Normalization and continuation-passing-style interpretation of simply-typed call-by-need λ-calculus with control

Hugo Herbelin[1] and Étienne Miquey[1,2]

1 πr^2 **(INRIA), IRIF, Université Paris-Diderot, Paris, France**
herbelin@inria.fr
2 **IMERL, Facultad de Ingeniería, Univ. de la República, Montevideo, Uruguay**
emiquey@irif.fr

Abstract

Ariola *et al* defined a call-by-need λ-calculus with control, together with a sequent calculus presentation of it. They mechanically derive from the sequent calculus presentation a continuation-passing-style transformation simulating the reduction. In this paper we consider the simply-typed version of the calculus and proove its normalization by means of a realizability interpretation. This justifies a posteriori the design choice of the translation, and is to be contrasted with Okasaki *et al.* semantics which is not normalizing even in the simply-typed case. Besides, we also present a type system for the target language of the continuation-passing-style translation.

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems

Keywords and phrases call by-need; type system; normalization; realizability; CPS translation

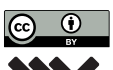
Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Call-by-name, call-by-value and call-by-need evaluation strategies

The call-by-name and call-by-value evaluation strategies are two basic strategies for evaluating the λ-calculus. The call-by-name evaluation strategy passes arguments to functions without evaluating them, postponing their evaluation to each place where the argument is needed, re-evaluating the argument several times if needed. Conversely, the call-by-value evaluation strategy evaluates the arguments of a function into so-called "values" prior to passing them to the function. The evaluation is then shared between the different places where the argument is needed. Yet, if the argument is not needed, it is evaluated uselessly.

The call-by-need evaluation strategy is a third evaluation strategy of the λ-calculus which evaluates arguments of functions only when needed, and, when needed, shares their evaluations across all places where the argument is needed. The call-by-need evaluation is at the heart of a functional programming language such as Haskell. It has in common with the call-by-value evaluation strategy that all places where a same argument is used share the same value. Nevertheless, it observationally behaves like the call-by-name evaluation strategy, in the sense that a given computation eventually evaluates to a value if and only if it evaluates to the same value (up to inner reduction) along the call-by-name evaluation. In particular, in a setting with non-terminating computations, it is not observationally equivalent to the call-by-value evaluation. Indeed, if the evaluation of a useless argument loops in the call-by-value evaluation, the whole computation loops, which is not the case of call-by-name and call-by-need evaluations.



Continuation-passing-style semantics for call-by-name, call-by-value and call-by-need calculi

The call-by-name, call-by-value and call-by-need evaluation strategies can be turned into equational theories. For call-by-name and call-by-value, this was done by Plotkin [19] through continuation-passing-style (CPS) semantics characterizing these theories. For call-by-name, the corresponding induced equational theory[1] is Church's original theory of the λ-calculus based on the operational rule β.

For call-by-value, Plotkin showed that the induced equational theory includes the key operational rule β_V. The induced equational theory was further completed implicitly by Moggi [16] with the convenient introduction of a native `let` operator[2]. Moggi's theory was then explicitly shown complete for cps semantics by Sabry and Felleisen [20].

For the call-by-need evaluation strategy, a specific equational theory reflecting the intentional behavior of the strategy into a semantics was proposed independently by Ariola and Felleisen [1] and by Maraist, Odersky and Wadler [15]. For call-by-need, a continuation-passing-style semantics was proposed in the 90s by Okasaki, Lee and Tarditi [17]. However, this semantics does not ensure normalization of simply-typed call-by-need evaluation, as shown in [2], thus failing to ensure a property which holds in the simply-typed call-by-name and call-by-value cases.

Call-by-name, call-by-value and call-by-need calculi with control

Continuation-passing-style semantics *de facto* gives a semantics to the extension of λ-calculus with control operators, i.e. with operators such as Scheme's `callcc`, Felleisen's $\mathcal{C}$, $\mathcal{K}$, or $\mathcal{A}$ operators [9], Parigot's μ and [ ] operators [18], Crolard's `catch` and `throw` operators [6]. In particular, even though call-by-name and call-by-need are observationally equivalent on pure λ-calculus, their different intentional behaviors induce different continuation-passing-style semantics, leading to different observational behaviors when control operators are considered.

Nonetheless, the semantics of calculi with control can also be reconstructed from an analysis of the duality between programs and their evaluation contexts, and the duality between the `let` construct (which binds programs) and a control operator such as Parigot's μ (which binds evaluation contexts). Such analysis can be done in the context of the $\mu\tilde{\mu}$-calculus [7, 10] and we consider for this purpose the following variant of the $\mu\tilde{\mu}$-calculus

[1] Later on, Lafont, Reus and Streicher [14] gave a more refined continuation-passing-style semantics which also validates the extensional rule η.

[2] In Plotkin, **let** $x = t$ **in** u is simulated by $(\lambda x.u)\, t$, but the latter fails to satisfy a Gentzen-style principle of "purity of methods" as it requires to know the constructor λ and destructor application of an arrow type for expressing something which is just a cut rule and has no reason to know about the arrow type. This is the same kind of purity of methods as in natural deduction compared to Frege-Hilbert systems: the latter uses the connective $\rightarrow$ to internalize derivability $\vdash$ leading to require $\rightarrow$ even when talking about the properties of say, $\wedge$. This is the same kind of purity of methods as in Parigot's classical natural deduction and $\lambda\mu$-calculus compared to say Prawitz's extension of natural deduction with Reduction ad absurdum: the latter uses the connective $\bot$ to internalize judgments with "no conclusion" and uses the connective $\neg$ to internalize the type of "evaluation contexts" (i.e. co-terms). The $\overline{\lambda}\mu\tilde{\mu}$-calculus [7] emphasizes a proof-as-program correspondence between "no conclusion" judgments and states of an abstract machine, between right-focused judgments and programs, and between left-focused judgments and evaluation contexts. Krivine [13], followed by Ariola *et al.* [3], used a notation $\perp\!\!\!\perp$ to characterize such "no conclusion" judgments.

which includes co-constants ranged over by $\boldsymbol{\alpha}$:

$$\begin{array}{llllllll}
\text{Strong values} & v & ::= & \lambda x.t \mid \mathbf{c} & \text{Strong contexts} & F & ::= & t \cdot e \mid \boldsymbol{\alpha} \\
\text{Weak values} & V & ::= & v \mid x & \text{Weak contexts} & E & ::= & F \mid \alpha \\
\text{Terms} & t, u & ::= & V \mid \mu\alpha.c & \text{Evaluation contexts} & e & ::= & E \mid \tilde{\mu}x.c
\end{array}$$

$$\text{Commands } c \ ::= \ \langle v \| e \rangle$$

Let us consider the following reduction rules parameterized over a sets of terms $\mathcal{V}$ and a set of evaluation contexts $\mathcal{E}$:

$$\begin{array}{lcll}
\langle t \| \tilde{\mu}x.c \rangle & \to & c[t/x] & v \in \mathcal{V} \\
\langle \mu\alpha.c \| e \rangle & \to & c[e/\alpha] & e \in \mathcal{E} \\
\langle \lambda x.t \| u \cdot e \rangle & \to & \langle u \| \tilde{\mu}x.\langle t \| e \rangle \rangle &
\end{array}$$

Then, the difference between call-by-name, call-by-value and call-by-need can be characterized by how the critical pair [3]

$$\begin{array}{ccc}
 & \langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle & \\
\swarrow & & \searrow \\
c[\tilde{\mu}x.c'/\alpha] & & c'[\mu\alpha.c/x]
\end{array}$$

is solved, which amounts to define $\mathcal{V}$ and $\mathcal{E}$ such that the two rules do not overlap:

- Call-by-name: $\mathcal{V}$ = Terms, $\mathcal{E}$ = Weak contexts
- Call-by-value: $\mathcal{V}$ = Weak values, $\mathcal{E}$ = Evaluation contexts
- Call-by-need: $\mathcal{V}$ = Weak values, $\mathcal{E}$ = Weak contexts $\cup$ Demanding contexts, where demanding contexts are expressions of the form $\tilde{\mu}x.C[\langle x \| F \rangle]$ where C, a meta-context, is a command with a hole as defined by the grammar

$$C[\,] \ ::= \ [\,] \mid \langle \mu\alpha.c \| \tilde{\mu}x.C \rangle$$

In particular, demanding contexts are evaluation contexts whose evaluation is blocked on the evaluation of x, therefore requiring the evaluation of what is bound to x. Also, meta-contexts are nesting of commands of the form $\langle v \| e \rangle$ for which neither v is in $\mathcal{V}$ (meaning it is some $\mu\alpha.c$) nor e in $\mathcal{E}$ (meaning it is an instance of some $\tilde{\mu}x.c$ which is not a forcing context).
The so-defined call-by-need calculus is close to the calculus called $\overline{\lambda}_{lv}$ in Ariola *et al* [2][4].

In the call-by-name and call-by-value cases, the approach based on $\mu\tilde{\mu}$-calculus leads to continuation-passing-style semantics similar to the ones given by Plotkin or, in the call-by-name case, also to the one by Lafont, Reus and Streicher [14]. In the case of call-by-need calculus, a continuation-passing-style semantics for $\overline{\lambda}_{lv}$ is defined in [2] via a calculus called $\overline{\lambda}_{[lv\tau\star]}$. This calculus is equivalent to $\overline{\lambda}_{lv}$ but is presented in such a way that the head redex of a command can be found by looking only at the surface of the command, from which a continuation-passing-style semantics directly comes. This semantics, distinct from the one in [17], is the object of study in this paper.

---

[3] There are different flavors of call-by-name, call-by-value and call-by-need calculi. We consider here flavors which are easy to explain in the given framework.

[4] The difference is in $t \cdot e$ which is $t \cdot E$ in [2]. Also, a similar calculus, which we shall call weak $\overline{\lambda}_{lv}$, was previously studied in [4] with $\mathcal{E}$ defined instead to be $\tilde{\mu}x.C[\langle x | E \|]\rangle$ (with same definition of C) and a definition of $\mathcal{V}$ which was different whether $\tilde{\mu}x.c$ was a forcing context ($\mathcal{V}$ was then the strong values) or not ($\mathcal{V}$ was then the weak values). Another variant is discussed in Section 6 of [2] where $\mathcal{E}$ is similarly defined to be $\tilde{\mu}x.C[\langle x | E \|]\rangle$ and $\mathcal{V}$ is defined to be (uniformly) the strong values. All three semantics seem to make sense to us.

$$\frac{(x:A)\in\Gamma}{\Gamma\vdash x:A\mid\Delta}\qquad\frac{\Gamma,x:A\vdash t:B\mid\Delta}{\Gamma\vdash\lambda x.t:A\to B\mid\Delta}\qquad\frac{c:(\Gamma\vdash\Delta,\alpha:A)}{\Gamma\vdash\mu\alpha.c:A\mid\Delta}$$

$$\frac{(\alpha:A)\in\Delta}{\Gamma\mid\alpha:A\vdash\Delta}\qquad\frac{\Gamma\vdash t:A\mid\Delta\qquad\Gamma\mid E:B\vdash\Delta}{\Gamma\mid t\cdot E\vdash\Delta}\qquad\frac{c:(\Gamma,x:A\vdash\Delta)}{\Gamma\mid\tilde{\mu}x.c:A\vdash\Delta}$$

$$\frac{\Gamma\vdash t:A\mid\Delta\qquad\Gamma\mid e:A\vdash\Delta}{\langle t\|e\rangle:(\Gamma\vdash\Delta)}\qquad\frac{(\boldsymbol{\alpha}:A)\in\mathcal{S}}{\Gamma\mid\boldsymbol{\alpha}:A\vdash\Delta}\qquad\frac{(\mathbf{c}:X)\in\mathcal{S}}{\Gamma\vdash\mathbf{c}:X\mid\Delta}$$

Figure 1 Typing rules for $\overline{\lambda}_{lv}$

Typing the continuation-passing-style for simply-typed call-by-need calculus

We shall concentrate on proving the normalization of the $\overline{\lambda}_{[lv\tau\star]}$-calculus and typing the continuation-passing-style transformation presented in [2]. Since evaluation of terms is shared, this continuation-passing-style is actually combined with a store-passing-style transformation. Moreover, the store can grow, so the translation also includes a Kripke-style forcing to address the extensibility of the store.

We shall first restate formally the calculus $\overline{\lambda}_{lv}$ from [2] which we shall equip with a system of simple types (Section 2). We shall then present $\overline{\lambda}_{[lv\tau\star]}$ (Section 3) which we shall also equip with a system of simple types (Section 4). The core of the paper consists first in giving a proof of normalization for the $\overline{\lambda}_{[lv\tau\star]}$-calculus (Section 5) by means of realizability techniques, second in typing the continuation-passing-style translation (Section 6).

For economy of space, most of the proofs of the results presented in this article are given in the appendices.

2 The simply-typed $\overline{\lambda}_{lv}$-calculus

We summarize the syntax of $\overline{\lambda}_{lv}$-calculus [2], which is a call-by-need instance of the $\bar{\lambda}\mu\tilde{\mu}$-calculus [7], as presented in the introduction[5].

Strong values	v	$::=$	$\lambda x.t \mid \mathbf{c}$	Forcing contexts	F	$::=$	$t\cdot E \mid \boldsymbol{\alpha}$
Weak values	V	$::=$	$v \mid x$	Catchable contexts	E	$::=$	$F \mid \alpha \mid \tilde{\mu}x.C[\langle x\|F\rangle]$
Terms	t,u	$::=$	$V \mid \mu\alpha.c$	Evaluation contexts	e	$::=$	$E \mid \tilde{\mu}x.c$

Commands	c	$::=$	$\langle v\|e\rangle$
Meta-contexts	C	$::=$	$[\,] \mid \langle\mu\alpha.c\|\tilde{\mu}x.C\rangle$

The $\overline{\lambda}_{lv}$ reduction, written as $\to_{lv}$, denotes the compatible reflexive transitive closure of the rules:

$$\begin{array}{lll}\langle V\|\tilde{\mu}x.c\rangle & \to_{lv} & c[V/x]\\ \langle\mu\alpha.c\|E\rangle & \to_{lv} & c[E/\alpha]\\ \langle\lambda x.t\|u\cdot E\rangle & \to_{lv} & \langle u\|\tilde{\mu}x.\langle t\|E\rangle\rangle\end{array}$$

A *forcing* contexts, which is either a stack $t\cdot E$ or a co-constant $\boldsymbol{\alpha}$, eagerly demands a value, and drives the computation forward. A variable is said to be *needed* or *demanded*

[5] In syntactic category, we implicitly assume $\tilde{\mu}x.c$ to only cover the cases which are not of the form $\tilde{\mu}x.C[\langle x\|F\rangle]$.

$$
\begin{array}{rcl}
\langle t \| \tilde{\mu} x.c \rangle \tau & \to & c\tau[x := t] \\
\langle \mu \alpha.c \| E \rangle \tau & \to & c\tau[\alpha := E] \\
\langle V \| \alpha \rangle \tau[\alpha := E] \tau' & \to & \langle V \| E \rangle \tau[\alpha := E] \tau' \\
\langle x \| F \rangle \tau[x := t] \tau' & \to & \langle t \| \tilde{\mu}[x].\langle x \| F \rangle \tau' \rangle \tau \\
\langle V \| \tilde{\mu}[x].\langle x \| F \rangle \tau' \rangle \tau & \to & \langle V \| F \rangle \tau[x := V] \tau' \\
\langle \lambda x.t \| u \cdot E \rangle \tau & \to & \langle u \| \tilde{\mu} x.\langle t \| E \rangle \rangle \tau
\end{array}
$$

Figure 2 Reduction rules of the $\overline{\lambda}_{[lv\tau\star]}$-calculus

if it is in a command with a forcing context, as in $\langle x \| F \rangle$. Furthermore, in a $\tilde{\mu}$-binding of the form $\tilde{\mu} x.C[\langle x \| F \rangle]$, we say that the bound variable x has been *forced*. The $C[\ ]$ is a meta-context, which identifies the standard redex in a command. Observe that the next reduction is not necessarily at the top of the command, but may be buried under several bound computations $\mu\alpha.c$. For instance, the command $\langle \mu\alpha.c \| \tilde{\mu} x_1.\langle x_1 \| \tilde{\mu} x_2.\langle x_2 \| F \rangle \rangle \rangle$, where x_1 is not needed, reduces to $\langle \mu\alpha.c \| \tilde{\mu} x_1.\langle x_1 \| F \rangle \rangle$, which now demands x_1.

The typing rules (see Figure 1) for the $\overline{\lambda}_{lv}$-calculus are the usual rules of the classical sequent calculus [7], where we adopt the convention that constants $\mathbf{c}$ and co-constants $\boldsymbol{\alpha}$ come with a signature $\mathcal{S}$ which assigns them a type.

3 The $\overline{\lambda}_{[lv\tau\star]}$-calculus syntax

While all the results that are presented in the sequel of this paper could be directly expressed using the $\overline{\lambda}_{lv}$-calculus, the continuation-passing-style translation we present naturally arises from the decomposition of this calculus into a different calculus with an explicit *environment*, the $\overline{\lambda}_{[lv\tau\star]}$-calculus [2]. Indeed, as we shall explain thereafter, the decomposition highlights different syntactic categories that are deeply involved in the definition and the typing of the continuation-passing-style translation.

The explicit environment of $\overline{\lambda}_{[lv\tau\star]}$-calculus, also called *substitution* of *store*, binds terms which are lazily evaluated. The reduction system resembles the one of an abstract machine.

Note that our approach slightly differ from [2] in that we split values into two categories: strong values (v) and weak values (V). The strong values correspond to values stricly speaking. The weak values include the variables which force the evaluation of terms to which they refer into shared strong value. Their evaluation may require capturing a continuation.

Besides, we reformulate the construction $\tilde{\mu} x.C[\langle x \| F \rangle]$ of $\overline{\lambda}_{lv}$ into $\tilde{\mu}[x].\langle x \| F \rangle \tau$. It expresses the fact that the variable x is forced at top-level. The syntax of the language is given by:

$$
\begin{array}{llcl@{\qquad}llcl}
\text{Strong values} & v & ::= & \lambda x.t \mid \mathbf{c} & \text{Forcing contexts} & F & ::= & \boldsymbol{\alpha} \mid t \cdot E \\
\text{Weak values} & V & ::= & v \mid x & \text{Catchable contexts} & E & ::= & F \mid \alpha \mid \tilde{\mu}[x].\langle x \| F \rangle \tau \\
\text{Terms} & t, u & ::= & V \mid \mu\alpha.c & \text{Evaluation contexts} & e & ::= & E \mid \tilde{\mu} x.c
\end{array}
$$

$$
\begin{array}{lcll}
\text{Closures} & l & ::= & c\tau \\
\text{Commands} & c & ::= & \langle t \| e \rangle \\
\text{Stores} & \tau & ::= & \varepsilon \mid \tau[x := t] \mid \tau[\alpha := E]
\end{array}
$$

and the reduction, written $\to$, is the compatible reflexive transitive closure of the rules given in Figure 2. The different syntactic categories can be understood as the different levels of alternation in a context-free abstract machine [2]: the priority is first given to contexts at level e (lazy storage of terms), then to terms at level p (evaluation of $\mu\alpha$ into values), then

$$\frac{(\mathbf{c} : A) \in \mathcal{S}}{\Gamma \vdash_v \mathbf{c} : A} \qquad \frac{\Gamma, x : A \vdash_t t : B}{\Gamma \vdash_v \lambda x.t : A \to B} \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash_V x : A} \qquad \frac{\Gamma \vdash_v v : A}{\Gamma \vdash_V v : A}$$

$$\frac{(\boldsymbol{\alpha} : A) \in \mathcal{S}}{\Gamma \vdash_F \boldsymbol{\alpha} : A^\perp} \qquad \frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_E E : B^\perp}{\Gamma \vdash_F t \cdot E : (A \to B)^\perp} \qquad \frac{(\alpha : A) \in \Gamma}{\Gamma \vdash_E \alpha : A^\perp} \qquad \frac{\Gamma \vdash_F F : A^\perp}{\Gamma \vdash_E F : A^\perp}$$

$$\frac{\Gamma \vdash_V V : A}{\Gamma \vdash_t V : A} \qquad \frac{\Gamma, \alpha : A^\perp \vdash_c c}{\Gamma \vdash_t \mu\alpha.c : A} \qquad \frac{\Gamma \vdash_E E : A^\perp}{\Gamma \vdash_e E : A^\perp} \qquad \frac{\Gamma, x : A \vdash_c c}{\Gamma \vdash_e \tilde{\mu}x.c : A^\perp}$$

$$\frac{\Gamma, x : A, \Gamma' \vdash_F F : A^\perp \quad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_E \tilde{\mu}[x].\langle x \| F \rangle \tau : A^\perp} \qquad \frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_e e : A^\perp}{\Gamma \vdash_c \langle t \| e \rangle} \qquad \frac{\Gamma, \Gamma' \vdash_c c \quad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l c\tau}$$

$$\frac{}{\Gamma \vdash_\tau \varepsilon : \varepsilon} \qquad \frac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_t t : A}{\Gamma \vdash_\tau \tau[x := t] : \Gamma', x : A} \qquad \frac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_E E : A^\perp}{\Gamma \vdash_\tau \tau[\alpha := E] : \Gamma', \alpha : A^\perp}$$

Figure 3 Typing rules of the $\overline{\lambda}_{[lv\tau\star]}$-calculus

back to contexts at level E and so on until level F. These different categories are directly reflected in the definition of the continuation-passing-style translation, and thus involved when typing it. We chose to highlight this by distinguishing different types of sequents already in the typing rules that we shall now present.

4 A type system for the $\overline{\lambda}_{[lv\tau\star]}$-calculus.

Unlike in the usual type system for sequent calculus where, as in the previous section, a judgment contains two typing contexts (one on the left for proofs, denoted by Γ, one on the right for contexts denoted by Δ), we group both of them into one single context, denoting the types for contexts (that used to be in Δ) with the exponent $\perp$. This allows to draw a strong connection in the sequel between the typing contexts Γ and the store τ, which contains both kind of terms.

We have nine kinds of sequents, one for typing each of the nine syntactic categories. We write them with an annotation on the $\vdash$ sign, using one of the letters v, V, t, F, E, e, l, c, τ. Sequents themselves are of four sorts: those typing values and terms are asserting a type, with the type written on the right; sequents typing contexts are expecting a type A with the type written $A^\perp$; sequents typing commands and closures are black boxes neither asserting nor expecting a type; sequents typing substitutions are instantiating a typing context. In other words, we have the following nine kinds of sequents:

$$\begin{array}{lll} \Gamma \vdash_l l & \Gamma \vdash_t t : A & \Gamma \vdash_e e : A^\perp \\ \Gamma \vdash_c c & \Gamma \vdash_V V : A & \Gamma \vdash_E E : A^\perp \\ \Gamma \vdash_\tau \tau : \Gamma' & \Gamma \vdash_v v : A & \Gamma \vdash_F F : A^\perp \end{array}$$

where types and typing contexts are defined by:

$$A, B ::= X \mid A \to B \qquad\qquad \Gamma ::= \varepsilon \mid \Gamma, x : A \mid \Gamma, \alpha : A^\perp$$

The typing rules are given on Figure 3 where we assume that a variable x (resp. covariable α) only occurs once in a context Γ (we implicitly assume the possibility of renaming variables by α-conversion). This type system enjoys the property of subject reduction.

▶ **Theorem 1** (Subject reduction). *If* $\Gamma \vdash_l c\tau$ *and* $c\tau \to c'\tau'$ *then* $\Gamma \vdash_l c'\tau'$.

