Under consideration for publication in Math. Struct. in Comp. Science

Forcing as a program transformation

Alexandre MIQUEL[1]

[1] *École Normale Supérieure de Lyon – Laboratoire d'Informatique du Parallélisme*
46, allée d'Italie – 69364 Lyon Cedex 07 – FRANCE

Received

This paper is a study of the forcing translation (in the sense of Cohen (Coh63; Coh64)) through the proofs as programs correspondence in classical logic, following the methodology introduced by Krivine in (Kri08; Kri10). For that, we introduce an extension of (classical) higher-order arithmetic suited to express the forcing translation, called PAω^+, as well as the corresponding proof system based on Curry-style proof terms with call/cc. Then, given a poset of conditions (represented in PAω^+ as an upwards closed subset of a fixed meet semi-lattice), we define the forcing translation $A \mapsto (p \Vdash A)$ (where A ranges over propositions) and show that the corresponding transformation of proofs is induced by a simple program transformation $t \mapsto t^*$ defined on raw proof terms (i.e. independently from the derivation). From an analysis of the computational behavior of transformed programs, we show how to avoid the cost of the transformation by introducing an extension of Krivine's abstract machine devoted to the execution of proofs constructed by forcing. We show that this machine induces new classical realizability models and present the corresponding adequacy results.

Contents

1. Introduction

The method of forcing was introduced by Cohen (Coh63; Coh64) to prove the relative consistency of the negation of the continuum hypothesis w.r.t. the axioms of set theory (ZFC). (The relative consistency of the continuum hypothesis was already proved by Gödel (Göd38) with the technique of constructible sets.) Since then, the forcing technique has been widely investigated, and it now constitutes a standard item in the toolbox of set theorists (Jec02). Cohen forcing is traditionally presented in a model-theoretic way, as a method to extend a given model $\mathscr{M}$ of ZF/ZFC into a larger model $\mathscr{M}[G]$ obtained by adding to $\mathscr{M}$ an 'ideal object' G (called a generic set) whose properties are deduced from a poset of conditions $(\mathsf{C}, \leq)$ given as a point of the initial model $\mathscr{M}$. This point of view is deeply connected to the construction of Boolean-valued models of ZF/ZFC (Bel85), that can actually be seen as an alternative presentation of forcing.

But from a proof theoretic point of view, all these model theoretic constructions ultimately rely on a formula translation mapping every formula A of a given theory $\mathscr{T}$ (typically: ZF, ZFC, or PAω^+ in this paper) to another formula written $p \Vdash A$ ('p forces A') that depends on an extra parameter p representing a forcing condition (taken in the poset $(\mathsf{C}, \leq)$ that parameterizes the construction). The formula translation $A \mapsto (1 \Vdash A)$ (writing 1 the largest element of C) is actually a *logical translation* since it preserves provability in the theory $\mathscr{T}$, so that it can be used as a device to extend the notion of provability that comes with $\mathscr{T}$. Indeed, if we consider the theory $\mathscr{T}[\mathsf{C}]$ formed by all formulæ A such that $\mathscr{T} \vdash (1 \Vdash A)$, then $\mathscr{T}[\mathsf{C}]$ is an extension—and in general a strict extension—of the initial theory $\mathscr{T}$ that is consistent relatively to $\mathscr{T}$. Depending on the choice of the poset $(\mathsf{C}, \leq)$ that parameterizes the construction, the extended theory $\mathscr{T}[\mathsf{C}]$ may prove interesting formulæ that are not provable in the initial theory $\mathscr{T}$—for instance: the continuum hypothesis or its negation.

Surprisingly, Cohen's forcing has received much less attention in proof theory than in model theory. One reason for this is that the forcing translation introduced by Cohen is intrinsicly classical, whereas proof theory is in general much better understood in the framework of intuitionistic logic, especially when working via the correspondence between proofs and programs (i.e. the Curry-Howard correspondence). Due to this, proof-theoretic analyses of Cohen forcing are usually carried out indirectly, through a suitable negative translation from classical logic to intuitionistic logic (Avi04; CJ10). However, it seems to be difficult to understand the computational meaning of the forcing translation—that is: at the level of proof terms—when forcing is studied through a negative translation from classical logic to intuitionistic logic.

In 1990, Griffin (Gri90) discovered that the control operator call/cc ('call with current continuation') of the Scheme programming language could be given the type corresponding to Peirce's law, which provided an elegant way to extend the correspondence between proofs and programs to classical logic. Since then, many classical λ-calculi have been introduced in the literature, such as Parigot's $\lambda\mu$-calculus (Par97), Barbanera and Berardi's symmetric λ-calculus (BB96) and Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus (CH00), together with type systems corresponding to classical logic. To analyze the computational behavior of classical proof terms, Krivine introduced the theory of *classical realizability* (Kri01; Kri03; Kri09), which is a reformulation of Kleene's realizability (Kle45) in which the computational contents of classical proofs can be analyzed directly, rather than through a negative translation. More recently, Krivine showed (Kri08; Kri10) how to combine Cohen forcing with classical realizability, and discovered the existence of a simple program transformation (defined on classical λ-terms) that turns any Curry-style proof term t of a formula A (in PA2/PA3) into a classical realizer t^* of the formula $1 \Vdash A$ in the suitable realizability model. From this, he deduced a method to realize a theorem whose proof relies on an axiom that can be forced using a suitable set of conditions.

The aim of this paper is to present and study in a more systematic way (a variant of) this program transformation in higher-order arithmetic using a fully typed setting, and, up to some extent, independently from the theory of classical realizability. For that, we shall present an extension of higher-order arithmetic with classical proof terms, called system PAω^+, and define the forcing relation $p \Vdash A$ in this framework. The subtle point is that the forcing relation has to be designed carefully throughout the hierarchy of finite types, so that the corresponding transformation of proofs can be lifted from the level of *typing derivations* to the level of (raw) *Curry-style proof terms*, that contain much less information. In this way, we shall deduce a program transformation $t \mapsto t^*$ of classical λ-terms, that can be studied *per se* independently from the forcing translation, exactly the same way as CPS-translations can be studied independently from the negative translations they correspond to via Curry-Howard.

From a fine-grained analysis of the computational behavior of transformed (classical) proof terms in the Krivine Abstract Machine (KAM), we shall explain the computational model underlying the forcing translation, showing that this computational model is reminiscent from well-known techniques in computer architecture, such as *virtualization* and *protection rings*. Exploiting this analogy, we shall see how to put the forcing translation 'into the hardware' in order to avoid the cost of the program transformation. For that, we shall introduce a new abstract machine, the *Krivine Forcing Abstract Machine* (KFAM), that extends Krivine's machine with an alternative execution mode devoted to the evaluation of 'proofs by forcing', which is reminiscent from the *protected mode* of modern computer architectures. We shall also present the realizability models coming with this new abstract machine, together with the corresponding adequacy results.

This paper presents the global computational architecture underlying Cohen forcing, but it does not yet explain how forced axioms may benefit from this particular architecture in concrete examples. In a future paper, we plan to present case studies illustrating how this architecture works for some forcing structures.

Outline of the paper

This article is divided into two parts.

The first part (Sections 2–4) is a purely syntactic study of the classical program transformation underlying Cohen's forcing, considering this transformation from the point of view of typing in classical higher-order arithmetic.

The second part (Sections 5–7) is a semantic study of the forcing translation from the point of view of classical realizability (Kri03; Kri09; Kri10), by introducing a new abstract machine devoted to the execution of proofs by forcing.

Syntactic study In Section 2, we introduce an extension of (classical) higher-order arithmetic, called PAω^+, with a system of Curry-style proof terms enriched with a control operator call/cc to prove Peirce's law (Gri90). We study the basic meta-theoretic properties of this system, and present the operational semantics of proof terms in the framework of Krivine's Abstract Machine (KAM). In Section 3, we introduce the notion of a forcing structure in system PAω^+ (following (Kri10)) and relate it with the traditional set-theoretic presentation of forcing from a poset of conditions. Given an arbitrary forcing structure, we define in Section 4 the corresponding forcing translation $A \mapsto (p \Vdash A)$ (where A is a proposition and p a forcing condition) and study its basic properties. Then we introduce a program transformation $t \mapsto t^*$ on the raw (Curry-style) proof terms of system PAω^+, and we show that this transformation maps every proof term of a proposition A to a proof term of the proposition $p \Vdash A$. We conclude this section by checking that all first-order propositions are invariant under the forcing translation, and by analyzing the computational behavior of transformed programs.

Semantic study In Section 5, we present (a variant of) the general notion of a classical realizability algebra introduced in (Kri10). We show how to build a classical realizability model $\mathscr{M}_{\mathscr{A}}$ of system PAω^+ from an arbitrary algebra $\mathscr{A}$, and prove the general theorem of adequacy. From the computational analysis of the program transformation $t \mapsto t^*$ presented in Section 4, we show in Section 6 how to hard-wire the program transformation into the abstract machine. For that, we introduce a new abstract machine—the *Krivine Forcing Abstract Machine* (KFAM)—that extends the usual KAM with an alternative execution mode devoted to the execution of proofs by forcing. Then we present two results of adequacy: one for the regular execution mode, and one for the forcing execution mode. To prove the latter result of adequacy, we show in Section 7 that the two execution modes of the KFAM are semantically reflected by two different ways of constructing a classical realizability algebra from the KFAM, respectively written $\mathscr{A}$ (for the regular mode) and $\mathscr{A}^*$ (for the forcing mode). We conclude by relating the denotations of a given proposition A in the two realizability models $\mathscr{M}_{\mathscr{A}}$ and $\mathscr{M}_{\mathscr{A}^*}$ (induced by the algebras $\mathscr{A}$ and $\mathscr{A}^*$) in the spirit of the definition of iterated forcing.

Contributions of the paper

This work is largely inspired by the methodology introduced by Krivine in (Kri08; Kri10). The author's own contributions are the following:

— A reformulation of the forcing translation in higher-order arithmetic (rather than in PA2/PA3), and the design of an expressive type system (PAω^+) in which the transformation preserves typability on proof terms. (In (Kri08; Kri10), well-typed proof terms are only transformed into classical realizers.)

— Some simplifications in the program transformation presented by Krivine. In particular, we get rid of the extra two instructions χ and χ' used in (Kri08; Kri10) by putting the computational condition on the top of the stack rather than at the bottom, as done in (Kri10). The removal of these extra instructions also simplifies the expression of the relationship with iterated forcing, that was already given in (Kri10).

— A slightly more abstract (but equivalent) presentation of classical realizability algebras that does not rely on the combinators used in (Kri10), which makes the underlying computational architecture more transparent.

— Finally, the main contribution of the paper is the Krivine Forcing Abstract Machine (KFAM), a new abstract machine specifically devoted to the evaluation of proofs by forcing that is deduced from the computational analysis of the program transformation, as well as the corresponding realizability models.

This paper is an expanded version of (Miq11).

2. An extension of higher-order arithmetic

Throughout this paper, we work in a presentation of higher-order arithmetic called PAω^+, that is basically an extension of (Curry-style) system $F\omega$. As for system $F\omega$, system PAω^+ is stratified into three syntactic categories: *kinds*, *higher-order terms* (that correspond to mathematical objects, including propositions) and *(Curry-style) proof terms.*

2.1. *Kinds*

Kinds (notation: τ, σ, etc.) of system PAω^+ are simple types based on two ground kinds: the kind ι of *individuals* and the kind o of *propositions*. We prefer here the terminology of a kind since in system PAω^+, the role of types is played by propositions, that belong to the syntactic category of higher-order terms. Formally:

Definition 1 (Kinds). — Kinds are inductively defined from the following rules:

(1) The symbol ι is a kind.
(2) The symbol o is a kind.
(3) If τ and σ are kinds, then so is $\tau \to \sigma$.

As usual, we consider that the symbol $\to$ associates to the right, so that $\tau_1 \to \tau_2 \to \sigma$ denotes the kind $\tau_1 \to (\tau_2 \to \sigma)$. We shall sometimes use the vector notation $\vec{\tau}$ to denote finite lists of kinds, and given a list $\vec{\tau} \equiv \tau_1, \ldots, \tau_n$ and a kind σ, we write $\vec{\tau} \to \sigma$ for $\tau_1 \to \cdots \to \tau_n \to \sigma$. In what follows, we shall call a *ι-kind* (resp. an o-kind) any kind of the form $\vec{\tau} \to \iota$ (resp. any kind of the form $\vec{\tau} \to o$).

2.2. *Higher-order terms*

We assume given an infinite set of variables (notation: x^τ, y^τ, z^τ, etc.) for every kind τ. Higher-order terms (notation: M, N, etc.) of PAω^+ are 'simply kinded' λ-terms enriched with extra constructions to represent arithmetic operations and logical constructions. Every higher-order term M has a particular kind, which is unique. (There is no 'kind' ambiguity since each variable comes with its kind.) Formally:

Definition 2 (Higher-order terms). — Higher-order terms of all kinds are inductively defined from the following rules:

Lambda-calculus:

(1) If x^τ is a variable of kind τ, then x^τ is a term of kind τ.
(2) If x^τ is a variable of kind τ and if M is a term of kind σ, then $\lambda x^\tau . M$ is a term of kind $\tau \to \sigma$.
(3) If M is a term of kind $\tau \to \sigma$ and if N is a term of kind τ, then MN is a term of kind σ.

Arithmetic constructions:

(4) The constant 0 ('zero') is a term of kind ι.
(5) The constant s ('successor') is a term of kind $\iota \to \iota$.
(6) For every kind τ, the constant rec_τ ('recursor') is a term of kind $\tau \to (\iota \to \tau \to \tau) \to \iota \to \tau$.

Logical constructions:

(7) If M and N are terms of kind o, then $M \Rightarrow N$ is a term of kind o.
(8) If M is a term of kind o possibly depending on a variable x of kind τ, then $\forall x^\tau M$ is a term of kind o.
(9) If M and M' are terms of kind τ, and if N is a term of kind o, then $M = M' \mapsto N$ is a term of kind o. This ternary construction represents an *equational implication* whose meaning will be explained below.

The notions of free and bound variables are defined as usual, keeping in mind that the constructions $\lambda x^\tau . M$ and $\forall x^\tau M$ are binders, which bind all the free occurrences of the variable x^τ in the term M. In what follows, we shall write $FV(M)$ the set of free variables of M, and $M\{x^\tau := N\}$ the term obtained by replacing in the term M (of some kind σ) all the free occurrences of the variable x^τ with the term N (of kind τ), possibly renaming bound variables of M to prevent variable captures.

Propositions We call a *proposition* any term A of kind o, preferring the letters A, B, C, etc. to denote them. Propositions of system PAω^+ are ultimately built from the traditional constructions $A \Rightarrow B$ (implication) and $\forall x^\tau A$ (universal quantification ranging over the kind τ) of minimal higher-order logic.

In addition, system PAω^+ provides a ternary construction written $M = M' \mapsto A$ and called an *equational implication*, whose intuitive meaning is:

$$M = M' \mapsto A \qquad \equiv \qquad \begin{cases} A & \text{if } M \text{ equals } M' \\ \top & \text{otherwise} \end{cases}$$

where $\top$ denotes the proposition proved by any proof term, that will be formally defined in section 2.6. As suggested by its name, the equational implication $M = M' \mapsto A$ is provably equivalent to the 'regular' implication $M =_\tau M' \Rightarrow A$, where the symbol $=_\tau$ stands for Leibniz equality (see below). In practice, the proposition $M = M' \mapsto A$ carries over the same logical contents as the proposition $M =_\tau M' \Rightarrow A$, but with more compact proof terms. While this compact form of an implication is not strictly needed to define the forcing translation, it helps to make the translation more understandable at the level of proof terms. However, the presence of this extra construction has a cost on the type system of PAω^+, since it makes the typing judgment $\mathcal{E}; \Gamma \vdash t : A$ not only depend on a typing context Γ, but also on an equational theory $\mathcal{E}$ (see Section 2.4).

In system PAω^+, absurdity, conjunction, disjunction, existential quantification and Leibniz equality on all kinds are defined using the standard second-order encodings:

$$\begin{array}{rcl}
\bot & \equiv & \forall z^o\, z \\
\neg A & \equiv & A \Rightarrow \bot \\
A \wedge B & \equiv & \forall z^o\, ((A \Rightarrow B \Rightarrow z) \Rightarrow z) \\
A \vee B & \equiv & \forall z^o\, ((A \Rightarrow z) \Rightarrow (B \Rightarrow z) \Rightarrow z) \\
A \Leftrightarrow B & \equiv & (A \Rightarrow B) \wedge (B \Rightarrow A) \\
\exists x^\tau A(x) & \equiv & \forall z^o\, (\forall x^\tau\, (A(x) \Rightarrow z) \;\Rightarrow\; z) \\
M =_\tau N & \equiv & \forall z^{\tau \to o} (z\, M \Rightarrow z\, N)
\end{array}$$

(where z^o and $z^{\tau \to o}$ are fresh variables).

2.3. *System T as a fragment of* PAω^+

The purely computational fragment of system PAω^+—which we obtain by excluding all the constructions that deal with propositions while keeping all the constructions that deal with individuals and functions—is actually isomorphic to Gödel's system T.

In what follows, we shall thus use the terminology of a *T-kind* (resp. of a *T-term*) to refer to a kind (resp. to a term) that belongs to the fragment of system PAω^+ that corresponds to Gödel's system T. Formally, a T-kind is any kind τ that is either of the form $\tau \equiv \iota$ or of the form $\tau \equiv \tau_1 \to \tau_2$, where τ_1 and τ_2 are T-kinds; and a T-term is any higher-order term M that has one of the following forms:

— $M \equiv x^\tau$, where τ is a T-kind,
— $M \equiv \lambda x^\tau . M$, where τ is a T-kind, and where M is a T-term,
— $M \equiv MN$, where M and N are T-terms,
— M is one of the constants 0, s or rec_τ, where τ is a T-kind.

Let us recall that the functions that can be implemented in system T (as terms of kind $\iota^k \to \iota$) are exactly the recursive functions that are provably total in first-order arithmetic, which includes all primitive recursive functions as well as many other total recursive functions such as Ackermann's.

Although T-terms may be computationally higher-order, they are logically first-order, and they play in system PAω^+ the same role as the terms of Gödel's system T in the arithmetic of finite types. (Intuitively, T-terms of functional kinds do not represent all functions, but only computable ones.) In particular, we shall see in Section 4 that T-kinds and T-terms are not affected by the forcing translation.

2.4. *The congruence $M \cong_{\mathcal{E}} M'$*

Definition 3 (Equational theories). — We call an *equational theory* any finite set of equations written

$$\mathcal{E} \;\equiv\; M_1 = M_1', \ldots, M_k = M_k'$$

where for all $i \in [1..k]$, M_i and M_i' are (open) higher-order terms of the same kind τ_i. (For simplicity, we assume that the equations $(M_i = M_i') \in \mathcal{E}$ are non oriented.)

Given an equational theory and an equation $M = M'$, we simply write $\mathcal{E}, M = M'$ for $\mathcal{E} \cup \{M = M'\}$. The notations $FV(\mathcal{E})$ and $\mathcal{E}\{x^\tau := N\}$ are extended to equational theories $\mathcal{E}$ in the obvious way.

Every equational theory $\mathcal{E}$ induces a binary relation $M \cong_{\mathcal{E}} M'$ between higher-order terms M and M' of the same kind, that expresses that the terms M and M' are equal modulo the equational theory $\mathcal{E}$. Formally:

Definition 4 (The relation $\cong_{\mathcal{E}}$). — The family of relations $\cong_{\mathcal{E}}$ (where $\mathcal{E}$ ranges over all equational theories) is inductively defined from the rules given in Table 1.

In the particular case where $\mathcal{E}$ is empty, we simply write $M \cong M'$ for $M \cong_{\mathcal{E}} M'$.

Remarks 5. (1) Table 1 contains the expected inference rules expressing that the relation $M \cong_{\mathcal{E}} M'$ is a congruence (cf Prop. 6 below). Note that the congruence rule for equational implications provides a mechanism of discharge that permits to derive that two equational implications $M_1 = M_2 \mapsto A$ and $M_1 = M_2 \mapsto A'$ are congruent modulo some equational theory $\mathcal{E}$ as soon as the propositions A and A' are congruent modulo the theory $\mathcal{E}$ extended with the equation $M_1 = M_2$. In practice, this means that the deeper

Table 1. *Inference rules of the relation $M \cong_{\mathcal{E}} M'$*

Reflexivity, symmetry, transitivity and base case

$$\frac{}{M \cong_{\mathcal{E}} M} \qquad \frac{}{M \cong_{\mathcal{E}} M'}\ {\scriptstyle (M=M')\in\mathcal{E}} \qquad \frac{M \cong_{\mathcal{E}} M'}{M' \cong_{\mathcal{E}} M} \qquad \frac{M \cong_{\mathcal{E}} M' \quad M' \cong_{\mathcal{E}} M''}{M \cong_{\mathcal{E}} M''}$$

Context closure

$$\frac{M \cong_{\mathcal{E}} M'}{\lambda x^\tau . M \ \cong_{\mathcal{E}}\ \lambda x^\tau . M'} \qquad \frac{M \cong_{\mathcal{E}} M' \quad N \cong_{\mathcal{E}} N'}{MN \ \cong_{\mathcal{E}}\ M'N'} \qquad \frac{A \cong_{\mathcal{E}} A' \quad B \cong_{\mathcal{E}} B'}{A \Rightarrow B \ \cong_{\mathcal{E}}\ A' \Rightarrow B'}$$

$$\frac{A \cong_{\mathcal{E}} A'}{\forall x^\tau A \ \cong_{\mathcal{E}}\ \forall x^\tau A'} \qquad \frac{M_1 \cong_{\mathcal{E}} M_1' \quad M_2 \cong_{\mathcal{E}} M_2' \quad A \cong_{\mathcal{E}, M_1 = M_2} A'}{M_1 = M_2 \mapsto A \ \cong_{\mathcal{E}}\ M_1' = M_2' \mapsto A'}$$

$\beta\eta\iota$-conversion

$$\frac{}{(\lambda x^\tau . M)N \cong_{\mathcal{E}} M\{x^\tau := N\}} \qquad \frac{}{\lambda x^\tau . Mx \ \cong_{\mathcal{E}}\ M}\ {\scriptstyle x^\tau \notin FV(M)}$$

$$\frac{}{\mathrm{rec}_\tau\, M\, M'\, 0 \ \cong_{\mathcal{E}}\ M} \qquad \frac{}{\mathrm{rec}_\tau\, M\, M'\,(s\, N) \ \cong_{\mathcal{E}}\ M'\, N\,(\mathrm{rec}_\tau\, M\, M'\, N)}$$

Semantically equivalent propositions

$$\frac{}{\forall x^\tau \forall y^\sigma A \ \cong_{\mathcal{E}}\ \forall y^\sigma \forall x^\tau A} \qquad \frac{}{\forall x^\tau A \ \cong_{\mathcal{E}}\ A}\ {\scriptstyle x^\tau \notin FV(A)}$$

$$\frac{}{A \Rightarrow \forall x^\tau B \ \cong_{\mathcal{E}}\ \forall x^\tau (A \Rightarrow B)}\ {\scriptstyle x^\tau \notin FV(A)}$$

$$\frac{}{M = M \mapsto A \ \cong_{\mathcal{E}}\ A} \qquad \frac{}{M = M' \mapsto A \ \cong_{\mathcal{E}}\ M' = M \mapsto A}$$

$$\frac{}{M = M' \mapsto N = N' \mapsto A \ \cong_{\mathcal{E}}\ N = N' \mapsto M = M' \mapsto A}$$

$$\frac{}{A \Rightarrow M = M' \mapsto B \ \cong_{\mathcal{E}}\ M = M' \mapsto A \Rightarrow B}$$

$$\frac{}{\forall x^\tau (M = M' \mapsto A) \ \cong_{\mathcal{E}}\ M = M' \mapsto \forall x^\tau A}\ {\scriptstyle x^\tau \notin FV(M, M')}$$

we go through equational implications (in order to derive that two terms are congruent), the more equations we add to the current equational theory.

(2) In addition to the expected rules of $\beta\eta\iota$-conversion, Table 1 provides many rules to identify propositions that are semantically equivalent in a Curry-style framework. These rules basically express the commutation properties of universal quantification and of equational implication w.r.t. other logical constructions.

We easily check that:

Proposition 6 (Monotonicity, substitutivity and congruence).

(1) If $M \cong_{\mathcal{E}} M'$ and $\mathcal{E} \subseteq \mathcal{E}'$, then $M \cong_{\mathcal{E}'} M'$.
(2) If $M \cong_{\mathcal{E}} M'$ and $N \cong_{\mathcal{E}} N'$, then $M\{x^\tau := N\} \cong_{\mathcal{E}\{x^\tau := N\}} M'\{x^\tau := N'\}$ (where N and N' are arbitrary higher-order terms of kind τ).
(3) If the equational theory $\mathcal{E}$ is closed (i.e. $FV(M_1) = FV(M_2) = \varnothing$ for every equation $(M_1 = M_2) \in \mathcal{E}$), then the relation $M \cong_{\mathcal{E}} M'$ is a congruence.

Proof. (1) By a straightforward induction on the derivation of $M \cong_{\mathcal{E}} M'$.

(2) We first prove the result in the particular case where $M \equiv M'$, reasoning by induction on the structure of M. The general result then follows by induction on the derivation of $M \cong_{\mathcal{E}} M'$, using (1) in the case where the derivation of $M \cong_{\mathcal{E}} M'$ ends with the congruence rule for equational implications.

(3) We already know that the relation $M \cong_{\mathcal{E}} M'$ is an equivalence relation, and that it is substitutive when $\mathcal{E}$ is closed, by (2). It suffices to show that it is context closed. Let us treat the case of an equational implication: if $M_1 \cong_{\mathcal{E}} M_1'$, $M_2 \cong_{\mathcal{E}} M_2'$ and $A \cong_{\mathcal{E}} A'$, we have $A \cong_{\mathcal{E}, M_1 = M_2} A'$ from (1), so that $M_1 = M_2 \mapsto A \cong_{\mathcal{E}} M_1' = M_2' \mapsto A'$ from the congruence rule of equational implications (Table 1). The other cases are obvious. $\square$

2.5. *The proof system of* PAω^+

Proof terms (notation: t, u, etc.) of system PAω^+ are pure λ-terms enriched with a constant $\mathbf{c\!c}$ (call/cc, for *call with current continuation*). They are defined from an auxiliary set of *proof variables*, that we still write using the letters x, y, z, etc. (But these variables should not be confused with the higher-order variables x^τ, y^τ, z^τ, etc. introduced in Section 2.2.) Formally:

Definition 7 (Proof terms). — Proof terms are inductively defined from the rules:

(1) If x is a proof variable, then x is a proof term.
(2) If x is a proof variable and if t is a proof term, then $\lambda x\,.\,t$ is a proof term.
(3) If t and u are proof terms, then so is tu.
(4) The constant $\mathbf{c\!c}$ is a proof term.

The set of free variables of a proof term t is written $FV(t)$ and the corresponding operation of substitution, whose result is written $t\{x := u\}$, is defined as expected. Given proof terms $t_1, t_2, \ldots, t_n$, we introduce the shorthand:

$$t_1 \circ t_2 \circ \cdots \circ t_n \equiv \lambda z\,.\,t_1\,(t_2 \cdots (t_n\,z) \cdots)$$

(where z is a fresh proof variable).

Definition 8 (Typing contexts). — A *typing context*, or simply a *context* (notation: Γ, Γ', etc.), is a finite ordered list of the form $\Gamma \equiv x_1 : A_1,\ \ldots,\ x_n : A_n$, where $x_1, \ldots, x_n$ are pairwise distinct proof variables and where $A_1, \ldots, A_n$ are arbitrary propositions.

