An operational domain-theoretic treatment of recursive types

WENG KIN HO

Mathematical Structures in Computer Science / Volume 24 / Issue 01 / February 2014 / e240101
DOI: 10.1017/S0960129512001004, Published online: 19 March 2013

Math. Struct. in Comp. Science (2014), vol. 24, iss. 1, e240101, 59 pages.
doi:10.1017/S0960129512001004

An operational domain-theoretic treatment of recursive types[†]

WENG KIN HO

Mathematics and Mathematics Education, National Institute of Education,
Nanyang Technological University,
1, Nanyang Walk, Singapore 637616, Singapore
Email: wengkin.ho@nie.edu.sg

Received 4 December 2009; revised 31 July 2012

We develop an operational domain theory for treating recursive types with respect to contextual equivalence. The principal approach we take deviates from classical domain theory in that we do not produce the recursive types using the usual inverse limits constructions – we get them for free by working directly with the operational semantics. By extending type expressions to functors between some 'syntactic' categories, we establish algebraic compactness. To do this, we rely on an operational version of the minimal invariance property, for which we give a purely operational proof.

1. Introduction

We develop a domain theory for treating recursive types with respect to contextual equivalence. The sequential language we consider has, in addition to recursive types, sum, product, function and lifted types. It is well known that the domain-theoretic model of such a language is *computationally adequate* but fails to be *fully abstract*, that is, the denotational equality of two terms implies their contextual equivalence but the converse does not hold (Plotkin 1977; Fiore and Plotkin 1994).

In order to cope with this mismatch, we develop the operational counterpart of domain theory that deals with the solutions of recursive domain equations, that is, using the functional programming language, FPC (Fixed-Point Calculus). In this paper, we export domain-theoretic tools directly to the operational setting of FPC. Such an enterprise can be seen as an extension of earlier programs taken on by Mason *et al.* (1996) and Escardó and Ho (2009) for the language PCF. As a foundation for our operational treatment of recursive types, we have put in place an operational domain-theoretic toolkit in the style of Pitts (1997) to suit the setting of FPC.

The principal approach taken in this paper deviates from classical domain theory in that we do not produce recursive types by inverse limits constructions, but get them for free by working directly with the recursive type declaration in FPC. By considering suitable syntactic categories, we extend type expressions to functors. Our approach is

[†] This work was supported by NTU AcRF under Grant No. RP1-10HWK.

similar to earlier work in Abadi and Fiore (1996) in that we too work with the diagonal category $\mathbf{FPC}_!^{\delta}$. The major differences are:

(1) We do not derive the functoriality using the domains model.
(2) We are not constrained by the syntax to work in the diagonal category $\mathbf{FPC}_!^{\delta}$ alone (*cf.* Abadi and Fiore (1996, page 5)). More precisely, we show that it is possible to work with the product category $\mathbf{F\check{P}C}_!$.

To establish functoriality of type expressions, we rely on an operational version of the minimal invariance theorem. An important contribution of this paper is a purely operational proof of this minimal invariance theorem. Similar results and proofs have already appeared in Birkedal and Harper (1999) and Lassen (1998a), but the languages they considered have only one top-level recursive type. Our work here is more general since FPC can deal with nested recursion for types. Furthermore, we develop an operational version of P. Freyd's algebraic compactness (Freyd 1991), which is founded on the preceding categorical constructions. This paper culminates in a new result relating the algebraic compactness results established earlier for $\mathbf{FPC}_!^{\delta}$ and $\mathbf{F\check{P}C}_!$.

1.1. *Structure of the paper*

In Section 2, we introduce the language FPC, together with the operational notions relevant to our study (for example, contextual pre-order and equivalence). In Section 3, we present the operational domain-theoretic toolkit in the style of Pitts (1997) – to avoid unnecessary technical details, we omit the proofs, but give precise references to where they can be found. Section 4 deals with the categorical constructions, where the bulk of the development involves the extension of formal type expressions to endofunctors on these categories. In Section 5, we state the minimal invariance theorem, and give a purely operational proof for it. In Section 6, we consider the (parameterised) algebraic compactness of the syntactic categories constructed in the preceding section.

Throughout the discussion, we assume familiarity with a sequential functional language such as PCF or Haskell. For details of the theory of recursive domain equations, see Abramsky and Jung (1994) and Gierz *et al.* (2003), and for category theory, see Mac Lane (1998) and Poigné (1992).

2. The programming language FPC

We choose to work with a call-by-name version of FPC[†] (Fixed-Point Calculus) whose call-by-value version was first introduced by G. D. Plotkin in his 1985 CSLI lecture notes (Plotkin 1985). In a nutshell, FPC does for recursive definitions of types what PCF does for recursive definitions of functions. In this section we will summarise the syntax and operational semantics of this sequential functional language – FPC experts may skip this section entirely.

[†] Further details on call-by-name FPC can be found in McCusker (2000).

$t :=$	x	term variable
	$\mid (s, t)$	pairs
	$\mid \mathrm{fst}(t)$	first projection
	$\mid \mathrm{snd}(t)$	second projection
	$\mid \mathrm{inl}(t)$	left injection
	$\mid \mathrm{inr}(t)$	right injection
	$\mid \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t\ \mathrm{or}\ \mathrm{inr}(y).\ t'$	case
	$\mid \mathrm{up}(t)$	lifting
	$\mid \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t$	case up
	$\mid \mathrm{fold}(t)$	fold
	$\mid \mathrm{unfold}(t)$	unfold
	$\mid \lambda x.\ t$	function abstraction
	$\mid s(t)$	function application

Fig. 1. FPC syntax.

2.1. *The syntax*

We assume a set of *type variables* (ranged over by X, Y, and so on). *Type expressions* are generated by the following grammar:

$$\sigma := X \mid \sigma \times \sigma \mid \sigma + \sigma \mid \sigma_\perp \mid \mu X.\ \sigma \mid \sigma \to \sigma.$$

For type expressions, we have type variables, product types, separated sum types, lifted types, recursive types and function types. Our choice of the type constructors is natural in the context of call-by-name, and similar choices have also been made in previous work such as McCusker (2000) and Rohr (2002). A *closed type* is a type expression containing no free type variables, that is, where every occurring type variable X is bound under the scope of a recursive type constructor μX.