5 Normalization of the $\overline{\lambda}_{[lv\tau\star]}$-calculus

We give here a proof of normalization for the $\overline{\lambda}_{[lv\tau\star]}$-calculus. The proof is inspired from techniques of Krivine's classical realizability [13], whose notations we borrow.

We proceed as follows: first, we generalize the usual notion of closed term to the notion of closed *context-in-store*. Intuitively, this means that we are no longer interested in closed terms and substitutions to close open terms, but rather in terms that are closed when considered in the current store. This is based on the simple observation that a store is nothing more than a shared substitution whose content might evolve along the execution. Second, we define a notion of *pole* $\perp\!\!\!\perp$, which are sets of closures allowing to relate terms and contexts thanks to a notion of orthogonality with respect to the pole. In particular, the set of normalizing closures is a valid pole. We then define for each formula A and typing level o (of e, t, E, V, F, v) a set $|A|_o$ (resp. $\|A\|_o$) of terms (resp. contexts) in the corresponding syntactic category. These sets correspond to reducibility candidates, or to what is usually called truth values and falsity values in realizability.

Finally, the core of the proof consists in the adequacy lemma, which shows that any closed term of type A at level o is in the corresponding set $|A|_o$. This guarantees that any typed closure is in any pole, and in particular in the pole of normalizing closures.

Actually, our proof is very closed to a proof by reducibility (see *e.g.* the proof of normalization for system D presented in [12, 3.2]), except that here the definitions are generalized over a parametric set $\perp\!\!\!\perp$ (which is usually taken to be SN in proofs by reducibility). The adequacy lemma, which is the central piece, evaluates in each case a state of an abstract machine (in our case a closure), so that this proof also proceeds by evaluation. A more detailed explanation of this observation as well as a more introductory presentation of normalization proofs by classical realizability are given in an article by Dagand and Scherer [8].

We begin by defining some key notions for stores that we shall need further in the proof.

▶ **Definition 2** (Closed store)**.** We extend the notion of free variable to stores:

$$\begin{array}{rcl} FV(\varepsilon) & \triangleq & \emptyset \\ FV(\tau[x := t]) & \triangleq & FV(\tau) \cup \{y \in FV(t) : y \notin \mathtt{dom}(\tau)\} \\ FV(\tau[\alpha := E]) & \triangleq & FV(\tau) \cup \{\beta \in FV(E) : \beta \notin \mathtt{dom}(\tau)\} \end{array}$$

so that we can define a *closed store* to be a store τ such that $FV(\tau) = \emptyset$.

▶ **Definition 3** (Compatible stores)**.** We say that two stores τ and τ' are *independent* and note $\tau \# \tau'$ when $\mathtt{dom}(\tau) \cap \mathtt{dom}(\tau') = \emptyset$. We say that they are *compatible* and note $\tau \diamond \tau'$ whenever for all variables x (resp. co-variables α) present in both stores: $x \in \mathtt{dom}(\tau) \cap \mathtt{dom}(\tau')$; the corresponding terms (resp. contexts) in τ and τ' coincide: $\tau(x) = \tau'(x)$. We say that τ' is an *extension* of τ and note $\tau \lhd \tau'$ whenever $\mathtt{dom}(\tau) \subseteq \mathtt{dom}(\tau')$ and $\tau \diamond \tau'$.

We denote by $\overline{\tau\tau'}$ the compatible union $\mathtt{join}(\tau, \tau')$, defined by:

$$\begin{array}{rcll} \mathtt{join}(\tau_0[x := t]\tau_1, \tau_0'[x := t]\tau_1') & \triangleq & \tau_0\tau_0'[x := t]\mathtt{join}(\tau_1, \tau_1') & \text{with } \tau \# \tau' \\ \mathtt{join}(\tau, \tau') & \triangleq & \tau\tau' & \text{if } \tau \# \tau' \\ \mathtt{join}(\varepsilon, \tau) & \triangleq & \tau & \\ \mathtt{join}(\tau, \varepsilon) & \triangleq & \tau & \end{array}$$

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

▶ **Lemma 4.** *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 $\tau_1 \lhd \overline{\tau_1}$.*

As we explained in the introduction of this section, we will not consider closed terms in the usual sense. Indeed, while it is frequent in the proofs of normalization (*e.g.* by realizability or reducibility) of a calculus to consider only closed terms and to perform substitutions to maintain the closure of terms, this only makes sense if it corresponds to the computational behavior of the calculus. For instance, to prove the normalization of $\lambda x.t$ in typed call-by-name $\lambda\mu\tilde{\mu}$-calculus, one would consider a substitution ρ that is suitable for with respect to the typing context Γ, then a context $u \cdot e$ of type $A \to B$, and evaluates :

$$\langle \lambda x.t_\rho \| u \cdot e \rangle \quad \to \quad \langle t_\rho[u/x] \| e \rangle$$

Then we would observe that $t_\rho[u/x] = t_{\rho[x:=u]}$ and deduce that $\rho[x := u]$ is suitable for $\Gamma, x : A$, which would allow us to conclude by induction.

However, in the $\overline{\lambda}_{[lv\tau\star]}$-calculus we do not perform global substitution when reducing a command, but rather add a new binding $[x := u]$ in the store:

$$\langle \lambda x.t \| u \cdot E \rangle \tau \quad \to \quad \langle t \| E \rangle \tau[x := u]$$

Therefore the natural notion of closed term invokes the closure under a store, which might evolve during the rest of the execution (this is to contrast with a substitution).

▶ **Definition 5** (Term-in-store)**.** We call *closed term-in-store* (resp. *closed context-in-store*, *closed closures*) the combination of a term t (resp. context e, command c) with a closed store τ such that $FV(t) \subseteq \mathtt{dom}(\tau)$. We use the notation $(t|\tau)$ to denote such a pair.

We should note that in particular, if t is a closed term, then $(t|\tau)$ is a term-in-store for any closed store τ. The notion of closed term-in-store is thus a generalization of the notion of closed terms, and we will (ab)use of this terminology in the sequel. We denote the sets of closed closures by $\mathcal{C}_0$, and will identify $(c|\tau)$ and the closure $c\tau$ when c is closed in τ. We are now equipped to define the notion of pole, and verify that the set of normalizing closures is indeed a valid pole.

▶ **Definition 6** (Pole)**.** A subset $\perp\!\!\!\perp \in \mathcal{C}_0$ is said to be *saturated* or *closed by anti-reduction* whenever for all $(c|\tau), (c'|\tau') \in \mathcal{C}_0$, if $c'\tau' \in \perp\!\!\!\perp$ and $c\tau \to c'\tau'$ then $c\tau \in \perp\!\!\!\perp$. It is said to be *closed by store extension* if whenever $c\tau \in \perp\!\!\!\perp$, for any store τ' extending τ: $\tau \lhd \tau'$, $c\tau' \in \perp\!\!\!\perp$. A *pole* is defined as any subset of $\mathcal{C}_0$ that is closed by anti-reduction and store extension.

▶ **Proposition 7.** *The set* $\perp\!\!\!\perp_\Downarrow = \{c\tau \in \mathcal{C}_0 : \ c\tau \text{ normalizes } \}$ *is a pole.*

We can now relate closed terms and contexts by orthogonality with respect to a given pole. This allows us to define for any formula A the sets $|A|_v, |A|_V, |A|_t$ (resp. $\|A\|_F, \|A\|_E, \|A\|_e$) of realizers (or reducibility candidates) at level v, V, t (resp F, E, e) for the formula A. It is to be observed that realizers are here closed terms-in-store.

▶ **Definition 8** (Orthogonality)**.** Given a pole $\perp\!\!\!\perp$, we say that a term-in-store $(t|\tau)$ is *orthogonal* to a context-in-store $(e|\tau')$ and write $(t|\tau)\perp\!\!\!\perp(e|\tau')$ if τ and τ' are compatible and $\langle t \| e \rangle \overline{\tau\tau'} \in \perp\!\!\!\perp$.

▶ **Definition 9** (Realizers)**.** We set:

$$\begin{array}{rcl}
|X|_v & = & \{(\mathbf{c}|\tau) : \ \vdash \mathbf{c} : X\} \\
|A \to B|_v & = & \{(\lambda x.t|\tau) : \forall u\tau', \tau \diamond \tau' \wedge (u|\tau') \in |A|_t \Rightarrow (t|\overline{\tau\tau'}[x := u]) \in |B|_t\} \\
\|A\|_F & = & \{(F|\tau) : \forall v\tau', \tau \diamond \tau' \wedge (v|\tau') \in |A|_v \Rightarrow (v|\tau')\perp\!\!\!\perp(F|\tau)\} \\
|A|_V & = & \{(V|\tau) : \forall F\tau', \tau \diamond \tau' \wedge (F|\tau') \in \|A\|_F \Rightarrow (V|\tau)\perp\!\!\!\perp(F|\tau')\} \\
\|A\|_E & = & \{(E|\tau) : \forall V\tau', \tau \diamond \tau' \wedge (V|\tau') \in |A|_V \Rightarrow (V|\tau')\perp\!\!\!\perp(E|\tau)\} \\
|A|_t & = & \{(t|\tau) : \forall E\tau', \tau \diamond \tau' \wedge (E|\tau') \in \|A\|_E \Rightarrow (t|\tau)\perp\!\!\!\perp(E|\tau')\} \\
\|A\|_e & = & \{(e|\tau) : \forall t\tau', \tau \diamond \tau' \wedge (t|\tau') \in |A|_t \Rightarrow (t|\tau')\perp\!\!\!\perp(e|\tau)\}
\end{array}$$

In comparison with the usual definition of Krivine's classical realizability, we only considered orthogonal sets restricted to some syntactical subcategories. However, the definition still satisfies the usual monotonicity properties of bi-orthogonal sets:

▶ **Proposition 10.** *For any type A and any given pole $\perp\!\!\!\perp$, we have the following inclusions*
1. $|A|_v \subseteq |A|_V \subseteq |A|_t$;
2. $\|A\|_F \subseteq \|A\|_E \subseteq \|A\|_e$.

We now extend the notion of realizers to stores, by stating that a store τ realizes a context Γ if it binds all the variables x and α in Γ to a realizer of the corresponding formula.

▶ **Definition 11.** Given a closed store τ, we say that τ realizes Γ and write $\tau \Vdash \Gamma$ if:
1. for any $(a : A) \in \Gamma$, $\tau \equiv \tau_0[a := t]\tau_1$ and $(t|\tau_0) \in |A|_t$
2. for any $(\alpha : A^\perp) \in \Gamma$, $\tau \equiv \tau_0[\alpha := E]\tau_1$ and $(E|\tau_0) \in \|A\|_E$

In the same way as weakening rules (for the typing context) were admissible for each level of the typing system :

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash_t t : A} \qquad \frac{\Gamma \vdash_e e : A^\perp \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash_e e : A^\perp} \qquad \cdots \qquad \frac{\Gamma \vdash_\tau \tau : \Gamma'' \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash_\tau \tau : \Gamma''}$$

the definition of realizers is compatible with a weakening of the store.

▶ **Lemma 12** (Store weakening)**.** *Let τ and τ' be two stores such that $\tau \lhd \tau'$, Γ be a typing context and let $\perp\!\!\!\perp$ be a pole. The following holds:*
1. *If $(t|\tau) \in |A|_t$ for some closed term $(t|\tau)$ and type A, then $(t|\tau') \in |A|_t$. The same holds for each level e, E, V, F, v of the typing rules.*
2. *If $\tau \Vdash \Gamma$ then $\tau' \Vdash \Gamma$.*

▶ **Definition 13** (Adequacy)**.** Given a fixed pole $\perp\!\!\!\perp$, we say that:
- A typing judgment $\Gamma \vdash_t t : A$ is *adequate* (w.r.t. the pole $\perp\!\!\!\perp$) if for all stores $\tau \Vdash \Gamma$, we have $(t|\tau) \in |A|_t$.
- More generally, we say that an inference rule

$$\frac{J_1 \quad \cdots \quad J_n}{J_0}$$

is adequate (w.r.t. the pole $\perp\!\!\!\perp$) if the adequacy of all typing judgments $J_1, \ldots, J_n$ implies the adequacy of the typing judgment J_0.

▶ Remark. From the latter definition, it is clear that a typing judgment that is derivable from a set of adequate inference rules is adequate too.

▶ **Lemma 14** (Adequacy)**.** *The typing rules of Figure 3 for the $\overline{\lambda}_{[lv\tau\star]}$-calculus without co-constants are adequate with any pole.*

The previous result required to consider the $\overline{\lambda}_{[lv\tau\star]}$-calculus without co-constants. Indeed, we consider co-constants as coming with their typing rules, potentially giving them any type (whereas constants can only be given an atomic type). That is why is *a priori* no reason[6] why their types should be adequate with any pole.

However, as observed in the previous remark, given a fixed pole it suffices to check whether the typing rules for a given co-constant are adequate with this pole. If they are, any judgment that is derivable using these rules will be adequate.

[6] Think for instance of a co-constant of type $(A \to B)^\perp$, there is no reason why it should be orthogonal to any function in $|A \to B|_v$.

▶ **Corollary 15.** *If $c\tau$ is a closure such that $\vdash_l c\tau$ is derivable, then for any pole $\perp\!\!\!\perp$ such that the typing rules for co-constants used in the derivation are adequate with $\perp\!\!\!\perp$, $c\tau \in \perp\!\!\!\perp$.*

We can now put our focus back on the normalization of typed closures. As we already saw in Proposition 7, the set $\perp\!\!\!\perp_{\Downarrow}$ of normalizing closure is a valid pole, so that it only remains to prove that any typing rule for constants and co-constants is adequate with $\perp\!\!\!\perp_{\Downarrow}$. This proposition directly stems from the observation that for any store τ and any closed strong value $(v|\tau') \in |A|_v$, $\langle v \| \boldsymbol{\alpha} \rangle \overline{\tau\tau'}$ does not reduce and thus belongs to the pole $\perp\!\!\!\perp_{\Downarrow}$.

▶ **Lemma 16.** *Any typing rule for co-constants is adequate with the pole $\perp\!\!\!\perp_{\Downarrow}$,* i.e. *if Γ is a typing context, and τ is a store such that $\tau \Vdash \Gamma$, if $\boldsymbol{\alpha}$ is a co-constant such that $\Gamma \vdash_F \boldsymbol{\alpha} : A^{\perp}$, then $(\boldsymbol{\alpha}|\tau) \in \|A\|_F$.*

As a consequence, we obtain the normalization of typed closures of the full calculus.

▶ **Theorem 17.** *If $c\tau$ is a closure of the $\overline{\lambda}_{[lv\tau\star]}$-calculus such that $\vdash_l c\tau$ is derivable, then $c\tau$ normalizes.*

Besides, the translations[7] from $\overline{\lambda}_{lv}$ to $\overline{\lambda}_{[lv\tau\star]}$ defined by Ariola *et al.* both preserve normalization [2, Theorem 2,4]. As it is clear that they also preserve typing, the previous result also implies the normalization of the $\overline{\lambda}_{lv}$-calculus:

▶ **Corollary 18.** *If c is a closure of the $\overline{\lambda}_{lv}$-calculus such that $c : (\vdash)$ is derivable, then c normalizes.*

6 A typed store-and-continuation-passing style translation

Guided by the normalization proof of the previous section, we shall now present a type system adapted to the continuation-passing style translation defined in [2]. The computational part is the very same, except for the fact that we explicitly handle renaming through a substitution σ that replaces names of the source language by names of the target. We consider in this section[8] that we dispose of a fresh names generator (for instance a global counter) and use names explicitly both in the language (for stores) and in the type system (for their types).

The transformation is actually not only a continuation-passing-style translation. Because of the sharing of the evaluation of arguments, the environment associating terms to variables behaves like a store which is passed around. Passing the store amounts to combine the continuation-passing-style translation with a store-passing-style translation. Additionally, the store is extensible, so, to anticipate extension of the store, Kripke style forcing has to be used too, in a way comparable to what is done in step-indexing translations.

Before presenting in detail the target system of the translation, let us explain step by step the rationale guiding the definition of the translation. In a first approximation, let us look only at the continuation-passing-style part of the translation of a $\overline{\lambda}_{[lv\tau\star]}$ sequent.

As shown in [2] and as emphasized by the definition of realizers (see Definition 9) reflecting the 6 nested syntactic categories used to define $\overline{\lambda}_{[lv\tau\star]}$, there are 6 different levels of control in call-by-need, leading to 6 mutually defined levels of interpretation. We define $[\![A \to B]\!]_v$ for strong values as $[\![A]\!]_t \to [\![B]\!]_E$, we define $[\![A]\!]_F$ for forcing contexts as $\neg\, [\![A]\!]_v$,

[7] There is actually an intermediate step to a calculus named $\bar{\lambda}_{[l\tau v]}$.
[8] A full presentation of this section using De Bruijn indexes instead of names is given in Appendix D.

$[\![A]\!]_V$ for weak values as $\neg\, [\![A]\!]_F = \overset{2}{\neg}\, [\![A]\!]_v$, and so on until $[\![A]\!]_e$ defined as $\overset{5}{\neg}\, [\![A]\!]_v$ (where $\overset{0}{\neg}\, A \triangleq A$ and $\overset{n+1}{\neg}\, A \triangleq \neg\, \overset{n}{\neg}\, A$).

As we already observed in the previous section (see Definition 11), hypothesis from a context Γ of the form $\alpha : A^{\perp}$ are to be translated as $[\![A]\!]_E = \overset{3}{\neg}\, [\![A]\!]_v$ while hypothesis of the form $x : A$ are to be translated as $[\![A]\!]_t = \overset{t}{\neg}\, [\![A]\!]_v$. Up to this point, if we denote this translation of Γ by $[\![\Gamma]\!]$, in the particular case of $\Gamma \vdash_t A$ the translation is $[\![\Gamma]\!] \vdash [\![A]\!]_t$ and similarly for other levels, *e.g.* $\Gamma \vdash_e A$ translates to $[\![\Gamma]\!] \vdash [\![A]\!]_e$.

The continuation-passing-style part being settled, the store-passing-style part should be considered. In particular, the translation of $\Gamma \vdash_t A$ is not anymore a sequent $[\![\Gamma]\!] \vdash [\![A]\!]_t$ but instead a sequent roughly of the form $\vdash [\![\Gamma]\!]_t \to [\![A]\!]_t$, with actually Γ being passed around not only at the top level of $[\![A]\!]_t$ but also every time a negation is used. Moreover, the translation of each type in Γ should itself be abstracted over the store at each use of a negation. So, for instance, at this stage of the explanation, the translation of a sequent $A^{\perp}, B \vdash_V C$ is $\vdash ([\![A]\!]_E, [\![B]\!]_t) \to [\![C]\!]_V$, where:

- $[\![A]\!]_E$ is $\overset{3}{\neg}\, [\![A]\!]_V$,
- $[\![B]\!]_t$, dependent on $[\![A]\!]_E$ at each level of negation is: $[\![A]\!]_E \to \neg([\![A]\!]_E \to \neg([\![A]\!]_E \to \neg([\![A]\!]_E \to \neg[\![B]\!]_v)))$,
- $[\![C]\!]_V$, also dependent on $[\![A]\!]_E$ and $[\![B]\!]_t$ is $[\![A]\!]_E \to [\![B]\!]_E \to \neg([\![A]\!]_E \to [\![B]\!]_t \to \neg[\![C]\!]_v)$.

The store-passing-style part being settled, it remains to anticipate that the store is extensible. This is done by supporting arbitrary insertions of any term at any place of the store. The extensibility is obtained by quantification over all possible extensions of the store at each level of the negation. To this end, we use as type system an adaptation of System $F_{<:}$ [5] extended with stores, defined as lists of assignations $[x := t]$. *Store types*, denoted by Υ, are defined as list of types of the form $(x : A)$ where x is a name and A is a type properly speaking and admit a subtyping notion $\Upsilon' <: \Upsilon$ to expressed that Υ' is an extension of Υ. The reader can think of subtyping as a sort of Kripke forcing [11], where *worlds* are store types Υ and *accessible worlds* from Υ are precisely all the possible $\Upsilon' <: \Upsilon$.

The target language is thus the usual λ-calculus extended with stores (defined lists of pairs of a name and a term) and second-order quantification over store types. We refer to this language as System F_{Υ}. We assume that types contain at least a constant for each atomic type X of the original system, still denote this constant by X. This allows us to define an embedding ι from the original type system to this one by:

$$\iota(X) = X \qquad\qquad \iota(A \to B) = \iota(A) \to \iota(B).$$

The syntax for terms and types is given by:

$$\begin{array}{rcl} t, u & ::= & \mathbf{c} \mid x \mid \lambda x.t \mid tu \mid \tau \\ & & \mid \mathbf{let}\ x_{\tau_0}, x, x_{\tau_1} = \mathbf{split}\, \tau''\, y\ \mathbf{in}\ t \\ \tau, \tau' & ::= & \varepsilon \mid \tau[x := t] \end{array} \qquad \begin{array}{rcl} A, B & ::= & X \mid \bot \mid \Upsilon \triangleright_{\tau} \Upsilon' \mid A \to B \mid \forall Y <: \Upsilon.\, A \\ \Upsilon, \Upsilon' & ::= & \varepsilon \mid (x : A) \mid (x : A^{\perp}) \mid Y \mid \Upsilon, \Upsilon' \\ \Gamma, \Gamma' & ::= & \varepsilon \mid \Gamma, x : A \mid \Gamma, Y <: \Upsilon \end{array}$$

We introduce a new symbol $\Upsilon \triangleright_{\tau} \Upsilon'$ to denote the fact that a store has a type conditioned by Υ (which should be the type of the head of the list). In order to ease the notations, we will denote Υ instead of $\varepsilon \triangleright_{\tau} \Upsilon$ in the sequel. On the contrary, $\Upsilon \triangleright_t A$ is a shorthand (defined in Figure 6). The type system is given in Figure 4 where we assume that a name can only occur once both in typing contexts Γ and stores types Υ.