The *domain* of a context $\Gamma \equiv x_1 : A_1,\ \ldots,\ x_n : A_n$ is the set of proof variables defined by $\mathrm{dom}(\Gamma) = \{x_1; \ldots; x_n\}$. The notations $FV(\Gamma)$ and $\Gamma\{x^\tau := N\}$ (where N is a higher-order term of kind τ) are extended to typing contexts by letting:

- $FV(\Gamma) = FV(A_1) \cup \cdots \cup FV(A_n)$
- $\Gamma\{x^\tau := N\} \equiv x_1 : A_1\{x^\tau := N\}, \ldots, x_n : A_n\{x^\tau := N\}$

Given two contexts Γ and Γ', we write $\Gamma \subseteq \Gamma'$ when all declarations of Γ also appear in Γ' (not necessarily in the same order). Finally, the concatenation of two contexts Γ and Γ' such that $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \varnothing$ is still a context, which we write Γ, Γ'. The latter notation can be generalized to the case where Γ and Γ' are *compatible*, in the sense

Table 2. *Deduction/typing rules of system* PAω^+

(Axiom), (Conversion)	$\dfrac{}{\mathcal{E};\Gamma\vdash x : A}\ {\scriptstyle (x:A)\in\Gamma}$	$\dfrac{\mathcal{E};\Gamma\vdash t : A}{\mathcal{E};\Gamma\vdash t : A'}\ {\scriptstyle A\cong_{\mathcal{E}}A'}$
($\Rightarrow$-intro/elim)	$\dfrac{\mathcal{E};\Gamma, x : A\vdash t : B}{\mathcal{E};\Gamma\vdash \lambda x\,.\,t : A\Rightarrow B}$	$\dfrac{\mathcal{E};\Gamma\vdash t : A\Rightarrow B \qquad \mathcal{E};\Gamma\vdash u : A}{\mathcal{E};\Gamma\vdash tu : B}$
($\mapsto$-intro/elim)	$\dfrac{\mathcal{E}, M = M';\Gamma\vdash t : A}{\mathcal{E};\Gamma\vdash t : M = M'\mapsto A}$	$\dfrac{\mathcal{E};\Gamma\vdash t : M = M\mapsto A}{\mathcal{E};\Gamma\vdash t : A}$
($\forall$-intro/elim)	$\dfrac{\mathcal{E};\Gamma\vdash t : A}{\mathcal{E};\Gamma\vdash t : \forall x^{\tau} A}\ {\scriptstyle x^{\tau}\notin FV(\mathcal{E};\Gamma)}$	$\dfrac{\mathcal{E};\Gamma\vdash t : \forall x^{\tau} A}{\mathcal{E};\Gamma\vdash t : A\{x^{\tau} := N\}}$
(Peirce's law)	$\dfrac{}{\mathcal{E};\Gamma\vdash \text{œ} : ((A\Rightarrow B)\Rightarrow A)\Rightarrow A}$	

that $(x : A)\in\Gamma$ and $(x : A')\in\Gamma'$ implies $A\equiv A'$ (syntactic identity) for all x, A and A'. In this case, the concatenation Γ, Γ' is defined by removing duplicates (in Γ or in Γ').

The proof system of system PAω^+ is based on a typing judgment written $\mathcal{E};\Gamma\vdash t : A$, that expresses that the (raw) proof term t is actually a proof term of the proposition A (i.e. of kind o) in the context Γ and in the equational theory $\mathcal{E}$. Formally:

Definition 9 (Derivable judgments). — The class of derivable judgments $\mathcal{E};\Gamma\vdash t : A$ of system PAω^+ is inductively defined from the rules of Table 2.

Remarks 10. (1) The elimination rule of equational implication is actually a particular case of the conversion rule, since the proposition $M = M\mapsto A$ is congruent to A modulo any equational theory $\mathcal{E}$. In what follows, we shall thus not consider the elimination rule of equational implication as a primitive rule, but only as a derived rule.

(2) In Table 2, the only typing rules participating to the construction of the current proof term are the axiom rule, the introduction and elimination rules of implication as well as Peirce's law. The remaining typing rules (conversion, introduction and elimination of universal quantification, introduction of equational implication) do not affect the current proof term, so that we shall say that they are *computationally transparent.*

We shall conclude this introduction to the proof system of PAω^+ by presenting some typing rules that are admissible in this system:

Proposition 11 (Admissible rules). — The following rules are admissible in system PAω^+:

(Weakening)
$$\frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E}';\Gamma' \vdash t : A}\ {\scriptstyle \mathcal{E}\subseteq\mathcal{E}',\ \Gamma\subseteq\Gamma'}$$

(Term substitutivity)
$$\frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E}\{x^\tau := N\};\Gamma\{x^\tau := N\} \vdash t : A\{x^\tau := N\}}$$

(Proof substitutivity)
$$\frac{\mathcal{E};\Gamma, z : B \vdash t : A \qquad \mathcal{E}';\Gamma' \vdash u : B}{\mathcal{E},\mathcal{E}';\Gamma,\Gamma' \vdash t\{z := u\} : A}\ {\scriptstyle \Gamma \text{ and } \Gamma' \text{ compatible}}$$

Proof. The admissibility of the first two rules is proved by induction on the derivation of $\mathcal{E};\Gamma \vdash t : A$. The admissibility of the last rule is proved by induction on the derivation of $\mathcal{E};\Gamma, z : B \vdash t : A$, using the admissible rule of weakening in the case where the last inference is an axiom introducing the variable z. □

2.6. *Expressiveness*

From the rules of Table 2 one can derive the usual introduction and elimination rules of falsity, negation, conjunction, disjunction, existential quantification and Leibniz equality using the encodings given in the end of Section 2.2. The typing rule of $\mathsf{c\!c}$ implements Peirce's law, from which we can derive all the usual reasoning principles of classical logic such as the excluded middle:

$$\mathsf{c\!c}\,(\lambda k\,.\,\mathsf{right}\,(\lambda x\,.\,k\,(\mathsf{left}\,x))) \quad : \quad \forall X^o\,(X \vee \neg X)\,,$$

where:

$$\begin{array}{rcl} \mathsf{left} & \equiv & \lambda xfg\,.\,f\,x \quad : \quad \forall X^o\,\forall Y^o\,(X \Rightarrow X \vee Y) \\ \mathsf{right} & \equiv & \lambda yfg\,.\,g\,y \quad : \quad \forall X^o\,\forall Y^o\,(Y \Rightarrow X \vee Y) \end{array}$$

Equational implication and the propositional constant $\top$ For all terms M, M' (of kind τ) and A (of kind o), the propositions $M = M' \mapsto A$ and $M =_\tau M' \Rightarrow A$ are provably equivalent in PAω^+ as shown by the following proof term:

$$\langle \lambda xy\,.\,y\,x,\ \lambda x\,.\,x\,(\lambda y\,.\,y)\rangle \quad : \quad (M = M' \mapsto A) \ \Leftrightarrow\ (M =_\tau M' \Rightarrow A)$$

(using the abbreviation $\langle t_1, t_2\rangle \equiv \lambda z\,.\,z\,t_1\,t_2$). Moreover, we can define a proposition $\top$ representing the type of all proof terms by letting

$$\top \ \equiv\ \mathsf{tt} = \mathsf{ff} \mapsto \bot\,,$$

where $\mathsf{tt} \equiv \lambda x^o y^o\,.\,x$ and $\mathsf{ff} \equiv \lambda x^o y^o\,.\,y$. Writing $\mathcal{E} \equiv \mathsf{tt} = \mathsf{ff}$, we easily check that

$$A \ \cong_{\mathcal{E}}\ \mathsf{tt}\,A\,A' \ \cong_{\mathcal{E}}\ \mathsf{ff}\,A\,A' \ \cong_{\mathcal{E}}\ A'$$

for all propositions A and A', which means that all propositions A and A' are congruent modulo the equational theory $\mathcal{E}$. From this, we immediately deduce that:

Lemma 12. — For all equational theories $\mathcal{E}$, for all contexts Γ and for all proof terms t such that $FV(t) \subseteq \mathrm{dom}(\Gamma)$, the judgment $\mathcal{E};\Gamma \vdash t : \top$ is derivable.

Proof. From the introduction rule of equational implication, it suffices to check that the judgment $\mathcal{E}, \mathsf{tt} = \mathsf{ff}; \Gamma \vdash t : \bot$ is derivable. This is easily proved by induction on the structure of t, using dummy conversions of the form $A \cong_{\mathcal{E},\mathsf{tt}=\mathsf{ff}} \bot$ at each step. □

As a consequence, the proof system of PAω^+ enjoys no normalization property. Nevertheless, system PAω^+ is logically consistent as we shall see in Section 5.

Arithmetic reasoning The family of recursors rec_τ of system PAω^+ allows to implement the predecessor and nullity test functions

$$\begin{array}{rcl} \mathsf{pred}^{\iota\to\iota} & \equiv & \mathrm{rec}_\iota\, 0\, (\lambda x\_ \,.\, x) \\ \mathsf{null}^{\iota\to o} & \equiv & \mathrm{rec}_o\, (\bot \Rightarrow \bot)\, (\lambda\_ \,.\, \lambda\_ \,.\, \bot) \end{array}$$

from which one easily derives that the successor function is injective (Peano 3rd axiom) and non surjective (Peano 4th axiom) using appropriate conversions:

$$\begin{array}{rcl} \lambda z \,.\, z & : & \forall x^\iota\, \forall y^\iota\, (s\, x =_\iota s\, y \Rightarrow x =_\iota y) \\ \lambda z \,.\, z\, (\lambda y \,.\, y) & : & \forall x^\iota\, \neg(0 =_\iota s\, x) \end{array}$$

(Note that these axioms are derivable without restriction on the objects of kind ι.)

To reason by induction, we proceed as in (GLT89; Kri93) by considering the *relativization predicate* nat (of kind $\iota \to o$) given by

$$\mathrm{nat} \;\equiv\; \lambda x^\iota \,.\, \forall z^{\iota\to o}\, (z\, 0 \Rightarrow \forall y^\iota\, (z\, y \Rightarrow z\, (s\, y)) \Rightarrow z\, x)\,,$$

that intuitively defines the smallest class of individuals containing zero and closed under the successor function (i.e. Dedekind's numerals). Provided we relativize all the quantifications over the kind ι to the class of Dedekind numerals using the shorthands[†]

$$\begin{array}{rcl} \forall x^{\mathrm{nat}}\, A(x) & \equiv & \forall x^\iota\, (\mathrm{nat}\, x \Rightarrow A(x)) \\ \exists x^{\mathrm{nat}}\, A(x) & \equiv & \forall z^o\, (\forall x^\iota (\mathrm{nat}\, x \Rightarrow A(x) \Rightarrow z) \Rightarrow z) \end{array}$$

we can derive the induction principle in system PAω^+:

$$\forall z^{\iota\to o}\, [z\, 0 \Rightarrow \forall x^{\mathrm{nat}} (z\, x \Rightarrow z\, (s\, x)) \Rightarrow \forall x^{\mathrm{nat}} z\, x]\,.$$

(Proof term: $\lambda x f n \,.\, n\, \langle \overline{0}, x\rangle\, (\lambda p \,.\, p\, (\lambda x y \,.\, \langle \overline{s}\, x,\ f\, x\, y\rangle))\, (\lambda x y \,.\, y)$, where $\overline{0} \equiv \lambda x f \,.\, x$ and $\overline{s} \equiv \lambda n x f \,.\, f(n x f)$ are Church's encodings of zero and the successor function).

However, the systematic use of the relativized quantifications $\forall x^{\mathrm{nat}} A(x)$ and $\exists x^{\mathrm{nat}} A(x)$ (instead of the non relativized ones) requires to prove that the individual N that is substituted to the variable x^ι is a Dedekind numeral at each elimination step of $\forall x^{\mathrm{nat}} A(x)$ and at each introduction step of $\exists x^{\mathrm{nat}} A(x)$:

$$\frac{\vdash t : \forall x^{\mathrm{nat}}\, A(x) \qquad \vdash u : \mathrm{nat}\, N}{\vdash tu : A(N)} \qquad\qquad \frac{\vdash t : A(N) \qquad \vdash u : \mathrm{nat}\, N}{\vdash \lambda z \,.\, z\, u\, t : \exists x^{\mathrm{nat}} A(x)}$$

[†] Through the proofs as programs correspondence, the relativized quantifications $\forall x^{\mathrm{nat}}\, A(x)$ and $\exists x^{\mathrm{nat}}\, A(x)$ play the same role as the dependent product $\Pi x\!:\!\mathrm{nat}\,.\, A(x)$ and the dependent sum $\Sigma x\!:\!\mathrm{nat}\,.\, A(x)$ in type theory. The only difference is that in our framework, these constructions are not atomic anymore.

For that, we need to check that all the constructions of the syntax of higher-order terms preserve this property, provided we assume that all the variables from which these terms are built already fulfill the property. Technically, we proceed as follows: to every ι-kind τ we associate a relativization predicate rel_τ of kind $\tau \to o$ that is inductively defined by:

$$\begin{array}{rcl} \mathrm{rel}_\iota & \equiv & \mathrm{nat} \\ \mathrm{rel}_{\sigma\to\tau} & \equiv & \begin{cases} \lambda f^{\sigma\to\tau}\,.\,\forall x^\sigma\,(\mathrm{rel}_\sigma x \Rightarrow \mathrm{rel}_\tau\,(f\,x)) & \text{if } \sigma \text{ is a } \iota\text{-kind} \\ \lambda f^{\sigma\to\tau}\,.\,\forall x^\sigma\,(\mathrm{rel}_\tau\,(f\,x)) & \text{if } \sigma \text{ is an } o\text{-kind} \end{cases} \end{array}$$

(where τ is a ι-kind). Using these abbreviations, we easily prove that:

Proposition 13. — For all higher-order terms M whose kind σ is a ι-kind, there exists a proof term $\overline{M}$ of free variables $\overline{x}_1, \ldots, \overline{x}_n$ such that the judgment

$$\overline{x}_1 : \mathrm{rel}_{\tau_1}\, x_1^{\tau_1}; \ldots; \overline{x}_n : \mathrm{rel}_{\tau_n}\, x_n^{\tau_n} \vdash \overline{M} : \mathrm{rel}_\sigma\, M$$

is derivable in system PAω^+, where $x_1^{\tau_1}, \ldots, x_n^{\tau_n}$ are the free variables of M whose kinds $\tau_1, \ldots, \tau_n$ are ι-kinds. (The other free variables of M do not need to be considered.)

Proof. For convenience, we associate a proof variable $\overline{x}$ to every higher-order variable x^τ of a ι-kind τ. The proof term $\overline{M}$ is defined by induction on M as follows:

$$\begin{array}{rclrcll} \overline{(x^\tau)} & \equiv & \overline{x} & & & & \\ \overline{\lambda x^\sigma\,.\,M^\tau} & \equiv & \lambda\overline{x}\,.\,\overline{M} & \overline{M^{\sigma\to\tau}N^\sigma} & \equiv & \overline{M}\,\overline{N} & (\text{if } \sigma \text{ is a } \iota\text{-kind}) \\ \overline{\lambda x^\sigma\,.\,M^\tau} & \equiv & \overline{M} & \overline{M^{\sigma\to\tau}N^\sigma} & \equiv & \overline{M} & (\text{if } \sigma \text{ is an } o\text{-kind}) \\ \overline{0} & \equiv & \lambda xf\,.\,x & \overline{s} & \equiv & \lambda nxf\,.\,f(nxf) & \\ \overline{\mathrm{rec}_\tau} & \equiv & \multicolumn{5}{l}{\lambda xfn\,.\,n\,\langle\overline{0}, x\rangle\,(\lambda p\,.\,p\,(\lambda xy\,.\,\langle\overline{s}\,x,\ f\,x\,y\rangle))\,(\lambda xy\,.\,y)} \end{array}$$

(assuming that τ is a ι-kind in all the above equations). $\square$

2.7. *Operational semantics of proof terms*

Unlike intuitionistic proof terms, the classical proof terms we presented in Section 2.5 are not subject to β-reduction, but they are intended to be evaluated (in front of a stack) in Krivine's Abstract Machine (KAM). Classical proof terms of system PAω^+ actually form a subset of the terms of Krivine's λ_c-calculus (Kri09), and their operational semantics is naturally described in the framework of the λ_c-calculus we shall now recall.

Formally, the λ_c-calculus distinguishes two kinds of syntactic expressions: *terms*—that represent programs—and *stacks*—that represent evaluation contexts. The syntax of terms and stacks is parameterized by two sets of symbols:

— A countable set $\mathcal{K}$ of *instructions*, that contains at least the instruction $\mathsf{c\!c}$;
— A nonempty countable set Π_0 of *stack constants* (a.k.a. *stack bottoms*).

Terms (notation: t, u, etc.) of the λ_c-calculus are ordinary λ-terms enriched with constants of two forms: *instructions* $\kappa \in \mathcal{K}$, including the instruction $\mathsf{c\!c}$ ('call/cc'), and *continuation constants* k_π, one for every stack π. *Stacks* (notation: π, π', etc.) are finite lists of *closed* λ_c-terms terminated by a stack constant $\alpha \in \Pi_0$. Formally:

Definition 14 (Terms and stacks). — Terms and stacks of the λ_c-calculus are defined by mutual induction from the following seven formation rules:

(1) If x is a proof variable, then x is a λ_c-term, and $FV(x) = \{x\}$.
(2) If $\kappa \in \mathcal{K}$ is an instruction, then κ is a λ_c-term, and $FV(\kappa) = \varnothing$.
(3) If π is a stack, then k_π is a λ_c-term, and $FV(\mathsf{k}_\pi) = \varnothing$.
(4) If t and u are λ_c-terms, then tu is a λ_c-term, and $FV(tu) = FV(t) \cup FV(u)$.
(5) If x is a proof variable and if t is a λ_c-term, then $\lambda x \,.\, t$ is a λ_c-term, and $FV(\lambda x \,.\, t) = FV(t) \setminus \{x\}$.
(6) If $\alpha \in \Pi_0$ is a stack constant, then α is a stack.
(7) If t is a closed λ_c-term (i.e. a term such that $FV(t) = \varnothing$) and if π is a stack, then $t \cdot \pi$ is a stack.

The set of closed terms (resp. the set of stacks) is denoted by Λ (resp. by Π).

In this definition, we introduce every λ_c-term with its set of free variables $FV(t)$ so that we can restrict the application of rule (7) to closed terms. As a consequence, stacks only contain closed terms and can thus be seen as closed objects themselves (so that each continuation constant k_π really deserves the name of a constant). It is immediate from the above definition that raw proof terms of system PAω^+ (Def. 7) constitute a strict subset of the set of open λ_c-terms, namely: the λ_c-terms that contain no continuation constant and no instruction but the control operator $\text{œ} \in \mathcal{K}$.

Evaluation In the λ_c-calculus, terms and stacks are computationally inert when taken separately, and computation only occurs through their interaction within processes:

Definition 15 (Processes). — A *process* is a pair formed by a closed λ_c-term t with a stack π, which is written $t \star \pi$. The set of all processes is written $\Lambda \star \Pi$ (which is just another notation for the Cartesian product $\Lambda \times \Pi$).

In the λ_c-calculus, computation is described by the means of a binary relation of *evaluation* between processes. This binary relation, which is written $p \succ p'$, is not defined but axiomatized as follows:

Definition 16 (Evaluation). — A relation of *evaluation* is any preorder $p \succ p'$ over the set of processes that fulfils the following axioms:

(Push)	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
(Grab)	$\lambda x \,.\, t \star u \cdot \pi$	$\succ$	$t\{x := u\} \star \pi$
(Save)	$\text{œ} \star t \cdot \pi$	$\succ$	$t \star \mathsf{k}_\pi \cdot \pi$
(Restore)	$\mathsf{k}_\pi \star t \cdot \pi'$	$\succ$	$t \star \pi$

In Krivine's Abstract Machine, the usual β-reduction rule is decomposed into two rules, called (Push) and (Grab). Note that these rules do not perform full β-reduction, but only weak-head β-reduction. (Redexes in the stack or below abstractions are never reduced until they come into head position.) Control is achieved via two rules (Save) and (Restore), that respectively describe the creation of a continuation constant (using the instruction œ) and its destruction.

Let us insist on the fact that we do not assume (in general) that the preorder $p \succ p'$ is generated from the four rules (Push), (Grab), (Save) and (Restore). Instead, the preorder $\succ$ may be generated using additional rules—typically, evaluation rules describing the computational behavior of extra instructions $\kappa \in \mathcal{K}$, such as 'quote', the 'clock' (Kri03), or the two instructions χ and χ' described in (Kri10). Formally, the relation of evaluation $\succ$ is thus another parameter of the definition of the calculus, just like the sets $\mathcal{K}$ and Π_0.

2.8. *Evaluation and typing*

The intuition behind the λ_c-calculus is that every classical proof term (and more generally: every realizer) of a proposition A is intended to be evaluated in front of a stack arguing against the proposition A. In this framework, a process $t \star \pi$ can be seen as a particular state of the discussion about the truth of a proposition A, the term t arguing for it while the stack π argues against it.

For instance, a proof term (or a realizer) t of the proposition $A \Rightarrow B$ is intended to be executed in front of a stack arguing against the implication $A \Rightarrow B$, that is, a stack of the form $u \cdot \pi$, where u argues for A while π argues against B. In the case where t is a λ-abstraction $\lambda x\,.\,t_0$, we can understand the evaluation rule

$$(\textsc{Grab}) \qquad \lambda x\,.\,t_0 \;\star\; u \cdot \pi \quad \succ \quad t_0\{x := u\} \;\star\; \pi$$

as the evolution of the discussion where the abstraction $\lambda x\,.\,t_0$ accepts the argument u defending A to produce an argument $t_0\{x := u\}$ defending B that will be evaluated against the stack π arguing against B. Note that although the proposition under discussion may change at each evaluation step, the two components of the process always agree on the currently discussed proposition at any time during evaluation.

It is possible to formalize these intuitions by extending the type system presented in Section 2.5 to stacks and processes of the λ_c-calculus as follows. First, we extend the definition of typing contexts by introducing new declarations of the form $\alpha : A^{\perp}$, where α is a stack constant (i.e. an element of Π_0) and where A is a proposition:

$$\textbf{Contexts} \qquad \Gamma \quad ::= \quad \cdots \quad | \quad \Gamma, \alpha : A^{\perp}$$

Here, the superscript $\perp$ is not an extra type former, but only a mark (belonging to the declaration itself) intended to recall that the stack constant α argues *against* A. Then we add two new typing judgments, namely:

— A typing judgment for stacks, written $\Gamma \vdash \pi : A^{\perp}$.
(Again, the superscript $\perp$ belongs to the judgment, not to the type.)

— A typing judgment for processes, written $\Gamma \vdash t \star \pi : \perp$.
(Using the symbol $\perp$ to denote the only possible type for processes.)

In this (extended) framework, we can introduce extra typing rules (such as given in Table 3) to type all the basic constructions of the λ_c-calculus, including continuation constants k_π in the syntactic category of terms. Thanks to this, we can type the four basic evaluation rules as follows, putting on the right-hand side of each line the formula

Table 3. *Typing stacks, continuations and processes*

$$\frac{}{\mathcal{E};\Gamma\vdash\alpha : A^{\perp}}\ {\scriptstyle(\alpha:A^{\perp})\in\Gamma}\qquad\qquad\frac{\mathcal{E};\Gamma\vdash t : A\qquad \mathcal{E};\Gamma\vdash\pi : B^{\perp}}{\Gamma\vdash t\cdot\pi : (A\Rightarrow B)^{\perp}}$$

$$\frac{\mathcal{E};\Gamma\vdash\pi : (A\{x^{\tau} := N\})^{\perp}}{\mathcal{E};\Gamma\vdash\pi : (\forall x^{\tau} A)^{\perp}}\ {\scriptstyle N\text{ of kind }\tau}\qquad\qquad\frac{\mathcal{E};\Gamma\vdash\pi : A^{\perp}}{\Gamma\vdash\pi : A'^{\perp}}\ {\scriptstyle A\cong_{\mathcal{E}}A'}$$

$$\frac{\mathcal{E};\Gamma\vdash\pi : A^{\perp}}{\mathcal{E};\Gamma\vdash\mathsf{k}_{\pi} : A\Rightarrow B}\qquad\qquad\frac{\mathcal{E};\Gamma\vdash t : A\qquad \mathcal{E};\Gamma\vdash\pi : A^{\perp}}{\mathcal{E};\Gamma\vdash t\star u : \perp}$$

that is currently debated:

$$\begin{array}{llcll}
(\textsc{Push}) & t^{A\Rightarrow B}u^{A} & \star & \pi^{B^{\perp}} & [B] \\
\succ & t^{A\Rightarrow B} & \star & u^{A}\cdot\pi^{B^{\perp}} & [A\Rightarrow B] \\[1ex]
(\textsc{Grab}) & \lambda x^{A}\,.\,t^{B} & \star & u^{A}\cdot\pi^{B^{\perp}} & [A\Rightarrow B] \\
\succ & t\{x^{A} := u^{A}\}^{B} & \star & \pi^{B^{\perp}} & [B] \\[1ex]
(\textsc{Save}) & \mathsf{cc} & \star & t^{(A\Rightarrow B)\Rightarrow A}\cdot\pi^{A^{\perp}} & [((A\Rightarrow B)\Rightarrow A)\Rightarrow A] \\
\succ & t^{(A\Rightarrow B)\Rightarrow A} & \star & (\mathsf{k}_{\pi})^{A\Rightarrow B}\cdot\pi^{A^{\perp}} & [(A\Rightarrow B)\Rightarrow A] \\[1ex]
(\textsc{Restore}) & (\mathsf{k}_{\pi})^{A\Rightarrow B} & \star & t^{A}\cdot\pi'^{B^{\perp}} & [A\Rightarrow B] \\
\succ & t^{A} & \star & \pi^{A^{\perp}} & [A]
\end{array}$$

Also notice that in this extended type system, the only objects that can be given a type in the empty context are the closed (and well-typed) proof terms that contain no continuation constant k_{π}. (The reason being that stacks, processes and continuation constants all involve at least a stack constant that has to be declared in the context.)

We shall not study further this extension of the type system (whose purpose is mainly pedagogical), but the reader is invited to check that the result of adequacy we shall prove in Section 5.7 holds for the extended system too, provided we interpret the two extra judgments $\mathcal{E};\Gamma\vdash\pi : A^{\perp}$ and $\mathcal{E};\Gamma\vdash t\star\pi : \perp$ the obvious way.

3. Representing forcing conditions

Let us recall that in set theory, the forcing translation is parameterized by a set of *forcing conditions* which is traditionally given as a poset $(C,\leq)$ with a largest element written 1 (Coh63; Coh64; Bel85; Jec02). In the framework of system PAω^{+}, we shall follow Krivine (Kri08; Kri10) by representing forcing conditions (intuitively) as the elements of an upwards closed subset C of a meet semi-lattice $(\kappa,\cdot,1)$, where κ is a fixed kind. This slightly non standard presentation of forcing conditions will be justified by the computational analysis of the underlying program transformation we will present in section 4.7, and we shall give an example of such a structure in section 3.3.