The set of all closed types is denoted by Type. A *type context* is a list of distinct type variables (which may be empty). We write $\Theta \vdash \sigma$ for the type σ in context Θ, indicating that the set of free type variables occurring in σ is a subset of the type context Θ.

The raw FPC *terms* are given by the syntax trees generated by the grammar given in Figure 1, modulo α-equivalence. Terms of the form $s(t)$ are called *applications*, while those of the form $\lambda x.\ t$ are called *abstractions*. For variables bound by λ's, we employ the usual convention of α-conversion according to which terms are considered as equal (denoted by $s \equiv t$) if they can be obtained from each other by an appropriate renaming of bound variables. The set of free variables appearing in a term t is denoted by $\mathrm{fv}(t)$. Terms containing no free variables are called *closed terms*; terms containing free variables are called *open terms*.

A *term context* is a list of distinct term variables with types. We write $\Theta; \Gamma \vdash t : \sigma$ for a term t in (term) context $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ where $\Theta \vdash \sigma_i$ $(i = 1, \ldots, n)$ are well-formed types-in-context. When there is no risk of confusion, we will omit the type context Θ. The typing rules of FPC are given in Figure 2. Collectively, the typing rules (var), (abs), (pair), (up), (inr), (inr) and (fold) are known as *introduction rules* and (app), (fst), (snd), (case up), (case) and (unfold) are known as *elimination rules*.

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ (\text{var}) \qquad \frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x^{\sigma}.\ t : \sigma \to \tau}\ (\text{abs})$$

$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash s(t) : \tau}\ (\text{app}) \qquad \frac{\Gamma \vdash s : \sigma \quad \Gamma \vdash t : \tau}{\Gamma \vdash (s, t) : \sigma \times \tau}\ (\text{pair})$$

$$\frac{\Gamma \vdash t : \sigma \times \tau}{\Gamma \vdash \text{fst}(t) : \sigma}\ (\text{fst}) \qquad \frac{\Gamma \vdash t : \sigma \times \tau}{\Gamma \vdash \text{snd}(t) : \tau}\ (\text{snd})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \text{up}(t) : \sigma_{\perp}}\ (\text{up}) \qquad \frac{\Gamma \vdash s : \sigma_{\perp} \quad \Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \text{case}(s)\ \text{of}\ \text{up}(x).\ t : \tau}\ (\text{case up})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \vdash \text{inl}(t) : \sigma + \tau}\ (\text{inl}) \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \vdash \text{inr}(t) : \sigma + \tau}\ (\text{inr})$$

$$\frac{\Gamma \vdash s : \sigma_1 + \sigma_2 \quad \Gamma, x : \sigma_1 \vdash t_1 : \tau \quad \Gamma, y : \sigma_2 \vdash t_2 : \tau}{\Gamma \vdash \text{case}(s)\ \text{of}\ \text{inl}(x).\ t_1\ \text{or}\ \text{inr}(y).\ t_2 : \tau}\ (\text{case})$$

$$\frac{\Gamma \vdash t : \sigma[\mu X.\ \sigma / X]}{\Gamma \vdash \text{fold}(t) : \mu X.\ \sigma}\ (\text{fold}) \qquad \frac{\Gamma \vdash t : \mu X.\ \sigma}{\Gamma \vdash \text{unfold}(t) : \sigma[\mu X.\ \sigma / X]}\ (\text{unfold})$$

Fig. 2. Typing rules of FPC.

Conventions 2.1. We use Θ to range over type contexts; X, Y, R, S to range over type variables; ρ, σ, τ to range over type expressions; Γ to range over term contexts; x, y, z, f, g, h to range over terms variables; and s, t, u, v to range over terms. We write $\sigma[\tau/X]$ to represent the result of replacing X with τ in the type expression σ (avoiding the capture of bound variables). Similarly, we write $s[t/x]$ to denote capture-free substitution of free occurrences of the variable x in s by the term t. We use the vector notation for sequences, for example, the term context $x_1 : \sigma_1, \ldots, x_n : \sigma_n$ is abbreviated as $\vec{x} : \vec{\sigma}$, with the default assumption that sequences are of length n.

The following lemma is routine.

Lemma 2.2.

(1) If $\Gamma \vdash t : \sigma$, then $\text{fv}(t) \subseteq \text{dom}(\Gamma)$.
(2) If $\Gamma \vdash t : \sigma$ and $x \notin \text{dom}(\Gamma)$, then $\Gamma, x : \tau \vdash t : \sigma$ for any τ.
(3) If $\Gamma, \Gamma' \vdash t : \sigma$ and $\text{fv}(t) \subseteq \text{dom}(\Gamma)$, then $\Gamma \vdash t : \sigma$.
(4) If $\Gamma \vdash t_i : \sigma_i$ for $i = 1, \ldots, n$ and $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash s : \sigma$, then $\Gamma \vdash s[\vec{t}/\vec{x}] : \sigma$.