▶ Remark. We shall make a few remarks about our choice of rules for typing stores. First, observe that we force elements of the store to have types of the form $\Upsilon \triangleright_t A$, that is having the structure of types obtained through the CPS translation. Even though this could appear

$$\frac{(\mathbf{c} : A) \in \mathcal{S}}{\Gamma \vdash \mathbf{c} : A} \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

$$\frac{\Gamma, Y <: \Upsilon \vdash t : A \quad Y \notin FV(\Gamma)}{\Gamma \vdash t : \forall X <: \Upsilon.A} \qquad \frac{\Gamma \vdash t : \forall Y <: \Upsilon.A \quad \Gamma \vdash \Upsilon' <: \Upsilon}{\Gamma \vdash t : A\{Y := \Upsilon'\}}$$

$$\frac{\Gamma, x_{\tau_0} : \Upsilon_0, x : \Upsilon_0 \triangleright_t A, x_{\tau_1} : (\Upsilon_0, y : A) \triangleright_\tau \Upsilon_1 \vdash t : A \quad \Gamma \vdash \tau : \Upsilon_0, y : B, \Upsilon_1}{\Gamma; \Sigma \vdash \mathbf{let}\ x_{\tau_0}, x, x_{\tau_1} = \mathbf{split}\, \tau\, y\ \mathbf{in}\ t : A}$$

$$\frac{}{\Gamma \vdash \varepsilon : \varepsilon \triangleright_\tau \varepsilon} \qquad \frac{\Gamma \vdash t : \Upsilon_0 \triangleright_t A}{\Gamma \vdash [x := t] : \Upsilon_0 \triangleright_\tau x : A} \qquad \frac{\Gamma \vdash t : \Upsilon_0 \triangleright_E A}{\Gamma \vdash [x := t] : \Upsilon_0 \triangleright_\tau x : A^\perp}$$

$$\frac{\Gamma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon \qquad \Gamma \vdash \tau' : (\Upsilon_0, \Upsilon) \triangleright_\tau \Upsilon'}{\Gamma \vdash \tau\tau' : \Upsilon_0 \triangleright_\tau \Upsilon, \Upsilon'} \qquad \frac{(\Upsilon' <: \Upsilon) \in \Gamma}{\Gamma \vdash \Upsilon' <: \Upsilon}\, <:_{\mathrm{ax}}$$

$$\frac{}{\Gamma \vdash \Upsilon <: \varepsilon}\, <:_\varepsilon \qquad \frac{\Gamma \vdash \Upsilon' <: \Upsilon}{\Gamma \vdash (\Upsilon', x : A) <: (\Upsilon, x : A)}\, <:_1 \qquad \frac{\Gamma \vdash \Upsilon' <: \Upsilon}{\Gamma \vdash \Upsilon', \Upsilon'' <: \Upsilon}\, <:_2$$

$$\frac{\Gamma \vdash \Upsilon'' <: \Upsilon' \quad \Gamma \vdash \Upsilon' <: \Upsilon}{\Gamma \vdash \Upsilon'' <: \Upsilon}\, <:_3 \qquad \frac{\Gamma \vdash \tau : \Upsilon'_0 \triangleright_\tau \Upsilon' \quad \Gamma \vdash \Upsilon' <: \Upsilon \quad \Gamma \vdash \Upsilon_0 <: \Upsilon'_0}{\Gamma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon}\, \tau <:$$

$$\frac{}{\Sigma \vdash Y <: Y}\, <:_Y \qquad \frac{\Gamma[(Y_0, x : A, Y_1)/\Upsilon'] \vdash t : B \qquad \Gamma \vdash Y <: (\Upsilon_0, x : A, \Upsilon_1)}{\Gamma \vdash t : B}\, <:_{\mathrm{split}}$$

Figure 4 Typing rules of System F_Υ

as a strong requirement, it appears naturally when giving a computational contents to the inclusion $\Upsilon <:' \Upsilon$ with De Bruijn indexes (see Appendix D.3). Indeed, a De Bruijn index (just as a name) can be understood as a pointer to a particular cell of the store. Therefore, we need to update pointers when inserting a new element (as in Proposition 19). Such an operation would not have any sense (and in particular be ill-typed) for an element that is not of type $\Upsilon \triangleright_t A$. This could be circumvent by tagging each cell of the store with a flag (using a sum type) indicating whether the corresponding elements have a type of this form or not. Second, note that each element of the store has a type depending on the type of the head of the store. Once again, this is natural and only reflects what was already happening in the source language or within the realizability interpretation.

The translation of judgments and types are given in Figures 5 and 6, where we made explicit the renaming procedure from the $\overline{\lambda}_{[lv\tau\star]}$-calculus to the target language. We denote by $\Gamma \,\mathfrak{s}\, \sigma$ the fact that σ is a substitution suitable to rename every names present in Γ.

As for the reduction rules of the language, there is only two of them, namely the usual β-reduction and the split of a store with respect to a name:

$$\begin{array}{rcll} \lambda x.t\, u & \to & t[u/x] & \\ \mathbf{let}\ x_0, x, x_1 = \mathbf{split}\, \tau\, y\ \mathbf{in}\ t & \to & t[\tau_0/x_0, u/x, \tau_1/x_1] & (\text{where } \tau = \tau_0[y := u]\tau_1) \end{array}$$

The translation of terms is given in Figure 7, where we assume that for each constant $\mathbf{c}$ of type A (resp. co-constant $\boldsymbol{\alpha}$ of type $A^\perp$) of the source system, we have a constant of type A in the signature $\mathcal{S}$ of target language, constant that we also denote by $\mathbf{c}$ (resp. $\boldsymbol{\alpha}$ of type $A \to \perp$). Except for the explicit renaming, the translation is the very same as in Ariola *et al.*, hence their results are preserved when considering a weak head-reduction strategy.

$$
\begin{array}{lcll}
[\![\Gamma \vdash_e e : A^\perp]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![e]\!]^\sigma_e : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_e \iota(A)) \\
[\![\Gamma \vdash_t t : A]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![t]\!]^\sigma_t : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_t \iota(A)) \\
[\![\Gamma \vdash_E E : A^\perp]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![E]\!]^\sigma_E : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_E \iota(A)) \\
[\![\Gamma \vdash_V V : A]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![V]\!]^\sigma_V : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_V \iota(A)) \\
[\![\Gamma \vdash_F F : A^\perp]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![F]\!]^\sigma_F : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_F \iota(A)) \\
[\![\Gamma \vdash_v v : A]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![v]\!]^\sigma_v : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_v \iota(A)) \\
[\![\Gamma \vdash_c c]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![c]\!]^\sigma_c : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_c \bot) \\
[\![\Gamma \vdash_l l]\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash [\![l]\!]^\sigma_l : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_c \bot) \\
[\![\Gamma \vdash_\tau \tau : \Gamma']\!] & \triangleq & \forall \sigma,\ \sigma \Subset \Gamma \Rightarrow & (\vdash \tau' : [\![\Gamma]\!]^{\sigma'}_\Gamma \triangleright_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma) \qquad (\text{where } \tau', \sigma' = [\![\tau]\!]^\sigma_\tau) \\
\\
\sigma \Subset \Gamma & \triangleq & \sigma \text{ injective } \wedge\, \mathtt{dom}(\Gamma) \subseteq \mathtt{dom}(\sigma) &
\end{array}
$$

Figure 5 Translation of judgments

$$
\begin{array}{ll}
\Upsilon \triangleright_c A & \triangleq \forall Y <: \Upsilon.\, Y \to \bot \\
\Upsilon \triangleright_e A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_t A) \to \bot \\
\Upsilon \triangleright_t A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_E A) \to \bot \\
\Upsilon \triangleright_E A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_V A) \to \bot \\
\Upsilon \triangleright_V A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_F A) \to \bot \\
\Upsilon \triangleright_F A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_v A) \to \bot \\
\Upsilon \triangleright_v A \to B & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_t A) \to (Y \triangleright_E B) \to \bot \\
\Upsilon \triangleright_v X & \triangleq X \\
\\
[\![\varepsilon]\!]^\sigma_\Gamma & \triangleq \varepsilon \\
[\![\Gamma, a : A]\!]^\sigma_\Gamma & \triangleq [\![\Gamma]\!]^\sigma_\Gamma, \sigma(a) : \iota(A) \\
[\![\Gamma, \alpha : A^\perp]\!]^\sigma_\Gamma & \triangleq [\![\Gamma]\!]^\sigma_\Gamma, \sigma(\alpha) : \iota(A)^\perp
\end{array}
$$

Figure 6 Translation of types

We only mention here the following proposition which is the analogue of Proposition 12 for typing and play a central part in the proof of Theorem 20.

▶ **Proposition 19.** *For any level o of the hierarchy $\tau, c, e, t, E, V, F, v$, the following rule is admissible:*

$$\frac{\Gamma \vdash t : \Upsilon_0 \triangleright_o A \quad \Gamma \vdash \Upsilon_1 <: \Upsilon_0}{\Gamma \vdash t : \Upsilon_1 \triangleright_o A}$$

▶ **Theorem 20.** *The translation is well-typed, i.e.*

1. *if* $\Gamma \vdash_v v : A$ *then* $[\![\Gamma \vdash_v v : A]\!]$
2. *if* $\Gamma \vdash_F F : A^\perp$ *then* $[\![\Gamma \vdash_F F : A^\perp]\!]$
3. *if* $\Gamma \vdash_V V : A$ *then* $[\![\Gamma \vdash_V V : A]\!]$
4. *if* $\Gamma \vdash_E E : A^\perp$ *then* $[\![\Gamma \vdash_E E : A^\perp]\!]$
5. *if* $\Gamma \vdash_t t : A$ *then* $[\![\Gamma \vdash_t t : A]\!]$
6. *if* $\Gamma \vdash_e e : A^\perp$ *then* $[\![\Gamma \vdash_e e : A^\perp]\!]$
7. *if* $\Gamma \vdash_c c$ *then* $[\![\Gamma \vdash_c c]\!]$
8. *if* $\Gamma \vdash_l l$ *then* $[\![\Gamma \vdash_l l]\!]$
9. *if* $\Gamma \vdash_\tau \tau$ *then* $[\![\Gamma \vdash_\tau \tau : \Gamma']\!]$

Combining the preservation of reduction through the CPS and a proof of normalization of our target language (that one could obtain for instance using realizability techniques again), the former theorem would provides us with an alternative proof of normalization

$$
\begin{array}{lcll}
[\![\mathbf{c}]\!]^{\sigma}_{v} & \triangleq & \mathbf{c} & \\
[\![\lambda x.t]\!]^{\sigma}_{v}\ \tau\, u\, E & \triangleq & [\![t]\!]^{\sigma[x:=n]}_{t}\ \tau[n := u]\ E & (n \text{ fresh}) \\
[\![\boldsymbol{\alpha}]\!]^{\sigma}_{F} & \triangleq & \boldsymbol{\alpha} & \\
[\![t \cdot E]\!]^{\sigma}_{F}\ \tau\, v & \triangleq & v\ \tau\ t\ E & \\
[\![v]\!]^{\sigma}_{V}\ \tau\ F & \triangleq & F\ \tau\ [\![v]\!]^{\sigma}_{v} & \\
[\![x]\!]^{\sigma}_{V}\ \tau[\sigma(x) := t]\tau'\ F & \triangleq & t\ \tau\ (\lambda\tau\lambda V.V\ \tau[\sigma(x) :=\uparrow^{t} V]\tau'\ F) & (\text{with } \uparrow^{t} V = \lambda\tau E.E\,\tau\,V) \\
[\![\alpha]\!]^{\sigma}_{E}\ \tau[\sigma(\alpha) := E]\tau'\ V & \triangleq & E\ \tau[\sigma(\alpha) := E]\tau'\ V & \\
[\![\tilde{\mu}[x].\langle x \| F\rangle\tau']\!]^{\sigma}_{E} E\ \tau\ V & \triangleq & V\ \tau[n :=\uparrow^{t} V]\tau''\ [\![F]\!]^{\sigma'}_{F} & (\text{where } n \text{ fresh}, \tau'', \sigma' = [\![\tau]\!]^{\sigma[x:=n]}_{\tau}) \\
[\![V]\!]^{\sigma}_{t}\ \tau\ E & \triangleq & E\ \tau\ [\![V]\!]^{\sigma}_{V} & \\
[\![\mu\alpha.c]\!]^{\sigma}_{t}\ \tau\ E & \triangleq & [\![c]\!]^{\sigma[\alpha:=n]}_{c}\ \tau[n := E] & (n \text{ fresh}) \\
[\![E]\!]^{\sigma}_{e}\ \tau\ t & \triangleq & t\ \tau\ [\![E]\!]^{\sigma}_{E} & \\
[\![\tilde{\mu}x.c]\!]^{\sigma}_{e}\ \tau\ t & \triangleq & [\![c]\!]^{\sigma[x:=n]}_{c}\ \tau[n := t] & (n \text{ fresh}) \\
[\![\langle t \| e\rangle]\!]^{\sigma}_{c}\ \tau & \triangleq & [\![e]\!]^{\sigma}_{e}\ \tau\ [\![t]\!]^{\sigma}_{t} & \\
[\![c\ \tau]\!]^{\sigma}_{l}\ \tau_0 & \triangleq & [\![c]\!]^{\sigma'}_{c}\ \tau_0\tau' & (\text{where } \tau', \sigma' = [\![\tau]\!]^{\sigma}_{\tau}) \\
[\![\varepsilon]\!]^{\sigma}_{\tau} & \triangleq & \varepsilon, \sigma & \\
[\![\tau'[x := t]]\!]^{\sigma}_{\tau} & \triangleq & \tau'[n := [\![t]\!]^{\sigma'}_{t}], \sigma[x := n] & (\text{where } \tau', \sigma' = [\![\tau]\!]^{\sigma}_{\tau},\ n \text{ fresh}) \\
[\![\tau'[\alpha := E]]\!]^{\sigma}_{\tau} & \triangleq & \tau'[n := [\![E]\!]^{\sigma'}_{E}], \sigma[\alpha := n] & (\text{where } \tau', \sigma' = [\![\tau]\!]^{\sigma}_{\tau},\ n \text{ fresh})
\end{array}
$$

Figure 7 Translation of terms

of the $\overline{\lambda}_{[lv\tau\star]}$-calculus. This is to be contrasted with Okasaki, Lee and Tarditi's semantics which is not normalizing in the simply-typed case, as shown in Ariola *et al* [2].

Acknowledgments

The first author is thanking Keiko Nakata, as well as José Carlos Espírito Santo, Luís Pinto, Zena Ariola, Paul Downen, Alexis Saurin and Rossen Mikhov for initial discussions on typing the continuation-passing-style semantics of call-by-need λ-calculus.

References

1 Zena Ariola and Matthias Felleisen. The call-by-need lambda calculus. *J. Funct. Program.*, 7(3):265–301, 1993. doi:10.1017/S0956796897002724.

2 Zena M. Ariola, Paul Downen, Hugo Herbelin, Keiko Nakata, and Alexis Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, Lecture Notes in Computer Science, pages 32–46. Springer, 2012. doi:10.1007/978-3-642-29822-6.

3 Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-theoretic foundation of continuations and prompts. In *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snowbird, UT, USA, September 19-21, 2004*, pages 40–53. ACM Press, New York, 2004. doi:10.1145/1016848.1016860.

4 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2011. doi:10.1007/978-3-642-21691-6_6.

5 Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. *An extension of system F with subtyping*, pages 750–770. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. URL: http://dx.doi.org/10.1007/3-540-54415-1_73, doi:10.1007/3-540-54415-1_73.

6 Tristan Crolard. A confluent lambda-calculus with a catch/throw mechanism. *J. Funct. Program.*, 9(6):625–647, 1999. doi:10.1017/S0956796899003512.

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

8 Pierre-Évariste Dagand and Gabriel Scherer. Normalization by realizability also evaluates. In David Baelde and Jade Alglave, editors, *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015)*, Le Val d'Ajol, France, January 2015. URL: https://hal.inria.fr/hal-01099138.

9 Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. Reasoning with continuations. In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 131–141. IEEE Computer Society, 1986.

10 Hugo Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Habilitation thesis, University Paris 11, December 2005.

11 Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.

12 Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood series in computers and their applications. Masson, 1993.

13 Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses*, 2004. To appear.

14 Yves Lafont, Bernhard Reus, and Thomas Streicher. Continuations semantics or expressing implication by negation. Technical Report 9321, Ludwig-Maximilians-Universität, München, 1993.

15 John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *J. Funct. Program.*, 8(3):275–317, 1998. doi:10.1017/S0956796898003037.

16 Eugenio Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Edinburgh Univ., 1988. doi:10.1109/LICS.1989.39155.

17 Chris Okasaki, Peter Lee, and David Tarditi. Call-by-need and continuation-passing style. *Lisp and Symbolic Computation*, 7(1):57–82, 1994. doi:10.1007/BF01019945.

18 Michel Parigot. Free deduction: An analysis of "computations" in classical logic. In Andrei Voronkov, editor, *Proceedings of LPAR*, volume 592 of *LNCS*, pages 361–380. Springer, 1991. URL: http://dx.doi.org/10.1007/3-540-55460-2_27.

19 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

20 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993. doi:10.1007/BF01019462.

A Proof of Section 4

▶ **Lemma 21.** *The following rule is admissible for any level o of the hierarchy $e, t, E, V, F, v, c, l, \tau$:*

$$\frac{\Gamma \vdash_o o : A \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash_o o : A}$$

Proof. Easy induction on typing rules. ◀

▶ **Theorem 1.** *If $\Gamma \vdash_l c\tau$ and $c\tau \to c'\tau'$ then $\Gamma \vdash_l c'\tau'$.*

Proof. **Case** $\langle t \| \tilde{\mu}x.c \rangle \tau \to c\tau[x := t]$.

A typing derivation of the closure of the left has the form:

$$\dfrac{\dfrac{\Pi_t}{\Gamma, \Gamma' \vdash_t t : A} \quad \dfrac{\dfrac{\Pi_c}{\Gamma, \Gamma', x : A \vdash_c c}}{\Gamma, \Gamma' \vdash_e \tilde{\mu}x.c : A} \quad \dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'}}{\dfrac{\Gamma, \Gamma' \vdash_c \langle t \| \tilde{\mu}x.c \rangle}{\Gamma \vdash_l \langle t \| \tilde{\mu}x.c \rangle \tau}}$$

hence we can derive:

$$\dfrac{\dfrac{\Pi_c}{\Gamma, \Gamma', x : A \vdash_c c} \quad \dfrac{\dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'} \quad \dfrac{\Pi_t}{\Gamma, \Gamma' \vdash_t t : A}}{\Gamma \vdash_\tau \tau[x := t] : (\Gamma', x : A)}}{\Gamma \vdash_l c\tau[x := t]}$$

Case $\langle \mu\alpha.c \| E \rangle \tau \to c\tau[\alpha := E]$.

A typing derivation of the closure of the left has the form:

$$\dfrac{\dfrac{\dfrac{\Pi_c}{\Gamma, \Gamma', \alpha : A^\perp \vdash_c c}}{\Gamma, \Gamma' \vdash_t \mu\alpha.c : A} \quad \dfrac{\dfrac{\Pi_E}{\Gamma, \Gamma' \vdash_E E : A^\perp}}{\Gamma, \Gamma' \vdash_e E : A^\perp}}{\Gamma, \Gamma' \vdash_c \langle \mu\alpha.c \| E \rangle} \quad \dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'}}{\Gamma \vdash_l \langle t \| \tilde{\mu}x.c \rangle \tau}$$

hence we can derive:

$$\dfrac{\dfrac{\Pi_c}{\Gamma, \Gamma', \alpha : A^\perp \vdash_c c} \quad \dfrac{\dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'} \quad \dfrac{\Pi_E}{\Gamma, \Gamma' \vdash_E E : A}}{\Gamma \vdash_\tau \tau[\alpha := E] : (\Gamma', \alpha : A^\perp)}}{\Gamma \vdash_l c\tau[\alpha := E]}$$

Case $\langle V \| \alpha \rangle \tau[\alpha := E]\tau' \to \langle V \| E \rangle \tau[\alpha := E]\tau'$.

A typing derivation of the closure of the left has the form:

$$\dfrac{\dfrac{\dfrac{\Pi_V}{\Gamma, \Gamma_0, \alpha : A^\perp, \Gamma_1 \vdash_t V : A} \quad \dfrac{\dfrac{\Gamma, \Gamma_0, \alpha : A^\perp, \Gamma_1 \vdash_F \alpha : A^\perp}{\Gamma, \Gamma_0, \alpha : A^\perp, \Gamma_1 \vdash_E \alpha : A^\perp}}{\Gamma, \Gamma_0, \alpha : A^\perp, \Gamma_1 \vdash_e \alpha : A^\perp}}{\Gamma, \Gamma_0, \alpha : A^\perp, \Gamma_1 \vdash_c \langle V \| \alpha \rangle} \quad \dfrac{\dfrac{\Pi_\tau}{\Gamma \vdash \tau : \Gamma_0} \quad \dfrac{\Pi_E}{\Gamma, \Gamma_0 \vdash_E E : A^\perp}}{\Gamma \vdash_\tau \tau[\alpha := E] : \Gamma_0, \alpha : A^\perp} \quad \Pi_{\tau'}}{\Gamma \vdash_l \langle V \| \alpha \rangle \tau[\alpha := E]\tau'}$$

where we cheated to compact $\Pi_{\tau'}$. Therefore, we can derive:

$$\dfrac{\dfrac{\Pi_V}{\Gamma,\Gamma_0,\alpha:A^\perp,\Gamma_1\vdash_t V:A}\quad\dfrac{\dfrac{\Pi_E}{\Gamma,\Gamma_0,\alpha:A^\perp,\Gamma_1\vdash_E E:A^\perp}}{\Gamma,\Gamma_0,\alpha:A^\perp,\Gamma_1\vdash_e E:A^\perp}}{\Gamma,\Gamma_0,\alpha:A^\perp,\Gamma_1\vdash_c\langle V\|E\rangle}\quad\dfrac{\dfrac{\Pi_\tau}{\Gamma\vdash\tau:\Gamma_0}\quad\dfrac{\Pi_E}{\Gamma,\Gamma_0\vdash_E E:A^\perp}}{\Gamma\vdash_\tau\tau[\alpha:=E]:\Gamma_0,\alpha:A^\perp}\quad\Pi_{\tau'}}{\Gamma\vdash_l\langle V\|\alpha\rangle\tau[\alpha:=E]\tau'}$$

Case $\langle x\|F\rangle\tau[x:=t]\tau'\to\langle t\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle\tau$.

A typing derivation of the closure of the left has the form:

$$\dfrac{\dfrac{\dfrac{}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_V x:A}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_t x:A}\quad\dfrac{\Pi_F}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_e F:A^\perp}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_c\langle x\|F\rangle}\quad\dfrac{\dfrac{\Pi_\tau}{\Gamma\vdash\tau:\Gamma_0}\quad\dfrac{\Pi_t}{\Gamma,\Gamma_0\vdash_t t:A}}{\Gamma\vdash_\tau\tau[x:=t]:\Gamma_0,x:A}\quad\Pi_{\tau'}}{\Gamma\vdash_l\langle V\|F\rangle\tau[x:=t]\tau'}$$

hence we can derive:

$$\dfrac{\dfrac{\Pi_t}{\Gamma,\Gamma_0,\Gamma_1\vdash_t t:A}\quad\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_V x:A}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_t x:A}\quad\dfrac{\Pi_F}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_e F:A^\perp}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_c\langle x\|F\rangle}\quad\dfrac{\Pi_{\tau'}}{\Gamma,\Gamma_0,x:A\vdash_\tau\tau':\Gamma_1}}{\Gamma,\Gamma_0,x:A\vdash_l\langle x\|F\rangle\tau'}}{\Gamma,\Gamma_0\vdash_E\tilde\mu[x].\langle x\|F\rangle\tau':A^\perp}}{\Gamma,\Gamma_0\vdash_e\tilde\mu[x].\langle x\|F\rangle\tau':A^\perp}\quad\dfrac{\Pi_\tau}{\Gamma\vdash\tau:\Gamma_0}}{\Gamma,\Gamma_0\vdash_c\langle t\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle}}{\Gamma\vdash_l\langle t\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle\tau}$$

Case $\langle V\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle\tau\to\langle V\|F\rangle\tau[x:=V]\tau'$.