3.1. *Forcing structures*

The parameters defining a notion of forcing in system PAω^+ are collected in what we call a *forcing structure*:

Definition 17 (Forcing structures). — A *forcing structure* is given by:

- A kind κ, whose elements are called *forcing conditions*, or simply *conditions*. In what follows, we shall use the letters p, q, r, etc. to denote conditions.
- A closed predicate C of kind $\kappa \to o$ delimiting *well formed conditions*. Given a condition p, the proposition $\mathsf{C}\,p$ is also written $\mathsf{C}[p]$ ('p is well formed').
- A closed term $(\cdot)$ of kind $\kappa \to \kappa \to \kappa$ defining a binary operation of *product*. In what follows, the product of two conditions p and q is simply written pq rather than $p \cdot q$. (But this notation should not be confused with application.)
- A closed term 1 of kind κ representing the largest condition.
- Nine closed proof terms α_*, α_1, α_2, α_3, α_4, α_5, α_6, α_7 and α_8 such that:

$$\begin{array}{lll}
\alpha_* & : & \mathsf{C}[1] \\
\alpha_1 & : & \forall p^\kappa\, \forall q^\kappa\, (\mathsf{C}[pq] \Rightarrow \mathsf{C}[p]) \\
\alpha_2 & : & \forall p^\kappa\, \forall q^\kappa\, (\mathsf{C}[pq] \Rightarrow \mathsf{C}[q]) \\
\alpha_3 & : & \forall p^\kappa\, \forall q^\kappa\, (\mathsf{C}[pq] \Rightarrow \mathsf{C}[qp]) \\
\alpha_4 & : & \forall p^\kappa\, (\mathsf{C}[p] \Rightarrow \mathsf{C}[pp])
\end{array}
\qquad
\begin{array}{lll}
\alpha_5 & : & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[(pq)r] \Rightarrow \mathsf{C}[p(qr)]) \\
\alpha_6 & : & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[(pq)r]) \\
\alpha_7 & : & \forall p^\kappa\, (\mathsf{C}[p] \Rightarrow \mathsf{C}[p1]) \\
\alpha_8 & : & \forall p^\kappa\, (\mathsf{C}[p] \Rightarrow \mathsf{C}[1p])
\end{array}$$

Remarks 18. (1) The closed proof terms $\alpha_*, \alpha_1, \ldots, \alpha_8$—that are also called *forcing combinators*—represent the axioms that must be fulfilled by the forcing structure. Intuitively, these axioms express that the set C is upwards closed w.r.t. the ordering induced by the product $(p, q) \mapsto pq$ (seen as a *meet* operation). However, this ordering is not a parameter of the forcing structure itself, but we shall see (Section 3.2) how to reconstruct it from the forcing parameters $\mathsf{C}, 1, \cdot$, etc.

(2) The set of combinators $\alpha_*, \alpha_1, \ldots, \alpha_8$ is not minimal: α_2 can be defined from α_1 and α_3 by letting $\alpha_2 \equiv \alpha_1 \circ \alpha_3$ (and vice-versa); α_8 can be defined from α_7 and α_3 by letting $\alpha_8 \equiv \alpha_7 \circ \alpha_3$ (and vice-versa); and α_6 can be defined from α_5 and α_3 by letting $\alpha_6 \equiv \alpha_3 \circ \alpha_5 \circ \alpha_3 \circ \alpha_5 \circ \alpha_3$ (and vice-versa).

Given two conditions p and q, we say that p and q are *compatible* when $\mathsf{C}[pq]$ holds. The proposition $\mathsf{C}[pq]$ ('p and q are compatible') obviously implies that both conditions p and q are well formed (from axioms α_1 and α_2), but the converse is not true in general. (This point is crucial in the definition of forcing.)

In what follows, we shall also need the following derived combinators:

$$\begin{array}{llll}
\alpha_9 & \equiv & \alpha_3 \circ \alpha_1 \circ \alpha_6 \circ \alpha_3 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[(pq)r] \Rightarrow \mathsf{C}[pr]) \\
\alpha_{10} & \equiv & \alpha_2 \circ \alpha_5 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[(pq)r] \Rightarrow \mathsf{C}[qr]) \\
\alpha_{11} & \equiv & \alpha_9 \circ \alpha_4 & : \forall p^\kappa\, \forall q^\kappa\, (\mathsf{C}[pq] \Rightarrow \mathsf{C}[p(pq)]) \\
\alpha_{12} & \equiv & \alpha_5 \circ \alpha_3 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[q(rp)]) \\
\alpha_{13} & \equiv & \alpha_3 \circ \alpha_{12} & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[(rp)q]) \\
\alpha_{14} & \equiv & \alpha_5 \circ \alpha_3 \circ \alpha_{10} \circ \alpha_4 \circ \alpha_2 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[q(rr)]) \\
\alpha_{15} & \equiv & \alpha_9 \circ \alpha_3 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[qp]) \\
\alpha_{16} & \equiv & \alpha_6 \circ \alpha_{10} \circ \alpha_5 & : \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, \forall s^\kappa\, (\mathsf{C}[((pq)r)s] \Rightarrow \mathsf{C}[(qr)s])
\end{array}$$

3.2. *Preorder on conditions*

Throughout this paper, we work with a fixed forcing structure $(\kappa, \mathsf{C}, \cdot, 1, \alpha_*, \alpha_1, \dots, \alpha_8)$, freely using the derived combinators α_9–α_{16}. The relation of (pre)ordering $p \le q$ between two conditions p and q is defined by

$$p \le q \;\equiv\; \forall r^\kappa\ (\mathsf{C}[pr] \Rightarrow \mathsf{C}[qr])\,.$$

It is easy to check that:

— The binary relation $\le$ is a preorder with largest element 1:

$$\begin{array}{lcl} \lambda c\,.\,c & : & \forall p^\kappa\ (p \le p) \\ \lambda xy\,.\,(y \circ x) & : & \forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (p \le q \Rightarrow q \le r \Rightarrow p \le r) \\ \alpha_8 \circ \alpha_2 & : & \forall p^\kappa\ (p \le 1) \end{array}$$

The equivalence induced by the preorder $p \le q$ is written $p \approx q$ and defined by

$$p \approx q \quad\equiv\quad \forall r^\kappa\,(\mathsf{C}[pr] \Leftrightarrow \mathsf{C}[qr]) \quad (\Leftrightarrow \quad p \le q \wedge q \le p)$$

— Non well formed conditions are smallest elements:

$$\lambda x\,.\,(x \circ \alpha_1) \quad : \quad \forall p^\kappa\ \forall q^\kappa\ (\neg\mathsf{C}[p] \Rightarrow p \le q)$$

A consequence of this property is that all the non well formed conditions are actually identified via the equivalence relation $p \approx q$:

$$\lambda xy\,.\,\langle x \circ \alpha_1, y \circ \alpha_1\rangle \quad : \quad \forall p^\kappa\,\forall q^\kappa\,(\neg\mathsf{C}[p] \Rightarrow \neg\mathsf{C}[q] \Rightarrow p \approx q)\,.$$

— The product of two conditions is their greatest lower bound:

$$\begin{array}{rcl} \alpha_9 & : & \forall p^\kappa\ \forall q^\kappa\ (pq \le p) \\ \alpha_{10} & : & \forall p^\kappa\ \forall q^\kappa\ (pq \le q) \\ \lambda xy\,.\,(\alpha_{13} \circ y \circ \alpha_{12} \circ x \circ \alpha_{11}) & : & \forall p^\kappa\ \forall q^\kappa\ \forall r^\kappa\ (r \le p \Rightarrow r \le q \Rightarrow r \le pq) \end{array}$$

3.3. *An example of a forcing structure*

The typical—and historical—example of a set of forcing conditions (in set theory) is the set of all finite functions from a given set X to the pair $\{0;1\}$. When X is taken large enough, the set of all finite functions from X to $\{0;1\}$ can be used to force the negation of the continuum hypothesis (Coh63; Coh64; Bel85; Jec02).

In system PAω^+, a *set* is naturally described as a triple $(\tau, X, =_X)$ that is formed by:

- a kind τ, together with
- a relativization predicate X (of kind $\tau \to o$), and
- an equivalence relation $=_X$ (of kind $\tau \to \tau \to o$).

For instance, the set of Booleans $\{0;1\}$ is represented by the triple $(\iota, \mathsf{bool}, =_\iota)$, where ι is the kind of individuals, where bool is the predicate $\lambda x^\iota\,.\,\forall z^{\iota\to o}\,(z\,0 \Rightarrow z\,1 \Rightarrow z\,x)$, and where $=_\iota$ stands for Leibniz equality over individuals.

Given a fixed set $X = (\tau, X, =_X)$, the set of all finite functions from X to $\{0;1\}$ can be turned into a forcing structure $(\kappa, \mathsf{C}, \cdot, 1, \alpha_*, \alpha_1, \dots, \alpha_8)$ by letting:

- $\kappa \equiv \tau \to \iota \to o$ (binary relations $\subseteq \tau \times \iota$)

- $\mathsf{C} \equiv \lambda p^\kappa . \mathsf{C}[p]$, where

$$\begin{array}{rcll} \mathsf{C}[p] & \equiv & \forall x^\tau \, \forall y^\iota \, (p\,x\,y \Rightarrow X\,x \wedge \mathsf{bool}\,y) \quad \wedge & (p \subseteq X \times \{0;1\}) \\ & & \forall x_1^\tau \, \forall x_2^\tau \, \forall y^\iota \, (x_1 =_X x_2 \Rightarrow p\,x_1\,y \Rightarrow p\,x_2\,y) \quad \wedge & (p \text{ compatible}) \\ & & \forall x^\tau \, \forall y_1^\iota \, \forall y_2^\iota \, (p\,x\,y_1 \Rightarrow p\,x\,y_2 \Rightarrow y_1 =_\iota y_2) \quad \wedge & (p \text{ functional}) \\ & & \forall z^{\kappa \to o} \, \big[\forall q^\kappa \, (\mathrm{empty}(q) \Rightarrow z\,q) \ \Rightarrow & (p \text{ finite}) \\ & & \qquad \forall q^\kappa \, \forall q'^\kappa \, (\mathrm{succ}(q,q') \Rightarrow z\,q \Rightarrow z\,q') \ \Rightarrow \ z\,p\big] & \end{array}$$

using the shorthands $\mathrm{empty}(q) \equiv \forall x^\tau \, \forall y^\iota \neg q\,x\,y$ and $\mathrm{succ}(q,q') \equiv \exists x_0^\tau \, \exists y_0^\iota \, \big[\neg q\,x_0\,y_0 \wedge \forall x^\tau \, \forall y^\iota \, (q'\,x\,y \Leftrightarrow q\,x\,y \vee (x =_X x_0 \wedge y =_\iota y_0))\big]$.
- $pq \equiv \lambda x^\tau . \lambda y^\iota . p\,x\,y \vee q\,x\,y$ (union of p and q)
- $\mathsf{1} \equiv \lambda x^\tau . \lambda y^\iota . \bot$ (empty relation)

Once the parameters κ, C, $\cdot$ and $\mathsf{1}$ have been defined, it is a standard exercise of formalization to build closed proof terms $\alpha_*, \alpha_1, \ldots, \alpha_8$ expressing the desired axioms:

— α_*: The empty relation $\varnothing$ is a finite function from X to $\{0;1\}$;
— α_1: If $p \cup q$ is a finite function from X to $\{0;1\}$, then so is p:
— α_2: If $p \cup q$ is a finite function from X to $\{0;1\}$, then so is q;
— α_3: If $p \cup q$ is a finite function from X to $\{0;1\}$, then so is $q \cup p$;
— α_4: If p is a finite function from X to $\{0;1\}$, then so is $p \cup p$;
— α_5: If $(p \cup q) \cup r$ is a finite function from X to $\{0;1\}$, then so is $p \cup (q \cup r)$;
— α_6: If $p \cup (q \cup r)$ is a finite function from X to $\{0;1\}$, then so is $(p \cup q) \cup r$;
— α_7: If p is a finite function from X to $\{0;1\}$, then so is $p \cup \varnothing$;
— α_8: If p is a finite function from X to $\{0;1\}$, then so is $\varnothing \cup p$.

It is worth to notice that in this example, the ordering $p \leq q$ can be characterized by

$$p \leq q \quad \Leftrightarrow \quad \mathsf{C}[p] \Rightarrow (\mathsf{C}[q] \wedge q \subseteq p)$$

(writing $q \subseteq p$ for $\forall x^\tau \, \forall y^\iota \, (q\,x\,y \Rightarrow p\,x\,y)$), which means that for any two well formed conditions p and q, the ordering $p \leq q$ defined in section 3.2 coincides with the reverse inclusion $p \supseteq q$ of finite functions:

$$\mathsf{C}[p] \Rightarrow \mathsf{C}[q] \Rightarrow (p \leq q \Leftrightarrow p \supseteq q)\,.$$

Of course, the above equivalence only holds for well formed conditions, since non well formed conditions are all identified via the equivalence relation $p \approx q$.

4. The forcing translation

From the forcing structure $(\kappa, \mathsf{C}, \cdot, \mathsf{1}, \alpha_*, \alpha_1, \ldots, \alpha_8)$ introduced in Section 3, we shall now define the forcing translation $A \mapsto (p \Vdash A)$ on propositions together with the corresponding program transformation $t \mapsto t^*$ on proof terms. Some care has to be taken when defining the proposition $p \Vdash A$ in PAω^+, since we do not only want to define the corresponding proof transformation at the level of *derivations*, but at the level of *proof terms*, that contain much less information. In practice, this means that the proposition $p \Vdash A$ has to be defined in such a way that all the computationally transparent deduction steps in the proof of A remain computationally transparent in the proof of $p \Vdash A$.

For that we shall proceed in two steps. First, we shall define an auxiliary translation $M \mapsto M^*$ over all the higher-order terms, through which a proposition A will be translated into an arbitrary set A^* of forcing conditions (i.e. a term A^* of kind $\kappa \to o$). Then we shall define the forcing relation $p \Vdash A$ from the set of conditions A^* in such a way that the set of all conditions p such that $p \Vdash A$ is an element of the complete Boolean algebra generated by the set of forcing conditions (Bel85). (See Section 4.2.)

4.1. *The auxiliary translation $M \mapsto M^*$*

We first define a translation of kinds:

Definition 19 (Translation of kinds). — To every kind τ of system PAω^+, we associate a kind τ^* that is defined by the equations

$$\iota^* \equiv \iota, \qquad o^* \equiv \kappa \to o, \qquad (\tau \to \sigma)^* \equiv \tau^* \to \sigma^* .$$

From this definition, we can already see that the forcing translation will essentially affect propositions (i.e. terms of kind o), that will be interpreted as sets of conditions (i.e. terms of kind $\kappa \to o$), while leaving individuals (of kind ι) unchanged. In particular, we have $\tau^* \equiv \tau$ if and only if τ is a T-kind (cf Section 2.3).

The translation of higher-order terms is parameterized by a function mapping every higher-order variable x^τ of kind τ to another higher-order variable of kind τ^* which we write x^{τ^*}, or more simply x^*. In the particular case where $\tau^* \equiv \tau$ (that is: when τ is a T-kind), we shall even assume that $x^{\tau^*} \equiv x^\tau$.

Definition 20 (Translation of higher-order terms). — Every higher-order term M of kind τ is translated into a higher-order term M^* of kind τ^* by letting:

$$\begin{array}{rclrcl}
(x^\tau)^* & \equiv & x^{\tau^*} & 0^* & \equiv & 0 \\
(\lambda x^\tau . M)^* & \equiv & \lambda x^{\tau^*} . M^* & s^* & \equiv & s \\
(MN)^* & \equiv & M^* N^* & (\mathrm{rec}_\tau)^* & \equiv & \mathrm{rec}_{\tau^*} \\
(M_1 = M_2 \mapsto A)^* & \equiv & \lambda r^\kappa . M_1^* = M_2^* \mapsto A^* \, r & (\forall x^\tau A)^* & \equiv & \lambda r^\kappa . \forall x^{\tau^*} A^* r \\
(A \Rightarrow B)^* & \equiv & \multicolumn{4}{l}{\lambda r^\kappa . \forall q^\kappa \, \forall r'^\kappa \, (r = qr' \mapsto (\forall s^\kappa \, (\mathsf{C}[qs] \Rightarrow A^* s)) \Rightarrow B^* r')}
\end{array}$$

(where q, r, r' and s are fresh condition variables).

The translation $M \mapsto M^*$ immediately extends to equational theories componentwise, by letting $\mathcal{E}^* \equiv M_1^* = M'^*_1, \ldots, M_k^* = M'^*_k$, where $\mathcal{E} \equiv M_1 = M'_1, \ldots, M_k = M'_k$.

Remarks 21. From the above definition, we can see that:

(1) The translation $M \mapsto M^*$ simply propagates through abstractions and applications, and it is trivial on arithmetic constructions. In particular, we have $M^* \equiv M$ if and only if M is a T-term (cf Section 2.3).
(2) Universal quantifications and equational implications—whose introduction and elimination rules are computationally transparent in system PAω^+—are translated as

sets of conditions in a quite obvious way, so that for all conditions r we have

$$\begin{array}{rcl}(\forall x^\tau\, A)^*\, r & \cong & \forall x^{\tau^*}\, (A^*\, r)\\ (M_1 = M_2 \mapsto A)^*\, r & \cong & M_1^* = M_2^* \mapsto (A^*\, r)\end{array}$$

(3) All the complexity of the translation actually lies in implication, which is the only connective whose deduction rules have a real computational contents:

$$(A \Rightarrow B)^*\, r \quad \equiv \quad \forall q^\kappa\, \forall r'^\kappa\, (r = qr' \mapsto (\forall s^\kappa\, (\mathsf{C}[qs] \Rightarrow A^* s)) \Rightarrow B^* r')\,.$$

The meaning of this definition will be explained in Section 4.3.

Let us now establish the basic properties of the auxiliary translation $M \mapsto M^*$:

Lemma 22 (Substitutivity and compatibility with the congruence $M \cong_{\mathcal{E}} M'$).

(1) $(M\{x^\tau := N\})^* \;\equiv\; M^*\{x^{\tau^*} := N^*\}$.
(2) If $M \cong_{\mathcal{E}} M'$, then $M^* \cong_{\mathcal{E}^*} M'^*$

Proof. (1) This property, which basically follows from the fact that variables of kind τ are transformed into variables of kind τ^*, is proved by a straightforward induction on M.

(2) This property is proved by induction on the derivation of $M \cong_{\mathcal{E}} M'$, distinguishing cases according to the last applied rule. We only treat the rules identifying semantically equivalent propositions (Table 1 p. 1) in the particular case where $\mathcal{E} \equiv \varnothing$, the general case following by monotonicity (Prop. 6 item (1)):

— Commutation $\forall/\forall$:

$$\begin{array}{rcll}(\forall x^\tau\, \forall y^\sigma\, A)^* & \equiv & \lambda r^\kappa\,.\,\forall x^{\tau^*}\,(\lambda r'^\kappa\,.\,\forall y^{\sigma^*}\, A^*\, r')\, r & \\ & \cong & \lambda r^\kappa\,.\,\forall x^{\tau^*}\,\forall y^{\sigma^*}\, A^*\, r & (\beta)\\ & \cong & \lambda r^\kappa\,.\,\forall y^{\sigma^*}\,\forall x^{\tau^*}\, A^*\, r & (\forall\text{-commut.})\\ & \cong & (\forall y^\sigma\, \forall x^\tau\, A)^* & (\beta)\end{array}$$

— Simplification of $\forall$ in the case where $x^\tau \notin FV(A)$:

$$\begin{array}{rcll}(\forall x^\tau\, A)^* & \equiv & \lambda r^\kappa\,.\,\forall x^{\tau^*}\, A^*\, r & \\ & \cong & \lambda r^\kappa\,.\, A^*\, r & (\text{since } x^{\tau^*} \notin FV(A^*))\\ & \cong & A^* & (\eta)\end{array}$$

— Commutation $\Rightarrow/\forall$ in the case where $x^\tau \notin FV(A)$:

$$\begin{array}{ll}(A \Rightarrow \forall x^\tau\, B)^* & \\ \cong\;\; \lambda r^\kappa\,.\,\forall q^\kappa\,\forall r'^\kappa\,(r = qr' \mapsto (\forall s^\kappa\,(\mathsf{C}[qs] \Rightarrow A^* s)) \Rightarrow \forall x^{\tau^*}\, B^*\, r') & (\beta)\\ \cong\;\; \lambda r^\kappa\,.\,\forall x^{\tau^*}\,\forall q^\kappa\,\forall r'^\kappa\,(r = qr' \mapsto (\forall s^\kappa\,(\mathsf{C}[qs] \Rightarrow A^* s)) \Rightarrow B^*\, r') & \\ \cong\;\; (\forall x^\tau\,(A \Rightarrow B))^* & (\beta)\end{array}$$

making $\forall x^{\tau^*}$ successively commute with $\Rightarrow$ (since $x^{\tau^*} \notin FV(A^*)$), $\mapsto$ and $\forall$.

— Simplification of $\mapsto$:

$$(M = M \mapsto A)^* \;\equiv\; \lambda r^\kappa\,.\,M^* = M^* \mapsto A^*\, r \;\cong\; \lambda r^\kappa\,.\,A^*\, r \;\cong\; A^* \qquad (\eta)$$

— Re-orientation of $\mapsto$:

$$\begin{array}{rcll}(M = M' \mapsto A)^* & \equiv & \lambda r^\kappa\,.\,M^* = M'^* \mapsto A^*\,r & \\ & \cong & \lambda r^\kappa\,.\,M'^* = M^* \mapsto A^*\,r & \equiv\ (M' = M \mapsto A)^*\end{array}$$

— Commutation $\mapsto/\mapsto$:

$$\begin{array}{rcll}(M = M' \mapsto N = N' \mapsto A)^* & \cong & \lambda r^\kappa\,.\,M^* = M'^* \mapsto N^* = N'^* \mapsto A^*\,r & (\beta)\\ & \cong & \lambda r^\kappa\,.\,N^* = N'^* \mapsto M^* = M'^* \mapsto A^*\,r & \\ & \cong & (N = N' \mapsto M = M' \mapsto A)^* & (\beta)\end{array}$$

— Commutation $\Rightarrow/\mapsto$:

$$\begin{array}{cll}\multicolumn{2}{l}{(A \Rightarrow M = M' \mapsto B)^*} & \\ \cong & \lambda r^\kappa\,.\,\forall q^\kappa\,\forall r'^\kappa\,(r = qr' \mapsto (\forall s^\kappa\,(\mathsf{C}[qs] \Rightarrow A^*\,s)) \Rightarrow M^* = M'^* \mapsto B^*\,r') & (\beta)\\ \cong & \lambda r^\kappa\,.\,M^* = M'^* \mapsto \forall q^\kappa\,\forall r'^\kappa\,(r = qr' \mapsto (\forall s^\kappa\,(\mathsf{C}[qs] \Rightarrow A^*\,s)) \Rightarrow B^*\,r') & \\ \cong & (M = M' \mapsto A \Rightarrow B)^* & (\beta)\end{array}$$

using the commutation rules $\Rightarrow/\mapsto$, $\mapsto/\mapsto$ and $\forall/\mapsto$.

— Commutation $\forall/\mapsto$, when $x^\tau \notin FV(M, M')$:

$$\begin{array}{rcll}(M = M' \mapsto \forall x^\tau\,A)^* & \cong & \lambda r^\kappa\,.\,M^* = M'^* \mapsto \forall x^{\tau^*}\,A^*\,r & (\beta)\\ & \cong & \lambda r^\kappa\,.\,\forall x^{\tau^*}\,(M^* = M'^* \mapsto A^*\,r) & (\mapsto/\forall\text{-commut.})\\ & \cong & (\forall x^\tau\,(M = M' \mapsto A))^* & (\beta)\end{array}$$

The cases corresponding to the remaining rules of Table 1 are straightforward. □

4.2. *The forcing translation $A \mapsto (p \Vdash A)$*

Given a condition p and a proposition A, we define the forcing relation $p \Vdash A$ by letting

$$p \Vdash A \;\equiv\; \forall r^\kappa\,(\mathsf{C}[pr] \Rightarrow A^* r)$$

(where r is a fresh variable). This definition immediately extends to all typing contexts $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ by letting $p \Vdash \Gamma \equiv x_1 : (p \Vdash A_1), \ldots, x_n : (p \Vdash A_n)$.

We easily check that the forcing relation is substitutive and compatible with the congruence $\cong_\mathcal{E}$, the latter property being crucial to ensure that any conversion step in the proof of A will be translated into a conversion step in the corresponding proof of $p \Vdash A$:

Lemma 23 (Substitutivity and compatibility with the congruence $M \cong_\mathcal{E} M'$).

(1) $p \Vdash (A\{x^\tau := N\}) \;\equiv\; (p \Vdash A)\{x^{\tau^*} := N^*\}$ (provided $x^\tau \notin FV(p)$)
(2) If $A \cong_\mathcal{E} A'$, then $(p \Vdash A) \cong_{\mathcal{E}^*} (p \Vdash A')$.

Proof. Immediately follows from the definition of $p \Vdash A$ and Lemma 22. □

Remark 24. The meaning of the definition of $p \Vdash A$ can be understood as follows. Given two conditions p and q, we write $p \perp q \equiv \neg\mathsf{C}[pq]$ ('p and q are *incompatible*'). The orthogonal of a set of conditions S (of kind $\kappa \to o$) is defined by

$$S^\perp \;\equiv\; \lambda q^\kappa\,.\,\forall p^\kappa\,(S\,p \Rightarrow p \perp q)\,,$$

that is, as the set of all conditions that are incompatible with all the elements of S. In the theory of Boolean valued models (Bel85), we know that the set of sets

$$\mathcal{B} \;=\; \{S \in \mathfrak{P}(\mathsf{C}) \;:\; S = S^{\perp\perp}\}$$

formed by all the sets of conditions $S \subseteq \mathsf{C}$ that are equal to their bi-orthogonal forms a complete Boolean algebra ordered by inclusion, which is known in the theory of forcing as *the Boolean algebra generated by the poset of conditions* $(\mathsf{C}, \leq)$[‡]. Coming back to the definition of the relation $p \Vdash A$ in system PAω^+, the following (classical) equivalence

$$p \Vdash A \quad\equiv\quad \forall r^\kappa(\mathsf{C}[pr] \Rightarrow A^* r) \quad\Leftrightarrow\quad \forall r^\kappa(\neg A^* r \Rightarrow p \perp r)$$

shows that the set of all conditions p forcing A is nothing but the orthogonal of the complement set of A^*, that is: $\{p : p \Vdash A\} = ((A^*)^c)^\perp$ (using suggestive notations). Therefore the set of all conditions p such that $p \Vdash A$ is equal to its bi-orthogonal, and thus belongs to the complete Boolean algebra $\mathcal{B}$. However, there is a small difference w.r.t. the traditional definition of $\mathcal{B}$, which is that in our framework, the elements of $\mathcal{B}$ are not defined as subsets of C (i.e. as sets of well-formed conditions), but as subsets of κ, which may contain ill-formed conditions as well. Technically, this extension slightly changes the implementation of $\mathcal{B}$—the bottom truth value of $\mathcal{B}$ is now represented by the set of all ill-formed formed conditions rather than by the empty set of (well-formed) conditions—but the underlying structure remains the same up to isomorphism. We can also notice that the inclusion $\{p : p \Vdash A\} \cap \mathsf{C} \subseteq A^*$ always holds (by combinator α_4).