We use $\text{Exp}_{\sigma}(\Gamma)$ to denote the set of FPC terms that can be assigned the closed type σ, given Γ, that is, $\text{Exp}_{\sigma}(\Gamma) := \{t | \Gamma \vdash t : \sigma\}$. We simply write Exp_{σ} for $\text{Exp}_{\sigma}(\varnothing)$.

$$\frac{}{v \Downarrow v}\ (\Downarrow \text{can}) \qquad \frac{s \Downarrow \lambda x.\ s' \quad s'[t/x] \Downarrow v}{s(t) \Downarrow v}\ (\Downarrow \text{app})$$

$$\frac{p \Downarrow (s,t) \quad s \Downarrow v}{\text{fst}(p) \Downarrow v}\ (\Downarrow \text{fst}) \qquad \frac{p \Downarrow (s,t) \quad t \Downarrow v}{\text{snd}(p) \Downarrow v}\ (\Downarrow \text{snd})$$

$$\frac{s \Downarrow \text{up}(t') \quad t[t'/x] \Downarrow v}{\text{case}(s)\ \text{of}\ \text{up}(x).\ t \Downarrow v}\ (\Downarrow \text{case up}) \qquad \frac{s \Downarrow \text{fold}(t) \quad t \Downarrow v}{\text{unfold}(s) \Downarrow v}\ (\Downarrow \text{unfold})$$

$$\frac{s \Downarrow \text{inl}(t) \quad t_1[t/x] \Downarrow v}{\text{case}(s)\ \text{of}\ \text{inl}(x).\ t_1\ \text{or}\ \text{inr}(y).\ t_2 \Downarrow v}\ (\Downarrow \text{case inl})$$

$$\frac{s \Downarrow \text{inr}(t) \quad t_2[t/y] \Downarrow v}{\text{case}(s)\ \text{of}\ \text{inl}(x).\ t_1\ \text{or}\ \text{inr}(y).\ t_2 \Downarrow v}\ (\Downarrow \text{case inr})$$

Fig. 3. Rules for evaluating FPC terms.

2.2. *Operational semantics*

The operational semantics is given by an evaluation relation $\Downarrow$ of the form $t \Downarrow v$, where t and v are closed FPC terms, and v is in canonical form:

$$v := (s,t) \mid \text{inl}(t) \mid \text{inr}(t) \mid \text{up}(t) \mid \text{fold}(t) \mid \lambda x.\ t.$$

A closed term v generated by the above grammar is called a *value*. Let Val_σ denote the set of values of the closed type σ, that is,

$$\text{Val}_\sigma := \{v \mid \varnothing \vdash v : \sigma\}.$$

The relation $\Downarrow$ is defined inductively in Figure 3.

Proposition 2.3. Evaluation is deterministic and preserves typing, that is:

(1) If $t \Downarrow v$ and $t \Downarrow v'$, then $v \equiv v'$.
(2) If $t \Downarrow v$ and $t \in \text{Exp}_\sigma$, then $v \in \text{Exp}_\sigma$.

2.3. *Fixed-point operator*

As in previous work such as Rohr (2002), we can define a fixed-point operator using the recursive types. This is done as follows:

$$\text{fix}_\sigma := \lambda f : (\sigma \to \sigma).\ k(\text{fold}^\tau(k))$$

with $\tau := \mu X.\ (X \to \sigma)$ and $k := \lambda x^\tau.\ f(\text{unfold}^\tau(x)x)$.

Those familiar with call-by-name PCF should note that the following fixed-point operator evaluation rule *does not* hold:

$$\frac{f(\text{fix}_\sigma(f)) \Downarrow v}{\text{fix}_\sigma(f) \Downarrow v}.$$

We will see later that what does turn out to be true for this fixed-point operator is the following operational property: $f(\mathrm{fix}_\sigma(f))$ is contextually equivalent to $\mathrm{fix}_\sigma(f)$ (see Property (3.48)) – the concept of contextual equivalence will be defined in Section 2.5.

2.4. *Some notation*

In this section, we gather together the notation we use for the FPC syntax.

For each FPC closed type σ, we define the *syntactic bottom* of type σ to be the term

$$\bot_\sigma := \mathrm{fix}_\sigma(\lambda x^\sigma.\ x).$$

The following three special closed types are worth mentioning:

$$1 := \mu X.\ X, \quad \Sigma := 1_\bot, \quad \overline{\omega} := \mu X.\ (X_\bot).$$

The type 1 is called the *void type* and contains no values. Lifting the type 1 produces the *unit type*, $1_\bot$, which we denote by Σ. The non-divergent element of Σ, $\mathrm{up}(\bot_1)$, is denoted by $\top$. We shall exploit Σ to make program observations.

Given $a : \Sigma$ and $b : \sigma$, we define

$$\mathrm{if}\ \ a\ \ \mathrm{then}\ \ b := \mathrm{case}(a)\ \ \mathrm{of}\ \ \mathrm{up}(x).\ b.$$

Note that 'if a then b' is an 'if-then' construct without the usual 'else'.

The *ordinal type* $\overline{\omega}$ has elements $0, 1, \ldots, \infty$, which can be encoded by defining

$$0 := \bot_{\overline{\omega}},\ n+1 = \mathrm{fold}(\mathrm{up}(n)) \text{ and } \infty := \mathrm{fix}(+1),$$

where $(+1) := \lambda x.\ x+1$. We define

$$n-1 := \mathrm{case}(\mathrm{unfold}(n))\ \ \mathrm{of}\ \ \mathrm{up}(x).\ x.$$

The Σ-valued convergence test

$$(>0) := \lambda x^{\overline{\omega}}.\ \mathrm{if}\ \ \mathrm{unfold}(x)\ \ \mathrm{then}\ \ \top$$

evaluates to $\top$ if and only if x evaluates to $n+1$ for some $n : \overline{\omega}$.

2.5. *FPC contexts*

The *FPC contexts*, C, are syntax trees generated by the grammar for FPC terms given in Figure 1 augmented by the clause

$$C ::= \ldots \mid \mathrm{p}$$

where p ranges over a fixed set of *parameters* (or holes).

Convention 2.4. We use capital letters, for instance, C, T and V to range over FPC contexts.

We assume a function that assigns types to parameters and write $-_\sigma$ to indicate that a parameter $-$ has closed type σ. We restrict ourselves to FPC contexts that are typable. The relation

$$\Gamma \vdash C : \sigma$$

assigning a closed type σ to an FPC context C given the typing context Γ is generated inductively by axioms and rules just like Figure 2, together with the following axiom for parameters:

$$\Gamma \vdash -_\sigma : \sigma.$$

We write $C[-_\sigma]$ to indicate that C is a context containing no parameters other than $-_\sigma$. If t is an FPC term, then $C[t]$ denotes the term resulting from choosing a representative syntax tree for t, substituting it for the parameter in C and forming the α-equivalence class of the resulting FPC syntax tree.