A typing derivation of the closure of the left has the form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_V x:A}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_t x:A}\quad\dfrac{\Pi_F}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_e F:A^\perp}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_c\langle x\|F\rangle}\quad\dfrac{\Pi_{\tau'}}{\Gamma,\Gamma_0,x:A\vdash_\tau\tau':\Gamma_1}}{\Gamma,\Gamma_0,x:A\vdash_l\langle x\|F\rangle\tau'}}{\Gamma,\Gamma_0\vdash_E\tilde\mu[x].\langle x\|F\rangle\tau':A^\perp}}{\Gamma,\Gamma_0\vdash_e\tilde\mu[x].\langle x\|F\rangle\tau':A^\perp}\quad\dfrac{\Pi_\tau}{\Gamma\vdash\tau:\Gamma_0}}{\Gamma,\Gamma_0\vdash_c\langle V\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle}}{\Gamma\vdash_l\langle V\|\tilde\mu[x].\langle x\|F\rangle\tau'\rangle\tau}$$

Therefore we can derive:

$$\dfrac{\dfrac{\Pi_V}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_t V:A}\quad\dfrac{\Pi_F}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_e F:A^\perp}}{\Gamma,\Gamma_0,x:A,\Gamma_1\vdash_c\langle V\|F\rangle}\quad\dfrac{\dfrac{\Pi_\tau}{\Gamma\vdash\tau:\Gamma_0}\quad\dfrac{\Pi_V}{\Gamma,\Gamma_0\vdash_t V:A}}{\Gamma\vdash_\tau\tau[x:=V]:\Gamma_0,x:A}\quad\Pi_{\tau'}}{\Gamma\vdash_l\langle V\|F\rangle\tau[x:=V]\tau'}$$

where we implicitly used Lemma 21 to weaken Π_V:

$$\dfrac{\dfrac{\Pi_V}{\Gamma,\Gamma_0 \vdash_t V : A} \quad \Gamma,\Gamma_0 \subseteq \Gamma,\Gamma_0,x:A,\Gamma_1}{\Gamma,\Gamma_0,x:A,\Gamma_1 \vdash_t V : A}$$

Case $\langle \lambda x.t \| u \cdot E \rangle \tau \to \langle u \| \tilde{\mu} x.\langle t \| E \rangle \rangle \tau$.

A typing proof for the closure on the left is of the form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma,\Gamma',x:A \vdash_t t : B}}{\Gamma,\Gamma' \vdash_v \lambda x.t : A \to B}}{\Gamma,\Gamma' \vdash_V \lambda x.t : A \to B}}{\Gamma,\Gamma' \vdash_t \lambda x.t : A \to B} \quad \dfrac{\dfrac{\Pi_u}{\Gamma,\Gamma' \vdash_t u : A} \quad \dfrac{\Pi_E}{\Gamma,\Gamma' \vdash_E E : B^\perp}}{\Gamma,\Gamma' \vdash_E u \cdot e : (A \to B)^\perp}}{\Gamma,\Gamma' \vdash_e u \cdot e : (A \to B)^\perp} \qquad \dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'}$$
$$\dfrac{\Gamma,\Gamma' \vdash_c \langle \lambda x.t \| u \cdot E \rangle \qquad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l \langle \lambda x.t \| u \cdot E \rangle \tau}$$

We can thus build the following derivation:

$$\dfrac{\dfrac{\Pi_u}{\Gamma,\Gamma' \vdash_t u : A} \quad \dfrac{\dfrac{\dfrac{\Pi_t}{\Gamma,\Gamma',x:A \vdash_t t : B} \quad \dfrac{\dfrac{\Pi_E}{\Gamma,\Gamma',x:A \vdash_E E : B^\perp}}{\Gamma,\Gamma',x:A \vdash_e E : B^\perp}}{\Gamma,\Gamma',x:A \vdash_c \langle t \| E \rangle}}{\Gamma,\Gamma' \vdash_e \tilde{\mu} x.\langle t \| E \rangle : A^\perp} \qquad \dfrac{\Pi_\tau}{\Gamma \vdash_\tau \tau : \Gamma'}$$
$$\dfrac{\Gamma,\Gamma' \vdash_c \langle u \| \tilde{\mu} x.\langle t \| E \rangle \rangle \qquad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l \langle u \| \tilde{\mu} x.\langle t \| E \rangle \rangle \tau}$$

where we implicitly used Lemma 21 to weaken Π_E:

$$\dfrac{\dfrac{\Pi_E}{\Gamma,\Gamma \vdash_E E : B^\perp} \quad \Gamma,\Gamma' \subseteq \Gamma,\Gamma',x:A}{\Gamma,\Gamma',x:A \vdash_E E : B^\perp}$$

◀

B Proofs of Section 5

▶ **Proposition 7.** *The set* $\perp\!\!\!\perp_\Downarrow = \{c\tau \in \mathcal{C}_0 : \ c\tau \text{ normalizes } \}$ *is a pole.*

Proof. As we only considered closures in $\mathcal{C}_0$, both conditions (closure by anti-reduction and store extension) are clearly satisfied:

- if $c\tau \to c'\tau'$ and $c'\tau'$ normalizes, then $c\tau$ normalizes too;
- if c is closed in τ and $c\tau$ normalizes, if $\tau \lhd \tau'$ then $c\tau'$ will reduce as $c\tau$ (since it only uses terms in τ' that already were in τ) and thus will normalize.

◀

▶ **Proposition 10.** *For any type A and any given pole $\perp\!\!\!\perp$, we have the following inclusions*

1. $|A|_v \subseteq |A|_V \subseteq |A|_t$;
2. $\|A\|_F \subseteq \|A\|_E \subseteq \|A\|_e$.

Proof. All the inclusions are proved in a similar way. We only give the proof for $|A|_v \subseteq |A|_V$. Let $\perp\!\!\!\perp$ be a pole and $(v|\tau)$ be in $|A|_v$. We want to show that $(v|\tau)$ is in $|A|_V$, that is that v is in the syntactic category V (which is true) and that for any $(F|\tau') \in \|A\|_F$ such that $\tau <: \tau'$, $(v|\tau)\perp\!\!\!\perp(F|\tau')$. But this holds by definition of $(F|\tau') \in \|A\|_F$, since $(v|\tau) \in |A|_v$. ◀

▶ **Lemma 12** (Store weakening). *Let τ and τ' be two stores such that $\tau \lhd \tau'$, Γ be a typing context and let $\perp\!\!\!\perp$ be a pole. The following holds:*

1. *If $(t|\tau) \in |A|_t$ for some closed term $(t|\tau)$ and type A, then $(t|\tau') \in |A|_t$. The same holds for each level e, E, V, F, v of the typing rules.*
2. *If $\tau \Vdash \Gamma$ then $\tau' \Vdash \Gamma$.*

Proof. 1. This essentially amounts to the following observations. First, one remark that if $(t|\tau)$ is a closed term, so is $(t|\overline{\tau\tau'})$ for any store τ' compatible with τ. Second, we observe that if we consider for instance a closed context $(E|\tau'') \in \|A\|_E$, then $\overline{\tau\tau'} <: \tau''$ implies $\tau <: \tau''$, thus $(t|\tau)\perp\!\!\!\perp(E|\tau'')$ and finally $(t|\overline{\tau\tau'})\perp\!\!\!\perp(E|\tau'')$ by closure of the pole under store extension.

2. By definition, for all $(x : A) \in \Gamma$, τ is of the form $\tau_0[x := t]\tau_1$ such that $(\tau(x)|\tau_0) \in |A|_t$. As τ and τ' are compatible, by Lemma 4 $\overline{\tau\tau'}$ is of the form $\tau_0'[x := t]\tau_1'$ with τ_0' an extension of τ_0, and using the first point we get that $(t|\tau_0') \in |A|_t$.

◀

▶ **Lemma 14.** *Let Γ be a typing context, $\perp\!\!\!\perp$ be a pole and τ be a store such that $\tau \Vdash \Gamma$. The following holds in the $\overline{\lambda}_{[lv\tau\star]}$-calculus without co-constants:*

1. *If v is a strong value such that $\Gamma \vdash_v v : A$, then $(v|\tau) \in |A|_v$.*
2. *If F is a forcing context such that $\Gamma \vdash_F F : A^\perp$, then $(F|\tau) \in \|A\|_F$.*
3. *If V is a weak value such that $\Gamma \vdash_V V : A$, then $(V|\tau) \in |A|_V$.*
4. *If E is a catchable context such that $\Gamma \vdash_E E : A^\perp$, then $(E|\tau) \in \|A\|_F$.*
5. *If t is a term such that $\Gamma \vdash_t t : A$, then $(t|\tau) \in |A|_t$.*
6. *If e is a context such that $\Gamma \vdash_e e : A^\perp$, then $(e|\tau) \in \|A\|_e$.*
7. *If c is a command such that $\Gamma \vdash_c c$, then $c\tau \in \perp\!\!\!\perp$.*
8. *If τ' is a store such that $\Gamma \vdash_\tau \tau' : \Gamma'$, then $\tau\tau' \Vdash \Gamma, \Gamma'$.*

Proof. We proceed by induction over the typing rules.

Case Constants.

This case stems directly from the definition of $|X|_v$ for X atomic.

Case $\vdash_v A \to B$.

This case exactly matches the definition of $|A \to B|_v$. Assume that

$$\frac{\Gamma, x : A \vdash_t t : B}{\Gamma \vdash_v \lambda x.t : A \to B}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. If $(u|\tau')$ is a closed term in the set $|A|_t$, then, up to α-conversion for the variable x, $\overline{\tau\tau'} \Vdash \Gamma$ by Lemma 12 and $\overline{\tau\tau'}[x := u] \Vdash \Gamma, x : A$. Using the induction hypothesis, $(t|\overline{\tau\tau'}[x := u])$ is indeed in $|B|_t$.

Case $\vdash_F \ (A \to B)^\perp$.

Assume that

$$\frac{\Gamma \vdash_t u : A \quad \Gamma \vdash_E E : B^\perp}{\Gamma \vdash_F u \cdot E : (A \to B)^\perp}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. Let $(\lambda x.t|\tau')$ be a closed term in the set $|A \to B|_v$ such that $\tau <: \tau'$, then we have:

$$\langle \lambda x.t \| u \cdot E \rangle \overline{\tau\tau'} \quad \to \quad \langle u \| \tilde{\mu} x.\langle t \| E \rangle \rangle \overline{\tau\tau'} \quad \to \quad \langle t \| E \rangle \overline{\tau\tau'}[x := u]$$

By definition of $|A \to B|_v$, this closure is in the pole, and we can conclude by anti-reduction.

Case $\Gamma \vdash_V \ v : A$.

This case, as well as every other case where typing a term (resp. context) at a higher level of the hierarchy (for instance F at level E) is a simple consequence of Proposition 10. Indeed, assume for instance that

$$\frac{\Gamma \vdash_v v : A}{\Gamma \vdash_V v : A}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. By induction hypothesis, we get that $(v|\tau) \in |A|_v$. Thus if $(F|\tau')$ is in $\|A\|_F$, by definition $(v|\tau) \perp\!\!\!\perp (F|\tau')$.

Case $\Gamma \vdash_V \ x : A$.

Assume that

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_V x : A}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. As $(x : A) \in \Gamma$, we know that τ is of the form $\tau_0[x := t]\tau_1$ with $(t|\tau_0) \in |A|_t$. Let $(F|\tau')$ be in $\|A\|_F$, with $\tau <: \tau'$. By Lemma 4, we know that $\overline{\tau\tau'}$ is of the form $\overline{\tau_0}[x := t]\overline{\tau_1}$. Hence we have:

$$\langle x \| F \rangle \overline{\tau_0}[x := t]\overline{\tau_1} \quad \to \quad \langle t \| \tilde{\mu}[x].\langle x \| F \rangle \overline{\tau_1} \rangle \overline{\tau_0}$$

and it suffices by anti-reduction to show that the last closure is in the pole $\perp\!\!\!\perp$. By induction hypothesis, we know that $(t|\tau) \in |A|_t$ thus we only need to show that it is in front of a catchable context in $\|A\|_E$. This corresponds exactly to the next case that we shall prove now.

Cases $\Gamma \vdash_E \alpha : A^\perp$.

This case is obvious from the definition of $\tau \Vdash \Gamma$.

Case $\Gamma \vdash_E \ \tilde{\mu}[x].\langle x \| F \rangle \tau' : A$.

Assume that

$$\frac{\Gamma, x : A, \Gamma' \vdash_F F : A \quad \Gamma, x : A \vdash \tau' : \Gamma'}{\Gamma \vdash_E \tilde{\mu}[x].\langle x \| F \rangle \tau' : A}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. Let $(V|\tau_0)$ be a closed term in $|A|_V$ such that $\tau_0 <: \tau$. We have that :

$$\langle V \| \tilde{\mu}[x].\langle x \| F \rangle \overline{\tau'} \rangle \overline{\tau_0\tau} \quad \to \quad \langle V \| F \rangle \overline{\tau_0\tau}[x := V]\tau'$$

By induction hypothesis, we obtain $\tau[x := V]\tau' \Vdash \Gamma, x : A, \Gamma'$. Up to α-conversion in F and τ' so that the variables in τ' are disjoint from those in τ_0, we have that $\overline{\tau_0\tau} \Vdash \Gamma$ (by Lemma 12) and then $\tau'' \triangleq \overline{\tau_0\tau}[x := V]\tau' \Vdash \Gamma, x : A, \Gamma'$. By induction hypothesis again, we obtain that $(F|\tau'') \in \|A\|_F$ (this was an assumption in the previous case) and as $(V|\tau_0) \in |A|_V$, we finally get that $(V|\tau_0) \perp\!\!\!\perp (F|\tau'')$ and conclude again by anti-reduction.

Case $\Gamma \vdash_t \mu\alpha.c : A$.

Assume that

$$\frac{\Gamma, \alpha : A^\perp \vdash_c c}{\Gamma \vdash_t \mu\alpha.c : A}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. Let $(E|\tau')$ be a closed context in $\|A\|_E$ such that $\tau <: \tau'$. We have that :

$$\langle \mu\alpha.c \| E \rangle \overline{\tau\tau'} \quad \to \quad c\overline{\tau\tau'}[\alpha := E]$$

Using the induction hypothesis, we only need to show that $\overline{\tau\tau'}[\alpha := E] \Vdash \Gamma, \alpha : A^\perp, \Gamma'$ and conclude by anti-reduction. This obviously holds, since $(E|\tau') \in \|A\|_E$ and $\overline{\tau\tau'} \Vdash \Gamma$ by Lemma 4.

Case $\Gamma \vdash_e \tilde{\mu}x.c : A^\perp$.

This case is identical to the previous one.

Case $\Gamma \vdash_c c$.

Assume that

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_e e : A^\perp}{\Gamma \vdash_c \langle t \| e \rangle}$$

and let $\perp\!\!\!\perp$ be a pole and τ a store such that $\tau \Vdash \Gamma$. Then by induction hypothesis $(t|\tau) \in |A|_t$ and $(e|\tau) \in \|A\|_e$, so that $\langle t \| e \rangle \tau \in \perp\!\!\!\perp$.

Case $\Gamma \vdash_\tau \tau' : \Gamma', x : A$.

This case is easy since if the induction hypothesis exactly matches the definition of $\tau\tau' \Vdash \Gamma\Gamma', x : A$. The case $\Gamma \vdash_\tau \tau' : \Gamma', \alpha : A^\perp$ is identical, and the case $\Gamma \vdash_\tau \varepsilon : \varepsilon$ is trivial. ◀

C Proofs of Section 6

We first prove a few technical results that we will use afterwards in the proof of the main theorem.

▶ **Lemma 22** (Suitable substitution). *For all σ and Γ such that σ is suitable for Γ, if τ is a store such that $\Gamma \vdash_\tau \tau : \Gamma'$ for some Γ', if $\tau', \sigma' = [\![\tau]\!]^\sigma_\tau$ then σ' is suitable for Γ, Γ' and $[\![\Gamma]\!]^\sigma_\Gamma = [\![\Gamma]\!]^{\sigma'}_\Gamma$.*

Proof. Obvious from the definition. ◀

▶ **Lemma 23** (Subtyping identity). *The following rule is admissible:* $\overline{\Sigma \vdash \Upsilon <: \Upsilon}$

Proof. Straightforward induction on the structure of Υ, applying repeatedly the $<:_1$-rule (or the $<:_Y$-rule). ◀

▶ **Lemma 24** (Weakening). *The following rules is admissible:*

$$\frac{\Gamma \vdash t : A \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash t : A}\ w$$

▶ **Lemma 25** (Terms subtyping). *The following rule is admissible:*

$$\frac{\Gamma \vdash t : \forall Y <: \Upsilon_0 . A \quad \Gamma \vdash \Upsilon_1 <: \Upsilon_0}{\Gamma \vdash t : \forall Y <: \Upsilon_1 . A}\ <:_\forall$$

Proof.

$$\dfrac{\dfrac{\Gamma, X <: \Upsilon_1 \vdash t : \forall Y <: \Upsilon_0 . A \qquad \dfrac{\dfrac{}{\Gamma, Y <: \Upsilon_1 \vdash Y <: \Upsilon_1}\ <:_{\text{ax}} \quad \Gamma \vdash \Upsilon_1 <: \Upsilon_0}{\Gamma, Y <: \Upsilon_1 \vdash Y <: \Upsilon_0}\ <:_3}{\Gamma, Y <: \Upsilon_1 \vdash t : A}\ \forall_E \qquad Y \notin FV(\Gamma)}{\Gamma \vdash t : \forall Y <: \Upsilon_1 . A}\ \forall_I$$

where we use Lemma 24 to weaken $\Gamma, X <: \Upsilon_1$ to Γ.

◀

▶ **Corollary 19.** *For any level o of the hierarchy e, t, E, V, F, v, the following rule is admissible:*

$$\frac{\Gamma \vdash t : \Upsilon_0 \triangleright_o A \quad \Gamma \vdash \Upsilon_1 <: \Upsilon_0}{\Gamma \vdash t : \Upsilon_1 \triangleright_o A}\ <:_\triangleright$$

▶ **Theorem 20.** *The translation is well-typed, i.e.*

1. *if* $\Gamma \vdash_v v : A$ *then* $[\![\Gamma \vdash_v v : A]\!]$
2. *if* $\Gamma \vdash_F F : A^\perp$ *then* $[\![\Gamma \vdash_F F : A^\perp]\!]$
3. *if* $\Gamma \vdash_V V : A$ *then* $[\![\Gamma \vdash_V V : A]\!]$
4. *if* $\Gamma \vdash_E E : A^\perp$ *then* $[\![\Gamma \vdash_E E : A^\perp]\!]$
5. *if* $\Gamma \vdash_t t : A$ *then* $[\![\Gamma \vdash_t t : A]\!]$
6. *if* $\Gamma \vdash_e e : A^\perp$ *then* $[\![\Gamma \vdash_e e : A^\perp]\!]$
7. *if* $\Gamma \vdash_c c$ *then* $[\![\Gamma \vdash_c c]\!]$
8. *if* $\Gamma \vdash_l l$ *then* $[\![\Gamma \vdash_l l]\!]$
9. *if* $\Gamma \vdash_\tau \tau$ *then* $[\![\Gamma \vdash_\tau \tau : \Gamma']\!]$

Proof. By induction over the typing rules. Let Γ be a typing context and σ be a suitable translation of names of Γ. We (ab)use of Lemma 24 to make the derivations more compact by systematically weakening contexts as soon as possible, and compact the first $\forall$- and λ-introductions in one rule.

1. Strong values

Case $[\![\mathbf{c}]\!]^\sigma_v$.

$[\![\mathbf{c}]\!]^\sigma_v = \mathbf{c}$, which has the desired type by hypothesis.

Case $[\![\lambda x_i.t]\!]^\sigma_v$.

In the source language, we have:

$$\frac{\Gamma, x : A \vdash_t t : B}{\Gamma \vdash_v \lambda x : A \to B}$$

Hence, if n is fresh (w.r.t σ), $\sigma[x := n]$ is suitable for $\Gamma, x : A$, and we get by induction a proof Π_t of $[\![t]\!]_t^{\sigma[x:=n]} : [\![\Gamma, x : A]\!]_\Gamma^{\sigma[x:=n]} \triangleright_t \iota(B)$. Observing that $[\![\Gamma, x : A]\!]_\Gamma^{\sigma[x:=n]} = [\![\Gamma]\!]^\sigma_\Gamma, n : \iota(A)$

we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\boldsymbol{\Pi_t}}{\vdash [\![t]\!]_t^{\sigma[x:=n]} : [\![\Gamma, x : A]\!]_\Gamma^{[\sigma,x:=n]} \triangleright_t \iota(B)} \quad \dfrac{\overline{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash Y <: [\![\Gamma]\!]_\Gamma^\sigma}^{<:_{\mathtt{ax}}}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash Y, n : A <: [\![\Gamma]\!]_\Gamma^\sigma, n : A}^{<:_2}\forall_E}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash [\![t]\!]_t^{\sigma[x:=n]} : Y, n : A \to Y, n : A \triangleright_E \iota(B) \to \bot} \quad \boldsymbol{\Pi_\tau}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma, \tau : Y, u : Y \triangleright_t \iota(A); \vdash [\![t]\!]_t\ \tau[u] : Y, n : A \triangleright_E \iota(B) \to \bot} \quad \boldsymbol{\Pi_E}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma, \tau : Y, u : Y \triangleright_t \iota(A), E : Y \triangleright_E \iota(B) \vdash [\![t]\!]_t^{\sigma[x:=n]}\ \tau[u]\ E : \bot}\lambda}{\vdash \lambda\tau u E.[\![t]\!]_t^{\sigma[x:=n]}\ \tau[u]\ E : \forall Y <: [\![\Gamma]\!]_\Gamma^\sigma. Y \to Y \triangleright_t \iota(A) \to Y \triangleright_E \iota(B) \to \bot}$$

where:

- Π_τ is the following subproof:

$$\dfrac{\dfrac{}{\tau : Y \vdash \tau : Y}\mathtt{ax} \quad \dfrac{\dfrac{}{u : Y \triangleright_t \iota(A) \vdash u : Y \triangleright_t \iota(A)}\mathtt{ax}}{Y \triangleright_t \iota(A) \vdash [n := u] : Y \triangleright_\tau \iota(A)}\tau_t}{\tau : Y, u : Y \triangleright_t \iota(A); \vdash \tau[n := u] : Y, n : A}\tau\tau'$$

- Π_E is the following proof (derivable using Corollary 19):

$$\dfrac{\dfrac{}{E : Y \triangleright_E \iota(B) \vdash E : (Y, n : \iota(A)) \triangleright_E \iota(B)}\mathtt{ax} \quad \dfrac{\dfrac{}{\vdash Y <: Y}^{<:_{\mathtt{ax}}}}{\vdash (Y, n : \iota(A)) <: Y}^{<:_2}}{E : Y \triangleright_E \iota(B) \vdash E : (Y, n : \iota(A)) \triangleright_E \iota(B)}^{<:_\triangleright}$$

2. Forcing contexts

Case $[\![\boldsymbol{\alpha}]\!]_F^\sigma$.

$[\![\boldsymbol{\alpha}]\!]_F^\sigma = \boldsymbol{\alpha}$, which has the desired type by hypothesis.

Case $[\![t.E]\!]_F^\sigma$.