Using the above intuitions, we can now build proof terms (in system PAω^+) expressing that the forcing relation $p \Vdash A$ is anti-monotonic:

Proposition 25. — In system PAω^+:

$$\begin{array}{lclcl}
\beta_1 & \equiv & \lambda xyc\,.\,y\,(x\,c) & : & \forall p^\kappa\ \forall q^\kappa\ (q \leq p \Rightarrow (p \Vdash A) \Rightarrow (q \Vdash A)) \\
\beta_2 & \equiv & \lambda xc\,.\,x\,(\alpha_1\,c) & : & \forall p^\kappa\ (\neg\mathsf{C}[p] \Rightarrow p \Vdash A) \\
\beta_3 & \equiv & \lambda xc\,.\,x\,(\alpha_9\,c) & : & \forall p^\kappa\ \forall q^\kappa\ ((p \Vdash A) \Rightarrow (pq \Vdash A)) \\
\beta_4 & \equiv & \lambda xc\,.\,x\,(\alpha_{10}\,c) & : & \forall p^\kappa\ \forall q^\kappa\ ((q \Vdash A) \Rightarrow (pq \Vdash A))
\end{array}$$

The first proof term β_1 expresses the desired property of anti-monotonicity: if a proposition A is forced by some condition p, then A is forced by all the conditions $q \leq p$. (In other words: the set $\{p : p \Vdash A\}$ is downwards closed.) Moreover, non well-formed conditions force all propositions (proof term β_2). Finally, the proof terms β_3 and β_4 express particular cases of the property of anti-monotonicity that will be useful in the following.

4.3. *Forcing logical constructions*

Let us now see how the forcing relation $p \Vdash A$ deals with the primitive logical constructions of PAω^+. We first treat the case of universal quantification and equational

[‡] The elements of this complete Boolean algebra are also known to be the *regular open subsets* of the set of conditions (for the topology whose open sets are the downwards closed sets of conditions).

implication, that are the logical constructions whose typing rules are computationally transparent in system PAω^+:

Fact 26 (Forcing universal quantification and equational implication).
(1) $p \Vdash \forall x^\tau A \;\cong\; \forall x^{\tau^*}(p \Vdash A)$ (if $x^\tau \notin FV(p)$)
(2) $p \Vdash M = M' \mapsto A \;\cong\; M^* = M'^* \mapsto p \Vdash A$

Proof. The following conversions

$$\begin{array}{rcll} p \Vdash \forall x^\tau A & \cong & \forall r^\kappa\,(\mathsf{C}[pr] \Rightarrow \forall x^{\tau^*} A^*\, r) & (\beta) \\ & \cong & \forall x^{\tau^*} \forall r^\kappa\,(\mathsf{C}[pr] \Rightarrow A^*\, r) \quad \equiv \quad \forall x^{\tau^*}\,(p \Vdash A) & \end{array}$$

$$\begin{array}{rcll} p \Vdash M = M' \mapsto A & \cong & \forall r^\kappa\,(\mathsf{C}[pr] \Rightarrow M^* = M'^* \mapsto A^*\, r) & (\beta) \\ & \cong & M^* = M'^* \mapsto \forall r^\kappa\,(\mathsf{C}[pr] \Rightarrow A^*\, r) & \\ & \equiv & M^* = M'^* \mapsto p \Vdash A & \end{array}$$

essentially rely on the commutation rules $\Rightarrow/\forall$, $\Rightarrow/\mapsto$ and $\mapsto/\forall$ (Table 1). $\square$

Remark 27. The above statement expresses that the forcing relation $p \Vdash A$ commutes with universal quantification (and with equational implication as well). From a purely logical point of view, this implies that the two propositions $p \Vdash (\forall x^\tau A)$ and $\forall x^{\tau^*}(p \Vdash A)$ are provably equivalent in system PAω^+—which is actually mandatory in any definition of forcing (Coh63; Coh64; Bel85). But in the proof-theoretic perspective, the (stronger) property of convertibility $p \Vdash (\forall x^\tau A) \cong \forall x^{\tau^*}(p \Vdash A)$ is much more interesting, since it means that any introduction/elimination step of a universal quantification can be translated into a similar deduction step (accompanied with a conversion step) during the translation of a derivation. Here we only show the case of the $\forall$-elimination rule

$$\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma \vdash \forall x^\tau A\end{array}}{\mathcal{E}; \Gamma \vdash A\{x^\tau := N\}}\,\forall\text{-elim} \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, d^* \\ \mathcal{E}^*(p \Vdash \Gamma) \vdash (p \Vdash \forall x^\tau A)\end{array}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \forall x^{\tau^*}\,(p \Vdash A)}\,\text{Conv.}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash (p \Vdash A)\{x^{\tau^*} := N^*\}}\,\forall\text{-elim}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash (p \Vdash A\{x^\tau := N\})}$$

(leaving proof terms implicit), but it is clear that the introduction rules of $\forall$ and $\mapsto$ can be translated in a similar way. When combined with Lemma. 23, the above conversions thus make possible to translate *any* computationally transparent deduction step into a combination of computationally transparent deduction steps during the translation of derivations. This is actually the key ingredient that will permit us to turn the translation $d \mapsto d^*$ of *derivations*—which is induced by the purely logical properties of forcing—into a translation $t \mapsto t^*$ of *proof terms*—thus giving us the desired program transformation.

Let us now consider the case of implication, that concentrates all the computational contents of the translation. The theory of forcing requires the logical equivalence

$$p \Vdash A \Rightarrow B \quad \Leftrightarrow \quad \forall q^\kappa\,((q \Vdash A) \Rightarrow (pq \Vdash B))\,.$$

And from our definition of the forcing relation, we get:

$$\begin{array}{rcll} p \Vdash A \Rightarrow B & \equiv & \forall r^{\kappa}\,(\mathsf{C}[pr] \Rightarrow (A \Rightarrow B)^{*} r) & \\ & \cong & \forall r^{\kappa}\,(\mathsf{C}[pr] \Rightarrow \forall q^{\kappa}\,\forall r'^{\kappa}\,(r = qr' \mapsto (q \Vdash A) \Rightarrow B^{*} r')) & \\ & \Leftrightarrow & \forall q^{\kappa}\,\forall r'^{\kappa}\,(\mathsf{C}[p(qr')] \Rightarrow (q \Vdash A) \Rightarrow B^{*} r') & \\ & \Leftrightarrow & \forall q^{\kappa}\,((q \Vdash A) \Rightarrow \forall r'^{\kappa}\,(\mathsf{C}[(pq)r'] \Rightarrow B^{*} r')) & (\text{using } \alpha_5, \alpha_6) \\ & \cong & \forall q^{\kappa}\,((q \Vdash A) \Rightarrow (pq \Vdash B)) & \end{array}$$

Formally:

Proposition 28 (Forcing an implication). — In system PAω^{+} we have:

$$\begin{array}{lcll} \gamma_1 & \equiv & \lambda xcy\,.\,x\,y\,(\alpha_6\,c) & :\ \forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B)) \ \Rightarrow\ (p \Vdash A \Rightarrow B) \\ \gamma_2 & \equiv & \lambda xyc\,.\,x\,(\alpha_5\,c)\,y & :\ (p \Vdash A \Rightarrow B) \ \Rightarrow\ \forall q\ ((q \Vdash A) \Rightarrow (pq \Vdash B)) \\ \gamma_3 & \equiv & \lambda xyc\,.\,x\,(\alpha_{11}\,c)\,y & :\ (p \Vdash A \Rightarrow B) \ \Rightarrow\ (p \Vdash A) \ \Rightarrow\ (p \Vdash B) \\ \gamma_4 & \equiv & \lambda xcy\,.\,x\,(y\,(\alpha_{15}\,c)) & :\ \neg A^{*}\,p \ \Rightarrow\ (p \Vdash A \Rightarrow B) \end{array}$$

Intuitively, the proof term γ_1 'folds' a proof of the proposition $\forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B))$ into a proof of $p \Vdash A \Rightarrow B$, while γ_2 performs the corresponding unfolding operation. The proof term γ_3 is a specialized form of γ_2 corresponding to the modus ponens through the forcing transformation. We also introduce a proof term γ_4 that will be a key ingredient of the translation of continuations.

Before presenting the proof transformation $t \mapsto t^{*}$ induced by Lemma 23 and Prop. 28, we can already see that Peirce's law is forced by any condition p:

Proposition 29 (Forcing Peirce's law). — In system PAω^{+} we have:

$$\text{œ}^{*} \ \equiv\ \lambda cx\,.\,\text{œ}\,(\lambda k\,.\,x\,(\alpha_{14}\,c)\,(\gamma_4\,k)) \ :\ p \Vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A\,.$$

4.4. *Translating proof terms*

We can now define the translation $t \mapsto t^{*}$ on *raw* proof terms as follows:

Definition 30 (Translating proof terms). — Every (raw) proof term t is translated as a proof term t^{*} with the same free variables using the following equations:

$$\begin{array}{rl} x^{*} & \equiv x \\ (t\,u)^{*} & \equiv \gamma_3\,t^{*}\,u^{*} \\ (\lambda x\,.\,t)^{*} & \equiv \gamma_1\,(\lambda x\,.\,t^{*}\{x_i := \beta_3 x_i\}_{i=1}^{n}\{x := \beta_4 x\}) \quad (\text{where } \{x_1; \ldots ; x_n\} = FV(t) \setminus \{x\}) \\ \text{œ}^{*} & \equiv \lambda cx\,.\,\text{œ}\,(\lambda k\,.\,x\,(\alpha_{14}\,c)\,(\gamma_4\,k)) \end{array}$$

Remark 31. Basically, the translation $t \mapsto t^{*}$ inserts the combinator γ_1 ('fold') in front of every abstraction, and the combinator γ_3 ('apply') in front of every application, while translating the call/cc constant into the proof term œ* introduced in Prop. 29.

The main subtlety of the translation lies in the treatment of free and bound variables: at each abstraction, the translation inserts the combinator β_3 in front of every free occurrence of a variable x_i that is not bound by the abstraction, while inserting the combinator β_4 in front of every free occurrence of the variable x (that is bound by the abstraction) in the term t. It is easy to see that, when applied to a closed term t, the translation reveals the De Bruijn structure of t, since every occurrence of a variable x in the term t is translated into the term $\beta_3^n(\beta_4 x)$, where n is the De Bruijn index of that occurrence (starting indices from 0).

We can now check that the program transformation $t \mapsto t^*$ is sound w.r.t. typing:

Theorem 32 (Soundness). — If the judgment $\mathcal{E}; \Gamma \vdash t : A$ is derivable (in PAω^+), then for all conditions p, the sequent $\mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A)$ is derivable too.

Proof. Without loss of generality, we can assume that p is a fresh condition variable, that is, a variable of kind κ that does not occur in the derivation of $\mathcal{E}; \Gamma \vdash t : A$. (The general case follows by substitutivity, second rule of Prop. 11.) The proof is done by induction on the derivation of $\mathcal{E}; \Gamma \vdash t : A$, distinguishing the following cases:

— Axiom. Obvious, since $x^* \equiv x$.

— Conversion. Immediate from Lemma. 23.

— $\Rightarrow$-intro. The conclusion $\mathcal{E}; \Gamma \vdash \lambda x\,.\,t : A \Rightarrow B$ comes from a unique premise $\mathcal{E}; \Gamma, x : A \vdash t : B$. By IH, the judgment $\mathcal{E}^*; (p \Vdash \Gamma), x : (p \Vdash A) \vdash t^* : (p \Vdash B)$ is derivable. Let q be a fresh condition variable. By substituting p with pq in the latter judgment, we get $\mathcal{E}^*; (pq \Vdash \Gamma), x : (pq \Vdash A) \vdash t^* : (pq \Vdash B)$ (2nd rule of Prop. 11). Since $x : (q \Vdash A) \vdash \beta_4\, x : (pq \Vdash A)$, we get

$$\mathcal{E}^*; (pq \Vdash \Gamma), x : (q \Vdash A) \vdash t^*\{x := \beta_4\, x\} : (pq \Vdash B)$$

using the 3rd rule of Prop. 11. Let us now write $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$. Since the judgment $x_i : (p \Vdash A_i) \vdash \beta_3\, x_i : (pq \Vdash A_i)$ is derivable for all $i \in [1..n]$, we get

$$\begin{array}{l} \mathcal{E}^*; x_1 : (p \Vdash A_1), \ldots, x_n : (p \Vdash A_n), x : (q \Vdash A) \\ \quad \vdash \; t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\} \; : \; (pq \Vdash B) \end{array}$$

by applying n times the third rule of Prop. 11, hence we can derive

$$\dfrac{\dfrac{\dfrac{\vdots \atop \mathcal{E}^*; (p \Vdash \Gamma), x : (q \Vdash A) \vdash t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\} : (pq \Vdash B)}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \lambda x\,.\,t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\} : (q \Vdash A) \Rightarrow (pq \Vdash B)}\,{\scriptstyle \Rightarrow\text{-intro}}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \lambda x\,.\,t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\} : \forall q\,((q \Vdash A) \Rightarrow (pq \Vdash B))}\,{\scriptstyle \forall\text{-intro}}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \gamma_1\,(\lambda x\,.\,t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\}) : (p \Vdash A \Rightarrow B)}\,{\scriptstyle \Rightarrow\text{-elim}}$$

and conclude using the fact that $\gamma_1\,(\lambda x\,.\,t^*\{x_i := \beta_3\, x_i\}_{i=1}^n\{x := \beta_4\, x\}) \equiv (\lambda x\,.\,t)^*$.

— $\Rightarrow$-elim. The conclusion $\mathcal{E}; \Gamma \vdash tu : B$ comes from two premises $\mathcal{E}; \Gamma \vdash t : A \Rightarrow B$ and $\mathcal{E}; \Gamma \vdash u : A$. By IH, the two judgments $\mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A \Rightarrow B)$ and $\mathcal{E}^*; (p \Vdash \Gamma) \vdash u^* : (p \Vdash A)$ are derivable, hence we can derive

$$\dfrac{\dfrac{\vdots \atop \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A \Rightarrow B)}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \gamma_3\, t^* : (p \Vdash A) \Rightarrow (p \Vdash B)} \qquad {\vdots \atop \mathcal{E}^*; (p \Vdash \Gamma) \vdash u^* : (p \Vdash A)}}{\mathcal{E}^*; (p \Vdash \Gamma) \vdash \gamma_3\, t^*\, u^* : (p \Vdash B)}$$

and conclude using the fact that $\gamma_3\, t^*\, u^* \equiv (tu)^*$.

— $\mapsto$-intro. The conclusion $\mathcal{E}; \Gamma \vdash t : M = M' \mapsto A$ comes from a unique premise $\mathcal{E}, M = M'; \Gamma \vdash t : A$. By IH, the judgment $\mathcal{E}^*, M^* = M'^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A)$ is derivable, hence:

$$\begin{array}{c} \vdots \\ \mathcal{E}^*, M^* = M'^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : M^* = M'^* \mapsto (p \Vdash A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash M = M' \mapsto A) \end{array} \begin{array}{l} \\ \mapsto\text{-intro} \\ \text{Conv.} \end{array}$$

— $\mapsto$-elim. This rule is a particular case of conversion (Remarks 10 (1)).

— $\forall$-intro. The conclusion $\mathcal{E}; \Gamma \vdash t : \forall x^\tau A$ comes from a unique premise $\mathcal{E}; \Gamma \vdash t : A$ such that $x^\tau \notin FV(\mathcal{E}; \Gamma)$. By IH, the judgment $\mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A)$ is derivable, hence:

$$\begin{array}{c} \vdots \\ \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : \forall x^{\tau^*} (p \Vdash A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash \forall x^\tau A) \end{array} \begin{array}{l} \\ \forall\text{-intro} \\ \text{Conv.} \end{array}$$

— $\forall$-elim. The conclusion $\mathcal{E}; \Gamma \vdash t : A\{x^\tau := N\}$ (where N is of kind τ) comes from a unique premise $\mathcal{E}; \Gamma \vdash t : \forall x^\tau A$. By IH, the judgment $\mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash \forall x^\tau A)$ is derivable, hence

$$\begin{array}{c} \vdots \\ \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash \forall x^\tau A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : \forall x^{\tau^*} (p \Vdash A) \\ \hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A)\{x^{\tau^*} := N^*\} \\ \hline\hline \mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : (p \Vdash A\{x^\tau := N\}) \end{array} \begin{array}{l} \\ \text{Conv.} \\ \forall\text{-elim} \\ \text{Prop. 23} \end{array}$$

— Peirce's law. Immediate from Prop. 29. □

4.5. *Using the program transformation $t \mapsto t^*$*

Let us now see how the proof/program transformation $t \mapsto t^*$ can be used in practice.

For that, let us imagine that we have a proof u of a theorem B that depends on an assumption $x : A$, where A is an axiom—for instance the negation of the continuum hypothesis expressed in system PAω^+—that cannot be proved in PAω^+:

$$x : A \vdash u : B\,.$$

On the other hand, let us imagine that we can 'force' the axiom A using a given forcing structure $(\kappa, \mathsf{C}, \cdot, 1, \alpha_*, \alpha_1, \ldots, \alpha_8)$, so that there is a closed proof term s such that

$$\vdash s : (1 \Vdash A)\,.$$

From Theorem 32 we get $x : (1 \Vdash A) \vdash u^* : (1 \Vdash B)$, so that using the property of proof substitutivity in system PAω^+ (Prop. 11) we can conclude that

$$\vdash u^*\{x := s\} : (1 \Vdash B)\,.$$

This gives us a uniform way to combine a proof of a theorem B depending on a axiom A together with a proof that A is forced (by some forcing structure) in order to get a proof that the theorem B is forced (by the same forcing structure). Moreover, the proof term $u^*\{x := s\}$ that we obtain in this way only depends on the proof terms u and s themselves, but not on the propositions A and B. (The proof term $u^*\{x := s\}$ also depends on the forcing combinators $\alpha_*, \alpha_1, \ldots, \alpha_8$ underlying the translation $u \mapsto u^*$.)

Of course, the resulting proof term $u^*\{x := s\}$ is not a proof of the theorem B, but only a proof of the proposition $\mathsf{1} \Vdash B$, that is not equivalent to B in general. However, there are many cases where the proposition $\mathsf{1} \Vdash B$ is actually equivalent to B, so that the proof $u^*\{x := s\}$ of the proposition $\mathsf{1} \Vdash B$ can be turned into a proof of B (using a suitable 'wrapper'), thus completely removing the need of the assumption A in the proof of B. This is typically the case when B is a first-order proposition, as shown below.

4.6. *Invariance under forcing*

In this section, we are interested in the propositions A that are invariant under forcing, in the sense that $p \Vdash A$ is provably equivalent to $\mathsf{C}[p] \Rightarrow A$. Notice that the equivalence between $p \Vdash A$ and $\mathsf{C}[p] \Rightarrow A$ has only a meaning when these two propositions have the same free variables, which is the case when the free variables of A are T-variables (i.e. whose kind is a T-kind), that are unaffected by the forcing translation.

Definition 33 (Invariance under forcing). — Let $A(\vec{x})$ be a proposition only depending on T-variables $\vec{x}$. We say that the proposition $A(\vec{x})$ is *invariant under forcing* when there are two closed proof terms ξ_A and ξ'_A such that both judgments

$$\begin{array}{rcl} \xi_A & : & \forall \vec{x}\, \forall p^\kappa\ [(p \Vdash A(\vec{x})) \ \Rightarrow\ (\mathsf{C}[p] \Rightarrow A(\vec{x}))] \\ \xi'_A & : & \forall \vec{x}\, \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow A(\vec{x})) \ \Rightarrow\ (p \Vdash A(\vec{x}))] \end{array}$$

are derivable (in the empty context) in system $\mathrm{PA}\omega^+$.

The following result expresses that the class of propositions that are invariant under forcing is closed under all first-order constructions, treating T-terms as first-order objects and T-kinds as the kinds of first-order objects:

Proposition 34 (Invariance of first-order constructions).

(1) The proposition $\bot \equiv \forall x^0\, x$ is invariant under forcing.
(2) Leibniz equality $x =_\tau y$ is invariant under forcing, for every T-kind τ.
(3) If both propositions $A(\vec{x})$ and $B(\vec{x})$ are invariant under forcing (where $\vec{x}$ are T-variables), then so is the proposition $A(\vec{x}) \Rightarrow B(\vec{x})$.
(4) If the proposition $A(x_0^\tau, \vec{x})$ is invariant under forcing (where $x_0^\tau, \vec{x}$ are T-variables), then so is the proposition $\forall x_0^\tau A(x_0^\tau, \vec{x})$.
(5) The proposition $\mathrm{rel}_\iota\, x \equiv \mathrm{nat}\, x$ (Section 2.6) is invariant under forcing, and more generally, the proposition $\mathrm{rel}_\tau\, x$ is invariant under forcing for every T-kind τ.

Proof. (1) We let

$$\begin{array}{lrcl} & \xi_\bot & \equiv & \lambda zc\,.\,z\,(\alpha_7\,c) \quad : \quad \forall p^\kappa\,[(p \Vdash \bot) \Rightarrow (\mathsf{C}[p] \Rightarrow \bot)] \\ \text{and} & \xi'_\bot & \equiv & \lambda zc\,.\,z\,(\alpha_1\,c) \quad : \quad \forall p^\kappa\,[(\mathsf{C}[p] \Rightarrow \bot) \Rightarrow (p \Vdash \bot)] \end{array}$$

(2) Given a T-kind τ, we let

$$\begin{array}{lrcl} & \xi_{=_\tau} & \equiv & \lambda zcx\,.\,\gamma_3\,z\,(\lambda\_\,.\,x)\,(\alpha_7\,c) \\ & & : & \forall x^\tau\ \forall y^\tau\ \forall p^\kappa\ [(p \Vdash x =_\tau y) \Rightarrow (\mathsf{C}[p] \Rightarrow x =_\tau y)] \\ \text{and} & \xi'_{=_\tau} & \equiv & \lambda z\,.\,\gamma_1\,(\lambda xc\,.\,z\,(\alpha_1\,(\alpha_1\,c))\,(x\,(\alpha_{10}\,c))) \\ & & : & \forall x^\tau\ \forall y^\tau\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow x =_\tau y) \Rightarrow (p \Vdash x =_\tau y)]\,. \end{array}$$

(3) Given four closed proof terms

$$\begin{array}{rcl} \xi_A & : & \forall\vec{x}\ \forall p^\kappa\ [(p \Vdash A(\vec{x})) \Rightarrow (\mathsf{C}[p] \Rightarrow A(\vec{x}))] \\ \xi'_A & : & \forall\vec{x}\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow A(\vec{x})) \Rightarrow (p \Vdash A(\vec{x}))] \\ \xi_B & : & \forall\vec{x}\ \forall p^\kappa\ [(p \Vdash B(\vec{x})) \Rightarrow (\mathsf{C}[p] \Rightarrow B(\vec{x}))] \\ \xi'_B & : & \forall\vec{x}\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow B(\vec{x})) \Rightarrow (p \Vdash B(\vec{x}))]\,, \end{array}$$

$$\begin{array}{lrcl} \text{we let} & \xi_{A\Rightarrow B} & \equiv & \lambda zcx\,.\,\xi_B\,(\gamma_3\,z\,(\xi'_A\,(\lambda\_\,.\,x)))\,c \\ & & : & \forall\vec{x}\ \forall p^\kappa\ [(p \Vdash A(\vec{x}) \Rightarrow B(\vec{x})) \Rightarrow (\mathsf{C}[p] \Rightarrow A(\vec{x}) \Rightarrow B(\vec{x}))] \\ \text{and} & \xi'_{A\Rightarrow B} & \equiv & \lambda z\,.\,\gamma_1\,(\lambda x\,.\,\xi'_B\,(\lambda c\,.\,z\,(\alpha_1\,c)\,(\xi_A\,x\,(\alpha_2\,c)))) \\ & & : & \forall\vec{x}\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow A(\vec{x}) \Rightarrow B(\vec{x})) \Rightarrow (p \Vdash A(\vec{x}) \Rightarrow B(\vec{x}))]\,. \end{array}$$

(4) Given two closed proof terms

$$\begin{array}{rcl} \xi_A & : & \forall x_0^\tau\ \forall\vec{x}\ \forall p^\kappa\ [(p \Vdash A(x_0^\tau,\vec{x})) \Rightarrow (\mathsf{C}[p] \Rightarrow A(x_0^\tau,\vec{x}))] \\ \xi'_A & : & \forall x_0^\tau\ \forall\vec{x}\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow A(x_0^\tau,\vec{x})) \Rightarrow (p \Vdash A(x_0^\tau,\vec{x}))]\,, \end{array}$$

$$\begin{array}{lrcl} \text{we let} & \xi_{\forall x_0 A} & \equiv & \lambda z\,.\,\xi_A\,z \ :\ \forall\vec{x}\ \forall p^\kappa\ [(p \Vdash \forall x_0^\tau\,A(x_0^\tau,\vec{x})) \Rightarrow (\mathsf{C}[p] \Rightarrow \forall x_0^\tau\,A(x_0^\tau,\vec{x}))] \\ \text{and} & \xi'_{\forall x_0 A} & \equiv & \lambda z\,.\,\xi'_A\,z \ :\ \forall\vec{x}\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow \forall x_0^\tau\,A(x_0^\tau,\vec{x})) \Rightarrow (p \Vdash \forall x_0^\tau\,A(x_0^\tau,\vec{x}))] \end{array}$$

$$\begin{array}{lrcl} \text{(5) We let} & \xi_{\mathrm{nat}} & \equiv & \lambda zcxy\,.\,\gamma_3\,(\gamma_3\,z\,(\lambda\_\,.\,x))\,(\gamma_1\,(\lambda uc'\,.\,y\,(u\,(\alpha_{10}\,c'))))\,(\alpha_7\,c) \\ & & : & \forall x^\iota\ \forall p^\kappa\ [(p \Vdash \mathrm{nat}\,x) \Rightarrow (\mathsf{C}[p] \Rightarrow \mathrm{nat}\,x)] \\ \text{and} & \xi'_{\mathrm{nat}} & \equiv & \lambda z\,.\,\gamma_1\,(\lambda x\,.\,\gamma_1\,(\lambda yc\,.\,z\,(\alpha_1^3\,c)\,(\beta_3\,x)\,(\gamma_3\,(\beta_4\,y))\,(\alpha_{16}\,c))) \\ & & : & \forall x^\iota\ \forall p^\kappa\ [(\mathsf{C}[p] \Rightarrow \mathrm{nat}\,x) \Rightarrow (p \Vdash \mathrm{nat}\,x)] \end{array}$$

More generally, the closed proof terms ξ_{rel_τ} and ξ'_{rel_τ} are built by induction on τ from the proof terms ξ_{nat} and ξ'_{nat} (corresponding to the base case) using the definition of the predicate rel_τ (Section 2.6) and the constructions depicted in items (3) and (4). □

The main consequence of the above proposition is that all first-order propositions are invariant under the forcing translation. Formally:

Definition 35 (First-order propositions). — We call a *first-order proposition* any proposition that is formed from the following grammar:

First-order propositions

$$\begin{array}{rcl} A, B & ::= & \bot \quad | \quad M =_\tau M' \quad | \quad \mathrm{rel}_\tau M \\ & | & A \Rightarrow B \quad | \quad \forall x^\tau A \end{array}$$

where τ ranges over T-kinds and where M, M' range over T-terms of kind τ.

Remark 36. As a particular case, the class of first-order propositions contains the (proper) subclass of *arithmetic propositions*, where all quantifications over a T-kind τ are systematically relativized using the predicate rel_τ of kind $\tau \to o$:

Arithmetic propositions $\qquad A, B \quad ::= \quad \bot \quad | \quad M =_\tau M'$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad | \quad A \Rightarrow B \quad | \quad \forall x^\tau\,(\mathrm{rel}_\tau\, x \Rightarrow A)$

(with the same restrictions on τ, M and M').

Proposition 37 (Invariance of first-order propositions). — All first-order propositions are invariant under forcing (in the sense of Def. 33).