We define

$$\mathrm{Ctx}_\sigma(\Gamma) := \{C | \Gamma \vdash C : \sigma\}$$

to be the set of FPC contexts that can be assigned to the closed type σ, given Γ. We write Ctx_σ for $\mathrm{Ctx}_\sigma(\varnothing)$.

Given Γ, Γ' and $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma')$, we say that Γ is *trapped within* $C[-_\sigma]$ if for each term variable x in Γ, every occurrence of $-_\sigma$ appears in the scope of a binder of x. For example, $\Gamma = x : \sigma$ is trapped in the FPC context

$$C_1[-_\sigma] := \lambda x.\ -_\sigma$$

but not in the FPC context

$$C_2[-_\sigma] := (\lambda x.\ -_\sigma)(\mathrm{if}\ \ -_\sigma\ \ \mathrm{then}\ 1\ \mathrm{else}\ 2).$$

Let $\Gamma \vdash s, t : \sigma$ be two FPC terms-in-context. We write

$$\Gamma \vdash s \sqsubseteq_\sigma t$$

to mean that for all *ground* contexts $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ with Γ trapped within $C[-_\sigma]$,

$$C[s] \Downarrow \top \implies C[t] \Downarrow \top.$$

The relation $\sqsubseteq$ is called the *contextual preorder* and its symmetrisation is called the *contextual equivalence*, which is denoted by $=$. For a given term σ, the order induced by the preorder $\sqsubseteq$ on the set of equivalence classes of closed terms of type σ is called the *contextual order*. Note that we have chosen the ground type Σ to be the type on which program observations are based.

Remark 2.5. Let $s, t : \sigma$ be closed terms. Then $s \sqsubseteq_\sigma t$ if and only if

$$\forall p : \sigma \to \Sigma.\ (p(s) \Downarrow \top \implies p(t) \Downarrow \top).$$

Proof.

($\Longrightarrow$): For each function $p : \sigma \to \Sigma$, we define the context $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ to be $p(-_\sigma)$.

($\Longleftarrow$): Given a context $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$, we define the function $p : \sigma \to \Sigma$ to be $\lambda x^\sigma.C[x]$ where x is a fresh variable not trapped within $C[-_\sigma]$. □

$$\forall s,\ s : 1,\ \text{define}\ \ s\langle\mathcal{R}\rangle_1 s'. \tag{3.1}$$

$$p\langle\mathcal{R}\rangle_{\sigma\times\tau} p' \iff \mathrm{fst}(p)\ \mathcal{R}_\sigma\ \mathrm{fst}(p')\ \wedge\ \mathrm{snd}(p)\ \mathcal{R}_\tau\ \mathrm{snd}(p') \tag{3.2}$$

$$\begin{aligned} s\langle\mathcal{R}\rangle_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma.\ \forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.\ s' \Downarrow \mathrm{inl}(a')\ \wedge\ a\ \mathcal{R}_\sigma\ a')\ \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.\ s' \Downarrow \mathrm{inl}(b')\ \wedge\ b\ \mathcal{R}_\tau\ b') \end{aligned} \tag{3.3}$$

$$\begin{aligned} t\langle\mathcal{R}\rangle_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s' \in \mathrm{Exp}_\sigma.\ t' \Downarrow \mathrm{up}(s')\ \wedge\ s\ \mathcal{R}_\sigma\ s') \end{aligned} \tag{3.4}$$

$$t\langle\mathcal{R}\rangle_{\mu X.\sigma} t' \iff \mathrm{unfold}(t)\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') \tag{3.5}$$

$$f\langle\mathcal{R}\rangle_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.\ (f(t)\ \mathcal{R}_\tau\ f'(t)) \tag{3.6}$$

Fig. 4. Definition of $\langle\mathcal{R}\rangle$ in FPC.

3. Foundations

In this section, we present the operational toolkit we shall use during the ensuing treatment of recursive types. The operational machinery described here is a reworking of Pitts (1997) to suit our needs in FPC. Crucially, we rely on it to reason about program equivalence without appeal to any denotational model. The operational proofs mentioned in this section (whether shown or cited) are independent of the properties described in the rest of the paper. Due to limited space, we will omit proofs of many technical lemmas in Section 3.1, but will give precise references whenever such omissions occur.

3.1. *FPC (bi)simulation and (bi)similarity*

As FPC (bis)simulation and (bi)similarity are the main operational tools through which we derive many properties of the contextual preorder and equivalence, we will devote this short section to them.

Let $\mathcal{R} = \{\mathcal{R}_\sigma |\ \sigma \in \mathrm{Type}\}$ be an type-indexed family of binary relations $\mathcal{R}_\sigma$ between closed FPC terms of type σ. Given $\mathcal{R}$, the definitions of $\langle\mathcal{R}\rangle$ and $[\mathcal{R}]$ are given in Figures 4 and 5. Because the operators $\mathcal{R} \mapsto \langle\mathcal{R}\rangle$ and $\mathcal{R} \mapsto [\mathcal{R}]$ are monotone on the set of all type-indexed families of binary relations between closed FPC terms, by the Tarski–Knaster's Fixed-Point Theorem they have greatest (post-)fixed points.

Definition 3.1. A type-indexed family $\mathcal{S}$ of binary relations $\mathcal{S}_\sigma$ between the closed FPC terms of closed type σ that satisfies $\mathcal{S} \subseteq \langle\mathcal{S}\rangle$ (respectively, $\mathcal{S} \subseteq [\mathcal{S}]$) is called an *FPC simulation* (respectively, *FPC bisimulation*), and the greatest such is called an *FPC similarity* (respectively, *FPC bisimilarity*) and is denoted by $\preceq$ (respectively, $\simeq$).