In the source language, we have:

$$\dfrac{\Gamma \vdash_t t : A \quad \Gamma \vdash_E E : B^\perp}{\Gamma \vdash_F t \cdot E : (A \to B)^\perp}$$

Hence we have by induction hypothesis a proof of $\vdash [\![t]\!]_t^\sigma : [\![\Gamma]\!]_\Gamma^\sigma \triangleright_t \iota(A)$ (and a proof of $\vdash [\![E]\!]_t^\sigma : [\![\Gamma]\!]_\Gamma^\sigma \triangleright_E \iota(B)$) that can be turn for any Y using Corollary 19 into a proof Π_t of $Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash [\![t]\!]_t^\sigma : Y \triangleright_t \iota(A)$ (resp. Π_E of $Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash [\![E]\!]_t^\sigma : Y \triangleright_E \iota(B)$). Thus we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{v : Y \triangleright_v (\iota(A) \to \iota(B)) \vdash v : Y \triangleright_v \iota(A) \to \iota(B)}\mathtt{ax} \quad \dfrac{}{\vdash Y <: Y}^{<:_{\mathtt{ax}}}}{v : Y \triangleright_v (\iota(A) \to \iota(B)) \vdash v : Y \to Y \triangleright_t \iota(A) \to Y \triangleright_E B \to \bot}\forall_E \quad \dfrac{}{\tau : Y \vdash \tau : Y}\mathtt{ax}}{\tau : Y, v : Y \triangleright_v (\iota(A) \to \iota(B)) \vdash v\ \tau : Y \triangleright_t \iota(A) \to Y \triangleright_E \iota(B) \to \bot}@ \quad \boldsymbol{\Pi_t}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma, \tau : Y, v : Y \triangleright_v (\iota(A) \to \iota(B)) \vdash v\ \tau\ [\![t]\!]_t^\sigma : Y \triangleright_E \iota(B) \to \bot}@ \quad \boldsymbol{\Pi_E}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma, \tau : Y, v : Y \triangleright_v (\iota(A) \to \iota(B))(\vdash v\ \tau\ [\![t]\!]_t^\sigma\ [\![E]\!]_E^\sigma : \bot}\lambda}{\vdash \lambda\tau v.v\ \tau\ [\![t]\!]_t^\sigma\ [\![E]\!]_E^\sigma : \forall Y <: [\![\Gamma]\!]_\Gamma^\sigma. Y \to Y \triangleright_v (\iota(A) \to \iota(B)) \to \bot}$$

3. Weak values

Case $[\![v]\!]_V^\sigma$.

In the source language, we have:

$$\dfrac{\Gamma \vdash_v v : A}{\Gamma \vdash_V v : A}$$

Hence we have by induction hypothesis a proof Π_v of $\vdash [\![v]\!]^\sigma_v : [\![\Gamma]\!]^\sigma_\Gamma \triangleright_v \iota(A)$ and we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F : Y \triangleright_F \iota(A) \vdash F : Y \triangleright_F \iota(A)}\,\text{ax} \quad \dfrac{}{\vdash Y <: Y}\,<:_{\text{ax}}}{F : Y \triangleright_F \iota(A) \vdash F : Y \to Y \triangleright_v \iota(A) \to \bot}\,\forall_E \quad \dfrac{}{\tau : Y \vdash \tau : Y}\,\text{ax}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma, \tau : Y, F : Y \triangleright_F \iota(A) \vdash F\ \tau : Y \triangleright_v \iota(A) \to \bot}\,@ \quad \dfrac{\boldsymbol{\Pi_v} \quad \dfrac{}{Y <: [\![\Gamma]\!]^\sigma_\Gamma \vdash Y <: [\![\Gamma]\!]^\sigma_\Gamma}\,\text{ax}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma \vdash [\![v]\!]^\sigma_v : Y \triangleright_v \iota(A)}\,<:_\triangleright}{Y <: [\![\Gamma]\!]^\sigma_\Gamma, \tau : Y, F : Y \triangleright_F \iota(A) \vdash F\ \tau\ [\![v]\!]^\sigma_v : \bot}\,@}{\vdash \lambda\tau F.F\ \tau\ [\![v]\!]^\sigma_v : \forall Y <: [\![\Gamma]\!]^\sigma_\Gamma.Y \to Y \triangleright_F \iota(A) \to \bot}\,\lambda$$

where we used Corollary 19 on the right part of the proof. Observe that $\uparrow^t V$ is in fact independent of the level t and that we could as well as written $[\![v]\!]^\sigma_V = \uparrow [\![v]\!]^\sigma_v$. We thus proved the admissibility of the following rule:

$$\dfrac{\Gamma \vdash V : \Upsilon \triangleright_V A}{\Gamma \vdash \uparrow^t V : \Upsilon \triangleright_t A}\,\uparrow$$

Case $[\![x]\!]^\sigma_V$.

In the source language, we have:

$$\dfrac{(x : A) \in \Gamma}{\Gamma \vdash_V x : A}$$

so that Γ is of the form $\Gamma_0, x : A, \Gamma_1$. By definition, we have:

$$[\![x]\!]^\sigma_V = \lambda\tau F.\mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\,\tau\,n\ \mathbf{in}\ t\ \tau_0\ (\lambda\tau'_0 V.V\ \tau'_0[n :=\uparrow^t V]\tau_1\ F) \qquad \text{where } n = \sigma(x)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{t : Y_0 \triangleright_t \iota(A) \vdash t : Y_0 \triangleright_t \iota(A)}\,\text{ax} \quad \dfrac{}{\vdash Y_0 <: Y_0}\,<:_{\text{ax}}}{t : Y_0 \triangleright_t \iota(A) \vdash t : Y_0 \to Y_0 \triangleright_E \iota(A) \to \bot}\,\forall_E \quad \dfrac{}{\tau_0 : Y_0 \vdash \tau_0 : Y_0}\,\text{ax}}{\tau_0 : Y_0, t : Y_0 \triangleright_t A \vdash t\ \tau_0 : Y_0 \triangleright_E \iota(A) \to \bot}\,@ \quad \boldsymbol{\Pi_E}}{\tau_0 : Y_0, t : Y_0 \triangleright_t \iota(A), \tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1, F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \vdash t\ \tau_0\ E : \bot}\,@}{\tau : (Y_0, n : \iota(A), Y_1), F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \vdash \mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\,\tau\,n\ \mathbf{in}\ t\ \tau_0\ E : \bot}\,\text{split} \quad \boldsymbol{\Pi_Y}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma, \tau : Y, F : Y \triangleright_F \iota(A) \vdash \mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\,\tau\,n\ \mathbf{in}\ t\ \tau_0\ E : \bot}\,<:_{\text{split}}}{\vdash \lambda\tau F.\mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\,\tau\,n\ \mathbf{in}\ t\ \tau_0\ E : \forall Y <: [\![\Gamma]\!]^\sigma_\Gamma.Y \to Y \triangleright_F \iota(A) \to \bot}\,\lambda$$

where:

- Π_Y is simply the axiom rule:

$$\dfrac{}{Y <: ([\![\Gamma_0]\!]^\sigma_\Gamma, n : \iota(A), [\![\Gamma_1]\!]^\sigma_\Gamma) \vdash Y <: ([\![\Gamma_0]\!]^\sigma_\Gamma, n : \iota(A), [\![\Gamma_1]\!]^\sigma_\Gamma)}\,<:_{\text{ax}}$$

- $E = (\lambda\tau'_0 V.V\ \tau'_0[n := V]\tau_1\ F)$ and Π_E is the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{V : Y'_0 \triangleright_V \iota(A) \vdash \uparrow^t V : Y'_0 \triangleright_t \iota(A)}\,\text{ax} \quad \dfrac{}{\vdash Y'_0, n : A, Y_1 <: Y'_0, n : A}\,\text{ax}}{V : Y'_0 \triangleright_V \iota(A) \vdash V : (Y'_0, n : A, Y_1) \to (Y'_0, n : A, Y_1) \triangleright_E \iota(A) \to \bot}\,\forall_E \quad \boldsymbol{\Pi_\tau}}{\tau_1 : (Y_0, n : A) \triangleright_\tau Y_1, Y'_0 <: Y_0, \tau'_0 : Y'_0, V : Y'_0 \triangleright_V \iota(A) \vdash V\ \tau'_0[n :=\uparrow^t V]\tau_1 : (Y'_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \to \bot}\,@ \quad \boldsymbol{\Pi_F}}{\tau_1 : (Y_0, n : A) \triangleright_\tau Y_1, F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A), Y'_0 <: Y_0, \tau'_0 : Y'_0, V : Y'_0 \triangleright_V \iota(A) \vdash V\ \tau'_0[n :=\uparrow^t V]\tau_1\ F : \bot}\,@}{\tau_1 : (Y_0, n : A) \triangleright_\tau Y_1, F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \vdash \lambda\tau'_0 V.V\ \tau'_0[n :=\uparrow^t V]\tau_1\ F : Y_0 \triangleright_E \iota(A)}\,\lambda$$

- Π_F is the following proof, obtained by Corollary 19:

$$\dfrac{\dfrac{}{F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \vdash F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A)}\,\text{ax} \quad \dfrac{\dfrac{\dfrac{}{Y'_0 <: Y_0 \vdash Y'_0 <: Y_0}\,<:_{\text{ax}}}{\vdots}\,<:_1}{Y'_0 <: Y_0 \vdash (Y'_0, n : \iota(A), Y_1) <: (Y_0, n : \iota(A), Y_1)}\,<:_1}{Y'_0 <: Y_0, F : (Y_0, n : \iota(A), Y_1) \triangleright_F \iota(A) \vdash F : (Y'_0, n : \iota(A), Y_1) \triangleright_F \iota(A)}\,<:_\triangleright$$

- Π_τ is the following derivation

$$\dfrac{\dfrac{}{\tau_0' : Y_0' \vdash \tau_0' : Y_0'}\,\mathsf{ax} \quad \dfrac{\dfrac{\dfrac{}{V : Y_0' \triangleright_V \iota(A) \vdash V : Y_0' \triangleright_V A}\,\mathsf{ax}}{V : Y_0' \triangleright_V \iota(A) \vdash \uparrow^t V : Y_0' \triangleright_t A}\uparrow}{V : Y_0' \triangleright_V \iota(A) \vdash [n := \uparrow^t V] : Y_0' \triangleright_\tau n : \iota(A)}\,\tau_t}{Y_0' <: Y_0, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A) \vdash \tau_0'[n := V] : Y_0', n : \iota(A)}\,\tau\tau' \qquad \boldsymbol{\Pi_{\tau_1}}}{\tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1, Y_0' <: Y_0, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A) \vdash \tau_0'[n := V]\tau_1 : Y_0', n : A, Y_1}\,\tau <:$$

- Π_{τ_1} is the following derivation:

$$\dfrac{\dfrac{}{\tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1 \vdash \tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1}\,\mathsf{ax} \quad \dfrac{\dfrac{}{Y_0' <: Y_0 \vdash Y_0' <: Y_0}\,<:_{\mathsf{ax}}}{Y_0' <: Y_0 \vdash Y_0', n : \iota(A) <: Y_0, n : \iota(A)}\,<:_1}{\tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1, Y_0' <: Y_0 \vdash \tau_1 : Y_0', n : \iota(A) \triangleright_\tau Y_1}\,\tau <:$$

4. Catchable contexts

Case $[\![F]\!]^\sigma_E$.

This case is similar to the case $[\![v]\!]^\sigma_V$.

Case $[\![\tilde{\mu}[x].\langle x \| F\rangle \tau']\!]^\sigma_E$.

In the source language, we have:

$$\dfrac{\Gamma, x : A, \Gamma' \vdash_F F : A^\perp \quad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_E \tilde{\mu}[x].\langle x \| F\rangle \tau : A^\perp}$$

If n is fresh (w.r.t σ), $\sigma[x := n]$ is suitable for $\Gamma, x : A$, and we then have by induction hypothesis a proof of $\vdash \tau'' : [\![\Gamma, x : A]\!]^{\sigma'} \triangleright_\tau [\![\Gamma']\!]^{\sigma'}$ and a proof Π_F of $\vdash [\![F]\!]^{\sigma'}_F : [\![\Gamma, x : A]\!]^{\sigma'} \triangleright_F \iota(A)$ where $\tau'', \sigma' = [\![\tau']\!]^{\sigma,[x:=n]}$ for some fresh n. We can thus derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{V : Y \triangleright_V \iota(A) \vdash Y \triangleright_V \iota(A)}\,\mathsf{ax} \quad \dfrac{\dfrac{}{\vdash Y <: Y}\,<:_{\mathsf{ax}}}{\vdash Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma <: Y}\,<:_2}{V : Y \triangleright_V \iota(A) \vdash V : (Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma) \to (Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma) \triangleright_F \iota(A) \to \bot}\,\forall_E \qquad \boldsymbol{\Pi_\tau}}{\tau : Y, V : Y \triangleright_V \iota(A) \vdash V\ \tau[n := \uparrow^t V]\tau'' : (Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma \iota(A) \triangleright_F \to \bot}\,@ \qquad \boldsymbol{\Pi_F}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma, \tau : Y, V : Y \triangleright_V \iota(A) \vdash V\ \tau[n := \uparrow^t V]\tau''\ [\![F]\!]^{\sigma'}_F : \bot}\,@}{\vdash \lambda\tau V.V\ \tau[n := \uparrow^t V]\tau''\ [\![F]\!]^{\sigma'}_F : \forall Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma . Y \to Y \triangleright_V \iota(A) \to \bot}\,\lambda$$

where:

- Π_F is the following proof, derived using Corollary 19 and Lemma 22:

$$\dfrac{\vdash [\![F]\!]^{\sigma'}_F : [\![\Gamma, n : \iota(A), \Gamma']\!]^{\sigma'}_\Gamma \triangleright_F \iota(A) \quad \dfrac{\dfrac{}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma}\,\mathsf{ax}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma <: [\![\Gamma, n : \iota(A), \Gamma']\!]^{\sigma'}_\Gamma}\,<:_1}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash [\![F]\!]^{\sigma'}_F : Y, n : \iota(A), [\![\Gamma']\!]^{\sigma'}_\Gamma \triangleright_F \iota(A)}\,\forall_E$$

- Π_τ is the following proof:

$$\dfrac{\dfrac{\dfrac{}{\tau : Y \vdash \tau : Y}\,\mathsf{ax} \quad \dfrac{\dfrac{\dfrac{}{V : Y \triangleright_V \iota(A) \vdash V : Y \triangleright_V \iota(A)}\,\mathsf{ax}}{V : Y \triangleright_V \iota(A) \vdash \uparrow^t V : Y \triangleright_t \iota(A)}\uparrow}{V : Y \triangleright_V \iota(A) \vdash [n := V] : Y \triangleright_\tau n : \iota(A)}\,\tau_t}{\tau : Y, V : Y \triangleright_V \iota(A) \vdash \tau[n := \uparrow^t V] : Y, n : \iota(A)}\,\tau\tau' \qquad \boldsymbol{\Pi_{\tau'}}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma, \tau : Y, V : Y' \triangleright_V \iota(A) \vdash \tau[n := \uparrow^t V][\![\tau']\!]^{\sigma[x:=n]}_\tau : (Y, n : \iota(A), [\![\Gamma']\!]^{\sigma[x:=n]})}\,\tau\tau'$$

- $\Pi_{\tau'}$ is the following proof, obtained from the induction hypothesis for τ':

$$\frac{\vdash [\![\tau']\!]_\tau^{\sigma[x:=n]} : [\![\Gamma]\!]_\Gamma^\sigma, n : \iota(A) \rhd_\tau [\![\Gamma']\!]^{\sigma[x:=n]} \quad \dfrac{\dfrac{}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash Y <: [\![\Gamma]\!]_\Gamma^\sigma}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash Y, n : \iota(A) <: [\![\Gamma]\!]_\Gamma^\sigma, n : \iota(A)}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash [\![\tau']\!]_\tau^{\sigma[x:=n]} : Y, n : \iota(A) \rhd_\tau [\![\Gamma']\!]^{\sigma[x:=n]}}$$

5. Terms

Case $[\![V]\!]_t^\sigma$.

This case is similar to the case $[\![v]\!]_V^\sigma$.

Case $[\![\mu\alpha.c]\!]_t^\sigma$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we have:

$$\frac{\Gamma, \alpha : A^\perp \vdash_c c}{\Gamma \vdash_t \mu\alpha.c : A}$$

If n is fresh (w.r.t σ), $\sigma[\alpha := n]$ is suitable for $\Gamma, \alpha : A^\perp$, and we then have by induction hypothesis a proof Π_c of $\vdash [\![c]\!]_c^{[\sigma, x:=n]} : [\![\Gamma, \alpha : A^\perp]\!]_\Gamma^{\sigma[\alpha:=n]} \rhd_c \perp$. We can thus derive, using Lemma 22 to identify $[\![\Gamma]\!]_\Gamma^\sigma$ and $[\![\Gamma]\!]_\Gamma^{\sigma,[\alpha:=n]}$:

$$\dfrac{\dfrac{\vdash [\![c]\!]_c^{\sigma[\alpha:=n]} : [\![\Gamma, \alpha : A^\perp]\!]_\Gamma^{\sigma[\alpha:=n]} \rhd_c \perp \quad \dfrac{\dfrac{}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash Y <: [\![\Gamma]\!]_\Gamma^{\sigma[\alpha:=n]}}\,{<:_{\mathsf{ax}}}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash (Y, n : \iota(A)^\perp) <: [\![\Gamma, \alpha : A^\perp]\!]^{\sigma[\alpha:=n]}}\,{<:_1}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma \vdash [\![c]\!]_c^{\sigma[\alpha:=n]} : (Y, n : \iota(A)^\perp) \to \perp}\,{\forall_E} \quad \boldsymbol{\Pi_\tau}}{Y <: [\![\Gamma]\!]_\Gamma^\sigma, \tau : Y, E : Y \rhd_E \iota(A) \vdash [\![c]\!]_c^{\sigma[\alpha:=n]}\ \tau[n := E] : \perp}\,{@}}{\vdash \lambda\tau E.[\![c]\!]_c^{\sigma[\alpha:=n]}\ \tau[n := E] : \forall Y <: [\![\Gamma]\!]_\Gamma^\sigma.Y \to Y \rhd_E \iota(A) \to \perp}\,{\lambda}$$

where Π_τ is the following derivation:

$$\dfrac{\dfrac{}{\tau : Y \vdash \tau : Y}\,{\mathsf{ax}} \quad \dfrac{\dfrac{}{E : Y \rhd_E \iota(A) \vdash E : Y \rhd_E \iota(A)}\,{\mathsf{ax}}}{E : Y \rhd_E \iota(A) \vdash [n := E] : (Y \rhd_\tau n : \iota(A)^\perp)}\,{\tau_t}}{\tau : Y, E : Y \rhd_E \iota(A) \vdash \tau[n := E] : (Y, n : \iota(A)^\perp)}\,{\tau\tau'}$$

6. Contexts

Case $[\![E]\!]_e^\sigma$.

This case is similar to the case $[\![v]\!]_V^\sigma$.

Case $[\![\tilde{\mu}x.c]\!]_e^\sigma$.

This case is similar to the case $[\![\mu\alpha.c]\!]_t^\sigma$.

7. Commands

Case $[\![\langle t \| e \rangle]\!]_c^\sigma$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we have:

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_e e : A^\perp}{\Gamma \vdash_c \langle t \| e \rangle}$$

We thus get by induction two proofs $\vdash [\![e]\!]^\sigma_e : [\![\Gamma]\!]^\sigma_\Gamma \rhd_e \iota(A)$ and $\vdash [\![t]\!]^\sigma_t : [\![\Gamma]\!]^\sigma_\Gamma \rhd_t \iota(A)$ We can derive:

$$\dfrac{\dfrac{\dfrac{\vdash [\![e]\!]^\sigma_e : [\![\Gamma]\!]^\sigma_\Gamma \rhd_e \iota(A) \qquad \boldsymbol{\Pi_Y}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma \vdash [\![e]\!]^\sigma_e : Y \to Y \rhd_t \iota(A) \to \bot}\forall_E \quad \dfrac{}{\tau : Y \vdash \tau : Y}\,\mathtt{ax}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma, \tau : Y \vdash [\![e]\!]^\sigma_e\ \tau : Y \rhd_t \iota(A) \to \bot}@ \quad \dfrac{\vdash [\![t]\!]^\sigma_t : [\![\Gamma]\!]^\sigma_\Gamma \rhd_t \iota(A) \quad \boldsymbol{\Pi_Y}}{Y <: [\![\Gamma]\!]^\sigma_\Gamma \vdash [\![t]\!]^\sigma_t : Y \rhd_t \iota(A)}\forall_E}{Y <: [\![\Gamma]\!]^\sigma_\Gamma, \tau : Y \vdash [\![e]\!]^\sigma_e\ \tau\ [\![t]\!]^\sigma_t : \bot}@}{\vdash \lambda\tau.[\![e]\!]^\sigma_t\ \tau\ [\![t]\!]^\sigma_t : \forall Y <: [\![\Gamma]\!]^\sigma_\Gamma.Y \to \bot}\lambda$$

where Π_Y is simply the axiom rule:

$$\dfrac{}{Y <: [\![\Gamma]\!]^\sigma_\Gamma \vdash Y <: [\![\Gamma]\!]^\sigma_\Gamma}\,<:_{\mathtt{ax}}$$

8. Closures

Case $[\![\langle t \| e \rangle \tau]\!]^\sigma_l$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we have:

$$\dfrac{\Gamma, \Gamma' \vdash_c c \quad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l c\tau}$$

We thus get by induction two proofs $\vdash \tau' : [\![\Gamma]\!]^{\sigma'}_\Gamma \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma$ and $\vdash [\![c]\!]^{\sigma'}_c : [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma \rhd_c \bot$ where $\tau', \sigma' = [\![\tau]\!]^\sigma_\tau$.

We can derive:

$$\dfrac{\dfrac{\vdash [\![c]\!]^{\sigma'}_c : [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma \rhd_c \bot \quad \dfrac{\dfrac{}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma}\,<:_{\mathtt{ax}}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash Y, [\![\Gamma']\!]^{\sigma'}_\Gamma <: [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma}\,<:_1}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash [\![c]\!]^{\sigma'}_c : Y, [\![\Gamma']\!]^{\sigma'}_\Gamma \to \bot}\forall_E \qquad \boldsymbol{\Pi_\tau}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma, \tau_0 : Y \vdash [\![c]\!]^{\sigma'}_c\ \tau_0\tau' : \bot}@}{\vdash \lambda\tau_0.[\![c]\!]^{\sigma'}_c\ \tau_0\tau' : \forall Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma.Y \to \bot}\lambda$$

where Π_τ is the following subderivation:

$$\dfrac{\dfrac{}{\tau_0 : Y \vdash \tau_0 : Y}\,\mathtt{ax} \quad \dfrac{\vdash \tau' : [\![\Gamma]\!]^{\sigma'}_\Gamma \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma \quad \dfrac{}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma}\,<:_{\mathtt{ax}}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma \vdash \tau' : Y \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma}\,\tau_{<:}}{Y <: [\![\Gamma]\!]^{\sigma'}_\Gamma, \tau_0 : Y \vdash \tau_0\tau' : Y, [\![\Gamma']\!]^{\sigma'}_\Gamma}\,\tau\tau'$$

9. Stores

Case $\tau[x := t]$.