Proof. By induction on the structure of A, distinguishing the following cases:
— $A \equiv \bot$. Obvious by Prop. 34 (1).
— $A \equiv M =_\tau M'$, where M, M' are T-terms of a T-kind τ. Follows from Prop. 34 (2) by substitutivity (Lemma 23), using the fact that $M^* \equiv M$ and $M'^* \equiv M'$.
— $A \equiv \mathrm{rel}_\tau M$, where M is a T-term of a T-kind τ. Follows from Prop. 34 (5) by substitutivity (Lemma 23), using the fact that $M^* \equiv M$.
— $A \equiv A_1 \Rightarrow A_2$, where A_1 and A_2 are first-order. Follows from Prop. 34 (3).
— $A \equiv \forall x_0^\tau A_0$, where A_0 is first-order. Follows from Prop. 34 (4). ☐

Theorem 38 (Elimination of a forced hypothesis). — If A and B are two closed propositions such that

(1) $\mathrm{PA}\omega^+ \vdash A \Rightarrow B$,
(2) $\mathrm{PA}\omega^+ \vdash (1 \Vdash A)$,
(3) B is a first-order proposition, or more generally a proposition that is invariant under forcing (in the sense of Def. 33);

then: $\mathrm{PA}\omega^+ \vdash B$.

Proof. Let u and s be proof terms such that $u : A \Rightarrow B$ and $s : (1 \Vdash A)$. Then from Theorem 32, Def. 33 and Prop. 37 we get $\xi_B(\gamma_3\, u^*\, s)\,\alpha_* : B$. ☐

4.7. *An analysis of the computational behavior of transformed proofs*

Let us first recall that the image of a proof term t via the transformation $t \mapsto t^*$ is another proof term t^* that has a type of the form $p \Vdash A \equiv \forall r\,(\mathsf{C}[pr] \Rightarrow A^* r)$, which means that t^* is intended to be evaluated in front of a stack whose first argument c has type $\mathsf{C}[pr]$ for some forcing conditions p and r. (See Section 2.8.) As we shall see, the condition p represents logical invariants attached to the current proof term t^* that is currently evaluated, whereas the condition r represents logical invariants attached to the stack facing t^* during evaluation. In what follows, we shall call a *computational condition*—as opposed to a logical condition—any closed term c of type $\mathsf{C}[pr]$ (or more generally: any realizer of $\mathsf{C}[pr]$) for some conditions p and r.

Using the definition of the combinators γ_1, γ_3, $\mathbb{c}^*$ and γ_4, we easily discover the following evaluation scheme for the translated program t^*:

Proposition 39 (Computational behavior of translated proof terms). — Let t_x be a proof term such that $FV(t_x) \subseteq \{x\}$, and t, u two closed proof terms. For all $c, v \in \Lambda$ and $\pi \in \Pi$:

$$\begin{array}{rcr}
(\lambda x\,.\,t_x)^* \star c\cdot v\cdot\pi & \succ^* & t_x^*\{x := \beta_4 v\} \star \alpha_6\ c\cdot\pi \\
(tu)^* \star c\cdot\pi & \succ^* & t^* \star \alpha_{11}c\cdot u^*\cdot\pi \\
\text{cc}^* \star c\cdot v\cdot\pi & \succ^* & v \star \alpha_{14}c\cdot \mathsf{k}_\pi^*\cdot\pi \\
\mathsf{k}_\pi^* \star c\cdot v\cdot\pi' & \succ^* & v \star \alpha_{15}c\cdot\pi
\end{array}$$

writing k_π^* as a shorthand for $\gamma_4\,\mathsf{k}_\pi$.

From this picture, we can see that the translated proof term t^* essentially behaves the same way as the initial proof term t, with the difference that the first slot of the stack is now reserved to the computational condition that evolves during evaluation. All the stack operations are thus performed one slot further in the stack (slots are thus intuitively re-indexed), while each operation updates the current computational condition (in the first slot of the stack) by inserting the appropriate combinator.

Most notably, the translated call/cc operator cc^* does not save the current computational condition, while the translated continuation constant k_π^* (that is dynamically generated by cc^*) does not restore any formerly saved computational condition—it just updates the current computational condition using the appropriate combinator. As noticed by Krivine (Kri08; Kri10), the first slot of the stack[§] is thus subtracted from the normal save/restore mechanism induced by call/cc, and now behaves as a mutable reference (or as a 'global memory' according to Krivine's terminology).

Let us now have a look at the types (cf Section 3.1) of the combinators α_6, α_{11}, α_{14} and α_{15} that are inserted in the first slot of the stack at each step of the evaluation of t^*:

(Grab)	α_6	:	$\mathsf{C}[p(qr)]$	$\Rightarrow$	$\mathsf{C}[(pq)r]$
(Push)	α_{11}	:	$\mathsf{C}[pr]$	$\Rightarrow$	$\mathsf{C}[p(pr)]$
(Save)	α_{14}	:	$\mathsf{C}[p(qr)]$	$\Rightarrow$	$\mathsf{C}[q(rr)]$
(Restore)	α_{15}	:	$\mathsf{C}[p(qr)]$	$\Rightarrow$	$\mathsf{C}[qp]$

These figures suggest the following scenario, which is that logical conditions are actually attached to pieces of data—and even to particular *closures* (as we shall see in Section 6)—within the currently executed process, and that the evolution of the current logical condition pr (which is given by the type $\mathsf{C}[pr]$ of the current computational condition) simply reflects the move of these pieces of data during evaluation. More precisely:

— The type $\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[(pq)r]$ of combinator α_6 reflects the fact that, during a (Grab) step, the first element of the stack (the condition q is attached to) is removed from the stack (the condition r is attached to) and then incorporated in the environment of the term t (the condition p is attached to).

— The type $\mathsf{C}[pr] \Rightarrow \mathsf{C}[p(pr)]$ of combinator α_{11} reflects the fact that, during a (Push)

§ In Krivine's work (Kri08; Kri10), the current computational condition c is actually stored in the very last slot of the stack, using two specific instructions χ and χ' swapping the first and last elements of the current stack. Our work shows that we can do the same by reserving the first slot of the stack instead of the last one, thus removing the need of the two extra instructions χ and χ'.

step, the environment of the currently executed term (the condition p is attached to) is duplicated and that a closure containing a copy of it is then put on the top of the stack (the condition r is attached to).

— The type $\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[q(rr)]$ of combinator α_{14} reflects the fact that, during a (SAVE) step, the first element of the stack (the continuation q is attached to) is given the control, while a continuation constant is built from the rest of the stack (the condition r is attached to) and put on the top of the stack, while being labelled with the same condition r the stack it comes from. Note that during this operation, the condition p attached to $\mathbb{c}$ is simply dropped.

— Finally, the type $\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[qp]$ of combinator α_{15} reflects the fact that, during a (RESTORE) step, the first argument of the stack (the condition q is attached to) is given the control, while the rest of the stack is replaced by the stack embedded in the continuation (the condition p is attached to). In particular, the initial stack (the condition r is attached to) is dropped, as well as the corresponding condition r.

In the author's opinion, all these features are reminiscent from well-known techniques in computer architecture, such as *virtualization* or *protection rings*, that allow the system to execute a program by only giving it access to a part of the resources (here: the tail of the stack), thus allowing the system to maintain and update critical information (here: the computational condition) in the back of the executed program. The difference is that in our framework, these features are implemented via a particular program transformation, whereas in most computer architectures, these features are hardwired using the memory virtualization facilities and the multiple protection rings provided by modern processors.

In Section 6, we shall see how to put the forcing transformation 'into the hardware' in order to avoid the cost of the program transformation $t \mapsto t^*$. But before doing that, we first need to present the classical realizability semantics of system PAω^+.

5. Classical realizability semantics

In Sections 3 and 4, we have shown how to reformulate the theory of Cohen forcing in system PAω^+, and we analyzed the computational meaning on the underlying translation at the level of proof terms. But in order to relate the computational behavior of proof terms with their types—that is: with the propositions which they prove—we now need to introduce the classical realizability semantics underlying system PAω^+. For that, we shall work in the general framework of *classical realizability algebras*, that has been introduced in (Kri10) to study the connections between realizability and forcing.

Before introducing the notion of a classical realizability algebra, we first need to introduce the notion of an environment.

5.1. *Environments*

Definition 40 (A-environments). — Given a fixed set A, we call an A-*environment* any finite list of bindings of the form $\sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]$, where $x_1, \ldots, x_n$ are proof variables and where $a_1, \ldots, a_n$ are elements of A.

Note that in this definition, we do not require that the variables $x_1, \ldots, x_n$ are pairwise distinct, so that the same variable x can be bound several times within an A-environment σ. In this case, we adopt the convention that only the rightmost binding of x is active in σ, the previous bindings of x being hidden by the last one.

Given an A-environment σ, we denote by $\mathrm{dom}(\sigma)$ the (finite) set of variables that are bound to an element of A in σ, and for every variable $x \in \mathrm{dom}(\sigma)$, we write $\sigma(x)$ the element of A that is bound to x in σ, always considering the rightmost binding when the variable x is bound several times in σ. The empty A-environment is denoted by $\emptyset$, whereas the concatenation of two A-environments σ and σ' is written σ, σ'.

5.2. *Classical realizability algebras*

Classical realizability algebras generalize the λ_c-calculus in the same way as *partial combinatory algebras* generalize λ-terms (or Gödel codes of recursive functions) in intuitionistic realizability. However, Krivine's original presentation (Kri10) relies on a particular encoding of the λ_c-calculus to a variant of combinatory logic (with control operators) that is suited to the call-by-name discipline of the KAM. In this section, we give a slightly different presentation of classical realizability algebras where most of Krivine's combinators are abstracted via a *compilation function* (from proof terms to the elements of the realizability algebra) that now constitutes a parameter of the algebra. Our presentation is more suited to the abstract machine we shall present in Section 6, but it is a simple exercise to show that it is equivalent to Krivine's.

Definition 41 (Classical realizability algebras). — A *classical realizability algebra* is a structure $\mathscr{A}$ given by:

— Three sets $\mathbf{\Lambda}$, $\mathbf{\Pi}$ and $\mathbf{\Lambda} \star \mathbf{\Pi}$, whose elements are respectively called the *$\mathscr{A}$-terms*, the *$\mathscr{A}$-stacks* and the *$\mathscr{A}$-processes.*
— An operation $(a, \pi) \mapsto a \cdot \pi$ from $\mathbf{\Lambda} \times \mathbf{\Pi}$ to $\mathbf{\Pi}$ ('consing').
— An operation $(a, \pi) \mapsto a \star \pi$ from $\mathbf{\Lambda} \times \mathbf{\Pi}$ to $\mathbf{\Lambda} \star \mathbf{\Pi}$ ('process formation').
— An operation $\pi \mapsto \mathsf{k}_\pi$ from $\mathbf{\Pi}$ to $\mathbf{\Lambda}$ ('continuation formation').
— A *compilation function* $(t, \sigma) \mapsto t[\sigma]$ that takes an open proof term t together with a $\mathbf{\Lambda}$-environment σ such that $FV(t) \subseteq \mathrm{dom}(\sigma)$, and builds an $\mathscr{A}$-term $t[\sigma] \in \mathbf{\Lambda}$.
— A set of processes $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$, called the *pole* of $\mathscr{A}$, such that:

- $\sigma(x) \star \pi \in \perp\!\!\!\perp$ implies $x[\sigma] \star \pi \in \perp\!\!\!\perp$; (if $x \in \mathrm{dom}(\sigma)$)
- $t[\sigma] \star u[\sigma] \cdot \pi \in \perp\!\!\!\perp$ implies $(tu)[\sigma] \star \pi \in \perp\!\!\!\perp$; (if $FV(tu) \subseteq \mathrm{dom}(\sigma)$)
- $t[\sigma, x := a] \star \pi \in \perp\!\!\!\perp$ implies $(\lambda x \,.\, t)[\sigma] \star a \cdot \pi$; (if $FV(\lambda x \,.\, t) \subseteq \mathrm{dom}(\sigma)$)
- $a \star \mathsf{k}_\pi \cdot \pi \in \perp\!\!\!\perp$ implies $\mathbb{c}\!\!\mathbb{c}[\sigma] \star a \cdot \pi \in \perp\!\!\!\perp$; and
- $a \star \pi \in \perp\!\!\!\perp$ implies $\mathsf{k}_\pi \star a \cdot \pi' \in \perp\!\!\!\perp$

for all (open) terms t, u, for all substitutions σ, and for all $a \in \mathbf{\Lambda}$, $\pi, \pi' \in \mathbf{\Pi}$.

Remarks 42.

(1) The above definition generalizes the λ_c-calculus by abstracting the syntactic constituents of the calculus via three (abstract) sets $\mathbf{\Lambda}$, $\mathbf{\Pi}$, $\mathbf{\Lambda} \star \mathbf{\Pi}$ and three (abstract)

operations $(t,\pi) \mapsto t \cdot \pi$, $(t,\pi) \mapsto t \star \pi$ and $\pi \mapsto \mathsf{k}_\pi$. Thanks to this abstraction, classical realizability algebras can be used to encapsulate other classical λ-calculi such as Parigot's $\lambda\mu$-calculus (Par97), Barbanera and Berardi's symmetric λ-calculus (BB96) or Curien and Herbelin $\bar{\lambda}\mu$-calculus (CH00), so that the theory of classical realizability can be extended to these classical λ-calculi as well.

(2) However, the above definition does not assume that the sets $\mathbf{\Lambda}$, $\mathbf{\Pi}$ and $\mathbf{\Lambda} \star \mathbf{\Pi}$ are denumerable. Indeed, classical realizability algebras are not limited to the encapsulation of syntactic entities, and they can be used to encapsulate semantic entities as well. We shall see in Section 7.1 an important example of a classical realizability algebra $\mathscr{A}^*$ whose constituents ($\mathscr{A}^*$-terms, $\mathscr{A}^*$-stacks and $\mathscr{A}^*$-processes) encapsulate semantic forcing conditions coming from a (previously defined) realizability model.

(3) The compilation function $(t,\sigma) \mapsto t[\sigma]$ comes with no particular equation. Even in the case where the proof term t is closed, the compiled $\mathscr{A}$-term $t[\sigma] \in \mathbf{\Lambda}$ may still depend on the environment σ—and even on the ordering of the bindings in σ. (Such dependencies will naturally appear in the classical realizability algebra $\mathscr{A}$ defined from the KFAM in regular mode, cf Section 6.2.)

(4) Unlike the λ_c-calculus, the set $\mathbf{\Lambda} \star \mathbf{\Pi}$ of $\mathscr{A}$-processes does not formally come with a binary relation of evaluation. (Remember that $\mathscr{A}$-processes are not necessarily syntactic entities.) To understand the conditions on the pole $\perp\!\!\!\perp \subseteq \mathbf{\Lambda}\star\mathbf{\Pi}$, it is however convenient to introduce a binary relation $\succ_{\mathscr{A}}$ of (pseudo-)evaluation between $\mathscr{A}$-processes, that is the preorder generated from the five rules

$$\begin{array}{lrcll}
(\textsc{Lookup}_{\mathscr{A}}) & x[\sigma] \star \pi & \succ_{\mathscr{A}} & \sigma(x) \star \pi & (x \in \mathrm{dom}(\sigma)) \\
(\textsc{Push}_{\mathscr{A}}) & (tu)[\sigma] \star \pi & \succ_{\mathscr{A}} & t[\sigma] \star u[\sigma] \cdot \pi & (FV(tu) \subseteq \mathrm{dom}(\sigma)) \\
(\textsc{Grab}_{\mathscr{A}}) & (\lambda x\,.\,t)[\sigma] \star a \cdot \pi & \succ_{\mathscr{A}} & t[\sigma, x := a] \star \pi & (FV(\lambda x\,.\,t) \subseteq \mathrm{dom}(\sigma)) \\
(\textsc{Save}_{\mathscr{A}}) & \mathsf{c\!c}[\sigma] \star a \cdot \pi & \succ_{\mathscr{A}} & a \star \mathsf{k}_\pi \cdot \pi & \\
(\textsc{Restore}_{\mathscr{A}}) & \mathsf{k}_\pi \star a \cdot \pi' & \succ_{\mathscr{A}} & a \star \pi &
\end{array}$$

With this definition, the conditions on the pole $\perp\!\!\!\perp$ simply express the fact that the set $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$ is saturated w.r.t. the relation $\succ_{\mathscr{A}}$, in the following sense:

Definition 43 (Saturated sets). — Given a set $\mathbf{P}$ equipped with a binary relation $\succ$, we say that a subset $\perp\!\!\!\perp \subseteq \mathbf{P}$ is *saturated* (w.r.t. the relation $\succ$) when the two conditions $p \succ p'$ and $p' \in \perp\!\!\!\perp$ together imply $p \in \perp\!\!\!\perp$ for all $p, p' \in \mathbf{P}$.

In the case where the binary relation $\succ$ denotes some notion of evaluation, we also say that the subset $\perp\!\!\!\perp \subseteq \mathbf{P}$ is *closed under anti-evaluation.*

5.3. Λ_c-*algebras*

In traditional presentations of classical realizability (Kri03; Kri09; Miq10), realizability models are parameterized by a particular instance of the λ_c-calculus (defined from the sets $\mathcal{K}$, Π_0 and the binary relation $\succ$) and by a saturated set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ (w.r.t. the relation $\succ$) that is used as the pole of the construction.

This construction of a classical realizability model from the λ_c-calculus—and formally:

from the four parameters $\mathcal{K}$, Π_0, $\succ$ and $⫫$—is actually a particular case of the construction we shall present in Section 5.6, that corresponds to the choice of a classical realizability algebra $\mathscr{A} = (\mathbf{\Lambda}, \mathbf{\Pi}, \mathbf{\Lambda} \star \mathbf{\Pi}, \ldots, ⫫)$ defined as follows:

— The sets $\mathbf{\Lambda}$, $\mathbf{\Pi}$ and $\mathbf{\Lambda} \star \mathbf{\Pi}$ are respectively defined as the sets Λ, Π and $\Lambda \star \Pi$ induced by the set of instructions $\mathcal{K}$ and the set of stack bottoms Π_0 (Def. 14 and 15).
— The three operations $(t, \pi) \mapsto t \cdot \pi$, $(t, \pi) \mapsto t \star \pi$ and $\pi \mapsto \mathsf{k}_\pi$ are defined in the realizability algebra $\mathscr{A}$ as in the λ_c-calculus.
— The compilation function $(t, \sigma) \mapsto t[\sigma]$ of the algebra $\mathscr{A}$ is defined by

$$t[\sigma] \;\equiv\; t\{x_1 := \sigma(x_1); \ldots ; x_n := \sigma(x_n)\}\,,$$

where $x_1, \ldots, x_n$ are the free variables of t.
— The pole of $\mathscr{A}$ is the saturated set $⫫$.

In what follows, we shall call a *λ_c-algebra* any classical realizability algebra of this form.

5.4. *Interpreting kinds*

From now on, we work in a fixed algebra $\mathscr{A} = (\mathbf{\Lambda}, \mathbf{\Pi}, \mathbf{\Lambda} \star \mathbf{\Pi}, \ldots, ⫫)$.

We call a *falsity value* any set of $\mathscr{A}$-stacks $S \subseteq \mathbf{\Pi}$. Every falsity value $S \subseteq \mathbf{\Pi}$ induces a *truth value* $S^{⫫} \subseteq \mathbf{\Lambda}$ that is the set of $\mathscr{A}$-terms defined by

$$S^{⫫} \;=\; \{a \in \mathbf{\Lambda} \;:\; \forall \pi \in S\ (a \star \pi) \in ⫫\}\,.$$

From this definition, it is clear that the larger the falsity value S, the smaller the corresponding truth value $S^{⫫}$, and vice-versa.

In classical realizability, individuals are interpreted as natural numbers, propositions as falsity values and higher-order functions as set-theoretic functions, that is:

$$[\![\iota]\!] = \mathbb{N}, \qquad [\![o]\!] = \mathfrak{P}(\mathbf{\Pi}), \qquad [\![\tau \to \sigma]\!] \;=\; [\![\sigma]\!]^{[\![\tau]\!]}\,.$$

Since the denotation of a kind τ depends on the choice of the classical realizability algebra $\mathscr{A}$, we shall write if sometimes $[\![\tau]\!]^{\mathscr{A}}$ to recall the dependency.

The classical realizability model $\mathscr{M}_{\mathscr{A}}$ induced by the classical realizability algebra $\mathscr{A}$ is defined as the union of the sets $[\![\tau]\!]^{\mathscr{A}}$ for all kinds τ.

5.5. *Valuations and higher-order terms with parameters*

A *valuation* (in the model $\mathscr{M}_{\mathscr{A}}$) is a partial function ρ from the set of higher-order variables to $\mathscr{M}_{\mathscr{A}}$ such that $\rho(x^\tau) \in [\![\tau]\!]$ for every variable $x^\tau \in \mathrm{dom}(\rho)$. We say that a valuation ρ is *finite* when its domain $\mathrm{dom}(\rho)$ is finite, and that it is *total* when its domain is the set of all higher-order variables. Given a valuation ρ, a variable x^τ and a denotation $v \in [\![\tau]\!]$, we write

$$\rho, x^\tau \leftarrow v \;=\; \rho_{|\mathrm{dom}(\rho)\setminus\{x^\tau\}} \cup \{x^\tau \leftarrow v\}$$

the valuation obtained by rebinding the variable x^τ to the denotation $v \in [\![\tau]\!]$ in ρ, possibly erasing the value formerly bound to x^τ in the case where $x^\tau \in \mathrm{dom}(\rho)$.

A *higher-order term with parameters in $\mathscr{M}_{\mathscr{A}}$* is a pair $M[\rho]$ formed by a higher-order term M and a valuation ρ. A higher-order term $M[\rho]$ with parameters in $\mathscr{M}_{\mathscr{A}}$ is *closed* when all the free variables of M are bound in ρ, in the sense that $FV(M) \subseteq \mathrm{dom}(\rho)$.

It is convenient to think of higher-order terms with parameters in $\mathscr{M}_{\mathscr{A}}$ as higher-order terms enriched with a constant symbol $\dot{v}$ of kind τ for every denotation $v \in [\![\tau]\!]$ and for every kind τ. The constant symbol $\dot{v}$ can be seen as a notation for the higher-order term $x^\tau[x^\tau \leftarrow v]$ with a single parameter $v \in [\![\tau]\!]$ (where x^τ is a fresh variable of kind τ), and we shall more generally use the notation

$$\dot{F}\,\dot{v}_1 \Rightarrow \dot{F}\,\dot{v}_2 \qquad\qquad (F \in [\![\tau \to o]\!],\ v_1, v_2 \in [\![\tau]\!])$$

to denote the higher-order term with parameters in $\mathscr{M}_{\mathscr{A}}$

$$(y^{\tau\to o}\,x_1^\tau \Rightarrow y^{\tau\to o}\,x_2^\tau)[y^{\tau\to o} \leftarrow F,\ x_1^\tau \leftarrow v_1,\ x_2^\tau \leftarrow v_2]\,.$$

With these notations, we can work with higher-order terms with parameters in $\mathscr{M}_{\mathscr{A}}$ using the same notations as for higher-order terms, thus leaving the valuation implicit.

In what follows, we shall say that a valuation ρ *closes* a higher-order term M (resp. an equational theory $\mathcal{E}$, a context Γ) when the higher-order term $M[\rho]$ with parameters is closed (resp. when $M_1[\rho]$ and $M_2[\rho]$ are closed for every equation $(M_1 = M_2) \in \mathcal{E}$, when $A[\rho]$ is closed for every declaration $(x : A) \in \Gamma$).

5.6. *Interpreting higher-order terms*

Every closed higher-order term $M[\rho]$ of kind τ with parameters in $\mathscr{M}_A$ is interpreted as an element $[\![M[\rho]]\!] \in [\![\tau]\!]$ that is defined by induction on the structure of the underlying higher-order term M. Variables are interpreted the obvious way whereas abstractions, applications and arithmetic constructions are interpreted by their obvious set-theoretic equivalents:

$$\begin{aligned} [\![x[\rho]]\!] &= \rho(x) & [\![0[\rho]]\!] &= 0 \\ [\![(\lambda x^\tau . M)[\rho]]\!] &= (v \in [\![\tau]\!] \mapsto [\![M[\rho, x^\tau \leftarrow v]]\!]) & [\![s[\rho]]\!] &= n \mapsto n+1 \\ [\![(MN)[\rho]]\!] &= [\![M[\rho]]\!]([\![N[\rho]]\!]) & [\![\mathrm{rec}_\tau[\rho]]\!] &= \mathrm{rec}_{[\![\tau]\!]} \end{aligned}$$

(where $\mathrm{rec}_{[\![\tau]\!]}$ denotes the expected set-theoretic recursor over the set $[\![\tau]\!]$). Implication and universal quantification (at every kind τ) are given their standard negative interpretation following (Kri09), whereas equational implication is interpreted as expected:

$$\begin{aligned} [\![(A \Rightarrow B)[\rho]]\!] &= [\![A[\rho]]\!]^{\perp\!\!\!\perp} \cdot [\![B[\rho]]\!] \ = \ \big\{a \cdot \pi \ :\ a \in [\![A[\rho]]\!]^{\perp\!\!\!\perp},\ \pi \in [\![B[\rho]]\!]\big\} \\ [\![(\forall x^\tau A)[\rho]]\!] &= \bigcup_{v \in [\![\tau]\!]} [\![A[\rho, x^\tau \leftarrow v]]\!] \end{aligned}$$

$$[\![(M_1 = M_2 \mapsto A)[\rho]]\!] = \begin{cases} [\![A[\rho]]\!] & \text{if } [\![M_1[\rho]]\!] = [\![M_2[\rho]]\!] \qquad \text{(equality of denotations)} \\ \varnothing & \text{otherwise} \qquad\qquad\qquad\qquad \text{(true formula)} \end{cases}$$

We immediately check the following:

Lemma 44 (Dependence w.r.t. free variables). — Let M be a higher-order term of kind τ, and ρ, ρ' two valuations closing M. If $\rho(x^\sigma) = \rho'(x^\sigma)$ for every free variable $x^\sigma \in FV(M)$, then $[\![M[\rho]]\!] = [\![M[\rho']]\!]$.

Proof. Obvious, by induction on the structure of M. □

Lemma 45 (Substitutivity). — Let M of kind τ and N of kind σ. For all valuations ρ closing $M\{x^\sigma := N\}$ and N: $[\![(M\{x^\sigma := N\})[\rho]]\!] = [\![M[\rho, x^\sigma \leftarrow [\![N[\rho]]\!]]]\!]$.

Proof. By induction on the structure of M. □

Using our abbreviations for higher-order terms with parameters in $\mathscr{M}_{\mathscr{A}}$ (Section 5.5), we can rephrase the interpretation function of closed higher-order terms (with parameters in $\mathscr{M}_{\mathscr{A}}$) using the following lighter notations:

$$\begin{aligned}
[\![\dot v]\!] &= v & [\![0]\!] &= 0\\
[\![\lambda x^\tau . M]\!] &= (v \in [\![\tau]\!] \mapsto [\![M\{x^\tau := \dot v\}]\!]) & [\![s]\!] &= n \mapsto n+1\\
[\![MN]\!] &= [\![M]\!]\big([\![N]\!]\big) & [\![\mathrm{rec}_\tau]\!] &= \mathrm{rec}_{[\![\tau]\!]}\\
[\![A \Rightarrow B]\!] &= [\![A]\!]^{\perp\!\!\!\perp} \cdot [\![B]\!] \quad = \quad \big\{a \cdot \pi \ :\ a \in [\![A]\!]^{\perp\!\!\!\perp},\ \pi \in [\![B]\!]\big\}\\
[\![\forall x^\tau A]\!] &= \bigcup_{v \in [\![\tau]\!]} [\![A\{x^\tau := \dot v\}]\!]
\end{aligned}$$

$$[\![M_1 = M_2 \mapsto A]\!] = \begin{cases} [\![A]\!] & \text{if } [\![M_1]\!] = [\![M_2]\!] \qquad \text{(equality of denotations)}\\ \varnothing & \text{otherwise} \qquad\qquad\qquad\qquad \text{(true formula)}\end{cases}$$

Note that in the interpretation of $\lambda x^\tau . M$ and $\forall x^\tau A$, we substitute the variable x^τ with the constant $\dot v$ associated to every denotation $v \in [\![\tau]\!]$ instead of rebinding x^τ with v in the hidden valuation. Of course, this abuse of notations is legitimated by Lemma 45.