The definition of $\langle\mathcal{R}\rangle$ should be compared with the extensionality properties (3.36)–(3.42), which we claim to hold. The idea is to first define $\langle\mathcal{R}\rangle$ (respectively, $[\mathcal{R}]$) in such a way as to 'model' the extensionality properties we have in mind, and once we have established that the contextual preorder is a bisimulation, it is immediate that it satisfies these extensionality properties.

$$\forall s,\ s : 1,\quad \text{define}\ \ s[\mathcal{R}]_1 s'. \tag{3.7}$$

$$p[\mathcal{R}]_{\sigma\times\tau} p' \iff \mathrm{fst}(p)\ \mathcal{R}_\sigma\ \mathrm{fst}(p')\ \wedge\ \mathrm{snd}(p)\ \mathcal{R}_\tau\ \mathrm{snd}(p') \tag{3.8}$$

$$\begin{aligned} s[\mathcal{R}]_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma.\ \forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.\ s' \Downarrow \mathrm{inl}(a')\ \wedge\ a\ \mathcal{R}_\sigma\ a')\ \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.\ s' \Downarrow \mathrm{inl}(b')\ \wedge\ b\ \mathcal{R}_\tau\ b') \end{aligned} \tag{3.9}$$

$$\begin{aligned} & \text{and} \\ & \forall a' \in \mathrm{Exp}_\sigma.\ \forall b' \in \mathrm{Exp}_\tau. \\ & (s' \Downarrow \mathrm{inl}(a') \implies \exists a \in \mathrm{Exp}_\sigma.\ s \Downarrow \mathrm{inl}(a)\ \wedge\ a\ \mathcal{R}_\sigma\ a')\ \wedge \\ & (s' \Downarrow \mathrm{inr}(b') \implies \exists b \in \mathrm{Exp}_\tau.\ s \Downarrow \mathrm{inl}(b)\ \wedge\ b\ \mathcal{R}_\tau\ b') \end{aligned} \tag{3.10}$$

$$\begin{aligned} t[\mathcal{R}]_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s' \in \mathrm{Exp}_\sigma.\ t' \Downarrow \mathrm{up}(s')\ \wedge\ s\ \mathcal{R}_\sigma\ s') \\ & \text{and} \\ & \forall s' \in \mathrm{Exp}_\sigma. \\ & (t' \Downarrow \mathrm{up}(s') \implies \exists s \in \mathrm{Exp}_\sigma.\ t \Downarrow \mathrm{up}(s)\ \wedge\ s\ \mathcal{R}_\sigma\ s') \end{aligned} \tag{3.11}$$

$$t[\mathcal{R}]_{\mu X.\sigma} t' \iff \mathrm{unfold}(t)\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') \tag{3.12}$$

$$f[\mathcal{R}]_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.\ (f(t)\ \mathcal{R}_\tau\ f'(t)) \tag{3.13}$$

Fig. 5. Definition of $[\mathcal{R}]$ in FPC.

We will often use the powerful proof principle given by the following proposition.

Proposition 3.2 (co-induction principle for $\preceq$ and $\simeq$). Given $s,\ t : \sigma$, to prove that $s \simeq_\sigma t$ (respectively, $s \preceq_\sigma t$), it suffices to find an FPC bisimulation $\mathcal{B}$ (respectively, an FPC simulation $\mathcal{S}$) such that $s\ \mathcal{B}_\sigma\ t$ (respectively, $s\ \mathcal{S}_\sigma\ t$).

Proof. If $\mathcal{B} \subseteq [\mathcal{B}]$, then $\mathcal{B} \subseteq \simeq$ since $\simeq$ is the greatest post-fixed point of $[-]$, so $\mathcal{B}_\sigma \subseteq \simeq_\sigma$. Thus, if $s\ \mathcal{B}_\sigma\ t$, then $s \simeq_\sigma t$. □

Once we have established that FPC bisimilarity and contextual equivalence coincide, the co-induction principle will provide a powerful tool for proving contextual equivalence. The following facts about (bi)similarity are a direct consequence of the co-induction principle.

Proposition 3.3. FPC similarity is a preorder and FPC bisimilarity is the equivalence relation induced by it, that is, for all closed types σ and all closed terms $t,\ t',\ t'' \in \mathrm{Exp}_\sigma$:

(1) $t \preceq_\sigma t$.
(2) $(t \preceq_\sigma t'\ \wedge\ t' \preceq_\sigma t'') \implies t \preceq_\sigma t''$.
(3) $(t \preceq_\sigma t'\ \wedge\ t' \preceq_\sigma t) \iff t \simeq_\sigma t'$.

Note that property (3) holds because the evaluation relation is deterministic.

3.2. *Operational extensionality theorem*

We can extend the definitions of $\preceq$ and $\simeq$ from FPC closed terms to all typable FPC terms by considering closed instantiations of open terms. This is done as follows.

Definition 3.4. Suppose $\mathcal{R}$ is a typable term context $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$. Then for any closed type σ and any terms $s,\ s' \in \mathrm{Exp}_\sigma(\Gamma)$, we define

$$\Gamma \vdash s\ \mathcal{R}^\circ_\sigma\ s' \iff \forall t_1 \in \mathrm{Exp}_{\sigma_1}, \ldots, t_n \in \mathrm{Exp}_{\sigma_n}.\ (s[\vec{t}/\vec{x}]\ \mathcal{R}_\sigma\ s'[\vec{t}/\vec{x}]).$$

We call $\mathcal{R}^\circ$ the open extension of $\mathcal{R}$. Applying this construction to $\preceq$ and $\simeq$, we obtain the relations $\preceq^\circ$ and $\simeq^\circ$ on open terms, which we call *open similarity* and *open bisimilarity*, respectively.

Proposition 3.5. FPC open similarity is a preorder and FPC open bisimilarity is the equivalence relation induced by it.

Proof. Because $\preceq^\circ$ is defined using an extension of $\preceq$ by considering closed instantiation of open terms, we can rely on the closed analogue in Proposition 3.3. □

FPC open similarity $\preceq^\circ$ respects substitution in the following sense.