We only consider the case $\tau[x := t]$, the proof for the case $\tau[\alpha := E]$ is identical. This corresponds to the typing rule:

$$\dfrac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_t t : A}{\Gamma \vdash_\tau \tau[x := t] : \Gamma', x : A}$$

By induction we obtain two proofs of $\vdash \tau' : [\![\Gamma]\!]^{\sigma'}_\Gamma \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma$ and $\vdash [\![t]\!]^{\sigma'}_t : [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma \rhd_t \iota(A)$ where $\tau', \sigma' = [\![\tau]\!]^\sigma_\tau$ We can thus derive:

$$\dfrac{\vdash \tau' : [\![\Gamma]\!]^{\sigma'}_\Gamma \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma \quad \dfrac{\vdash [\![t]\!]^{\sigma'}_t : [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma \rhd_t n : \iota(A)}{\vdash [n := [\![t]\!]^{\sigma'}_t] : [\![\Gamma, \Gamma']\!]^{\sigma'}_\Gamma \rhd_\tau n : \iota(A)}\,\tau_t}{\vdash \tau'[n := [\![t]\!]^{\sigma'}_t] : [\![\Gamma]\!]^{\sigma'}_\Gamma \rhd_\tau [\![\Gamma']\!]^{\sigma'}_\Gamma, n : \iota(A)}\,\tau\tau'$$

◀

D Introducing De Bruijn indexes

As for the proof of normalization, we observe in Figure 7 that the translation relies on names which implicitly suggests ability to perform α-conversion at run-time. Let us take a closer look to better understand this phenomenon. Let us consider a typed closure $\langle t \| e \rangle \tau$ such that:

$$\dfrac{\dfrac{\dfrac{\pi_t}{\Gamma \vdash_t t : A} \quad \dfrac{\pi_e}{\Gamma \vdash_e e : A^\perp}}{\Gamma \vdash_c \langle t \| e \rangle} \quad \dfrac{\pi_\tau}{\vdash_\tau \tau : \Gamma}}{\vdash_l \langle t \| e \rangle \tau}$$

Assume that both t and e introduce a new variable x in their sub-derivations π_t and π_e, which will be the case for instance if $t = \mu\alpha.\langle u \| \tilde{\mu} x.\langle x \| \alpha \rangle\rangle$ and $e = \tilde{\mu} x.\langle x \| F \rangle$. This is perfectly suitable for typing, however, through the translation (without α-conversion) we obtain:

$$\begin{aligned}
[\![c\varepsilon]\!] &= [\![e]\!]_e \; \varepsilon \; [\![t]\!]_t \\
&= [\![\langle x \| F \rangle]\!]_c \; [x := [\![t]\!]_t] \\
&= [\![x]\!]_x \; [x := [\![t]\!]_t] \; [\![F]\!]_F \\
&= [\![\mu\alpha.\langle u \| \tilde{\mu} x.\langle x \| \alpha \rangle\rangle]\!]_t \; \varepsilon \; (\lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F) \\
&= [\![\langle u \| \tilde{\mu} x.\langle x \| \alpha \rangle\rangle]\!]_t \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F] \\
&= [\![\tilde{\mu} x.\langle x \| \alpha \rangle]\!]_e \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F] \; [\![u]\!]_t \\
&= [\![\langle x \| \alpha \rangle]\!]_c \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F, x := [\![u]\!]_t] \\
&= [\![\alpha]\!]_E \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F, x := [\![u]\!]_t][\![x]\!]_V \\
&= (\lambda\tau\lambda V.V \; \tau[x := V]) \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F, x := [\![u]\!]_t][\![x]\!]_V \\
&\to [\![x]\!]_V \; [\alpha := \lambda\tau\lambda V.V \; \tau[x := V] \; [\![F]\!]_F, x := [\![u]\!]_t, x := [\![x]\!]_V]
\end{aligned}$$

We obtain on the last line a term executing $[\![x]\!]_V$ applied to a store containing a cyclic reference $[x := [\![x]\!]_V]$, which will loop forever.

Thus we would need to be able to perform α-conversion while executing the translation of a command, assuming that we can find a smooth way to do it. Another way of ensuring the correctness of our translation is to use De Bruijn indexes already in the $\overline{\lambda}_{[lv\tau\star]}$.

D.1 The $\overline{\lambda}_{[lv\tau\star]}$-calculus with De Bruijn

For binders of terms, we now use De Bruijn levels, i.e. names of the form x_i where x is a fixed name serving just the purpose of looking like a name and where the relevant information is the number i which counts how many binders of terms have been already traversed from the root of the term. For binders of evaluation contexts, we similarly use De Bruijn levels, but with variables of the form α_i, where, again, α is a fixed name indicating that the variable is binding evaluation contexts, and the relevant information is the index i. Note that substituting a term t within another term requires in general to update the number used to represent the bound variables of t, shifting them by the number of binders traversed by t at each position where it is substituted.

$$\frac{(\mathbf{c} : A) \in \mathcal{S}}{\Gamma \vdash_v \mathbf{c} : A} \qquad \frac{\Gamma, x_n : A \vdash_t t : B \quad |\Gamma| = n}{\Gamma \vdash_v \lambda x_n.t : A \to B} \qquad \frac{\Gamma(n) = (x_n : A)}{\Gamma \vdash_V x_n : A} \qquad \frac{\Gamma \vdash_v v : A}{\Gamma \vdash_V v : A}$$

$$\frac{(\boldsymbol{\alpha} : A) \in \mathcal{S}}{\Gamma \vdash_F \boldsymbol{\alpha} : A^\perp} \qquad \frac{\Gamma \vdash_t t : A \qquad \Gamma \vdash_E E : B^\perp}{\Gamma \vdash_F t \cdot E : (A \to B)^\perp} \qquad \frac{\Gamma, \alpha_n : A^\perp \vdash_c c \quad |\Gamma| = n}{\Gamma \vdash_t \mu\alpha_n.c : A} \qquad \frac{\Gamma \vdash_F F : A^\perp}{\Gamma \vdash_E F : A^\perp}$$

$$\frac{\Gamma \vdash_V V : A}{\Gamma \vdash_t V : A} \qquad \frac{\Gamma, \alpha_n : A^\perp \vdash_c c \quad |\Gamma| = n}{\Gamma \vdash_t \mu\alpha_n.c : A} \qquad \frac{\Gamma \vdash_E E : A^\perp}{\Gamma \vdash_e E : A^\perp} \qquad \frac{\Gamma, x_n : A \vdash_c c \quad |\Gamma| = n}{\Gamma \vdash_e \tilde{\mu}x_n.c : A^\perp}$$

$$\frac{\Gamma, x_i : A, \Gamma' \vdash_F F : A^\perp \quad \Gamma, x_i : A \vdash_\tau \tau : \Gamma' \quad |\Gamma| = i}{\Gamma \vdash_E \tilde{\mu}[x_i].\langle x_i \| F \rangle \tau : A^\perp} \qquad \frac{}{\Gamma \vdash_\tau \varepsilon : \varepsilon}$$

$$\frac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_t t : A \quad |\Gamma, \Gamma'| = n}{\Gamma \vdash_\tau \tau[x_n := t] : \Gamma', x_n : A} \qquad \frac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_E E : A^\perp \quad |\Gamma, \Gamma'| = n}{\Gamma \vdash_\tau \tau[\alpha_n := E] : \Gamma', \alpha_n : A^\perp}$$

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_e e : A^\perp}{\Gamma \vdash_c \langle t \| e \rangle} \qquad \frac{\Gamma, \Gamma' \vdash_c c \quad \Gamma \vdash_\tau \tau : \Gamma'}{\Gamma \vdash_l c\tau}$$

Figure 8 Typing rules for the $\overline{\lambda}_{[lv\tau\star]}$-calculus with De Bruijn

Closures	l	::=	$c\tau$
Commands	c	::=	$\langle t \| e \rangle$
Stores	τ	::=	$\varepsilon \mid \tau[x_i := t] \mid \tau[\alpha_i := E]$
Strong values	v	::=	$\mathbf{c} \mid \lambda x_i.t$
Weak values	V	::=	$v \mid x_i$
Terms	t, u	::=	$V \mid \mu\alpha_i.c$
Forcing contexts	F	::=	$\boldsymbol{\alpha} \mid t \cdot E$
Catchable contexts	E	::=	$F \mid \alpha_i \mid \tilde{\mu}[x_i].\langle x_i \| F \rangle \tau$
Evaluation contexts	e	::=	$E \mid \tilde{\mu}x_i.c$

We define the lifted term (and contexts, values, etc...) $\uparrow_n^{+i} t$ as the term t where all the free variables x_j with $j > n$ (resp. α_j) have been replaced by x_{j+i}. Formally, this can be defined as follows:

$$
\begin{array}{lcll}
(\uparrow_n^{+i} c\tau) & \triangleq & (\uparrow_n^{+i} c)(\uparrow_n^{+i} \tau) & \\
(\uparrow_n^{+i} \langle t \| e \rangle) & \triangleq & \langle \uparrow_n^{+i} t \| \uparrow_n^{+i} e \rangle & \\
\\
\uparrow_n^{+i} \varepsilon & \triangleq & \varepsilon & \\
\uparrow_n^{+i} (\tau[x_j := t]) & \triangleq & \uparrow_n^{+i} (\tau)([\uparrow_n^{+i} x_j := \uparrow_n^{+i} t] & \\
\uparrow_n^{+i} (\tau[\alpha_j := E]) & \triangleq & \uparrow_n^{+i} (\tau[\uparrow_n^{+i} \alpha_j := \uparrow_n^{+i} E] & \\
\\
\uparrow_n^{+i} (\mathbf{c}) & \triangleq & \mathbf{c} & \\
\uparrow_n^{+i} (\lambda x_j.t) & \triangleq & \lambda(\uparrow_n^{+i} x_j).(\uparrow_n^{+i} t) & \\
\uparrow_n^{+i} (x_j) & \triangleq & x_j & \text{if } j < n \\
\uparrow_n^{+i} (x_j) & \triangleq & x_{j+i} & \text{if } j \geq n \\
\uparrow_n^{+i} (\mu\alpha_j.c) & \triangleq & \mu(\uparrow_n^{+i} \alpha_i).(\uparrow_n^{+i} c) & \\
\\
\uparrow_n^{+i} (\boldsymbol{\alpha}) & \triangleq & \boldsymbol{\alpha} & \\
\uparrow_n^{+i} (t \cdot E) & \triangleq & (\uparrow_n^{+i} t) \cdot (\uparrow_n^{+i} E) & \\
\uparrow_n^{+i} (\alpha_j) & \triangleq & \alpha_j & \text{if } j < n \\
\uparrow_n^{+i} (\alpha_j) & \triangleq & \alpha_{j+i} & \text{if } j \geq n \\
\uparrow_n^{+i} (\tilde{\mu}[x_j].\langle x_j \| F \rangle \tau) & \triangleq & \tilde{\mu}[\uparrow_n^{+i} x_i].(\uparrow_n^{+i} \langle x_i \| F \rangle \tau) & \\
\uparrow_n^{+i} (\tilde{\mu} x_j.c) & \triangleq & \tilde{\mu}(\uparrow_n^{+i} x_i).(\uparrow_n^{+i} c) &
\end{array}
$$

The reduction rules became:

$$
\begin{array}{ccccl}
\langle t \| \tilde{\mu} x_i.c \rangle \tau & \to & c[x_n/x_i]\tau[x_n := t] & \text{with } |\tau| = n \\
\langle \mu\alpha.c \| E \rangle \tau & \to & c[\alpha_n/\alpha_i]\tau[\alpha := E] & \text{with } |\tau| = n \\
\langle V \| \alpha_n \rangle \tau & \to & \langle V \| \tau(n) \rangle \tau & \\
\langle x_n \| F \rangle \tau[x_n := t]\tau' & \to & \langle t \| \tilde{\mu}[x_n].\langle x_n \| F \rangle \tau' \rangle \tau & \\
\langle V \| \tilde{\mu}[x_i].\langle x_i \| F \rangle \tau' \rangle \tau & \to & \langle V \| \uparrow_n^{+i} F \rangle \tau[x_n := V](\uparrow_n^{+i} \tau') & \text{with } |\tau| = n \\
\langle \lambda x_i.t \| u \cdot E \rangle \tau & \to & \langle u \| \tilde{\mu} x_n.\langle t[x_n/x_i] \| E \rangle \rangle \tau & \text{with } |\tau| = n
\end{array}
$$

Note that we choose to perform indexes substitutions as soon as they come (maintaining the property that x_n is a variable referring to the $(n+1)^{\text{th}}$ element of the store), while it would also have been possible to store and compose them along the execution (so that x_n is a variable referring to the $(\sigma(n)+1)^{\text{th}}$ element of the store where σ is the current substitution). This could have seemed more natural for the reader familiar with compilation procedures that do not modify at run time but rather maintain the location of variables through this kind of substitution.

The typing rules are unchanged except for the one where indexes should now match the length of the typing context. The resulting type system is given in Figure 8.

D.2 System F_Υ with De Bruijn levels

We keep the same translation for judgments and types (Figures 11,10), except that we avoid using names and rather use De Bruijn indexes. The target language is as in Section 6 an adaptation of system F with stores (lists), to which we now add explicit coercions for store subtyping.

▶ **Definition 26** (Coercion)**.** We defined coercions to witness store subtyping $\Upsilon' <: \Upsilon$ as finite monotonic functions σ such that $\mathtt{dom}(\sigma) = [\![0, |\Upsilon| - 1]\!]$, $\mathtt{codom}(\sigma) \subseteq [\![0, |\Upsilon'| - 1]\!]$ and such that for all $i < |\Upsilon|$, $\Upsilon_i = \Upsilon'_{\sigma(i)}$.

Otherwise said, σ indicates where to find each type of the list Υ in the list Υ'. We denote by $\sigma_{|n}$ the restriction of σ to $[0, n-1]$ and $\mathtt{id}_n$ the identity on $[0, n-1]$. We also define σ_p^+

$$\frac{(x:A)\in\Gamma}{\Gamma;\Sigma\vdash x:A}\,\mathtt{ax} \qquad \frac{\Gamma,x:A;\Sigma\vdash t:B \quad |\Gamma|=n}{\Gamma;\Sigma\vdash \lambda x.t:A\to B}\,\lambda \qquad \frac{\Gamma;\Sigma\vdash t:A\to B \quad \Gamma\vdash u:A}{\Gamma;\Sigma\vdash t\,u:B}\,@$$

$$\frac{\Gamma;\Sigma,\sigma:X<:\Upsilon\vdash t:A \quad X\notin FV(\Gamma,\Sigma)}{\Gamma;\Sigma\vdash \lambda\sigma.t:\forall X<:\Upsilon.A}\,\forall_I \qquad \frac{\Gamma;\Sigma\vdash t:\forall X<:\Upsilon.A \quad \Sigma\vdash\sigma:\Upsilon'<:\Upsilon}{\Gamma;\Sigma\vdash t\,\sigma:A\{X:=\Upsilon'\}}\,\forall_E$$

$$\frac{(c:A)\in\mathcal{S}}{\Gamma;\Sigma\vdash \mathbf{c}:A}\,c \qquad \frac{\Gamma,x_{\tau_0}:\Upsilon_0,x:A,x_{\tau_1}:\Upsilon_1;\Sigma\vdash t:A \quad \Gamma\vdash\tau:\Upsilon_0,B,\Upsilon_1 \quad |\Upsilon_0|=n}{\Gamma;\Sigma\vdash \mathbf{let}\ x_{\tau_0},x,x_{\tau_1}=\mathbf{split}\,\tau\,n\ \mathbf{in}\ t:A}\,\mathtt{split}$$

$$\frac{}{\Gamma;\Sigma\vdash\varepsilon:\varepsilon\triangleright_\tau\varepsilon}\,\varepsilon \qquad \frac{\Gamma;\Sigma\vdash t:\Upsilon\triangleright_t A}{\Gamma;\Sigma\vdash[t]:\Upsilon\triangleright_\tau A}\,\tau_t \qquad \frac{\Gamma;\Sigma\vdash t:\Upsilon\triangleright_E A}{\Gamma;\Sigma\vdash[t]:\Upsilon\triangleright_\tau A^\perp}\,\tau_E$$

$$\frac{\Gamma\vdash\tau:\Upsilon_0\triangleright_\tau\Upsilon \qquad \Gamma\vdash\tau':(\Upsilon_0,\Upsilon)\triangleright_\tau\Upsilon'}{\Gamma\vdash\tau\tau':\Upsilon_0\triangleright_\tau\Upsilon,\Upsilon'}\,\tau\tau' \qquad \frac{}{\Sigma\vdash\sigma:\Upsilon'<:\varepsilon}\,{<:_\varepsilon}$$

$$\frac{(\sigma:\Upsilon'<:\Upsilon)\in\Sigma}{\Sigma\vdash\sigma:\Upsilon'<:\Upsilon}\,{<:_{\mathtt{ax}}} \qquad \frac{\Sigma\vdash\sigma:\Upsilon'<:\Upsilon \quad \sigma(|\Upsilon|)=|\Upsilon'|}{\Sigma\vdash\sigma:(\Upsilon',A)<:(\Upsilon,A)}\,{<:_1} \qquad \frac{\Sigma\vdash\sigma:\Upsilon'<:\Upsilon}{\Sigma\vdash\sigma:(\Upsilon',A)<:\Upsilon}\,{<:_2}$$

Figure 9 Typing rules of the target language

the canonical extension of a function σ whose domain is $[\![0,n-1]\!]$ for some n and whose codomain is included in $[\![0,p-1]\!]$ for some p by:

$$\sigma_p^+ : \begin{cases} [0,n] & \to & [0,p] \\ i<n & \mapsto & \sigma(i) \\ n & \mapsto & p \end{cases}$$

▶ **Lemma 27.** *If σ witnesses $\Upsilon'<:\Upsilon$ for some Υ,Υ', then $\sigma^+_{|\Upsilon'|}$ witnesses $\Upsilon',A<:\Upsilon,A$ for any type A.*

As we now got rid of names, we will now split stores with respect to an index. So that if you consider for instance a store of type $\Upsilon'<:(\Upsilon_0,A,\Upsilon_1)$, the knowledge of the position where to find the expected element of type A becomes crucial. In practice, it will be guided by the coercion witnessing $\Upsilon'<:(\Upsilon_0,A,\Upsilon_1)$. But to ensure the correctness of our typing rules, we need now to consider second-order variables (which are in fact vectors of second-order variables) with their arities. That is to say that we should denote by Y^p the vector of variables $Y_0,\dots,Y_{p-1}$ and that $\forall Y<:\Upsilon.A$ is equivalent

$$\forall p_0\forall Y^{p_0}\dots\forall p_n\forall Y^{p_n}.(Y^{p_0}\Upsilon(0)Y^{p_1}\Upsilon(1)\dots Y^{p_n})<:\Upsilon\to A$$

where we have in fact $p_0=\sigma(0)$, $p_1=\sigma(1)-p_0-1$, etc... In particular, a careful manipulation of variables with their arities allows us to prove the following lemma:

▶ **Lemma 28.** *The typing rules given for coercions in Figure 9 are equivalent to Definition 26,* i.e. *for all Υ,Υ', forall $i<|\Upsilon|$, $\Upsilon_i=\Upsilon'_{\sigma(i)}$.*

Even though arities are crucial to ensure the correctness of the definition in Figure 9 (in particular to define the relation $\sigma:\Upsilon'<:\Upsilon$ by means of inference rules), to easen the notation we will omit the arity most of the time. We will use the notation $\forall Y^n<:\Upsilon.A$ only when necessary.

The syntax of terms and types is given by:

$$\begin{array}{rcl} t,u & ::= & x \mid \lambda x.t \mid t\,u \mid \tau \mid \lambda\sigma.t \mid t\,\sigma \\ & \mid & \mathbf{let}\ \tau, x, \tau' = \mathbf{split}\ \tau''\ n\ \mathbf{in}\ t \\ \tau,\tau' & ::= & \varepsilon \mid \tau[t] \end{array} \qquad \begin{array}{rcl} A,B & ::= & X \mid \bot \mid \Upsilon \triangleright_\tau \Upsilon' \mid A \to B \mid \forall Y <: \Upsilon.\, A \\ \Upsilon,\Upsilon' & ::= & \varepsilon \mid \Upsilon, A \mid \Upsilon, A^\bot \mid Y \end{array}$$

Once again, we will use Υ as a shorthand for typing stores of type $\varepsilon \triangleright_\tau A$. The typing rules are given in Figure 9 where the typing contexts is divided in two parts, Γ containing typing hypothesis and Σ the subtyping hypothesis, that are defined by:

$$\Gamma,\Gamma' ::= \varepsilon \mid \Gamma, x : A \qquad\qquad \Sigma,\Sigma' ::= \varepsilon \mid \Sigma, \sigma : (\Upsilon' <: \Upsilon)$$

Now that we gave a computational contents to the subtyping relation, some properties that were defined axiomatically in Section 6 are now deducible from the characteristics of the coercions σ.

▶ **Proposition 29.** *The subtyping relation $<:$ is an order relation on store types.*

1. *For any Υ, $\Sigma \vdash \mathtt{id}_{|\Upsilon|} : \Upsilon <: \Upsilon$*
2. *If $\Sigma \vdash \sigma : \Upsilon <: \Upsilon'$ and $\Sigma \vdash \sigma' : \Upsilon' <: \Upsilon''$, then $\Sigma \vdash \sigma' \circ \sigma : \Upsilon <: \Upsilon''$.*
3. *If $\Sigma \vdash \sigma : \Upsilon <: \Upsilon'$ and $\Sigma \vdash \sigma' : \Upsilon' <: \Upsilon$, then $\sigma' \circ \sigma = \sigma' \circ \sigma = \mathtt{id}_{|\Upsilon|}$ and $\Upsilon = \Upsilon'$.*

Proof. Straightforward from the definition of $\sigma : \Upsilon' <: \Upsilon$:

2. for all $i < |\Upsilon|$, we have $\Upsilon''_{\sigma'(\sigma(i))} = \Upsilon'_{\sigma(i))} = \Upsilon_i$.
3. using the second item, we deduce that $\sigma' \circ \sigma$ witnesses $\Upsilon <: \Upsilon$. Both σ and σ' being monotonic functions, we deduce that $\sigma' = \sigma = \mathtt{id}_{|\Upsilon|}$ and that for all $i < |\Upsilon|$, $\Upsilon_i = \Upsilon'_i$.

◀

▶ **Proposition 30.** *For any function σ and any types Υ, Υ', if $\vdash \sigma : \Upsilon' <: \Upsilon$ and Υ is of the form $\Upsilon = \Upsilon_0, A, \Upsilon_1$, then Υ' is of the form $\Upsilon' = \Upsilon'_0, A, \Upsilon'_1$ such that $|\Upsilon'_0| = \sigma(|\Upsilon_0|)$ and $|\Upsilon'_1| = \sigma(|\Upsilon|) - |\Upsilon'_0| - 1$.*

Proof. Straightforward from the definitions. ◀

The former propositions shows that the following subtyping rules (where we use a compact version of the second-order variable) are admissible:

$$\frac{\Sigma \vdash \sigma : \Upsilon <: \Upsilon' \quad \Sigma \vdash \sigma' : \Upsilon' <: \Upsilon''}{\Sigma \vdash \sigma' \circ \sigma : \Upsilon <: \Upsilon''}\ {<:_3} \qquad\qquad \frac{\Gamma';\Sigma' \vdash t : B \quad \Sigma \vdash \sigma : Y <: \Upsilon_0, A, \Upsilon_1}{\Gamma;\Sigma \vdash t : B}\ {<:_{\mathtt{split}}}$$

where $\Gamma' = \Gamma[(Y_0^{\sigma(n)}, A, Y_1)/Y]$, $\Sigma' = \Sigma[(Y_0^{\sigma(n)}, A, Y_1)/X]$, and $Y_0^{\sigma(n)}, Y_1$ are fresh variables. Observe that the second one is a tautology that we only used to avoid the heavy syntactical manipulation of vectors of variables within proof trees.