Again, the denotation $[\![M[\rho]]\!]$ of the closed higher-order term $M[\rho]$ with parameters in $\mathscr{M}_{\mathscr{A}}$ depends on the choice of the classical realizability algebra $\mathscr{A}$, so that we shall write it sometimes $[\![M[\rho]]\!]^{\mathscr{A}}$ to recall the dependency. Given a closed proposition A (with parameters in $\mathscr{M}_{\mathscr{A}}$) and an $\mathscr{A}$-term $a \in \mathbf{\Lambda}$, we say that a *realizes* A and write $a \Vdash_{\mathscr{A}} A$ when $a \in [\![A]\!]^{\perp\!\!\!\perp}$, that is:

$$a \Vdash_{\mathscr{A}} A \quad \equiv \quad \forall \pi \in [\![A]\!]^{\mathscr{A}} \ \ a \star \pi \in \perp\!\!\!\perp$$

With these notations, we can already check that the instruction $\mathbf{c\!c}$, when compiled in the algebra $\mathscr{A}$ using an arbitrary $\mathbf{\Lambda}$-environment σ, provides an $\mathscr{A}$-term $\mathbf{c\!c}[\sigma]$ that realizes Peirce's law. (Note that the $\mathscr{A}$-term $\mathbf{c\!c}[\sigma]$ may depend on the environment σ, depending on the actual implementation of the compilation function $(t, \sigma) \mapsto t[\sigma]$.)

Lemma 46 (Realizing Peirce's law). — Let A and B be two closed propositions with parameters in $\mathscr{M}_{\mathscr{A}}$.

(1) For every $\mathscr{A}$-stack $\pi \in [\![A]\!]$, we have $\mathsf{k}_\pi \Vdash_{\mathscr{A}} A \Rightarrow B$.
(2) For every $\mathbf{\Lambda}$-environment σ, we have $\mathbf{c\!c}[\sigma] \Vdash_{\mathscr{A}} ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$.

Proof. (1) Let $\pi \in [\![A]\!]$. To prove that $\mathsf{k}_\pi \Vdash_{\mathscr{A}} A \Rightarrow B$, it suffices to check that

$\mathsf{k}_\pi \star a \cdot \pi' \in \perp\!\!\!\perp$ for all $a \in [\![A]\!]^{\perp\!\!\!\perp}$ and $\pi' \in [\![B]\!]$. But since $\mathsf{k}_\pi \star a \cdot \pi' \succ_{\mathscr{A}} a \star \pi$ (Restore$_{\mathscr{A}}$) and since $a \star \pi \in \perp\!\!\!\perp$, we get $\mathsf{k}_\pi \star a \cdot \pi' \in \perp\!\!\!\perp$ by saturation.

(2) To prove that $\text{œ}[\sigma] \Vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$ (where σ is any $\mathbf{\Lambda}$-environment), we need to check that $\text{œ}[\sigma] \star a \cdot \pi \in \perp\!\!\!\perp$ for all $a \in [\![(A \Rightarrow B) \Rightarrow A]\!]^{\perp\!\!\!\perp}$ and $\pi \in [\![A]\!]$. From (1) we get $\mathsf{k}_\pi \in [\![A \Rightarrow B]\!]^{\perp\!\!\!\perp}$, hence $\mathsf{k}_\pi \cdot \pi \in [\![(A \Rightarrow B) \Rightarrow A]\!]$, and thus $a \star \mathsf{k}_\pi \cdot \pi \in \perp\!\!\!\perp$. But since $\text{œ}[\sigma] \star a \cdot \pi \succ_{\mathscr{A}} a \star \mathsf{k}_\pi \cdot \pi \in \perp\!\!\!\perp$ (Save$_{\mathscr{A}}$), we get $\text{œ}[\sigma] \star a \cdot \pi \in \perp\!\!\!\perp$ by saturation. □

Finally, we say that a valuation ρ *models* an equational theory $\mathcal{E}$ (Def. 3) and write $\rho \models \mathcal{E}$ when ρ closes $\mathcal{E}$ and when $[\![M_1[\rho]]\!] = [\![M_2[\rho]]\!]$ for every equation $(M_1 = M_2) \in \mathcal{E}$.

Lemma 47 (Soundness of conversion). — If the conversion $M_1 \cong_{\mathcal{E}} M_2$ is derivable from the rules of Table. 1, then for every valuation ρ closing M_1 and M_2 and such that $\rho \models \mathcal{E}$, we have $[\![M_1[\rho]]\!] = [\![M_2[\rho]]\!]$.

Proof. By induction on the derivation of $M_1 \cong_{\mathcal{E}} M_2$, using Lemma 45. □

5.7. *The general property of adequacy*

Given a closed context Γ with parameters in $\mathscr{M}_{\mathscr{A}}$, we say that a $\mathbf{\Lambda}$-environment σ *realizes* the closed context Γ (with parameters in $\mathscr{M}_{\mathscr{A}}$) and write $\sigma \Vdash_{\mathscr{A}} \Gamma$ when:

— $\mathrm{dom}(\sigma) \supseteq \mathrm{dom}(\Gamma)$, and

— $\sigma(x) \Vdash_{\mathscr{A}} A$ for every declaration $(x : A) \in \Gamma$.

With this notation, the general property of adequacy w.r.t. an arbitrary classical realizability algebra $\mathscr{A}$ can be stated as follows:

Proposition 48 (Adequacy). — If the typing judgement $\mathcal{E}; \Gamma \vdash t : A$ is derivable from the rules of Table 2, then for all total valuations ρ such that $\rho \models \mathcal{E}$ and for all $\mathbf{\Lambda}$-environments $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$, we have $t[\sigma] \Vdash_{\mathscr{A}} A[\rho]$.

Proof. We prove the property by induction on the derivation of $\Gamma \vdash t : A$, distinguishing cases depending on the last applied rule.

— Axiom. The conclusion $\mathcal{E}; \Gamma \vdash x : A$ comes from the side condition $(x : A) \in \Gamma$. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and a $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $x[\sigma] \Vdash_{\mathscr{A}} A[\rho]$, let take an $\mathscr{A}$-stack $\pi \in [\![A[\rho]]\!]$ and let us show that $x[\sigma] \star \pi \in \perp\!\!\!\perp$. From the side condition $(x : A) \in \Gamma$ and the assumption $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$, we have $\sigma(x) \Vdash_{\mathscr{A}} A[\rho]$, so that $\sigma(x) \star \pi \in \perp\!\!\!\perp$. But since

$$x[\sigma] \star \pi \succ_{\mathscr{A}} \sigma(x) \star \pi \qquad (\textsc{Lookup}_{\mathscr{A}})$$

and since $\sigma(x) \star \pi \in \perp\!\!\!\perp$, we get $x[\sigma] \star \pi \in \perp\!\!\!\perp$ by saturation.

— Conversion. This case immediately follows from Lemma 47.

— Introduction of $\Rightarrow$. The judgment $\mathcal{E}; \Gamma \vdash \lambda x \,.\, t : A \Rightarrow B$ comes from a unique premise $\mathcal{E}; \Gamma, x : A \vdash t : B$. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and a $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $(\lambda x \,.\, t)[\sigma] \Vdash_{\mathscr{A}} (A \Rightarrow B)[\rho]$, take an arbitrary $\mathscr{A}$-stack in $[\![(A \Rightarrow B)[\rho]]\!]$—that is, an $\mathscr{A}$-stack of the form $a \cdot \pi$ where $a \Vdash_{\mathscr{A}} A[\rho]$ and $\pi \in [\![B[\rho]]\!]$—and let us show that $(\lambda x \,.\, t)[\sigma] \star a \cdot \pi \in \perp\!\!\!\perp$. We have:

$$(\lambda x \,.\, t)[\sigma] \star a \cdot \pi \;\succ_{\mathscr{A}}\; t[\sigma, x := a] \star \pi \,. \qquad (\textsc{Grab}_{\mathscr{A}})$$

It is easy to check that $(\sigma, x := a) \Vdash_{\mathscr{A}} (\Gamma, x : A)[\rho]$, and since $\pi \in [\![B[\rho]]\!]$ we get $t[\sigma, x := a] \star \pi \in \perp\!\!\!\perp$ from the induction hypothesis. We can then conclude that $(\lambda x\,.\,t)[\sigma] \star a \cdot \pi \in \perp\!\!\!\perp$ by saturation.

— Elimination of $\Rightarrow$. The judgment $\mathcal{E}; \Gamma \vdash tu : B$ ending the derivation comes from two premises $\mathcal{E}; \Gamma \vdash t : A \Rightarrow B$ and $\mathcal{E}; \Gamma \vdash u : A$. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and a $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $(tu)[\sigma] \Vdash_{\mathscr{A}} B[\rho]$, take an arbitrary $\mathscr{A}$-stack $\pi \in [\![B[\rho]]\!]$, and let us show that $(tu)[\sigma] \star \pi \in \perp\!\!\!\perp$. We have

$$(tu)[\sigma] \star \pi \ \succ_{\mathscr{A}}\ t[\sigma] \star u[\sigma] \cdot \pi\,. \qquad (\textsc{Push}_{\mathscr{A}})$$

From the induction hypothesis applied to the second premise, we have $u[\sigma] \Vdash_{\mathscr{A}} A[\rho]$, and since $\pi \in [\![B[\rho]]\!]$ we get $u[\sigma] \cdot \pi \in [\![(A \Rightarrow B)[\rho]]\!]$. Now applying the induction hypothesis to the first premise, we get $t[\sigma] \Vdash (A \Rightarrow B)[\rho]$, so that $t[\sigma] \star u[\sigma] \cdot \pi \in \perp\!\!\!\perp$. Hence $(tu)[\sigma] \star \pi \in \perp\!\!\!\perp$ by saturation.

— Introduction of $\mapsto$. The judgment $\mathcal{E}; \Gamma \vdash t : M_1 = M_2 \mapsto A$ comes from a unique premise $(\mathcal{E}, M_1 = M_2); \Gamma \vdash t : A$. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and a $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $t[\sigma] \Vdash_{\mathscr{A}} (M_1 = M_2 \mapsto A)[\rho]$, we distinguish the following two cases:

- $[\![M_1[\rho]]\!] = [\![M_2[\rho]]\!]$. In this case, we have $[\![(M_1 = M_2 \mapsto A)[\rho]]\!] = [\![A[\rho]]\!]$. But since $\rho \models (\mathcal{E}, M_1 = M_2)$, we get $t[\sigma] \Vdash_{\mathscr{A}} A[\rho]$ from the induction hypothesis, hence $t[\sigma] \Vdash_{\mathscr{A}} (M_1 = M_2 \mapsto A)[\rho]$.
- $[\![M_1[\rho]]\!] \neq [\![M_2[\rho]]\!]$. In this case, we have $[\![(M_1 = M_2 \mapsto A)[\rho]]\!] = \varnothing$, so that the desired relation $t[\sigma] \Vdash_{\mathscr{A}} (M_1 = M_2 \mapsto A)[\rho]$ holds vacuously.

— Elimination of $\mapsto$. This case is a particular case of conversion.

— Introduction of $\forall$. The judgment $\mathcal{E}; \Gamma \vdash t : \forall x^\tau A$ comes from a premise $\mathcal{E}; \Gamma \vdash t : A$, where $x^\tau \notin FV(\mathcal{E}; \Gamma)$. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and a $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $t[\sigma] \Vdash_{\mathscr{A}} (\forall x^\tau A)[\rho]$, take an $\mathscr{A}$-stack $\pi \in [\![(\forall x^\tau A)[\rho]]\!]$—that is, a stack $\pi \in [\![A[\rho, x^\tau \leftarrow v]]\!]$ for some $v \in [\![\tau]\!]$—and let us show that $t[\sigma] \star \pi \in \perp\!\!\!\perp$. Let us consider the valuation $\rho' = \rho, x^\tau \leftarrow v$. Since $x^\tau \notin FV(\mathcal{E})$, we have $[\![M_1[\rho']]\!] = [\![M_1[\rho]]\!] = [\![M_2[\rho]]\!] = [\![M_2[\rho']]\!]$ for every equation $(M_1 = M_2) \in \mathcal{E}$, so that $\rho' \models \mathcal{E}$. And since $x^\tau \notin FV(\Gamma)$, we have $[\![B[\rho']]\!] = [\![B[\rho]]\!]$ for every declaration $(z : B) \in \Gamma$, hence $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho']$. By induction hypothesis we thus get $t[\sigma] \Vdash_{\mathscr{A}} A[\rho']$, hence $t[\sigma] \star \pi \in \perp\!\!\!\perp$.

— Elimination of $\forall$. The judgment $\mathcal{E}; \Gamma \vdash t : A\{x^\tau := N\}$ comes from a unique premise $\mathcal{E}; \Gamma \vdash t : \forall x^\tau A$, where N is of kind τ. Let us consider a total valuation ρ such that $\rho \models \mathcal{E}$ and $\mathbf{\Lambda}$-environment $\sigma \Vdash_{\mathscr{A}} \Gamma[\rho]$. To show that $t[\sigma] \Vdash_{\mathscr{A}} (A\{x^\tau := N\})[\rho]$, take an arbitrary $\mathscr{A}$-stack $\pi \in [\![(A\{x^\tau := N\})[\rho]]\!] = [\![A[\rho, x^\tau \leftarrow [\![N[\rho]]\!]]]\!]$ (by Lemma 45), and let us show that $t[\sigma] \star \pi \in \perp\!\!\!\perp$. We have

$$\pi \ \in \ \bigcup_{v \in [\![\tau]\!]} [\![A[\rho, x^\tau \leftarrow v]]\!] \ = \ [\![(\forall x^\tau A)[\rho]]\!]$$

(taking $v = [\![N[\rho]]\!]$), so that by induction hypothesis we get $t[\sigma] \star \pi \in \perp\!\!\!\perp$.

— Peirce's law. Obvious by Lemma 46. □

5.8. *Adequacy w.r.t. the KAM*

In the particular case where the classical realizability algebra $\mathscr{A}$ is a λ_c-algebra (Section 5.3) induced by a particular instance $(\mathcal{K}, \Pi_0, \succ)$ of the λ_c-calculus and by a particular saturated set $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$, the general property of adequacy (Prop. 48) reduces to the following statement:

Corollary 49 (Adequacy w.r.t. the KAM). — If the judgment

$$\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

is derivable in system PAω^+, then for all total valuations ρ such that $\rho \models \mathcal{E}$ and for all realizers $u_1 \Vdash_{\mathscr{A}} A_1[\rho]$, …, $u_n \Vdash_{\mathscr{A}} A_n[\rho]$, we have

$$t\{x_1 := u_1; \ldots; x_n := u_n\} \Vdash_{\mathscr{A}} A[\rho]\,.$$

Up to the fact that we work in higher-order arithmetic (rather than in second-order arithmetic) and that the typing judgment $\mathcal{E}; \Gamma \vdash t : A$ now depends on an equational theory $\mathcal{E}$, the above statement is exactly the property of adequacy in the sense of the traditional presentation of classical realizability (Kri03; Kri09; Miq10).

Coming back to the forcing translation defined in Section 4, we can combine the above corollary with Theorem 32, so that we get:

Corollary 50 (Adequacy w.r.t. forcing). — If the judgment

$$\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

is derivable in system PAω^+, then for all total valuations ρ such that $\rho \models \mathcal{E}^*$ and for all realizers $u_1 \Vdash_{\mathscr{A}} (p \Vdash A_1)[\rho]$, …, $u_n \Vdash_{\mathscr{A}} (p \Vdash A_n)[\rho]$, we have

$$t^*\{x_1 := u_1; \ldots; x_n := u_n\} \Vdash_{\mathscr{A}} (p \Vdash A)[\rho]$$

(where p is a fresh condition variable).

However, the main drawback of the above result is that we need to transform the proof term t of A in order to turn it into a realizer of the proposition $p \Vdash A$. If we want to see forcing as an *extension* of provability in system PAω^+, we should not change the proof term t, but we should rather extend the KAM to make it understand forcing directly.

Let us now see how forcing can be hard-wired into Krivine's Abstract Machine.

6. An abstract machine for forcing

We now present an extension of Krivine's Abstract Machine (with explicit environments) that is devoted to the evaluation of proofs by forcing, which we call the *Krivine Forcing Abstract Machine* (KFAM). The main novelty of this new abstract machine is that it distinguishes two kinds of closures:

— *regular closures*, that are intended to be executed the usual way, and
— *forcing closures* (bearing a star as a superscript), that are intended to be executed as through the program transformation $t \mapsto t^*$ defined in section 4.4.

At each evaluation step, the current execution mode of the abstract machine is given by the mode of the currently executed closure. In particular, there is no need to introduce a special instruction to switch from one mode to the other.

6.1. *Krivine's Forcing Abstract Machine (KFAM)*

Formally, the KFAM manipulates five kinds of syntactic entities: terms, environments, closures, stacks and processes, whose BNF is given in Table 4.

Table 4. *Syntax and evaluation rules of the KFAM*

Syntax of the KFAM

Terms	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu \mid \text{œ}$	
Environments	e	$::=$	$\emptyset \mid e, x := c$	
Closures	c	$::=$	$t[e] \mid \mathsf{k}_\pi \mid t[e]^* \mid \mathsf{k}^*_\pi$	$(FV(t) \subseteq \mathrm{dom}(e))$
Stacks	π	$::=$	$\diamond \mid c \cdot \pi$	
Processes	P	$::=$	$c \star \pi$	

Evaluation rules for regular closures

(Skip)	$x[e, y := c] \star \pi$	$\succ$	$x[e] \star \pi$	$(y \not\equiv x)$
(Access)	$x[e, x := c] \star \pi$	$\succ$	$c \star \pi$	
(Grab)	$(\lambda x\,.\,t)[e] \star c \cdot \pi$	$\succ$	$t[e, x := c] \star \pi$	
(Push)	$(tu)[e] \star \pi$	$\succ$	$t[e] \star u[e] \cdot \pi$	
(Save)	$\text{œ}[e] \star c \cdot \pi$	$\succ$	$c \star \mathsf{k}_\pi \cdot \pi$	
(Restore)	$\mathsf{k}_\pi \star c \cdot \pi'$	$\succ$	$c \star \pi$	

Evaluation rules for forcing closures

(Skip*)	$x[e, y := c]^* \star c_0 \cdot \pi$	$\succ$	$x[e]^* \star \alpha_9\ c_0 \cdot \pi$	$(y \not\equiv x)$
(Access*)	$x[e, x := c]^* \star c_0 \cdot \pi$	$\succ$	$c \star \alpha_{10} c_0 \cdot \pi$	
(Grab*)	$(\lambda x\,.\,t)[e]^* \star c_0 \cdot c \cdot \pi$	$\succ$	$t[e, x := c]^* \star \alpha_6\ c_0 \cdot \pi$	
(Push*)	$(tu)[e]^* \star c_0 \cdot \pi$	$\succ$	$t[e]^* \star \alpha_{11} c_0 \cdot u[e]^* \cdot \pi$	
(Save*)	$\text{œ}[e]^* \star c_0 \cdot c \cdot \pi$	$\succ$	$c \star \alpha_{14} c_0 \cdot \mathsf{k}^*_\pi \cdot \pi$	
(Restore*)	$\mathsf{k}^*_\pi \star c_0 \cdot c \cdot \pi'$	$\succ$	$c \star \alpha_{15} c_0 \cdot \pi$	

— The *terms* of the KFAM are simply the proof terms of system PAω^+ (Def. 7), that is: pure λ-terms enriched with the constant œ.

— The *environments* of the KFAM are finite association lists mapping closures to proof variables. Note that the same variable x can be bound several times in an environment e, but in this case, only the rightmost binding will be considered during evaluation, the other bindings being hidden. Actually, the environments of the KFAM are exactly the A-environments of Def. 40 when taking the set of all closures as A—with this subtlety that environments and closures are defined here by mutual induction.

— The KFAM distinguishes two forms of *regular closures*, as well as two forms of *forcing closures*. Regular closures are either term closures of the form $t[e]$, where t is a term and e an environment that closes t (in the sense that $FV(t) \subseteq \mathrm{dom}(e)$), or continuation closures of the form k_π, where π is a stack. Forcing closures are just starred versions $t[e]^*$ and k_π^* of regular closures. (Here, the star '$*$' in superscript is just a mark indicating that the closure is intended to be executed in forcing mode.)

— The *stacks* of the KFAM are finite lists of closures, whereas *processes* are pairs $c \star \pi$ formed by a closure c and a stack π.

In what follows, we shall write $\mathbf{\Lambda}$ (resp. $\mathbf{\Pi}$, $\mathbf{\Lambda} \star \mathbf{\Pi}$) the set of all closures (resp. of all stacks, of all processes) in the sense of the KFAM.

The set $\mathbf{\Lambda} \star \mathbf{\Pi}$ of all processes is equipped with a preorder $P \succ P'$ that is generated from the evaluation rules given in Table 4. Evaluation may proceed in regular mode or in forcing mode depending on whether the currently evaluated closure is a regular closure or a forcing closure.

Evaluation in regular mode The evaluation rules for regular closures (Table 4) are the standard evaluation rules of the KAM with continuations and explicit environments. Here we can find the usual evaluation rules (Grab), (Push), (Save) and (Restore) (adapted to the presence of explicit environments) as well as two specific rules (Skip) and (Access) to lookup a variable in the current environment.

Evaluation in forcing mode The evaluation rules for forcing closures (Table 4) are similar to the evaluation rules for regular closures, except that in forcing mode, the first slot of the stack is now reserved for the computational condition, represented here as a closure c_0. This computational condition is updated at each evaluation step by the insertion of one of the six combinators α_6, α_9, α_{10}, α_{11}, α_{14} and α_{15}. These combinators are now fixed closures—or closed proof terms—that are parameterizing the abstract machine. (At this stage, these parameters can be taken arbitrarily.) The insertion of a particular combinator α_i on the top of the current computational condition c_0 is achieved using *closure application*, which is defined by

$$\alpha_i\, c_0 \;\equiv\; (yx)[y := \alpha_i, x := c_0]\,.$$

(Or as $\alpha_i\, c_0 \equiv (\alpha_i\, x)[x := c_0]$ in the case where α_i is a closed proof term.)
The four evaluation rules (Grab*), (Push*), (Save*) and (Restore*) mimic the computational behavior of transformed programs such as described by Prop. 39 in Section 4.7, using the same combinators α_6, α_{11}, α_{14} and α_{15} as before. Also note that lookup operations (Skip*) and (Access*) also insert their own combinators α_9 and α_{10} in the first slot of the stack. These extra two combinators—that were actually hidden in the proof terms β_3 and β_4 defined in Prop. 25 and used in the translation of abstraction (Def. 30)—simply reflect the destruction of the current environment during lookup:

$\alpha_9 \;:\; \mathsf{C}[(pq)r] \Rightarrow \mathsf{C}[pr]$ 'skip the rightmost closure'

$\alpha_{10} \;:\; \mathsf{C}[(pq)r] \Rightarrow \mathsf{C}[qr]$ 'extract the rightmost closure'

(Here, the condition pq is attached to the environment of the currently evaluated closure while the condition r is attached to the current stack.)

Switching between the two execution modes The KFAM provides no instruction to switch between the two execution modes; instead, every closure bears its own execution mode. However, environments may contain both regular closures and forcing closures, so that each access to a closure in the current environment is an opportunity to switch from one execution mode to the other. In practice, this may only occur in the following two situations:

— When, during an (ACCESS) step, the regular closure $x[e, x := c]$ gives the control to the closure c that is bound to x in the current environment. In the case where c is a regular closure, evaluation continues in regular mode. But when c is a forcing closure, the abstract machine switches to the forcing mode.

— When, during an (ACCESS*) step, the forcing closure $x[e, x := c]^*$ gives the control to the closure c that is bound to x in the current environment. In the case where c is a forcing closure, evaluation continues in forcing mode. But when c is a regular closure, the abstract machine switches back to the regular mode.

As we shall see below, the main interest of the KFAM is that it allows us to use the same proof term t of a proposition A both as a realizer of A and as a realizer of $p \Vdash A$, depending on whether we embed t into a regular closure $t[e]$ or into a forcing closure $t[e]^*$. To prove this, we need to present the realizability semantics of the KFAM.

6.2. *Adequacy in regular mode*

The same way as the λ_c-calculus induces a particular family of classical realizability algebras (Section 5.2), called the λ_c-algebras (Section 5.3), the KFAM also induces its own family of classical realizability algebras.

Formally, every set of processes $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$ of the KFAM that is saturated w.r.t. the evaluation rules of Table 4 induces a classical realizability algebra $\mathscr{A}$ whose components $\mathbf{\Lambda}$, $\mathbf{\Pi}$, $\mathbf{\Lambda} \star \mathbf{\Pi}$, etc. are defined as follows:

— The set $\mathbf{\Lambda}$ of $\mathscr{A}$-terms is the set of all closures (in the sense of the KFAM).

— The set $\mathbf{\Pi}$ of $\mathscr{A}$-stacks is the set of all stacks (ditto).

— The set $\mathbf{\Lambda} \star \mathbf{\Pi}$ of $\mathscr{A}$-processes is the set of all processes (ditto).

— The three operations $\pi \mapsto \mathsf{k}_\pi$, $(c, \pi) \mapsto c \cdot \pi$ and $(c, \pi) \mapsto c \star \pi$ are defined in the algebra $\mathscr{A}$ the same way as they are defined in the KFAM.

— The compilation function $(t, \sigma) \mapsto t[\sigma]$ is simply the operation that consists to form a regular term closure $t[\sigma]$ from a proof term t and an environment σ such that $FV(t) \subseteq \mathrm{dom}(\sigma)$. (Remember that the environments of the KFAM are the same as the $\mathbf{\Lambda}$-environments in the sense of Def. 40.)

— The pole of $\mathscr{A}$ is the saturated set $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$.

Proposition 51. — The structure $\mathscr{A}$ defined above is a classical realizability algebra.

Proof. If suffices to check that the set $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$ (which is saturated w.r.t. the rules of Table 4) is also saturated w.r.t. the pseudo-evaluation preorder $\succ_{\mathscr{A}}$ that is generated from $\mathscr{A}$ (cf Section 5.2). But this is obvious, since the preorder $\succ_{\mathscr{A}}$ is included into the relation of evaluation defined from the rules of Table 4—simulating the pseudo-evaluation rule (LOOKUP$_{\mathscr{A}}$) by the real rules (SKIP) and (ACCESS). □

The classical realizability algebra $\mathscr{A}$ induces a classical realizability model $\mathscr{M}_{\mathscr{A}}$ (Section 5) and a relation of realizability $c \Vdash_{\mathscr{A}} A$ between a closure c and a closed proposition A (with parameters in $\mathscr{M}_{\mathscr{A}}$). In this realizability model, the general property of adequacy (Prop. 48) instantiates as follows:

Proposition 52 (Adequacy in regular mode). — If the judgment

$$\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

is derivable in system PAω^+, then for all total valuations ρ such that $\rho \models \mathcal{E}$ and for all closures $c_1 \Vdash_{\mathscr{A}} A_1[\rho]$, …, $c_n \Vdash_{\mathscr{A}} A_n[\rho]$, we have:

$$t[x_1 := c_1, \ldots, x_n := c_n] \Vdash_{\mathscr{A}} A[\rho]\,.$$

Intuitively, the above result expresses that a proof term $t : A$ behaves as a realizer of the proposition A as soon as we embed it into a regular closure (using a suitable environment). We can notice that this adequacy result only relies on the operational semantics of regular closures, which is defined from the rules (Skip), (Access), (Push), (Grab), (Save) and (Restore). At this stage, the operational semantics of forcing closures is simply irrelevant, and we could have replaced the starred rules (Skip*)–(Restore*) by completely different rules without altering the property of adequacy in regular mode. In particular, we do not need to make any assumption on the six combinators α_6, α_9, α_{10}, α_{11}, α_{14} and α_{15} which parameterize the abstract machine.