Lemma 3.6. If $\Gamma \vdash t : \sigma$ and $\Gamma, x : \sigma \vdash s \preceq^\circ_\tau s'$ (where $x \notin \mathrm{dom}(\Gamma)$), then

$$\Gamma \vdash s[t/x] \preceq^\circ_\tau s'[t/x].$$

Proof. Let $\Gamma \equiv x_1 : \sigma_1, \ldots x_n : \sigma_n$ and suppose that $t_i \in \mathrm{Exp}_{\sigma_i}$ (for $i = 1, \ldots, n$). Since $\Gamma, x : \sigma \vdash s \preceq^\circ_\tau s'$, it follows that

$$s[t/x, \vec{t}/\vec{x}] \preceq_\tau s'[t/x, \vec{t}/\vec{x}].$$

But

$$(s[t/x])[\vec{t}/\vec{x}] \equiv s[t/x, \vec{t}/\vec{x}]$$

since $x \notin \mathrm{dom}(\Gamma)$, so it follows that

$$(s[t/x])[\vec{t}/\vec{x}] \preceq_\tau (s'[t/x])[\vec{t}/\vec{x}]$$

as required. □

We will now state (without detailed proof) the main result of this section.

Theorem 3.7 (operational extensionality theorem for FPC). Contextual preorder (respectively, equivalence) coincides with similarity (respectively, bisimilarity):

$$\Gamma \vdash t \sqsubseteq_\sigma t' \iff \Gamma \vdash t \preceq^\circ_\sigma t'$$

and

$$\Gamma \vdash t =_\sigma t' \iff \Gamma \vdash t \simeq^\circ_\sigma t'.$$

In particular, the following coinduction principle for contextual equivalence holds: to prove that two closed FPC terms are contextually equivalent, it suffices to to find an FPC bisimulation which relates them.

Proof sketch. The detailed proof is technical. We begin by defining the notions of FPC precongruence and congruence. We then define an auxiliary relation $\preceq^*$ in terms of $\preceq^\circ$, and establish that $\preceq^\circ$ and $\preceq^*$ are equivalent. Exploiting this equivalence, it can be shown that $\preceq^\circ$ is an FPC precongruence and, as a consequence, is contained in the contextual preorder. This special technique is an adaptation of Howe's method (Howe 1989; Howe 1996). We then need to verify that the contextual preorder, when restricted to closed terms, is an FPC simulation. It will then follow, by the co-induction principle, that the contextual preorder is contained in the FPC similarity. The proof will finally be complete once we show that the above required implications can be extended from the closed terms to the open terms. For details of the proof, see Ho (2006b, Chapter 8, pages 92–117). □

3.3. *Kleene preorder and equivalence*

In this section, we look at a very useful notion of program equivalence called *Kleene equivalence*, which turns out to be an FPC bismulation. In certain cases, establishing Kleene equivalence is a convenient way of proving contextual equivalence in the light of Theorem 3.7.

Definition 3.8. For each closed type σ, consider the following binary relations on Exp_σ:

$$t \sqsubseteq^{\mathrm{kl}}_\sigma t' \iff \forall v \in \mathrm{Val}_\sigma.(t \Downarrow v \implies t' \Downarrow v)$$
$$t \simeq^{\mathrm{kl}}_\sigma t' \iff (t \sqsubseteq^{\mathrm{kl}}_\sigma t') \wedge (t' \sqsubseteq^{\mathrm{kl}}_\sigma t).$$

The relation $\sqsubseteq^{\mathrm{kl}}$ is called the *Kleene preorder*. If $t \simeq^{\mathrm{kl}}_\sigma t'$, we say that t and t' are *Kleene equivalent*.

From the evaluation rules of FPC, the following Kleene equivalences hold (the type information has been suppressed):

$$(\lambda x.\ s)t \simeq^{\mathrm{kl}} s[t/x] \tag{3.14}$$
$$\mathrm{fst}(s,t) \simeq^{\mathrm{kl}} s \tag{3.15}$$
$$\mathrm{snd}(s,t) \simeq^{\mathrm{kl}} t \tag{3.16}$$
$$\mathrm{case}(\mathrm{inl}(t))\ \mathrm{of}\ \mathrm{inl}(x).\ s\ \mathrm{or}\ \mathrm{inr}(y).\ s' \simeq^{\mathrm{kl}} s[t/x] \tag{3.17}$$
$$\mathrm{case}(\mathrm{inr}(t))\ \mathrm{of}\ \mathrm{inl}(x).\ s\ \mathrm{or}\ \mathrm{inr}(y).\ s' \simeq^{\mathrm{kl}} s'[t/y] \tag{3.18}$$
$$\mathrm{case}(\mathrm{up}(t))\ \mathrm{of}\ \mathrm{up}(x).\ y \simeq^{\mathrm{kl}} y[t/x] \tag{3.19}$$
$$\mathrm{unfold}(\mathrm{fold}(t)) \simeq^{\mathrm{kl}} t. \tag{3.20}$$

Proposition 3.9. For any closed type σ and any $t,\ t' \in \mathrm{Exp}_\sigma$:

(1) $t \sqsubseteq^{\mathrm{kl}}_\sigma t' \implies t \preceq_\sigma t'$.
(2) $t \simeq^{\mathrm{kl}}_\sigma t' \implies t \simeq_\sigma t'$.

Proof. Because $\simeq$ is the symmetrisation of $\preceq$, (2) follows immediately from (1). To prove (1), it is easy to check that the relation $\sqsubseteq^{\mathrm{kl}}$ is an FPC simulation, and we then invoke Theorem 3.7. □

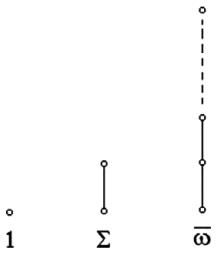


Fig. 6. 1, Σ, $\overline{\omega}$.