▶ **Lemma 31** (Weakening). *The following rules are admissible:*

$$\frac{\Gamma;\Sigma \vdash t : A \quad \Sigma \subseteq \Sigma'}{\Gamma;\Sigma' \vdash t : A}\ {\Gamma_w} \qquad\qquad \frac{\Gamma;\Sigma \vdash t : A \quad \Gamma \subseteq \Gamma'}{\Gamma';\Sigma \vdash t : A}\ {\Sigma_w}$$

Proof. Easy induction on typing rules. In the case of second-order quantification, we might need to rename the second-order variable X if it occurs in Σ' (resp.Γ') and not in Σ (resp. Γ). ◀

$$
\begin{array}{lcl}
[\![\Gamma \vdash_e e : A^\perp]\!] & \triangleq & \vdash [\![e]\!]_e : [\![\Gamma]\!]_\Gamma \triangleright_e \iota(A) \\
[\![\Gamma \vdash_t t : A]\!] & \triangleq & \vdash [\![t]\!]_t : [\![\Gamma]\!]_\Gamma \triangleright_t \iota(A) \\
[\![\Gamma \vdash_E E : A^\perp]\!] & \triangleq & \vdash [\![E]\!]_E : [\![\Gamma]\!]_\Gamma \triangleright_E \iota(A) \\
[\![\Gamma \vdash_V V : A]\!] & \triangleq & \vdash [\![V]\!]_V : [\![\Gamma]\!]_\Gamma \triangleright_V \iota(A) \\
[\![\Gamma \vdash_F F : A^\perp]\!] & \triangleq & \vdash [\![F]\!]_F : [\![\Gamma]\!]_\Gamma \triangleright_F \iota(A) \\
[\![\Gamma \vdash_v v : A]\!] & \triangleq & \vdash [\![v]\!]_v : [\![\Gamma]\!]_\Gamma \triangleright_v \iota(A) \\
[\![\Gamma \vdash_c c]\!] & \triangleq & \vdash [\![c]\!]_c : [\![\Gamma]\!]_\Gamma \triangleright_c \bot \\
[\![\Gamma \vdash_l l]\!] & \triangleq & \vdash [\![l]\!]_l^{|\Gamma|} : [\![\Gamma]\!]_\Gamma \triangleright_c \bot \\
[\![\Gamma \vdash_\tau \tau : \Gamma']\!] & \triangleq & \vdash [\![\tau]\!]_\tau : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma
\end{array}
$$

Figure 10 Translation of judgments

$$
\begin{array}{ll}
\Upsilon \triangleright_c A & \triangleq \forall Y <: \Upsilon.\, Y \to \bot \\
\Upsilon \triangleright_e A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_t A) \to \bot \\
\Upsilon \triangleright_t A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_E A) \to \bot \\
\Upsilon \triangleright_E A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_V A) \to \bot \\
\Upsilon \triangleright_V A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_F A) \to \bot \\
\Upsilon \triangleright_F A & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_v A) \to \bot \\
\Upsilon \triangleright_v A \to B & \triangleq \forall Y <: \Upsilon.\, Y \to (Y \triangleright_t A) \to (Y \triangleright_E B) \to \bot \\
\Upsilon \triangleright_v X & \triangleq X \\
\\
[\![\varepsilon]\!]_\Gamma & \triangleq \varepsilon \\
[\![\Gamma, x_i : A]\!]_\Gamma & \triangleq [\![\Gamma]\!]_\Gamma, \iota(A) \\
[\![\Gamma, \alpha_i : A^\perp]\!]_\Gamma & \triangleq [\![\Gamma]\!]_\Gamma, \iota(A)^\perp
\end{array}
$$

Figure 11 Translation of types

D.3 A typed CPS translation

We shall now present the translation of terms and prove its correctness with respect to types. The translation, which is given in Figure 12, is similar to the translation with names in Section 6 plus the manipulation of coercions. Once again we assume that for each constant $\mathbf{c}$ of type A (resp. co-constant $\boldsymbol{\alpha}$ of type $A^\perp$) of the source system, we have a constant of type A in the signature of the target language that we also denote by $\mathbf{c}$ (resp. $\boldsymbol{\alpha}$ of type $A \to \bot$). We will now prove a bunch of lemmas that will be useful in the proof of the main theorem.

▶ **Lemma 32.** *The following rule is admissible for any level o of the hierarchy e, t, E, V, F, v:*

$$\frac{\Gamma; \Sigma \vdash t : \Upsilon \triangleright_o A}{\Gamma; \Sigma \vdash t : \Upsilon, B \triangleright_o A}$$

Proof. Directly follows from the observation that we can always derive:

$$\frac{\Sigma \vdash \sigma : \Upsilon' <: \Upsilon, B}{\Sigma \vdash \sigma : \Upsilon' <: \Upsilon}$$

◀

$$
\begin{array}{ll}
(\uparrow^{\sigma} t)\ \sigma' & \triangleq\ t\ (\sigma' \circ \sigma) \\
(\uparrow^{\sigma} \tau[t]) & \triangleq\ (\uparrow^{\sigma} \tau)[\uparrow^{\sigma} t] \\
\\
[\![\mathbf{c}]\!]_v & \triangleq\ \mathbf{c} \\
[\![\lambda x_i.t]\!]_v\ \sigma\ \tau\ u\ E & \triangleq\ [\![t]\!]_t\ \sigma^{+}_{|\tau|}\ \tau[u]\ E \\
\\
[\![\boldsymbol{\alpha}]\!]_F & \triangleq\ \boldsymbol{\alpha} \\
[\![t \cdot E]\!]_F\ \sigma\ \tau\ v & \triangleq\ v\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^{\sigma} [\![t]\!]_t)\ (\uparrow^{\sigma} [\![E]\!]_E) \\
\\
[\![v]\!]_V\ \sigma\ \tau\ F & \triangleq\ F\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^{\sigma} [\![v]\!]_v) \\
[\![x_i]\!]_V\ \sigma\ \tau[t]\tau'\ F & \triangleq\ t\ \mathtt{id}_{|\tau|}\ \tau\ (\lambda\sigma'\tau''\lambda V.V\ \tau''[\uparrow^{t} V](\uparrow^{\sigma''}\tau')\ (\uparrow^{\sigma''} F)) \\
 & \quad \text{where } n = |\tau| = \sigma(i),\ k = |\tau''| - n,\ p = n + |\tau'|,\ \sigma'' = \sigma' \circ \delta^{+k}_{[n,p]} \\
 & \quad \text{and } \uparrow^{t} V = \lambda\sigma\tau E.E\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^{\sigma} V)) \\
\\
[\![\alpha_i]\!]_E\ \sigma\ \tau\ V & \triangleq\ \mathbf{let}\ \tau', x, \tau'' = \mathbf{split}\ \sigma(i)\ \tau\ \mathbf{in}\ x\ \mathtt{id}_{|\tau|}\ \tau\ V \\
[\![\tilde{\mu}[x_i].\langle x_i \| F \rangle \tau']\!]_E\ \sigma\ \tau\ V & \triangleq\ V\ \mathtt{id}_{|\tau|}\ \tau[\uparrow^{t} V](\uparrow^{\sigma'}[\![\tau']\!]_\tau)\ (\uparrow^{\sigma'}[\![F]\!]_F) \\
 & \quad \text{where } n = |\tau|,\ k = n - i,\ p = n + |\tau'|,\ \sigma' = \sigma \circ \delta^{+k}_{[i,p]} \\
\\
[\![V]\!]_t\ \sigma\ \tau\ E & \triangleq\ E\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^{\sigma} [\![V]\!]_V) \\
[\![\mu\alpha_i.c]\!]_t\ \sigma\ \tau\ E & \triangleq\ [\![c]\!]_c\ \sigma^{+}_{|\tau|}\ \tau[E] \\
\\
[\![E]\!]_e\ \sigma\ \tau\ t & \triangleq\ t\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^{\sigma} [\![E]\!]_E) \\
[\![\tilde{\mu} x_i.c]\!]_e\ \sigma\ \tau\ t & \triangleq\ [\![c]\!]_c\ \sigma^{+}_{|\tau|}\ \tau[t] \\
\\
[\![\langle t \| e \rangle]\!]_c\ \sigma\ \tau & \triangleq\ [\![e]\!]_e\ \sigma\ \tau\ (\uparrow^{\sigma} [\![t]\!]_t) \\
[\![c\tau]\!]^{n}_{l}\ \sigma\ \tau' & \triangleq\ [\![c]\!]_c\ \sigma'\ \tau'(\uparrow^{\sigma'}[\![\tau]\!]_\tau) \\
 & \quad \text{where } k = |\tau'| - n,\ p = n + |\tau|,\ \sigma' = \sigma \circ \delta^{+k}_{[n,p]} \\
\\
[\![\varepsilon]\!]_\tau & \triangleq\ \varepsilon \\
[\![\tau_0[x_i := t]]\!]_\tau & \triangleq\ [\![\tau_0]\!]_\tau[[\![t]\!]_t] \\
[\![\tau_0[\alpha_i := E]]\!]_\tau & \triangleq\ [\![\tau_0]\!]_\tau[[\![E]\!]_E] \\
\\
\delta^{+i}_{[n,p]} & \triangleq\ \begin{cases} j \mapsto j + i & \text{if } n \le j < p \\ j \mapsto j & \text{if } j < n \end{cases}
\end{array}
$$

Figure 12 Translation of terms

▶ **Lemma 33.** *The following rules is admissible:*

$$\frac{\Gamma;\Sigma \vdash t : \forall Y <: \Upsilon_0.A \quad \Sigma \vdash \sigma : \Upsilon_1 <: \Upsilon_0}{\Gamma;\Sigma \vdash (\uparrow^\sigma t) : \forall Y <: \Upsilon_1.A}$$

Proof. We assume that the variable X is not $FV(\Gamma,\Sigma)$, otherwise it suffices to rename it. Unfolding the definition of $\uparrow^\sigma t$, we can derive:

$$\dfrac{\dfrac{\dfrac{\Gamma;\Sigma \vdash t : \forall X <: \Upsilon_0.A}{\Gamma;\Sigma,\sigma' : X <: \Upsilon_1 \vdash t : \forall X <: \Upsilon_0.A} \quad \dfrac{\Sigma \vdash \sigma : \Upsilon' <: \Upsilon_1 \quad \Sigma,\sigma' : X <: \Upsilon_1 \vdash \sigma' : X <: \Upsilon_1}{\Sigma,\sigma' : X <: \Upsilon_1 \vdash \sigma' \circ \sigma : X <: \Upsilon_0}}{\Gamma;\Sigma,\sigma' : X <: \Upsilon_1 \vdash t\ (\sigma' \circ \sigma) : A} X \notin FV(\Gamma,\Sigma)}{\Gamma;\Sigma \vdash \lambda\sigma'.t\ (\sigma' \circ \sigma) : \forall X <: \Upsilon_1.A}$$

where we use Lemma 31 to weaken $\Sigma,\sigma : X <: \Upsilon_1$.

◀

We deduce from the former lemma the following corollary that will be crucial when typing the translation of terms.

▶ **Corollary 34.** *For any level o of the hierarchy e,t,E,V,F,v, the following rule are admissible:*

$$\frac{\Gamma;\Sigma \vdash t : \Upsilon_0 \triangleright_o A \quad \Sigma \vdash \sigma : \Upsilon_1 <: \Upsilon_0}{\Gamma;\Sigma \vdash (\uparrow^\sigma t) : \Upsilon_1 \triangleright_o A} \qquad \frac{\Gamma;\Sigma \vdash \tau : \Upsilon_0 \triangleright_\tau \Upsilon \quad \Sigma \vdash \sigma : \Upsilon_1\Upsilon <: \Upsilon_0\Upsilon}{\Gamma;\Sigma \vdash (\uparrow^\sigma \tau) : \Upsilon_1 \triangleright_\tau \Upsilon}$$

▶ **Lemma 35** (Lifting values)**.** *The following rule is admissible:*

$$\frac{\Gamma;\Sigma \vdash V : \Upsilon \triangleright_V A}{\Gamma;\Sigma \vdash \uparrow^t V : \Upsilon \triangleright_t A}\uparrow$$

Proof.

$$\dfrac{\dfrac{\dfrac{\Pi_E}{\Gamma,\tau : Y,E : \Upsilon \triangleright_E A;\Sigma;\sigma : Y <: \Upsilon \vdash E\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma V) : \bot}@}{\Gamma;\Sigma \vdash \lambda\sigma\tau E.E\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma V) : \Upsilon \triangleright_t A}\lambda}{\dfrac{\Gamma;\Sigma \vdash V : \Upsilon \triangleright_V A \quad \dfrac{}{\sigma : Y <: \Upsilon \vdash \sigma : Y <: \Upsilon}{<:_{\mathtt{ax}}}}{\Gamma;\Sigma,\sigma : Y <: \Upsilon \vdash \uparrow^\sigma V : Y \triangleright_V A}}$$

where we used Corollay 34 and Π_E is the following derivation:

$$\dfrac{\dfrac{\dfrac{}{E : \Upsilon \triangleright_E A;\vdash E : Y \triangleright_E A \to \bot}{\mathtt{ax}} \quad \dfrac{}{\vdash \mathtt{id}_{|\tau|} : Y <: Y}{<:_{\mathtt{ax}}}}{E : \Upsilon \triangleright_E A;\vdash E\ \mathtt{id}_{|\tau|} : Y \to Y \triangleright_V A \to \bot}\forall_E \quad \dfrac{}{\tau : Y;\vdash \tau : Y}{\mathtt{ax}}}{\tau : Y,E : \Upsilon \triangleright_E A;\vdash E\ \mathtt{id}_{|\tau|}\ \tau : Y \triangleright_V A \to \bot}@$$

◀

▶ **Lemma 36** (Store formation)**.** *The following rules are admissible:*

$$\frac{\Gamma;\Sigma \vdash \tau : \Upsilon \quad \Gamma;\Sigma \vdash t : \Upsilon \triangleright_t A}{\Gamma;\Sigma \vdash \tau[t] : \Upsilon, A} \qquad \frac{\Sigma \vdash \sigma : \Upsilon <: [\![\Gamma_0]\!]}{\Sigma \vdash \sigma^+_{|\Upsilon|} : (\Upsilon, A) <: [\![\Gamma_0, A]\!]}$$

The same holds for $\Gamma \vdash E : \Upsilon \triangleright_E \iota(A)$ and $\Gamma \vdash \tau[E] : \Upsilon, A^\perp$.

Proof. The left rule is a straightforward application of$_{\tau\tau'}$- and$_{\tau_t}$-rules:

$$\dfrac{\Gamma;\Sigma\vdash\tau[t]:Y,A \quad \dfrac{\Gamma;\Sigma\vdash t:Y\rhd_t\iota(A)}{\Gamma;\Sigma\vdash[t]:Y\rhd_\tau\iota(A)^\perp}\tau_t}{\Gamma;\Sigma\vdash\tau[t]:Y,A}\tau\tau'$$

The right one is a reformulation of Lemma 27.

◀

▶ **Lemma 37** (Shifts). *For any $\Upsilon_0,\Upsilon'_0,\Upsilon_1$, if $\sigma:\Upsilon'_0<:\Upsilon_0$ and $n=|\Upsilon_0|, p=n+|\Upsilon_1|$, $k=[\Upsilon'_0|-|\Upsilon_0|$, if we define $\sigma'=\sigma\circ\delta^{+k}_{[n,p]}$ then $\sigma':(\Upsilon'_0\Upsilon_1)<:(\Upsilon_0\Upsilon_1)$.*
In particular, the following rules are admissible for any level o:

$$\dfrac{\Gamma;\Sigma\vdash t:\Upsilon_0\Upsilon_1\rhd_o A \quad \Sigma\vdash\sigma:\Upsilon'_0<:\Upsilon_0}{\Gamma;\Sigma\vdash(\uparrow^{\sigma'}t):\Upsilon'_0\Upsilon_1\rhd_o A} \qquad \dfrac{\Gamma;\Sigma\vdash\tau:\Upsilon_0\rhd_\tau\Upsilon_1 \quad \Sigma\vdash\sigma:\Upsilon'_0<:\Upsilon_0}{\Gamma;\Sigma\vdash(\uparrow^{\sigma'}\tau):\Upsilon'_0\rhd_\tau\Upsilon_1}$$

Proof. We denote by $\Upsilon(i)$ the i^{th}-element of the list Υ. By definition, we have:

$$\sigma'(i)=\begin{cases}i+k & \text{if } n\le i<p\\ \sigma'(i) & \text{if } j<n\end{cases}$$

Thus we have:

- if $i<n$:
$$(\Upsilon'_0\Upsilon_1)(\sigma'(i))=\Upsilon'_0(\sigma'(i))=\Upsilon'_0(\sigma(i))=\Upsilon_0(i)$$
- otherwise:
$$(\Upsilon'_0\Upsilon_1)(\sigma'(i))=(\Upsilon'_0\Upsilon_1)(i+k)=\Upsilon_1(i+k-|\Upsilon'_0|)=\Upsilon_1(i-|\Upsilon_0|)=(\Upsilon_0\Upsilon_1)(i)$$

Hence $\sigma':(\Upsilon'_0\Upsilon_1)<:(\Upsilon_0\Upsilon_1)$. ◀

We are now equipped to prove the main theorem of this section, that is the correctness of the translation with respect to types.

▶ **Theorem 38.** *The translation is well-typed, i.e.*

1. *if* $\Gamma\vdash_v v:A$ *then* $[\![\Gamma\vdash_v v:A]\!]$
2. *if* $\Gamma\vdash_F F:A^\perp$ *then* $[\![\Gamma\vdash_F F:A^\perp]\!]$
3. *if* $\Gamma\vdash_V V:A$ *then* $[\![\Gamma\vdash_V V:A]\!]$
4. *if* $\Gamma\vdash_E E:A^\perp$ *then* $[\![\Gamma\vdash_E E:A^\perp]\!]$
5. *if* $\Gamma\vdash_t t:A$ *then* $[\![\Gamma\vdash_t t:A]\!]$
6. *if* $\Gamma\vdash_e e:A^\perp$ *then* $[\![\Gamma\vdash_e e:A^\perp]\!]$
7. *if* $\Gamma\vdash_c c$ *then* $[\![\Gamma\vdash_c c]\!]$
8. *if* $\Gamma\vdash_l l$ *then* $[\![\Gamma\vdash_l l]\!]$
9. *if* $\Gamma\vdash_\tau \tau$ *then* $[\![\Gamma\vdash_\tau \tau:\Gamma']\!]$

Proof. The proof is very almost the same as the proof of Theorem 20, using the previous lemmas. We reason by induction over the typing rules of Figure 8. We (ab)use of Lemma 31 to make the derivations more compact by systematically weakening contexts as soon as possible, and compact the first ∀- and λ-introductions in one rule.

1. Strong values

Case $[\![\mathbf{c}]\!]_v$.

$[\![\mathbf{c}]\!]_v=\mathbf{c}$, which has the desired type by hypothesis.

Case $\lambda x_i.t$.

In the source language, we have:

$$\frac{\Gamma, x_i : A \vdash_t t : B \quad |\Gamma| = i}{\Gamma \vdash_v \lambda x_i : A \to B}$$

Hence, we get by induction a proof Π_t of $[\![t]\!]_t : [\![\Gamma, x_i : A]\!] \rhd_t \iota(B)$ and we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_t}{\vdash [\![t]\!]_t : \forall Y' <: [\![\Gamma, x_i : A]\!].Y' \to Y' \rhd_E \iota(B) \to \bot}\ \boldsymbol{\Pi_\sigma}}{; \sigma : Y <: [\![\Gamma]\!] \vdash [\![t]\!]_t\ \sigma^+_{|\tau|} : (Y, \iota(A)) \to (Y, \iota(A)) \rhd_E \iota(B) \to \bot}\ \forall_E \quad \boldsymbol{\Pi_\tau}}{\tau : Y, u : Y \rhd_t \iota(A); \sigma : Y <: [\![\Gamma]\!] \vdash [\![t]\!]_t\ \sigma^+_{|\tau|}\ \tau[u] : (Y, \iota(A)) \rhd_E \iota(B) \to \bot}\ @ \quad \boldsymbol{\Pi_E}}{\tau : Y, u : Y \rhd_t \iota(A), E : Y \rhd_E \iota(B); \sigma : Y <: [\![\Gamma]\!] \vdash [\![t]\!]_t\ \sigma^+_{|\tau|}\ \tau[u]\ E^+ : \bot}\ @}{\vdash \lambda\sigma\tau u E.[\![t]\!]_t\ \sigma^+_{|\tau|}\ \tau[u]\ E : \forall Y <: [\![\Gamma]\!].Y \to Y \rhd_t \iota(A) \to Y \rhd_E \iota(B) \to \bot}\ \lambda$$

where:

- Π_E is a proof of $E : Y \rhd_E \iota(B) \vdash E : (Y, \iota(A)) \rhd_E \iota(B)$ (derivable according to Lemma 32);
- Π_τ is a proof of $\tau : Y, u : Y \rhd_t \iota(A); \vdash \tau[u] : Y, \iota(A)$ (derivable according to Lemma 36);
- Π_σ is a obtained by Lemma 36:

$$\frac{\dfrac{}{\sigma : Y <: [\![\Gamma]\!] \vdash \sigma : Y <: [\![\Gamma]\!]}\ <:_{\mathrm{ax}}}{\sigma : Y <: [\![\Gamma]\!] \vdash \sigma^+_{|\tau|} : (Y, \iota(A)) <: [\![\Gamma, x_i : A]\!]}$$

2. Forcing contexts

Case $[\![\alpha]\!]_F$.

$[\![\alpha]\!]_F = \alpha$, which has the desired type by hypothesis.

Case $[\![t.E]\!]_F$.

In the source language, we have:

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_E E : B^\perp}{\Gamma \vdash_F t \cdot E : (A \to B)^\perp}$$

Hence we have by induction hypothesis that $\vdash [\![t]\!]_t : [\![\Gamma]\!]_\Gamma \rhd_t \iota(A)$ and $\vdash [\![E]\!]_t : [\![\Gamma]\!]_\Gamma \rhd_E \iota(B)$, so that we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{v : Y \rhd_v \iota(A) \to \iota(B); \vdash v : \forall Y' <: Y : Y' \to Y' \rhd_t \iota(A) \to Y' \rhd_E \iota(B) \to \bot}\ \mathtt{ax} \quad \boldsymbol{\Pi_\sigma}}{\tau : Y, v : Y \rhd_v \iota(A) \to \iota(B); \vdash v\ \mathtt{id}_{|\tau|} : Y \to Y \rhd_t \iota(A) \to Y \rhd_E B \to \bot}\ \forall_I \quad \boldsymbol{\Pi_\tau}}{\tau : Y, v : Y \rhd_v \iota(A) \to \iota(B); \vdash v\ \mathtt{id}_{|\tau|}\ \tau\ : Y \rhd_t \iota(A) \to Y \rhd_E \iota(B) \to \bot}\ @ \quad \boldsymbol{\Pi_t}}{\tau : Y, v : Y \rhd_v \iota(A) \to \iota(B); \sigma : Y <: [\![\Gamma]\!] \vdash v\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma [\![t]\!]_t) : Y \rhd_E \iota(B) \to \bot}\ @ \quad \boldsymbol{\Pi_E}}{\tau : Y, v : Y \rhd_v \iota(A) \to \iota(B); \sigma : Y <: [\![\Gamma]\!] \vdash v\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma [\![t]\!]_t)\ (\uparrow^\sigma [\![E]\!]_E) : \bot}\ @}{\vdash \lambda\sigma\tau v.v\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma [\![t]\!]_t)\ (\uparrow^\sigma [\![E]\!]_E) : \forall Y <: [\![\Gamma]\!]_\Gamma.Y \to Y \rhd_v \iota(A) \to \iota(B) \to \bot}\ \lambda$$

where:

- Π_E is a proof of $\varepsilon; \sigma : Y <: [\![\Gamma]\!] \vdash (\uparrow^\sigma [\![E]\!]_E) : Y \rhd_E \iota(B)$, derivable from the induction hypothesis for t and Corollary 34.
- Π_t is a proof of $\varepsilon; \sigma : Y <: [\![\Gamma]\!] \vdash (\uparrow^\sigma [\![t]\!]_t) : Y \rhd_t \iota(A)$, derivable from the induction hypothesis for E and Corollary 34.
- Π_τ is the axiom rule $\tau : Y; \vdash \tau : Y$
- Π_σ is a proof of $\mathtt{id}_{|\tau|} : Y <: Y$ (Proposition 29)

3. Weak values

Case $[\![v]\!]_V$.