However, things become different when considering adequacy in forcing mode.

6.3. *Adequacy in forcing mode*

For each combinator α_i (where $i \in \{*; 1; \ldots; 15\}$), let us write type($\alpha_i$) the 'type of α_i' such as given in Section 3.1. In particular, we have:

$$\begin{array}{rcl}
\mathrm{type}(\alpha_6) & \equiv & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[(pq)r]) \\
\mathrm{type}(\alpha_9) & \equiv & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[(pq)r)] \Rightarrow \mathsf{C}[p(qr)]) \\
\mathrm{type}(\alpha_{10}) & \equiv & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[(pq)r)] \Rightarrow \mathsf{C}[qr]) \\
\mathrm{type}(\alpha_{11}) & \equiv & \forall p^\kappa\, \forall q^\kappa\, (\mathsf{C}[pq] \Rightarrow \mathsf{C}[p(pq)]) \\
\mathrm{type}(\alpha_{14}) & \equiv & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[q(rr)]) \\
\mathrm{type}(\alpha_{15}) & \equiv & \forall p^\kappa\, \forall q^\kappa\, \forall r^\kappa\, (\mathsf{C}[p(qr)] \Rightarrow \mathsf{C}[qp])
\end{array}$$

From now on, we assume that the six combinators α_6, α_9, α_{10}, α_{11}, α_{14} and α_{15} parameterizing evaluation in the forcing mode of the KFAM are *closures* that realize the corresponding types, in the sense that $\alpha_i \Vdash_{\mathscr{A}}$ type(α_i) for all $i \in \{6; 9; 10; 11; 14; 15\}$. From the property of adequacy in regular mode (Prop. 52), such closures can be easily built from closed proof terms (also named $\alpha_6, \ldots, \alpha_{15}$ in Section 3) of the propositions type(α_6), …, type(α_{15}), simply by turning these proof terms into regular closures.

Proposition 53 (Adequacy in forcing mode). — Under the assumption that the closures α_6, α_9, α_{10}, α_{11}, α_{14} and α_{15} realize their types, if the judgment

$$\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

is derivable in system PAω^+, then for all total valuations ρ such that $\rho \models \mathcal{E}^*$ and for all closures $c_1 \Vdash_{\mathscr{A}} (p_1 \Vdash A_1)[\rho]$, …, $c_n \Vdash_{\mathscr{A}} (p_n \Vdash A_n)[\rho]$, we have:

$$t[x_1 := c_1, \ldots, x_n := c_n]^* \Vdash_{\mathscr{A}} \big(((p_0 p_1) \cdots p_n) \Vdash A\big)[\rho]\,,$$

where $p_0, p_1, \ldots, p_n$ are $n+1$ fresh condition variables.

In particular, the above proposition expresses that if t is a closed proof term of a closed proposition A, then the corresponding forcing closure $t[]^*$ is now a realizer of the proposition $p \Vdash A$ (for an arbitrary condition p).

Remark 54. We can notice that in the case where the context is not empty, the free variables of t are not treated exactly the same way as in Corollary 50 (for the usual KAM), which is due to the fact that we now work with explicit environments (i.e. delayed substitutions) rather than with ordinary parallel substitutions. In the above adequacy result, each free variable $x_i : A_i$ of the proof term $t : A$ has to be bound (in the current environment) to a closure c_i realizing the proposition $p_i \Vdash A_i$, where p_i is an arbitrary condition attached to the particular hypothesis A_i. The resulting forcing closure $t[x_1 := c_1, \ldots, x_n := c_n]^*$ is then a realizer of the proposition $((p_0 p_1) \cdots p_n) \Vdash A$, where $((p_0 p_1) \cdots p_n)$ denotes the (left-associative) product of the conditions $p_0, p_1, \ldots, p_n$, and where p_0 is an arbitrary extra condition attached to the proof term t itself.

We could readily prove the above property of adequacy in forcing mode (Prop. 53) by adapting the ingredients of the proof of Theorem 32 to the framework of the realizability model $\mathscr{M}_{\mathscr{A}}$ induced by the algebra $\mathscr{A}$. However, it is much more interesting to *deduce* Prop. 53 from the general property of adequacy (Prop. 48), by introducing a classical realizability algebra $\mathscr{A}^*$ describing the forcing mode of the KFAM, and by studying the relationship between the two realizability models $\mathscr{M}_{\mathscr{A}}$ and $\mathscr{M}_{\mathscr{A}^*}$ that are induced by the two algebras $\mathscr{A}$ and $\mathscr{A}^*$. This is what we shall do in Section 7, and for this reason, the proof of Prop. 53 is postponed to the end of Section 7.

6.4. *Combining the two execution modes together*

To illustrate the use of the KFAM, let us come back to the situation depicted in Section 4.5 and consider two proof terms terms u ('user program') and s ('system program') such that $x : A \vdash u : B$ and $\vdash s : (1 \Vdash A)$. (Recall that B is the theorem we want to prove from an axiom A that is forced using a suitable forcing structure.)

From the property of adequacy in regular mode (Prop. 52), we know that the regular closure $s[]$ is a realizer of the proposition $1 \Vdash A$. Now using the property of adequacy in forcing mode (Prop. 53), we thus deduce that the forcing closure $u[x := s[]]^*$ is a realizer of the proposition $11 \Vdash B$. Intuitively, the forcing closure $u[x := s[]]^*$ is obtained by embedding the system program s (that is intended to be evaluated in regular mode) into the environment of the user program u (intended to be evaluated in forcing mode)[¶].

[¶] Here, the regular mode corresponds to the *supervisor mode* of modern CPUs (also called the *real mode* in x86-compatible CPUs from the 286 architecture), whereas the forcing mode corresponds to the *user mode* (or the *protected mode* in x86-compatible CPUs).

If moreover the proposition B is invariant under forcing (which is automatically the case if B is a first-order proposition by Prop. 37), then we easily derive

$$z : (11 \Vdash B) \vdash \xi_B\, z\, (\alpha_7\, \alpha_*) : B$$

(using the proof term ξ_B introduced in Def. 33), so that we can again embed the forcing closure $u[x := s[]]^*$ into a regular closure $(\xi'_B\, z\, (\alpha_7\, \alpha_*))[z := u[x := s[]]^*]$ and deduce from the property of adequacy in regular mode (Prop. 52) that:

$$(\xi'_B\, z\, (\alpha_7\, \alpha_*))[z := u[x := s[]]^*] \ \Vdash_{\mathscr{A}} B\,.$$

7. The connection theorem

In Section 6.2, we have seen that every saturated set of processes $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$ (w.r.t. the evaluation rules of Table 4) induces a classical realizability algebra $\mathscr{A}$ whose $\mathscr{A}$-terms, $\mathscr{A}$-stacks and $\mathscr{A}$-processes are precisely the closures, the stacks and the processes of the KFAM, and whose pole $\perp\!\!\!\perp$ is the saturated set $\perp\!\!\!\perp$. We shall now see that the same saturated set $\perp\!\!\!\perp \subseteq \mathbf{\Lambda} \star \mathbf{\Pi}$ induces another classical realizability algebra written $\mathscr{A}^*$ and whose components are built from the forcing mode of the KFAM.

7.1. *The algebra $\mathscr{A}^*$ induced by the forcing mode*

Formally, the classical realizability algebra $\mathscr{A}^*$ induced by the forcing mode is defined from the forcing structure $(\kappa, \mathsf{C}, \cdot, 1, \alpha_*, \alpha_1, \ldots, \alpha_8)$ introduced in Section 3, and more precisely: from the semantics of this structure in the realizability model $\mathscr{M}_{\mathscr{A}}$ induced by the classical realizability algebra $\mathscr{A}$ corresponding to the regular execution mode.

In what follows, we shall write $[\![\_]\!] = [\![\_]\!]^{\mathscr{A}}$ the interpretation function that comes with the realizability model $\mathscr{M}_{\mathscr{A}}$, and we shall extend the syntactic product pq of two conditions p, q to all pairs of semantic conditions $\boldsymbol{p}, \boldsymbol{q} \in [\![\kappa]\!]$ (writing them using bold letters $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{r}$, etc.), letting $\boldsymbol{pq} = [\![(\cdot)]\!](\boldsymbol{p})(\boldsymbol{q})$, where $(\cdot)$ denotes the binary product (of kind $\kappa \to \kappa \to \kappa$) that comes with the forcing structure parameterizing the construction. (We shall also write $\mathbf{1} = [\![1]\!]^{\mathscr{A}}$ the denotation of the top condition 1 in $\mathscr{M}_{\mathscr{A}}$.) Notice that this semantic product is in general neither associative, commutative nor idempotent in the model $\mathscr{M}_{\mathscr{A}}$, although we can realize the equivalences $\forall p\, \forall q\, \forall r\, ((pq)r \approx p(qr))$, $\forall p\, \forall q\, (pq \approx qp)$ and $\forall p\, (p \approx pp)$ (cf Section 3.2).

The components of the algebra $\mathscr{A}^*$ are formally defined as follows:

— The sets of $\mathscr{A}^*$-terms, of $\mathscr{A}^*$-stacks and of $\mathscr{A}^*$-processes—which we now respectively write $\mathbf{\Lambda}^*$, $\mathbf{\Pi}^*$ and $(\mathbf{\Lambda} \star \mathbf{\Pi})^*$—are defined by

$$\mathbf{\Lambda}^* = \mathbf{\Lambda} \times [\![\kappa]\!], \qquad \mathbf{\Pi}^* = \mathbf{\Pi} \times [\![\kappa]\!] \qquad \text{and} \qquad (\mathbf{\Lambda} \star \mathbf{\Pi})^* = (\mathbf{\Lambda} \star \mathbf{\Pi}) \times [\![\kappa]\!]\,.$$

Notice that every element of the above sets is an order pair formed by a syntactic object (a closure, a stack, a process of the KFAM) together with a semantic condition in the set $[\![\kappa]\!]$. (So that the sets $\mathbf{\Lambda}^*$, $\mathbf{\Pi}^*$ and $(\mathbf{\Lambda} \star \mathbf{\Pi})^*$ are in general not countable.)

— The three operations $(a,\pi)\mapsto a\cdot\pi$ (‘consing’), $(a,\pi)\mapsto a\star\pi$ (‘process formation’) and $\pi\mapsto \mathsf{k}_\pi$ (‘continuation formation’) are defined in the algebra $\mathscr{A}^*$ by letting

$$\begin{array}{rcll}(c,\boldsymbol{p})\cdot(\pi,\boldsymbol{r}) &=& (c\cdot\pi,\ \boldsymbol{pr}) &\in\ \mathbf{\Pi}^*\\ (c,\boldsymbol{p})\star(\pi,\boldsymbol{r}) &=& (c\star\pi,\ \boldsymbol{pr}) &\in\ (\mathbf{\Lambda}\star\mathbf{\Pi})^*\\ \mathsf{k}_{(\pi,\boldsymbol{r})} &=& (\mathsf{k}^*_\pi,\boldsymbol{r}) &\in\ \mathbf{\Lambda}^*\end{array}$$

for all $(c,\boldsymbol{p})\in\mathbf{\Lambda}^*$ and $(\pi,\boldsymbol{r})\in\mathbf{\Pi}^*$.

— The compilation function $(t,\sigma)\mapsto t[\sigma]$ of the algebra $\mathscr{A}^*$ is defined by letting

$$\begin{array}{l} t[x_1:=(c_1,\boldsymbol{p}_1),\ldots,x_n:=(c_n,\boldsymbol{p}_n)] \ = \\ \quad\big(t[x_1:=c_1,\ldots,x_n:=c_n]^*,\ ((\mathbf{1}\boldsymbol{p}_1)\cdots\boldsymbol{p}_n)\big)\end{array}$$

for all proof terms t such that $FV(t)\subseteq\{x_1;\ldots;x_n\}$ and for all $\mathbf{\Lambda}^*$-environments $\sigma\equiv[x_1:=(c_1,\boldsymbol{p}_1),\ldots,x_n:=(c_n,\boldsymbol{p}_n)]$.

— Finally, the pole of the algebra $\mathscr{A}^*$, which we write $\perp\!\!\!\perp^*$, is defined by:

$$\perp\!\!\!\perp^*\ =\ \big\{(c\star\pi,\boldsymbol{p})\in(\mathbf{\Lambda}\star\mathbf{\Pi})^*\ :\ \forall c_0\in(\llbracket\mathsf{C}\rrbracket(\boldsymbol{p}))^{\perp\!\!\!\perp}\ \ (c\star c_0\cdot\pi)\in\perp\!\!\!\perp\big\}\,.$$

We first need to check that:

Proposition 55. — The structure $\mathscr{A}^*$ defined above is a classical realizability algebra.

Proof. As noticed in Section 5.2, it suffices to show that the set of $\mathscr{A}^*$-processes $\perp\!\!\!\perp^*$ defined above is saturated (Def. 43) w.r.t. the preorder $\succ_{\mathscr{A}^*}$ generated from the five rules (Lookup$_{\mathscr{A}^*}$), (Push$_{\mathscr{A}^*}$), (Grab$_{\mathscr{A}^*}$), (Save$_{\mathscr{A}^*}$) and (Restore$_{\mathscr{A}^*}$) as shown in Remark 42(4). To prove this, we shall not directly work with the preorder $\succ_{\mathscr{A}^*}$ induced by the operations of $\mathscr{A}^*$, but we shall consider instead another (finer) preorder $\succ^\bullet$ that is generated from the following six rules:

(Skip$^\bullet$)	$(x[e,y:=c]^*\star\pi,\ (\boldsymbol{pq})\boldsymbol{r})$	$\succ^\bullet$	$(x[e]^*\star\pi,\ \boldsymbol{pr})$	$(y\not\equiv x)$
(Access$^\bullet$)	$(x[e,x:=c]^*\star\pi,\ (\boldsymbol{pq})\boldsymbol{r})$	$\succ^\bullet$	$(c\star\pi,\ \boldsymbol{qr})$	
(Push$^\bullet$)	$((tu)[e]^*\star\pi,\ \boldsymbol{pr})$	$\succ^\bullet$	$(t[e]^*\star u[e]^*\cdot\pi,\ \boldsymbol{p}(\boldsymbol{pr}))$	
(Grab$^\bullet$)	$((\lambda x\,.\,t)[e]^*\star c\cdot\pi,\ \boldsymbol{p}(\boldsymbol{qr}))$	$\succ^\bullet$	$(t[e,x:=c]^*\star\pi,\ (\boldsymbol{pq})\boldsymbol{r})$	
(Save$^\bullet$)	$(\text{œ}[e]^*\star c\cdot\pi,\ \boldsymbol{p}(\boldsymbol{qr}))$	$\succ^\bullet$	$(c\star\mathsf{k}^*_\pi\cdot\pi,\ \boldsymbol{q}(\boldsymbol{rr}))$	
(Restore$^\bullet$)	$(\mathsf{k}^*_\pi\star c\cdot\pi',\ \boldsymbol{p}(\boldsymbol{qr}))$	$\succ^\bullet$	$(c\star\pi,\ \boldsymbol{qp})$	

(for all $c\in\mathbf{\Lambda}$, $\pi\in\mathbf{\Lambda}$ and $\boldsymbol{p},\boldsymbol{q},\boldsymbol{r}\in\llbracket\kappa\rrbracket$.)

Remark 56. The six rules (Skip$^\bullet$)–(Restore$^\bullet$) closely reflect the structure of the evaluation rules (Skip*)–(Restore*) defining the forcing mode of the KFAM (Table 4). In particular, notice that for each of these rules, the semantic condition evolves accordingly to the type of the combinator α_i attached to the corresponding evaluation rule in the forcing mode of the KFAM—which is crucial in the proof of Prop. 58 below.

Lemma 57. — The preorder $\succ_{\mathscr{A}^*}$ induced by the structure $\mathscr{A}^*$ (as shown in Remark 42(4)) is included in the preorder $\succ^\bullet$ defined above.

Proof. Unfolding the definition of the compilation function $(t,\sigma)\mapsto t[\sigma]$ provided with the structure $\mathscr{A}^*$, we simply notice that:

$$\begin{array}{rcl}(\textsc{Lookup}_{\mathscr{A}^*}) & \subseteq & (\textsc{Skip}^\bullet)^*; (\textsc{Access}^\bullet)\\ (\textsc{Push}_{\mathscr{A}^*}) & \subseteq & (\textsc{Push}^\bullet)\\ (\textsc{Grab}_{\mathscr{A}^*}) & \subseteq & (\textsc{Grab}^\bullet)\\ (\textsc{Save}_{\mathscr{A}^*}) & \subseteq & (\textsc{Save}^\bullet)\\ (\textsc{Restore}_{\mathscr{A}^*}) & \subseteq & (\textsc{Restore}^\bullet)\end{array}$$ □

From the above inclusion, it is obvious that every set of $\mathscr{A}^*$-processes that is saturated w.r.t. the preorder $\succ^\bullet$ is also saturated w.r.t. the preorder $\succ_{\mathscr{A}^*}$. To conclude the proof of Prop. 55, we just need to check that:

Proposition 58. — The set of $\mathscr{A}^*$-processes

$$\perp\!\!\!\perp^* \;=\; \big\{(c \star \pi, \boldsymbol{p}) \in (\boldsymbol{\Lambda} \star \boldsymbol{\Pi})^* \;:\; \forall c_0 \in \boldsymbol{\Lambda}\ ((c_0 \Vdash_{\mathscr{A}} \mathsf{C}[\dot{\boldsymbol{p}}]) \;\Rightarrow\; (c \star c_0 \cdot \pi) \in \perp\!\!\!\perp\big\}$$

is saturated w.r.t. the preorder $\succ^\bullet$.

Proof. The six cases corresponding to the rules ($\textsc{Skip}^\bullet$)–($\textsc{Restore}^\bullet$) follow the same pattern, using Remark 56. Let us treat for instance the case of rule ($\textsc{Skip}^\bullet$). For that, let us assume that $(x[e]^* \star \pi,\ \boldsymbol{pr}) \in \perp\!\!\!\perp^*$, which means that:

$$\forall c_0 \in \boldsymbol{\Lambda}\ \big((c_0 \Vdash_{\mathscr{A}} \mathsf{C}[\dot{\boldsymbol{p}}\dot{\boldsymbol{r}}]) \;\Rightarrow\; (x[e]^* \star c_0 \cdot \pi) \in \perp\!\!\!\perp\big)\,. \tag{1}$$

Applying Prop. 52 (adequacy in regular mode) to the derivable judgment

$$y : \mathrm{type}(\alpha_9),\ x : \mathsf{C}[(pq)r] \;\vdash\; yx \;: \mathsf{C}[pr]\,,$$

we deduce that

$$\forall c_0 \in \boldsymbol{\Lambda}\ \big((c_0 \Vdash_{\mathscr{A}} \mathsf{C}[(\dot{\boldsymbol{p}}\dot{\boldsymbol{q}})\dot{\boldsymbol{r}}]) \;\Rightarrow\; (\alpha_9\, c_0 \Vdash_{\mathscr{A}} \mathsf{C}[\dot{\boldsymbol{p}}\dot{\boldsymbol{r}}])\big)\,. \tag{2}$$

(Recall that the closure $\alpha_9\, c_0$ is implemented as $\alpha_9\, c_0 \equiv (yx)[y \leftarrow \alpha_9, x \leftarrow c_0]$.) Combining (1) and (2), we deduce that

$$\forall c_0 \in \boldsymbol{\Lambda}\ \big((c_0 \Vdash_{\mathscr{A}} \mathsf{C}[(\dot{\boldsymbol{p}}\dot{\boldsymbol{q}})\dot{\boldsymbol{r}}]) \;\Rightarrow\; (x[e]^* \star \alpha_9\, c_0 \cdot \pi) \in \perp\!\!\!\perp\big)\,.$$

But since the set $\perp\!\!\!\perp \subseteq \boldsymbol{\Lambda} \star \boldsymbol{\Pi}$ is saturated w.r.t. the rule ($\textsc{Skip}^*$), we get

$$\forall c_0 \in \boldsymbol{\Lambda}\ \big((c_0 \Vdash_{\mathscr{A}} \mathsf{C}[(\dot{\boldsymbol{p}}\dot{\boldsymbol{q}})\dot{\boldsymbol{r}}]) \;\Rightarrow\; (x[e, y := c]^* \star c_0 \cdot \pi) \in \perp\!\!\!\perp\big)\,,$$

which precisely means that $(x[e, y := c]^* \star \pi,\ (\boldsymbol{pq})\boldsymbol{r}) \in \perp\!\!\!\perp^*$. □

The proof that $\mathscr{A}^*$ is a classical realizability algebra (Prop. 55) is now complete. □

In what follows, we shall call the *forcing model* and write $\mathscr{M}_{\mathscr{A}^*}$ the realizability model induced by the algebra $\mathscr{A}^*$, as opposed to the *regular model* $\mathscr{M}_{\mathscr{A}}$ induced by the algebra $\mathscr{A}$. The interpretation function $[\![\_]\!]^{\mathscr{A}^*}$ of kinds and higher-order terms of PAω^+ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ will be simply written $[\![\_]\!]^*$.

7.2. *The connection theorem*

The property of adequacy in forcing mode (Prop. 53) follows from the general property of adequacy (Prop. 48) instantiated to the algebra $\mathscr{A}^*$ by noticing that a $\mathscr{A}^*$-term

$(c, \boldsymbol{p}) \in \boldsymbol{\Lambda}^*$ realizes a (closed) proposition A in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ if and only if the closure $c \in \boldsymbol{\Lambda}$ realizes the proposition $\dot{\boldsymbol{p}} \mathrel{\mathsf{I\!F}} A$ in the regular model $\mathscr{M}_{\mathscr{A}}$:

$$(c, \boldsymbol{p}) \Vdash_{\mathscr{A}^*} A \quad \Leftrightarrow \quad c \Vdash_{\mathscr{A}} (\dot{\boldsymbol{p}} \mathrel{\mathsf{I\!F}} A)\,.$$

Making explicit the underlying valuations, we formally get:

Proposition 59 (Realizability equivalence). — For all propositions A, for all valuations ρ closing A in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ and for all $(c, \boldsymbol{p}) \in \boldsymbol{\Lambda}^*$, we have

$$(c, \boldsymbol{p}) \Vdash_{\mathscr{A}^*} A[\rho] \quad \Leftrightarrow \quad c \Vdash_{\mathscr{A}} (p \mathrel{\mathsf{I\!F}} A)[\psi(\rho), p \leftarrow \boldsymbol{p}]\,,$$

where $\psi(\rho)$ is the valuation in $\mathscr{M}_{\mathscr{A}}$ that is defined from ρ as shown in Def. 62 below.

Remark 60. As suggested by Krivine (Kri10), the above equivalence is reminiscent of the method of iterated forcing (Jec02, p. 267). Intuitively, the forcing algebra $\mathscr{A}^*$ plays here the same role as the set of conditions $\mathsf{D} * \dot{\mathsf{C}}$ we obtain by iterating two notions of forcing D and C, with this difference that the outermost notion of forcing D is replaced here by the classical realizability algebra $\mathscr{A}$ describing the KFAM (in regular mode).

The above result (Prop. 59) is actually a particular case of a more general result—the *connection theorem* (Theorem 63 below)—that relates the forcing model $\mathscr{M}_{\mathscr{A}^*}$ to the regular model $\mathscr{M}_{\mathscr{A}}$ throughout the hierarchy of kinds using the auxiliary translation $M \mapsto M^*$ (Def. 20). Basically, this theorem says that the denotation $\llbracket M \rrbracket^* \in \llbracket \tau \rrbracket^*$ of any higher-order term M of kind τ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ is the same as the denotation $\llbracket M^* \rrbracket \in \llbracket \tau^* \rrbracket$ of the translated term M^* of kind τ^* in the regular model $\mathscr{M}_{\mathscr{A}}$, through a natural isomorphism $\psi_\tau : \llbracket \tau \rrbracket^* \xrightarrow{\sim} \llbracket \tau^* \rrbracket$ (Fact 61 below). In other words, the connection theorem establishes that the following diagram commutes for every kind τ:

$$\begin{tikzcd}
\underline{\textbf{Syntax of system } \mathrm{PA}\omega^+} & \underline{\textbf{Realizability semantics}} \\
\mathrm{Term}(\tau) \arrow[r, "\llbracket\text{-}\rrbracket^{\mathscr{A}^*}"] \arrow[d, "*"'] & \llbracket \tau \rrbracket^* \arrow[d, "\psi_\tau"] \\
\mathrm{Term}(\tau^*) \arrow[r, "\llbracket\text{-}\rrbracket^{\mathscr{A}}"'] & \llbracket \tau^* \rrbracket
\end{tikzcd}$$

To establish this theorem, we first need to notice that:

Fact 61 (Isomorphism between $\llbracket \tau \rrbracket^*$ and $\llbracket \tau^* \rrbracket$). — For every kind τ of $\mathrm{PA}\omega^+$, there is a natural isomorphism $\psi_\tau : \llbracket \tau \rrbracket^* \xrightarrow{\sim} \llbracket \tau^* \rrbracket$, which is defined from the equations

$$\begin{array}{rcll}
\psi_o(S) & = & (p \in \llbracket \kappa \rrbracket \mapsto \{\pi \in \boldsymbol{\Pi} : (\pi, p) \in S\}) & (\text{for all } S \in \mathfrak{P}(\boldsymbol{\Pi}^*)) \\
\psi_\iota(n) & = & n & (\text{for all } n \in \mathbb{N}) \\
\psi_{\tau \to \sigma}(f) & = & \psi_\sigma \circ f \circ \psi_\tau^{-1} & (\text{for all } f \in \llbracket \tau \to \sigma \rrbracket^*)
\end{array}$$

The family of isomorphisms $(\psi_\tau)_{\tau \in \mathrm{Kind}}$ induces a mapping $\rho \mapsto \psi(\rho)$ between the valuations in $\mathscr{M}_{\mathscr{A}^*}$ and the valuations in $\mathscr{M}_{\mathscr{A}}$ that is defined as follows:

Definition 62 (Mapping valuations from $\mathscr{M}_{\mathscr{A}^*}$ to $\mathscr{M}_{\mathscr{A}}$). — To every valuation ρ in $\mathscr{M}_{\mathscr{A}^*}$, we associate a (partial) valuation $\psi(\rho)$ in $\mathscr{M}_{\mathscr{A}}$ that is defined by

— $\mathrm{dom}(\psi(\rho)) = \{x^{\tau^*} : x^\tau \in \mathrm{dom}(\rho)\}$;
— $\psi(\rho)(x^{\tau^*}) = \psi_\tau(\rho(x^\tau))$ for every $x^\tau \in \mathrm{dom}(\rho)$.

With this notation, the connection theorem states as follows:

Theorem 63 (Connection theorem). — Let M be a higher-order term of kind τ.