3.4. *Contextual orders of* 1, Σ *and* $\overline{\omega}$

We define the type-indexed relation $\mathcal{I}$ on the FPC closed terms by enforcing

$$\forall s,\ s' : 1.\ s\ \mathcal{I}_1\ s',$$

and replacing all occurrences of $[\mathcal{R}]$ in equations (3.8)–(3.13) by $\mathcal{I}$. It is obvious that $\mathcal{I}$ is an FPC bisimulation. So, for any $t : 1$, we have $t\ \mathcal{I}_1\ \bot_1$, which then implies, by the co-induction principle, that $t =_1 \bot_1$. This proves that the contextual order of the void type is the trivial singleton order. Similarly, by choosing suitable FPC bisimulations and applying the co-induction principle, it can be shown that the contextual orders of the unit type Σ and the ordinal type $\overline{\omega}$ are the usual ones interpreted by the Scott model (see Ho (2006b, page 66) for details). The contextual orders of 1, Σ and $\overline{\omega}$ are given in Figure 6.

3.5. *Properties of FPC contextual preorder and equivalence*

With the operational machinery just developed, we are now ready to state and prove the following groups of properties concerning FPC contextual preorder and equivalence.

3.5.1. *Inequational logic.*

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash t \sqsubseteq_\sigma t \tag{3.21}$$

$$\Gamma \vdash t \sqsubseteq_\sigma t' \ \wedge\ \Gamma \vdash t' \sqsubseteq_\sigma t'' \implies \Gamma \vdash t \sqsubseteq_\sigma t'' \tag{3.22}$$

$$\Gamma \vdash t \sqsubseteq_\sigma t' \ \wedge\ \Gamma \vdash t' \sqsubseteq_\sigma t \implies \Gamma \vdash t =_\sigma t' \tag{3.23}$$

$$\Gamma,\ x : \sigma \vdash t \sqsubseteq_\tau t' \implies \Gamma \vdash \lambda x.\ t \sqsubseteq_{\sigma\to\tau} \lambda x.\ t' \tag{3.24}$$

$$\Gamma \vdash s \sqsubseteq_{\sigma_\bot} s' \ \wedge\ \Gamma,\ x : \sigma \vdash t \sqsubseteq_\rho t' \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).\ t' \tag{3.25}$$

$$\Gamma \vdash s \sqsubseteq_{\sigma+\tau} s' \ \wedge\ \Gamma,\ x : \sigma \vdash t_1 \sqsubseteq_\rho t_1' \ \wedge\ \Gamma,\ y : \tau \vdash t_2 \sqsubseteq_\rho t_2' \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2 \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).\ t_1'\ \mathrm{or}\ \mathrm{inr}(y).\ t_2' \tag{3.26}$$

$$\Gamma \vdash t \sqsubseteq_\sigma t' \ \wedge\ \Gamma \subseteq \Gamma' \implies \Gamma' \vdash t \sqsubseteq_\sigma t' \tag{3.27}$$

$$\Gamma \vdash t \sqsubseteq_\sigma t' \ \wedge\ \Gamma,\ x : \sigma \vdash s : \tau \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s[t'/x] \tag{3.28}$$

$$\Gamma \vdash t : \sigma \ \wedge\ \Gamma,\ x : \sigma \vdash s \sqsubseteq_\tau s' \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s'[t/x]. \tag{3.29}$$

Properties (3.21)–(3.28) (which is known as *inequational logic*) are direct consequences of the definitions of $\sqsubseteq_\sigma$ and $=_\sigma$. However, (3.29) cannot be immediately established since the operation $s \mapsto s[t/x]$ is not necessarily of the form $s \mapsto C[s]$ for some FPC context $C[-]$. Instead, it is easy to establish that

$$\Gamma, x : \sigma \vdash s \sqsubseteq_\tau s' \implies \Gamma \vdash (\lambda x.\ s)t \sqsubseteq_\tau (\lambda x.\ s')t$$

using the definition of contextual preorder directly. We can then use the following β-equality (3.30) to achieve the desired result.

3.5.2. *β-equalities.*

$$\Gamma,\ x : \sigma \vdash s : \tau \ \wedge\ \Gamma \vdash t : \sigma \implies \Gamma \vdash (\lambda x.\ s)t =_\tau s[t/x] \tag{3.30}$$

$$\Gamma \vdash s : \sigma \ \wedge\ \Gamma \vdash t : \tau \implies \Gamma \vdash \mathrm{fst}(s,t) =_\sigma s \ \wedge\ \Gamma \vdash \mathrm{snd}(s,t) =_\tau t \tag{3.31}$$

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash \mathrm{case}(t)\ \mathrm{of}\ \mathrm{up}(x).\ s =_\sigma s[t/x] \tag{3.32}$$

$$\Gamma \vdash t : \sigma \implies \forall \Gamma \vdash s : \tau,\ \Gamma \vdash s' : \tau.\ (\Gamma \vdash \mathrm{case}(\mathrm{inl}(t))\ \mathrm{of}\ \mathrm{inl}(x).\ s\ \mathrm{or}\ \mathrm{inr}(y).\ s' =_\sigma s[t/x]) \tag{3.33}$$

$$\Gamma \vdash t : \sigma \implies \forall \Gamma \vdash s : \tau,\ \Gamma \vdash s' : \tau.\ (\Gamma \vdash \mathrm{case}(\mathrm{inr}(t))\ \mathrm{of}\ \mathrm{inl}(x).\ s\ \mathrm{or}\ \mathrm{inr}(y).\ s' =_\sigma s'[t/y]) \tag{3.34}$$

$$\Gamma \vdash t : \sigma[\mu X.\ \sigma/X] \implies \Gamma \vdash \mathrm{unfold}(\mathrm{fold}(t)) =_{\sigma[\mu X.\ \sigma/X]} t. \tag{3.35}$$

These *β-equalities* are valid because of the characterisation of contextual equivalence in terms of the FPC bisimilarity given in Theorem 3.7. This is because in each case the closed instantiations of the term on the left-hand side of $=_\sigma$ are Kleene equivalent to closed instantiations of the right-hand term by (3.14)–(3.20). The β-equalities then follow from Proposition 3.9, together with (3.36) below, which is the first of the following *extensionality properties.*

