *If $\Gamma \vdash^{\sigma} c$, then c is normalizable.*

Theorem 8.24 (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. □

8.4.2 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:

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

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[12] 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 8.25. *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. □

[12] 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.

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 $|A|_v^\rho, \|A\|_f^\rho, \ldots$ of formulas with parameters exactly as above with the additional rule[13]:

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

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 8.26. *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 $|A|_v^\rho = |A[B(t)/X(t)]|_v$*

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 8.27. *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:

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

This definition satisfies the expected equality:

Proposition 8.28. *We have:*

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

Proof. Observe first that by definition, the formula $B(z) = |\nu_{Xx}^z A|_v$ satisfies the hypotheses of Lemma 8.26 and that $\mathtt{gfp}(\varphi_A) = t \mapsto B(t)$. Then we can deduce :

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

□

Back to the original language, it only remains to define $|\nu_{fx}^t A|_v$ as the set $|\nu_{Xx}^t A[X(y)/f(y) = 0]|_v$ that we just defined. This concludes our proof that the interpretation of coinductive formulas through the equation in Proposition 8.28 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:

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

[13] Observe that this rule is exactly the same as in the previous section (see Figure 8.8).

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[14] 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[15] 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[16] (the definition of falsity values involves double-orthogonal sets which do not preserve supremums), and this does not apply.

8.5 Conclusion and perspectives

Recap of the journey In the end, 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. The calculus indeed includes co-fixpoint operators, which are lazily evaluated. To this end, dLPA$^\omega$ uses the design of the $\overline{\lambda}_{[lv\tau\star]}$-calculus of Ariola *et al.* [4], which we equipped in Chapter 6 with a type system and which we proved to be normalizing. In addition, the proof terms of dLPA$^\omega$ are dependently typed thanks to a restriction of dependencies to the negative-elimination-free fragment which makes them compatible with classical logic. These computational features allows dLPA$^\omega$ to internalize the realizability approach of [15, 40] as a direct proofs-as-programs interpretation: both proof terms for countable and dependent choices furnish a

[14] 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.

[15] 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 [30]. Yet, we doubt that the gain of the desired equality is worth a transfinite definition of the realizability interpretation.

[16] 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.

lazy witness for the ideal choice function which is evaluated on demand. At the risk of repeating ourself, 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*.

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 Chapter 7 the difficulties related to the definition of a sequent calculus with dependent types. We proposed a formalism, $\mathrm{dL}_{\hat{\mathfrak{tp}}}$, which restricts the dependencies to proofs in the NEF fragment together. This restriction is strengthened with an evaluation of dependently typed computations within delimited continuations; while the type system is enriched with an explicit list of dependencies. This provides us with a calculus whose reduction is safe, and which has the advantage of being suitable for a typed continuation-passing style translation carrying the dependencies.

On the other hand, we defined a typed continuation-and-store passing style translation for the $\overline{\lambda}_{[lv\tau\star]}$-calculus, which we related to Kripke forcing semantics. Besides, we saw how to handle laziness and explicit stores in a realizability interpretation *à la* Krivine. This might be a first step toward new interpretations of different axioms using laziness within Krivine classical realizability. In a long term perspective, it would be interesting to understand the impact of laziness and stores on the corresponding realizability algebras. More generally, the algebraic models that we study in the last part of this thesis are oriented toward the call-by-name and the call-by-value evaluation strategies. While it is probably the case that these structures could be modified to obtain call-by-need algebras, the structure of such algebras is not obvious *a priori*.

Comparison with Krivine's interpretations of dependent choice At the end of the day, we presented a calculus, dLPA$^{\omega}$, with the nice property of allowing for the direct definition of a proof term for the axiom of dependent choice. Beside, we prove the normalization and soundness of dLPA$^{\omega}$ by means of a realizability interpretation *à la* Krivine. Yet, the computational content we give to the axiom of dependent choice is pretty different of Krivine's usual realizer of the same [94]. 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 particular, such an instruction allows to compare the codes of two programs, so that terms of the λ_c-calculus extended with `quote` can reduce differently according to the code of the arguments they are given. In a more recent work [102], 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 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. We did not study at all this question for dLPA$^{\omega}$, especially we do not know whether it is suitable to define the same model of ZF + ¬ AC + ¬ CH (set theory with the negation of the axiom of choice and the negation of continuum hypothesis). Neither do we know if the induced model is compatible with the `quote` instruction (we conjecture that it is). It might be the case that our approach can be related with the one with a bar-recursor in [102]. 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.

The study of the structures of Krivine realizability models is already a hard question, and so is in general the problem of determining the consequences that a particular set of instructions or a specific

pole might have on on the model[17]. Nonetheless, the question still holds.

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 [150] in the context of Gödel's functional interpretation, and later adapted to the context of modified realizability by Berardi *et al* [15]. In the different settings of second-order arithmetic [97] and classical realizability, Krivine [94] 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 [16] 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^ω or dLPA^ω 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 [149, 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 [18] a translation of System F into a simply-typed total language with a variant of bar recursion.

[17] To quote the last PhD student in date who attempted to define purpose-oriented realizability models [2], they are like Forrest's Gump chocolates boxes, *"you never know what you're gonna get"*.