(1) For all valuations ρ closing M in the forcing model $\mathscr{M}_{\mathscr{A}^*}$, we have:

$$\psi_\tau\big([\![M[\rho]]\!]^*\big) \;=\; [\![M^*[\psi(\rho)]]\!]\,.$$

(2) If moreover $M \equiv A$ is a proposition, then for all valuations ρ closing A in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ and for all $(c, \boldsymbol{p}) \in \boldsymbol{\Lambda}^*$ we have

$$(c, \boldsymbol{p}) \Vdash_{\mathscr{A}^*} A[\rho] \quad\Leftrightarrow\quad c \Vdash_{\mathscr{A}} (p \Vdash A)[\psi(\rho), p \leftarrow \boldsymbol{p}]\,,$$

where p is a fresh condition variable (bound to the semantic condition $\boldsymbol{p}$).

Proof. We first show that (2) follows from (1) in the case where $M \equiv A$ is a proposition. Let us assume that $\psi_o([\![A[\rho]]\!]^*) = [\![A^*[\psi(\rho)]]\!]$, so that for all $(\pi, \boldsymbol{r}) \in \boldsymbol{\Pi}^*$ we have

$$\begin{array}{rcll}
(\pi, \boldsymbol{r}) \in [\![A[\rho]]\!]^* & \Leftrightarrow & \pi \in \psi_o([\![A[\rho]]\!]^*)(\boldsymbol{r}) & \text{(Def. of } \psi_o) \\
& \Leftrightarrow & \pi \in [\![A^*[\psi(\rho)]]\!](\boldsymbol{r}) & \text{(from (1))} \\
& \Leftrightarrow & \pi \in [\![(A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}]]\!] &
\end{array}$$

(where r is a fresh condition variable). Therefore, for all $(c, \boldsymbol{p}) \in \boldsymbol{\Lambda}^*$, we get:

$$\begin{array}{ll}
& (c, \boldsymbol{p}) \Vdash_{\mathscr{A}^*} A[\rho] \\
\Leftrightarrow & \forall(\pi, \boldsymbol{r}) \in [\![A[\rho]]\!]^* \;\; (c \star \pi, \boldsymbol{pr}) \in \perp\!\!\!\perp^* \\
\Leftrightarrow & \forall \boldsymbol{r} \in [\![\kappa]\!]\; \forall \pi \in [\![(A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}]]\!]\; \forall c_0 \in ([\![\mathsf{C}]\!](\boldsymbol{pr}))^{\perp\!\!\!\perp} \;\; (c \star c_0 \cdot \pi) \in \perp\!\!\!\perp \\
\Leftrightarrow & \forall \boldsymbol{r} \in [\![\kappa]\!]\; \forall c_0 \in [\![(\mathsf{C}[pr])[p \leftarrow \boldsymbol{p}, r \leftarrow \boldsymbol{r}]]\!]^{\perp\!\!\!\perp}\; \forall \pi \in [\![(A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}]]\!] \;\; (c \star c_0 \cdot \pi) \in \perp\!\!\!\perp \\
\Leftrightarrow & \forall \boldsymbol{r} \in [\![\kappa]\!] \;\; c \Vdash_{\mathscr{A}} (\mathsf{C}[pr] \Rightarrow A^* r)[\psi(\rho), p \leftarrow \boldsymbol{p}, r \leftarrow \boldsymbol{r}] \\
\Leftrightarrow & c \Vdash_{\mathscr{A}} (\forall r^\kappa (\mathsf{C}[pr] \Rightarrow A^* r))[\psi(\rho), p \leftarrow \boldsymbol{p}] \\
\Leftrightarrow & c \Vdash_{\mathscr{A}} (p \Vdash A)[\psi(\rho), p \leftarrow \boldsymbol{p}]
\end{array}$$

Let us now prove (1) by induction on M, distinguishing the following cases:

— Variable. Given a variable x^τ of kind τ and a valuation ρ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ such that $x^\tau \in \mathrm{dom}(\rho)$, we have

$$\psi_\tau([\![x^\tau[\rho]]\!]^*) \;=\; \psi_\tau(\rho(x^\tau)) \;=\; (\psi(\rho))(x^{\tau^*}) \;=\; [\![(x^\tau)^*[\psi(\rho)]]\!]$$

— Zero, Successor. The equalities $\psi_\iota([\![0]\!]^*) = [\![0^*]\!]$ and $\psi_{\iota\to\iota}([\![s]\!]^*) = [\![s^*]\!]$ are obvious, since both interpretation functions $[\![\_]\!]^*$ and $[\![\_]\!]$ coincide on the T-kinds ι and $\iota \to \iota$.

— Recursor. Writing $\sigma \equiv \tau \to (\iota \to \tau \to \tau) \to \iota \to \tau$, we want to prove the equality $\psi_\sigma([\![\mathrm{rec}_\tau]\!]^*) = [\![(\mathrm{rec}_\tau)^*]\!]$. For that, it suffices to check the equality

$$\psi_\sigma([\![\mathrm{rec}_\tau]\!]^*)(v)(f)(n) \;=\; [\![\mathrm{rec}_{\tau^*}]\!](v)(f)(n)$$

for all $v \in [\![\tau^*]\!]$, $f \in [\![\iota \to \tau^* \to \tau^*]\!]$ and $n \in \mathbb{N}$, which is done by induction on n.

— Abstraction. Let us assume that the property (1) holds for a term M of kind σ (IH), and consider a valuation ρ that closes the term $\lambda x^\tau . M$ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$. For all $v \in [\![\tau^*]\!]$, we have

$$\begin{array}{rcll}
\psi_{\tau\to\sigma}([\![(\lambda x^\tau . M)[\rho]]\!]^*)(v) & = & \psi_\sigma\big([\![(\lambda x^\tau . M)[\rho]]\!]^*(\psi_\tau^{-1}(v))\big) & (\text{Def. of } \psi_{\tau\to\sigma}) \\
& = & \psi_\sigma\big([\![M[\rho, x^\tau \leftarrow \psi_\tau^{-1}(v)]]\!]^*\big) & \\
& = & [\![M^*[\psi(\rho), x^{\tau^*} \leftarrow v]]\!] & (\text{IH}) \\
& = & [\![(\lambda x^{\tau^*} . M^*)[\psi(\rho)]]\!](v) & \\
& = & [\![(\lambda x^\tau . M)^*[\psi(\rho)]]\!](v)\,, &
\end{array}$$

hence $\psi_{\tau\to\sigma}([\![(\lambda x^\tau . M)[\rho]]\!]^*) = [\![(\lambda x^\tau . M)^*[\psi(\rho)]]\!]$.

— Application. Let us assume that the property (1) holds for a term M of kind $\tau \to \sigma$ and for a term N of kind τ (IH), and consider a valuation ρ that closes the term MN in the forcing model $\mathscr{M}_{\mathscr{A}^*}$. We have

$$\begin{array}{rcll}
\psi_\sigma\big([\![(MN)[\rho]]\!]^*\big) & = & \psi_\sigma\big([\![M[\rho]]\!]^*([\![N[\rho]]\!]^*)\big) & \\
& = & \psi_{\tau\to\sigma}([\![M[\rho]]\!]^*)\big(\psi_\tau([\![N[\rho]]\!]^*)\big) & (\text{Def. of } \psi_{\tau\to\sigma}) \\
& = & [\![M^*[\psi(\rho)]]\!]\big([\![N^*[\psi(\rho)]]\!]\big) & (\text{IH}) \\
& = & [\![(M^*N^*)[\psi(\rho)]]\!] \;=\; [\![(MN)^*[\psi(\rho)]]\!] &
\end{array}$$

— Universal quantification. Let us assume that the property (1) holds for a proposition A (IH), and consider a valuation ρ that closes the proposition $\forall x^\tau A$ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$. For all $\pi \in \mathbf{\Pi}$ and $\boldsymbol{r} \in [\![\kappa]\!]$, we have

$$\begin{array}{cll}
& \pi \in \psi_o\big([\![(\forall x^\tau A)[\rho]]\!]^*\big)(\boldsymbol{r}) & \\
\Leftrightarrow & (\pi, \boldsymbol{r}) \in [\![(\forall x^\tau A)[\rho]]\!]^* & (\text{Def. of } \psi_o) \\
\Leftrightarrow & \exists v \in [\![\tau]\!]^* \;\; (\pi, \boldsymbol{r}) \in [\![A[\rho, x \leftarrow v]]\!]^* & \\
\Leftrightarrow & \exists v \in [\![\tau]\!]^* \;\; \pi \in \psi_o([\![A[\rho, x^\tau \leftarrow v]]\!]^*)(\boldsymbol{r}) & (\text{Def. of } \psi_o) \\
\Leftrightarrow & \exists v \in [\![\tau]\!]^* \;\; \pi \in [\![A^*[\psi(\rho), x^{\tau^*} \leftarrow \psi_\tau(v)]]\!](\boldsymbol{r}) & (\text{IH}) \\
\Leftrightarrow & \exists v' \in [\![\tau^*]\!] \;\; \pi \in [\![A^*[\psi(\rho), x^{\tau^*} \leftarrow v']]\!](\boldsymbol{r}) & \\
\Leftrightarrow & \exists v' \in [\![\tau^*]\!] \;\; \pi \in [\![(A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}, x^{\tau^*} \leftarrow v']]\!] & \\
\Leftrightarrow & \pi \in [\![(\forall x^{\tau^*} (A^* r))[\psi(\rho), r \leftarrow \boldsymbol{r}]]\!] & \\
\Leftrightarrow & \pi \in [\![(\lambda r^\kappa . \forall x^{\tau^*} (A^* r))[\psi(\rho)]]\!](\boldsymbol{r}) & \\
\Leftrightarrow & \pi \in [\![(\forall x^\tau A)^*[\psi(\rho)]]\!](\boldsymbol{r})\,, &
\end{array}$$

hence $\psi_o([\![(\forall x^\tau A)[\rho]]\!]^*) = [\![(\forall x^\tau A)^*[\psi(\rho)]]\!]$.

— Equational implication. Let us assume that the property (1) holds for two terms M_1 and M_2 of kind τ and for a proposition A (IH), and consider a valuation ρ that closes the proposition $M_1 = M_2 \mapsto A$ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$. For all $\pi \in \mathbf{\Pi}$ and $\boldsymbol{r} \in [\![\kappa]\!]$

we have the equivalences:

$$\begin{array}{cll}
 & \pi \in \psi_o\big(\llbracket (M_1 = M_2 \mapsto A)[\rho] \rrbracket^*\big)(\boldsymbol{r}) & \\
\Leftrightarrow & (\pi, \boldsymbol{r}) \in \llbracket (M_1 = M_2 \mapsto A)[\rho] \rrbracket^* & \text{(Def of } \psi_o) \\
\Leftrightarrow & \llbracket M_1[\rho] \rrbracket^* = \llbracket M_2[\rho] \rrbracket^* \ \wedge\ (\pi, \boldsymbol{r}) \in \llbracket A[\rho] \rrbracket^* & \\
\Leftrightarrow & \llbracket M_1[\rho] \rrbracket^* = \llbracket M_2[\rho] \rrbracket^* \ \wedge\ \pi \in \psi_o(\llbracket A[\rho] \rrbracket^*)(\boldsymbol{r}) & \text{(Def of } \psi_o) \\
\Leftrightarrow & \psi_\tau(\llbracket M_1[\rho] \rrbracket^*) = \psi_\tau(\llbracket M_2[\rho] \rrbracket^*) \ \wedge\ \pi \in \psi_o(\llbracket A[\rho] \rrbracket^*)(\boldsymbol{r}) & (\psi_\tau \text{ into}) \\
\Leftrightarrow & \llbracket M_1^*[\psi(\rho)] \rrbracket = \llbracket M_2^*[\psi(\rho)] \rrbracket \ \wedge\ \pi \in \llbracket A^*[\psi(\rho)] \rrbracket(\boldsymbol{r}) & \text{(IH)} \\
\Leftrightarrow & \llbracket M_1^*[\psi(\rho)] \rrbracket = \llbracket M_2^*[\psi(\rho)] \rrbracket \ \wedge\ \pi \in \llbracket (A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}] \rrbracket & \\
\Leftrightarrow & \pi \in \llbracket (M_1^* = M_2^* \mapsto A^* r)[\psi(\rho), r \leftarrow \boldsymbol{r}] \rrbracket & \\
\Leftrightarrow & \pi \in \llbracket (\lambda r^\kappa . M_1^* = M_2^* \mapsto A^* r)[\psi(\rho)] \rrbracket(\boldsymbol{r}) & \\
\Leftrightarrow & \pi \in \llbracket (M_1 = M_2 \mapsto A)^*[\psi(\rho)] \rrbracket(\boldsymbol{r})\,. &
\end{array}$$

Hence $\psi_o(\llbracket (M_1 = M_2 \mapsto A)[\rho] \rrbracket^*) = \llbracket (M_1 = M_2 \mapsto A)^*[\psi(\rho)] \rrbracket$.

— Implication. Let us assume that the property (1) (and thus also (2)) holds for two propositions A and B (IH), and consider a valuation ρ that closes the proposition $A \Rightarrow B$ in the forcing model $\mathscr{M}_{\mathscr{A}^*}$. We first notice that

$$\begin{array}{cll}
 & ((c, \boldsymbol{q}) \Vdash_{\mathscr{A}^*} A[\rho]) \ \wedge\ (\pi', \boldsymbol{r}') \in \llbracket B[\rho] \rrbracket^* & \\
\Leftrightarrow & ((c, \boldsymbol{q}) \Vdash_{\mathscr{A}^*} A[\rho]) \ \wedge\ \pi' \in \psi_o(\llbracket B[\rho] \rrbracket^*)(\boldsymbol{r}') & \text{(Def. of } \psi_o) \\
\Leftrightarrow & (c \Vdash_{\mathscr{A}} (q \Vdash A)[\psi(\rho), q \leftarrow \boldsymbol{q}]) \ \wedge\ \pi' \in \llbracket B^*[\psi(\rho)] \rrbracket(\boldsymbol{r}') & \text{(IH)} \\
\Leftrightarrow & (c \Vdash_{\mathscr{A}} (q \Vdash A)[\psi(\rho), q \leftarrow \boldsymbol{q}]) \ \wedge\ \pi' \in \llbracket (B^* r')[\psi(\rho), r' \leftarrow \boldsymbol{r}'] \rrbracket & \\
\Leftrightarrow & (c \cdot \pi') \in \llbracket ((q \Vdash A) \Rightarrow B^* r')[\psi(\rho), q \leftarrow \boldsymbol{q}, r' \leftarrow \boldsymbol{r}'] \rrbracket & (*)
\end{array}$$

for all $c \in \boldsymbol{\Lambda}$, $\pi' \in \boldsymbol{\Pi}$ and $\boldsymbol{q}, \boldsymbol{r}' \in \llbracket \kappa \rrbracket$. From the above equivalence $(*)$, we deduce that for all $\pi \in \boldsymbol{\Pi}$ and $\boldsymbol{r} \in \llbracket \kappa \rrbracket$, we have

$$\begin{array}{cll}
 & \pi \in \psi_o\big(\llbracket (A \Rightarrow B)[\rho] \rrbracket^*\big)(\boldsymbol{r}) & \\
\Leftrightarrow & (\pi, \boldsymbol{r}) \in \llbracket (A \Rightarrow B)[\rho] \rrbracket^* & \text{(Def. of } \psi_o) \\
\Leftrightarrow & \exists (c, \boldsymbol{q}) \in \boldsymbol{\Lambda}^*\ \exists (\pi', \boldsymbol{r}') \in \boldsymbol{\Pi}^*\ [\pi \equiv c \cdot \pi' \ \wedge\ \boldsymbol{r} = \boldsymbol{q}\boldsymbol{r}' \ \wedge & \\
 & \qquad ((c, \boldsymbol{q}) \Vdash_{\mathscr{A}^*} A[\rho]) \ \wedge\ (\pi', \boldsymbol{r}') \in \llbracket B[\rho] \rrbracket^*] & \\
\Leftrightarrow & \exists (c, \boldsymbol{q}) \in \boldsymbol{\Lambda}^*\ \exists (\pi', \boldsymbol{r}') \in \boldsymbol{\Pi}^*\ [\pi \equiv c \cdot \pi' \ \wedge\ \boldsymbol{r} = \boldsymbol{q}\boldsymbol{r}' \ \wedge & \\
 & \qquad (c \cdot \pi') \in \llbracket ((q \Vdash A) \Rightarrow B^* r')[\psi(\rho), q \leftarrow \boldsymbol{q}, r' \leftarrow \boldsymbol{r}'] \rrbracket] & (*) \\
\Leftrightarrow & \exists \boldsymbol{q} \in \llbracket \kappa \rrbracket\ \exists \boldsymbol{r}' \in \llbracket \kappa \rrbracket\ [\boldsymbol{r} = \boldsymbol{q}\boldsymbol{r}' \ \wedge & \\
 & \qquad \pi \in \llbracket ((q \Vdash A) \Rightarrow B^* r')[\psi(\rho), q \leftarrow \boldsymbol{q}, r' \leftarrow \boldsymbol{r}'] \rrbracket] & \\
\Leftrightarrow & \exists \boldsymbol{q} \in \llbracket \kappa \rrbracket\ \exists \boldsymbol{r}' \in \llbracket \kappa \rrbracket & \\
 & \quad \pi \in \llbracket (r = q r' \mapsto (q \Vdash A) \Rightarrow B^* r')[\psi(\rho), r \leftarrow \boldsymbol{r}, q \leftarrow \boldsymbol{q}, r' \leftarrow \boldsymbol{r}'] \rrbracket & \\
\Leftrightarrow & \pi \in \llbracket (\forall q^\kappa\, \forall r'^\kappa\, (r = q r' \mapsto (q \Vdash A) \Rightarrow B^* r'))[\psi(\rho), r \leftarrow \boldsymbol{r}] \rrbracket & \\
\Leftrightarrow & \pi \in \llbracket (\lambda r^\kappa . \forall q^\kappa\, \forall r'^\kappa\, (r = q r' \mapsto (q \Vdash A) \Rightarrow B^* r'))[\psi(\rho)] \rrbracket(\boldsymbol{r}) & \\
\Leftrightarrow & \pi \in \llbracket (A \Rightarrow B)^*[\psi(\rho)] \rrbracket(\boldsymbol{r})\,, &
\end{array}$$

hence $\psi_o\big(\llbracket (A \Rightarrow B)[\rho] \rrbracket^*\big) = \llbracket (A \Rightarrow B)^*[\psi(\rho)] \rrbracket$. □

7.3. *Proof of adequacy in forcing mode*

We now have all the ingredients to prove the property of adequacy in forcing mode:

Proof of Prop. 53 p. 45. Let us assume that the judgment

$$\mathcal{E}; x_1 : A_1, \ldots, x_n : A_n \vdash t : A$$

is derivable in system PAω^+, and consider:

— a total valuation ρ in the model $\mathscr{M}_{\mathscr{A}}$ such that $\rho \models \mathcal{E}^*$;
— n fresh condition variables $p_1, \ldots, p_n$;
— n closures $c_1, \ldots, c_n \in \mathbf{\Lambda}$ such that $c_1 \Vdash_{\mathscr{A}} (p_1 \Vdash A_1)[\rho]$, $\ldots$, $c_n \Vdash_{\mathscr{A}} (p_n \Vdash A_n)[\rho]$.

Let us write $\boldsymbol{p}_i = \rho(p_i)$ for all $i \in [1..n]$, and consider the total valuation ρ_* in the forcing model $\mathscr{M}_{\mathscr{A}^*}$ that is defined by $\rho_*(x^\tau) = \psi_\tau^{-1}(\rho(x^{\tau^*})) \in [\![\tau]\!]^*$ for all x^τ. By construction, the two valuations $\psi(\rho_*)$ and ρ coincide on the set of all transformed variables x^{τ^*}. Therefore, for each $i \in [1..n]$, the valuations ρ and $\psi(\rho_*), p_i \leftarrow \boldsymbol{p}_i$ coincide on the set of free variables of the proposition $p_i \Vdash A_i$, so that by Lemma 44 we have

$$c_i \Vdash_{\mathscr{A}} (p_i \Vdash A_i)[\psi(\rho_*), p_i \leftarrow \boldsymbol{p}_i]\,, \qquad (i \in [1..n])$$

hence from the connection theorem (Theorem 63) we get

$$(c_i, \boldsymbol{p}_i) \Vdash_{\mathscr{A}^*} A_i[\rho_*] \qquad (i \in [1..n])\,.$$

To apply the property of adequacy (Prop. 48) in the algebra $\mathscr{A}^*$, we still need to check that $\rho_* \models \mathcal{E}$. For that, we notice that for every equation $(M_1 = M_2) \in \mathcal{E}$ (where M_1, M_2 are of kind τ), we have

$$[\![M_1^*[\rho]]\!] \;=\; [\![M_2^*[\rho]]\!]$$

hence (by Lemma 44)

$$[\![M_1^*[\psi(\rho_*)]]\!] \;=\; [\![M_2^*[\psi(\rho_*)]]\!]$$

so that from the connection theorem we get

$$\psi_\tau([\![M_1[\rho_*]]\!]^*) \;=\; \psi_\tau([\![M_2[\rho_*]]\!]^*)\,,$$

and thus

$$[\![M_1[\rho_*]]\!]^* \;=\; [\![M_2[\rho_*]]\!]^*$$

since ψ_τ is bijective. Therefore $\rho_* \models \mathcal{E}$ (in the realizability algebra $\mathscr{A}^*$). Applying the general property of adequacy (Prop. 48) in this algebra $\mathscr{A}^*$, we thus have:

$$t[x_1 := (c_1, \boldsymbol{p}_1), \ldots, x_n := (c_n, \boldsymbol{p}_n)] \Vdash_{\mathscr{A}^*} A[\rho_*]\,,$$

that is:

$$(t[x_1 := c_1, \ldots, x_n := c_n]^*,\ ((\mathbf{1}\boldsymbol{p}_1)\cdots\boldsymbol{p}_n)) \Vdash_{\mathscr{A}^*} A[\rho_*]$$

unfolding the definition of the compilation function in the algebra $\mathscr{A}^*$. Applying the connection theorem again, we deduce that

$$t[x_1 := c_1, \ldots, x_n := c_n]^* \Vdash_{\mathscr{A}} (p \Vdash A)[\psi(\rho_*), p \leftarrow ((\mathbf{1}\boldsymbol{p}_1)\cdots\boldsymbol{p}_n)]\,,$$

(where p is a fresh condition variable), so that

$$t[x_1 := c_1, \ldots, x_n := c_n]^* \Vdash_{\mathscr{A}} (((1p_1)\cdots p_n) \Vdash A)[\psi(\rho_*), p_1 \leftarrow \boldsymbol{p}_1, \ldots, p_n \leftarrow \boldsymbol{p}_n]$$

by Lemma 45. But since the two valuations ρ and $\psi(\rho_*), p_1 \leftarrow \boldsymbol{p}_1, \ldots, p_n \leftarrow \boldsymbol{p}_n$ coincide on all the free variables of the proposition $((1p_1)\cdots p_n) \Vdash A$, we conclude that

$$t[x_1 := c_1, \ldots, x_n := c_n]^* \Vdash_{\mathscr{A}} (((1p_1)\cdots p_n) \Vdash A)[\rho]$$

by Lemma 44. $\square$

Remark 64. The statement we gave in Prop. 53 is actually slightly more general than the statement we proved above, since its conclusion is

$$t[x_1 := c_1, \ldots, x_n := c_n]^* \Vdash_{\mathscr{A}} (((p_0p_1)\cdots p_n) \Vdash A)[\rho]$$

where p_0 is an extra condition variable attached to the proof term t. We thus only proved the desired result in the particular case where $\rho(p_0) = \mathbf{1} = [\![1]\!]$. To generalize the above proof to the case where $\rho(p_0)$ is an arbitrary semantic condition $\boldsymbol{p}_0 \in [\![\kappa]\!]$, we need to alter the definition of the compilation function in the algebra $\mathscr{A}^*$, letting

$$\begin{aligned} t[x_1 := (c_1, \boldsymbol{p}_1), \ldots, x_n := (c_n, \boldsymbol{p}_n)] \ = \\ \big(t[x_1 := c_1, \ldots, x_n := c_n]^*,\ ((\boldsymbol{p}_0\boldsymbol{p}_1)\cdots \boldsymbol{p}_n)\big)\,, \end{aligned}$$

where $\boldsymbol{p}_0 \in [\![\kappa]\!]$ is a fixed semantic condition that is now uniformly attached to all proof terms. Formally, we thus get another forcing algebra written $\mathscr{A}^{*\boldsymbol{p}_0}$, whose compilation function is parameterized by the semantic condition $\boldsymbol{p}_0 \in [\![\kappa]\!]$. (The other components of the algebra $\mathscr{A}^{*\boldsymbol{p}_0}$ are defined as in the algebra $\mathscr{A}^* = \mathscr{A}^{*\mathbf{1}}$.) It is then a simple exercise to check that Prop. 55 and the connection theorem (Theorem 63) still hold if we replace the algebra $\mathscr{A}^* = \mathscr{A}^{*\mathbf{1}}$ by the algebra $\mathscr{A}^{*\boldsymbol{p}_0}$. The general case of Prop. 53 is then proved as above, simply by working in the forcing algebra $\mathscr{A}^{*\boldsymbol{p}_0}$ where $\boldsymbol{p}_0 = \rho(p_0)$.

References

J. Avigad. Forcing in proof theory. *Bulletin of Symbolic Logic*, 10(3):305–333, 2004.

F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Inf. Comput.*, 125(2):103–117, 1996.

J. L. Bell. *Boolean-Valued Models and Independence Proofs in Set Theory.* Oxford, 1985.

P.-L. Curien and H. Herbelin. The duality of computation. In *ICFP*, pages 233–243, 2000.

T. Coquand and G. Jaber. A note on forcing and type theory. *Fundam. Inform.*, 100(1-4):43–52, 2010.

P. J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):1143–1148, December 1963.

P. J. Cohen. The independence of the continuum hypothesis II. *Proceedings of the National Academy of Sciences of the United States of America*, 51(1):105–110, January 1964.

J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

K. Gödel. Consistency of the axiom of choice and of the generalized continuum-hypothesis with the axioms of set theory. *Proceedings of the National Academy of Sciences of the United States of America*, 24(12), 1938.

T. Griffin. A formulae-as-types notion of control. In *Principles Of Programming Languages (POPL'90)*, pages 47–58, 1990.

T. Jech. *Set theory, third millennium edition (revised and expanded).* Springer, 2002.

S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

J. L. Krivine. *Lambda-calculus, types and models.* Masson, 1993.

J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

J.-L. Krivine. Dependent choice, 'quote' and the clock. *Theor. Comput. Sci.*, 308(1-3):259–276, 2003.

J.-L. Krivine. Structures de réalisabilité, RAM et ultrafiltre sur N. Manuscript, available on the author's web page, 2008.

J.-L. Krivine. Realizability in classical logic. In *Interactive models of computation and program behaviour*, volume 27 of *Panoramas et synthèses*, pages 197–229. Société Mathématique de France, 2009.

J.-L. Krivine. Realizability algebras : a program to well order R. Manuscript, available on the author's web page, 2010.

A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods for Computer Science*, 2010.

Alexandre Miquel. Forcing as a program transformation. In *LICS*, pages 197–206. IEEE Computer Society, 2011.

M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. Symb. Log.*, 62(4):1461–1479, 1997.

C. Raffalli and F. Ruyer. Realizability of the axiom of choice in HOL (An analysis of Krivine's work). *Fundamenta Informaticae*, 84(2):241–258, 2008.