In the source language, we have:

$$\frac{\Gamma \vdash_v v : A}{\Gamma \vdash_V v : A}$$

Hence we have by induction hypothesis that $\vdash [\![v]\!]_v : [\![\Gamma]\!]_\Gamma \triangleright_v \iota(A)$ and we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{F : Y \triangleright_F \iota(A) \vdash F : \forall Y' <: Y.Y' \to Y' \triangleright_v \iota(A) \to \bot \quad \Pi_Y}{F : Y \triangleright_F \iota(A); \sigma : Y <: [\![\Gamma]\!] \vdash F\ \mathtt{id}_{|\tau|} : Y \to Y \triangleright_v \iota(A) \to \bot}\,@ \quad \dfrac{}{\tau : Y; \vdash \tau : Y}}{\tau : Y, F : Y \triangleright_F \iota(A) \vdash F\ \mathtt{id}_{|\tau|}\ \tau : Y \triangleright_v \iota(A) \to \bot}\,@ \quad \Pi_v}{\tau : Y, F : Y \triangleright_F \iota(A); \sigma : Y <: [\![\Gamma]\!] \vdash F\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma [\![v]\!]_v) : \bot}\,@}{\vdash \lambda\sigma\tau F.F\ \mathtt{id}_{|\tau|}\ \tau\ (\uparrow^\sigma [\![v]\!]_v) : \forall Y <: [\![\Gamma]\!].Y \to Y \triangleright_F \iota(A) \to \bot}\,\lambda$$

where:

- Π_v is a proof of $\varepsilon; \sigma : Y <: [\![\Gamma]\!] \vdash (\uparrow^\sigma [\![v]\!]_v) : Y \triangleright_v \iota(A)$, derivable from the induction hypothesis and Corollary 34.
- Π_τ is the axiom rule $\tau : Y; \vdash \tau : Y$
- Π_Y is a proof of $\mathtt{id}_{|\tau|} : Y <: Y$ (Proposition 29)

Case $[\![x_i]\!]_V$.

In the source language, we have:

$$\frac{\Gamma(i) = (x_i : A)}{\Gamma \vdash_V x_i : A}$$

so that Γ is of the form $\Gamma', x_i : A, \Gamma''$. By definition, we have:

$$[\![x_i]\!]_V = \lambda\sigma\tau F.\mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\, n\, \tau\ \mathbf{in}\ t\ \mathtt{id}_n\ \tau_0\ (\lambda\sigma'\tau_0'\lambda V.V\ \tau''[\uparrow^t V](\uparrow^{\sigma''}\tau_1)\ (\uparrow^{\sigma''}F))$$

where $n = \sigma(i)$,$k = |\tau_0| - n$, $p = n + |\tau_1|$, $\sigma'' = \sigma' \circ \delta^{+k}_{[n,p]}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{t : Y_0^n \triangleright_t \iota(A) \vdash t : Y_0^n \triangleright_t \iota(A)}\,\mathtt{ax} \quad \dfrac{}{\vdash \mathtt{id}_n : Y_0^n <: Y_0^n}\,<:_{\mathtt{ax}}}{t : Y_0^n \triangleright_t \iota(A); \vdash t\ \mathtt{id}_n : Y_0^n \to Y_0^n \triangleright_E \iota(A) \to \bot}\,\forall_E \quad \dfrac{}{\tau_0 : Y_0^n \vdash \tau_0 : Y_0^n}\,\mathtt{ax}}{\tau_0 : Y_0^n, t : Y_0^n \triangleright_t A; \vdash t\ \mathtt{id}_n\ \tau_0 : Y_0^n \triangleright_E \iota(A) \to \bot}\,@ \quad \boldsymbol{\Pi_E}}{\tau_0 : Y_0^n, t : Y_0^n \triangleright_t \iota(A), \tau_1 : (Y_0^n, n : \iota(A)) \triangleright_\tau Y_1, F : (Y_0^n, n : \iota(A), Y_1) \triangleright_F \iota(A); \vdash t\ \mathtt{id}_n\ \tau_0\ E : \bot}\,@ \quad |Y_0^n| = n}{\tau : (Y_0^n, n : \iota(A), Y_1), F : (Y_0^n, n : \iota(A), Y_1) \triangleright_F \iota(A); \vdash \mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\, \tau\, n\ \mathbf{in}\ t\ \mathtt{id}_n\ \tau_0\ E : \bot}\,\mathtt{split} \quad \boldsymbol{\Pi_\sigma}}{\tau : Y, F : Y \triangleright_F \iota(A); \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\, \tau\, n\ \mathbf{in}\ t\ \mathtt{id}_n\ \tau_0\ E : \bot}\,<:_{\mathtt{split}}$$
$$\frac{}{\vdash \lambda\sigma\tau F.\mathbf{let}\ \tau_0, t, \tau_1 = \mathbf{split}\, \tau\, n\ \mathbf{in}\ t\ \mathtt{id}_n\ \tau_0\ E : \forall Y <: [\![\Gamma]\!]_\Gamma.Y \to Y \triangleright_F \iota(A) \to \bot}\,\lambda$$

where:

- Π_σ is simply the axiom rule:

$$\frac{}{\sigma : Y <: ([\![\Gamma_0]\!]_\Gamma, n : \iota(A), [\![\Gamma_1]\!]_\Gamma) \vdash \sigma : Y <: ([\![\Gamma_0]\!]_\Gamma, n : \iota(A), [\![\Gamma_1]\!]_\Gamma)}\,<:_{\mathtt{ax}}$$

- $E = \lambda\sigma'\tau''\lambda V.V\ \tau_0'[\uparrow^t V](\uparrow^{\sigma''}\tau_1)\ (\uparrow^{\sigma''}F))$ and Π_E is the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{V : Y_0' \triangleright_V \iota(A); \vdash \uparrow^t V : Y_0' \triangleright_t \iota(A)}\,\mathtt{ax} \quad \dfrac{}{\vdash \mathtt{id}_p : Y_0', A, Y_1 <: Y_0', A, Y_1}}{V : Y_0' \triangleright_V \iota(A); \vdash V\ \mathtt{id}_p : (Y_0', \iota(A), Y_1) \to (Y_0', \iota(A), Y_1) \triangleright_F \iota(A) \to \bot}\,\forall_E \quad \boldsymbol{\Pi_\tau}}{\tau_1 : (Y_0^n, \iota(A)) \triangleright_\tau Y_1, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A); \vdash V\ \mathtt{id}_p\ \tau_0'[\uparrow^t V](\uparrow^{\sigma'}\tau_1) : (Y_0', \iota(A), Y_1) \triangleright_F \iota(A) \to \bot}\,@ \quad \boldsymbol{\Pi_F}}{\Gamma, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A); \sigma' : Y_0' <: Y_0^n \vdash V\ \mathtt{id}_p\ \tau_0'[\uparrow^t V](\uparrow^{\sigma''}\tau')\ (\uparrow^{\sigma''}F) : \bot}\,@$$
$$\frac{}{\Gamma \vdash \lambda\sigma'\tau_0'V.V\ \mathtt{id}_p\ \tau_0'[\uparrow^t V](\uparrow^{\sigma''}\tau_1)\ (\uparrow^{\sigma''}F) : Y_0^n \triangleright_F \iota(A)}\,\lambda$$

where $\Gamma = \tau_1 : (Y_0^n, \iota(A)) \triangleright_\tau Y_1, F : (Y_0^n, \iota(A), Y_1) \triangleright_F \iota(A)$.

- Π_F is the following proof, obtained by Lemma 37:

$$\dfrac{\dfrac{}{F : (Y_0^n, \iota(A), Y_1) \triangleright_F \iota(A); \vdash F : (Y_0^n, \iota(A), Y_1) \triangleright_F \iota(A)}\,\text{ax} \quad \dfrac{}{\sigma' : Y_0' <: Y_0^n \vdash \sigma' : Y_0' <: Y_0^n}\,\text{ax}}{F : (Y_0^n, \iota(A), Y_1) \triangleright_F \iota(A); \sigma_1 : Y_0' <: Y_0^n \vdash (\uparrow^{\sigma''} F) : (Y_0', \iota(A), Y_1) \triangleright_F \iota(A)}$$

- Π_τ is the following derivation

$$\dfrac{\dfrac{\dfrac{}{\tau_0' : Y_0' \vdash \tau_0' : Y_0'}\,\text{ax} \quad \dfrac{\dfrac{\dfrac{}{V : Y_0' \triangleright_V \iota(A) \vdash V : Y_0' \triangleright_V A}\,\text{ax}}{V : Y_0' \triangleright_V \iota(A) \vdash \uparrow^t V : Y_0' \triangleright_t A}\uparrow}{V : Y_0' \triangleright_V \iota(A) \vdash [\uparrow^t V] : Y_0' \triangleright_\tau \iota(A)}\,\tau_t}{Y_0' <: Y_0^n, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A) \vdash \tau_0'[\uparrow^t V] : Y_0', n : \iota(A)}\,\tau\tau' \qquad \boldsymbol{\Pi_{\tau_1}}}{\tau_1 : (Y_0^n, \iota(A)) \triangleright_\tau Y_1, Y_0' <: Y_0^n, \tau_0' : Y_0', V : Y_0' \triangleright_V \iota(A) \vdash \tau_0'[V](\uparrow^{\sigma''} \tau_1) : Y_0', \iota(A), Y_1}\,\tau <:$$

- Π_{τ_1} is obtained by Lemma 37:

$$\dfrac{\dfrac{}{\tau_1 : (Y_0, n : \iota(A)) \triangleright_\tau Y_1 \vdash \tau_1 : (Y_0, \iota(A)) \triangleright_\tau Y_1}\,\text{ax} \quad \dfrac{}{\sigma' : Y_0' <: Y_0^n \vdash \sigma' : Y_0' <: Y_0^n}\,<:\text{ax}}{\tau_1 : (Y_0^n, \iota(A)) \triangleright_\tau Y_1; \sigma' : Y_0' <: Y_0 \vdash (\uparrow^{\sigma''} \tau_1) : Y_0', n : \iota(A) \triangleright_\tau Y_1}$$

4. Catchable contexts

Case $[\![F]\!]_E$.

This case is similar to the case $[\![v]\!]_V$.

Case $[\![\tilde{\mu}[x_i].\langle x_i \| F \rangle \tau']\!]_E$.

In the source language, we have:

$$\dfrac{\Gamma, x_i : A, \Gamma' \vdash_F F : A^\perp \quad \Gamma, x_i : A \vdash_\tau \tau' : \Gamma' \quad |\Gamma| = i}{\Gamma \vdash_E \tilde{\mu}[x_i].\langle x_i \| F \rangle \tau' : A^\perp}$$

We have by induction hypothesis a proof of $\vdash [\![\tau']\!]_\tau : [\![\Gamma, x_i : A]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma$ and a proof Π_F of $\vdash [\![F]\!]_F : [\![\Gamma, x_i : A, \Gamma']\!]_\Gamma \triangleright_F \iota(A)$. We can thus derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{V : Y \triangleright_V \iota(A); \vdash V : Y \triangleright_t \iota(A)}\,\text{ax} \quad \dfrac{}{\vdash \mathtt{id}_p : (Y, \iota(A), [\![\Gamma']\!]_\Gamma) <: Y}}{V : Y \triangleright_V \iota(A); \vdash V\ \mathtt{id}_p : (Y, \iota(A), [\![\Gamma']\!]_\Gamma) \to (Y, \iota(A), [\![\Gamma']\!]_\Gamma) \triangleright_F \iota(A) \to \bot}\,\forall_E \qquad \boldsymbol{\Pi_\tau}}{\tau : Y, V : Y \triangleright_V \iota(A); \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash V\ \mathtt{id}_p\ \tau[\uparrow^t V](\uparrow^{\sigma'} [\![\tau']\!]_\tau) : (Y, \iota(A), [\![\Gamma']\!]_\Gamma) \triangleright_F \iota(A) \to \bot}\,@ \qquad \boldsymbol{\Pi_F}}{\dfrac{\Gamma, \tau : Y, V : Y \triangleright_V \iota(A); \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash V\ \mathtt{id}_p\ \tau[\uparrow^t V](\uparrow^{\sigma'} [\![\tau']\!]_\tau)\ (\uparrow^{\sigma'} [\![F]\!]_F) : \bot}{\Gamma \vdash \lambda\sigma\tau V.V\ \mathtt{id}_p\ \tau[\uparrow^t V](\uparrow^{\sigma'} [\![\tau']\!]_\tau)\ (\uparrow^{\sigma'} [\![F]\!]_F) : [\![\Gamma]\!]_\Gamma \triangleright_F \iota(A)}\,\lambda}\,@$$

where:

- $n = |\tau|,\ k = n - i,\ p = n + |\tau'|,\ \sigma' = \sigma \circ \delta^{+k}_{[i,p]}$
- Π_F is the following proof, obtained by Lemma 37:

$$\dfrac{;\vdash F : ([\![\Gamma]\!]_\Gamma, \iota(A), [\![\Gamma']\!]_\Gamma) \triangleright_F \iota(A) \quad \dfrac{}{\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}\,\text{ax}}{;\sigma : Y <: [\![\Gamma']\!]_\Gamma \vdash (\uparrow^{\sigma'} F) : (Y, \iota(A), [\![\Gamma']\!]_\Gamma) \triangleright_F \iota(A)}$$

- Π_τ is the following proof:

$$\dfrac{\dfrac{\dfrac{}{\tau : Y \vdash \tau : Y}\,\mathtt{ax} \quad \dfrac{\dfrac{\dfrac{}{V : Y \triangleright_V \iota(A) \vdash V : Y \triangleright_V \iota(A)}\,\mathtt{ax}}{V : Y \triangleright_V \iota(A) \vdash \uparrow^t V : Y \triangleright_t \iota(A)}\,\uparrow}{V : Y \triangleright_V \iota(A) \vdash [V] : Y \triangleright_\tau \iota(A)}\,\tau_t}{\tau : Y, V : Y \triangleright_V \iota(A) \vdash \tau[\uparrow^t V] : Y, \iota(A)}\,\tau\tau' \qquad \boldsymbol{\Pi_{\tau'}}}{\tau : Y, V : Y' \triangleright_V \iota(A); \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \tau[\uparrow^t V][\![\tau']\!]_\tau : (Y, \iota(A), [\![\Gamma']\!]^{\sigma[x:=n]})}\,\tau\tau'$$

- $\Pi_{\tau'}$ is the following proof, obtained from the induction hypothesis for τ' and Lemma 37:

$$\dfrac{\vdash [\![\tau']\!]_\tau : [\![\Gamma]\!]_\Gamma, \iota(A) \triangleright_\tau [\![\Gamma']\!]_\Gamma \quad \dfrac{}{\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}\,<:\mathtt{ax}}{; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \uparrow^{\sigma'} [\![\tau']\!]_\tau : Y, \iota(A) \triangleright_\tau [\![\Gamma']\!]_\Gamma}$$

5. Terms

Case $[\![V]\!]_t$.

This case is similar to the case $[\![v]\!]_V$.

Case $[\![\mu\alpha_i.c]\!]_t$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we have:

$$\dfrac{\Gamma, \alpha_i : A^\perp \vdash_c c \quad |\Gamma| = i}{\Gamma \vdash_t \mu\alpha_i.c : A}$$

Hence we have by induction a proof of $; \vdash [\![c]\!]_c : [\![\Gamma, x_i : A^\perp]\!]_\Gamma \triangleright_c \perp$ and we can derive:

$$\dfrac{\dfrac{\dfrac{; \vdash [\![c]\!]_c : [\![\Gamma, x_i : A^\perp]\!]_\Gamma \triangleright_c \perp \qquad \boldsymbol{\Pi_\sigma}}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![c]\!]_c\ \sigma^+_{|\tau|} : (Y, \iota(A)^\perp) \to \perp}\,\forall_E \qquad \boldsymbol{\Pi_\tau}}{\tau : Y, E : Y \triangleright_E \iota(A); \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![c]\!]_c\ \sigma^+_{|\tau|}\ \tau[E] : \perp}\,@}{; \vdash \lambda\sigma\tau E.[\![c]\!]_c\ \sigma^+_{|\tau|}\ \tau[E] : [\![\Gamma]\!]_\Gamma \triangleright_t \iota(A)}\,\lambda$$

where

- Π_σ is the following derivation, obtained by Lemma 36 (since $|\tau|$ matches $|Y|$):

$$\dfrac{\dfrac{}{\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}\,<:\mathtt{ax}}{\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma^+_{|\tau|} : (Y, \iota(A)^\perp) <: [\![\Gamma, x_i : \iota(A)^\perp]\!]_\Gamma}$$

- Π_E is also obtained by Lemma 36:

$$\dfrac{\dfrac{}{\tau : Y, E : Y \triangleright_E \iota(A); \vdash \tau[E] : Y, \iota(A)^\perp}\,\mathtt{ax} \quad \dfrac{}{E : Y \triangleright_E \iota(A); \vdash E : Y \triangleright_E \iota(A)}\,\mathtt{ax}}{\tau : Y, E : Y \triangleright_E \iota(A); \vdash \tau[E] : Y, \iota(A)^\perp}$$

6. Contexts

Case $[\![E]\!]_e$.

This case is similar to the case $[\![v]\!]_V$.

Case $[\![\tilde{\mu}x_i.c]\!]_e$.

This case is similar to the case $[\![\mu\alpha_i.c]\!]_t$.

7. Commands

Case $[\![\langle t \| e \rangle]\!]_c$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus we have:

$$\frac{\Gamma \vdash_t t : A \quad \Gamma \vdash_e e : A^{\perp}}{\Gamma \vdash_c \langle t \| e \rangle}$$

thus we get by induction two proofs of $;\vdash [\![t]\!]_t : [\![\Gamma]\!]_\Gamma \triangleright_t \iota(A)$ and $;\vdash [\![e]\!]_c : [\![\Gamma]\!]_\Gamma \triangleright_e \iota(A)$. We can then derive:

$$\dfrac{\dfrac{\dfrac{;\vdash [\![e]\!]_e : [\![\Gamma]\!]_\Gamma \triangleright_e \iota(A)}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![e]\!]_e\ \sigma : Y \to Y \triangleright_t \iota(A) \to \perp}\forall_E \quad \boldsymbol{\Pi_\sigma}}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![e]\!]_e\ \sigma\ \tau : Y \triangleright_t \iota(A) \to \perp}@ \quad \dfrac{}{\tau : Y; \vdash \tau : Y}\mathtt{ax}}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![e]\!]_e\ \sigma\ \tau\ (\uparrow^\sigma [\![t]\!]_t) : \perp}@ \quad \boldsymbol{\Pi_t}}{;\vdash \lambda\sigma\tau.[\![e]\!]_e\ \sigma\ \tau\ (\uparrow^\sigma [\![t]\!]_t) : [\![\Gamma]\!]_\Gamma \triangleright_c \perp}\lambda$$

where:

- Π_σ is the axiom rule:
$$\frac{}{\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}<:_{\mathtt{ax}}$$
- Π_t is obtained using Lemma 33:
$$\frac{;\vdash [\![t]\!]_t : [\![\Gamma]\!]_\Gamma \triangleright_t \iota(A) \quad \dfrac{}{;\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}<:_{\mathtt{ax}}}{;\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \uparrow^\sigma [\![t]\!]_t : Y \triangleright_t \iota(A)}$$

8. Closures

Case $[\![c\tau']\!]^n_l$.

In the $\overline{\lambda}_{[lv\tau\star]}$-calculus, we have:

$$\frac{\Gamma, \Gamma' \vdash_c c \quad \Gamma \vdash_\tau \tau' : \Gamma'}{\Gamma \vdash_l c\tau'}$$

where n matches $|\Gamma|$. We thus get by induction two proofs $;\vdash [\![\tau']\!]_\tau : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma$ and $\vdash [\![c]\!]_c : [\![\Gamma, \Gamma']\!]_\Gamma \triangleright_c \perp$. We can derive:

$$\dfrac{\dfrac{\vdash [\![c]\!]_c : [\![\Gamma, \Gamma']\!]_\Gamma \triangleright_c \perp \qquad \boldsymbol{\Pi_\sigma}}{;\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![c]\!]_c\ \sigma' : (Y, [\![\Gamma']\!]_\Gamma) \to \perp}\forall_E \quad \dfrac{\dfrac{}{\tau : Y; \vdash \tau : Y}\mathtt{ax} \qquad \boldsymbol{\Pi_\tau}}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \tau(\uparrow^{\sigma'} [\![\tau']\!]_\tau) : Y[\![\Gamma']\!]_\Gamma}\tau\tau'}{\tau : Y; \sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash [\![c]\!]_c\ \sigma'\ \tau'(\uparrow^{\sigma'} [\![\tau']\!]_\tau : \perp}@}{;\vdash \lambda\sigma\tau.[\![c]\!]_c\ \sigma'\ \tau'(\uparrow^{\sigma'} [\![\tau']\!]_\tau)}\lambda$$

where:

- $k = |\tau'| - n,\ p = n + |\tau|,\ \sigma' = \sigma \circ \delta^{+k}_{[n,p]}$
- Π_σ is a proof of $\sigma : Y <: [\![\Gamma]\!]_\Gamma \vdash \sigma' : (Y, [\![\Gamma']\!]_\Gamma) <: [\![\Gamma, \Gamma']\!]_\Gamma$ obtained by Lemma 37;
- $\Pi_{\tau'}$ is the following proof also obtained by Lemma 37:
$$\frac{;\vdash [\![\tau']\!]_\tau : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma \quad \dfrac{}{\vdash \sigma : Y <: [\![\Gamma]\!]_\Gamma}<:_{\mathtt{ax}}}{\vdash (\uparrow^{\sigma'} [\![\tau']\!]_\tau) : Y \triangleright_\tau [\![\Gamma']\!]_\Gamma}$$

9. Stores

Case $[\![\tau[x_i := t]]\!]_\tau$.

We only consider the case $\tau[x_i := t]$, the proof for the case $\tau[\alpha_i := E]$ is identical. This corresponds to the typing rules:

$$\frac{\Gamma \vdash_\tau \tau : \Gamma' \qquad \Gamma, \Gamma' \vdash_t t : A \quad |\Gamma, \Gamma'| = i}{\Gamma \vdash_\tau \tau[x_i := t] : \Gamma', x_i : A}$$

By induction we obtain two proofs of $\vdash [\![\tau]\!]_\tau : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma$ and $\vdash [\![t]\!]_t : [\![\Gamma, \Gamma']\!]_\Gamma \triangleright_t \iota(A)$. We can thus derive:

$$\frac{\vdash [\![\tau]\!]_\tau : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma \qquad \dfrac{\vdash [\![t]\!]_t : [\![\Gamma, \Gamma']\!]_\Gamma \triangleright_t \iota(A)}{\vdash [[\![t]\!]_t] : [\![\Gamma, \Gamma']\!]_\Gamma \triangleright_\tau \iota(A)}\,\tau_t}{\vdash [\![\tau]\!]_\tau[[\![t]\!]_t] : [\![\Gamma]\!]_\Gamma \triangleright_\tau [\![\Gamma']\!]_\Gamma, \iota(A)}\,\tau\tau'$$

◀