3.5.3. *Extensionality properties.*

For all $s,\ s' \in \mathrm{Exp}_\sigma(\vec{x} : \vec{\sigma})$,

$$\vec{x} : \vec{\sigma} \vdash s \sqsubseteq_\sigma s' \iff \forall t_i \in \mathrm{Exp}_{\sigma_i}(i = 1, \ldots, n).(s[\vec{t}/\vec{x}] \sqsubseteq_\sigma s'[\vec{t}/\vec{x}]). \tag{3.36}$$

For all $s,\ s' \in \mathrm{Exp}_\Sigma$,

$$s \sqsubseteq_\Sigma s' \iff (s \Downarrow \top \implies s' \Downarrow \top). \tag{3.37}$$

For all $p,\ p' \in \mathrm{Exp}_{\sigma\times\tau}$,

$$p \sqsubseteq_{\sigma\times\tau} p' \iff (\mathrm{fst}(p) \sqsubseteq_\sigma \mathrm{fst}(p') \ \wedge\ \mathrm{snd}(p) \sqsubseteq_\tau \mathrm{snd}(p')). \tag{3.38}$$

For all $s,\ s' \in \mathrm{Exp}_{\sigma+\tau}$,

$$\begin{aligned} s \sqsubseteq_{\sigma+\tau} s' \iff & \forall a \in \mathrm{Exp}_\sigma.\ \forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.\ s' \Downarrow \mathrm{inl}(a') \ \wedge\ a \sqsubseteq_\sigma a') \ \vee \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.\ s' \Downarrow \mathrm{inr}(b') \ \wedge\ a \sqsubseteq_\sigma a'). \end{aligned} \tag{3.39}$$

For all $t,\ t' \in \mathrm{Exp}_{\sigma_\perp}$,

$$\begin{aligned} t \sqsubseteq_{\sigma_\perp} t' \iff & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s' \in \mathrm{Exp}_\sigma.\ t' \Downarrow \mathrm{up}(s') \ \wedge\ s \sqsubseteq_\sigma s'). \end{aligned} \tag{3.40}$$

For all $t,\ t' \in \mathrm{Exp}_{\mu X.\ \sigma}$,

$$t \sqsubseteq_{\mu X.\sigma} t' \iff \mathrm{unfold}(t) \sqsubseteq_{\sigma[\mu X.\ \sigma/X]} \mathrm{unfold}(t'). \tag{3.41}$$

For all $f,\ f' \in \mathrm{Exp}_{\sigma\to\tau}$,

$$f \sqsubseteq_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.\ (f(t) \sqsubseteq_\tau f'(t)). \tag{3.42}$$

Extensionality properties analogous to equations (3.37)–(3.42) (see Figure 4) hold by construction for the notion of FPC similarity defined in Section 3.1. Thus, equations (3.37)–(3.42) follow immediately from the coincidence of FPC bisimilarity with contextual equivalence (Theorem 3.7). Note that by virtue of Theorem 3.7, (3.36) then follows from Lemma 3.6.

3.5.4. *η-equalities.* The following η-equalities hold by combining the extensionality properties with the corresponding β-equality:

$$\Gamma \vdash f : \sigma \to \tau \ \wedge\ x \notin \mathrm{dom}(\Gamma) \implies \Gamma \vdash f =_{\sigma\to\tau} \lambda x.\ f(x) \tag{3.43}$$

$$\Gamma \vdash p : \sigma \times \tau \implies \Gamma \vdash p =_{\sigma\times\tau} (\mathrm{fst}(p), \mathrm{snd}(p)) \tag{3.44}$$

$$\Gamma \vdash t : \sigma + \tau \implies \Gamma \vdash t =_{\sigma+\tau} \mathrm{case}(t)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{inl}(x)\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{inr}(y) \tag{3.45}$$

$$\Gamma \vdash t : \sigma_\perp \implies \Gamma \vdash t =_{\sigma_\perp} \mathrm{case}(t)\ \mathrm{of}\ \mathrm{up}(x).\ \mathrm{up}(x) \tag{3.46}$$

$$\Gamma \vdash t : \mu X.\sigma \implies \Gamma \vdash t =_{\mu X.\ \sigma} \mathrm{fold}(\mathrm{unfold}(t)). \tag{3.47}$$

For instance, to prove (3.47), by virtue of (3.41), we just need to show that

$$\Gamma \vdash \mathrm{unfold}(t) =_{\mu X.\ \sigma} \mathrm{unfold}(\mathrm{fold}(\mathrm{unfold}(t))).$$

But the β-equality (3.35) guarantees this contextual equivalence, so (3.47) holds. Note that properties (3.35) and (3.47) together imply the following proposition.

Proposition 3.10. With respect to the contextual equivalence, fold and unfold are mutual inverses.

This fact will be used frequently in the development of an operational domain theory for treating recursive types in FPC.

3.5.5. *Fixed-point operator.* Recall that

$$\mathrm{fix}_\sigma := \lambda f : (\sigma \to \sigma).\ k(\mathrm{fold}^\tau(k)),$$

where $\tau := \mu X.\ (X \to \sigma)$ and $k := \lambda x^\tau.\ f(\mathrm{unfold}^\tau(x)x)$.

Using the β-equality (3.30), we have

$$\Gamma \vdash \mathrm{fix}_\sigma(f) =_\sigma k(\mathrm{fold}^\tau(k)).$$

But writing k explicitly, it follows from (3.30) and (3.35) that

$$\begin{aligned}\Gamma \vdash \mathrm{fix}_\sigma(f) &\equiv (\lambda x^\tau.\ f(\mathrm{unfold}^\tau(x)x))(\mathrm{fold}^\tau(k))\\ &=_\sigma f(\mathrm{unfold}^\tau(\mathrm{fold}^\tau(\mathrm{fold}^\tau(k))))\\ &=_\sigma f(k(\mathrm{fold}^\tau(k))).\end{aligned}$$

Thus, we have:

$$\Gamma \vdash f : \sigma \to \sigma \implies \mathrm{fix}(f) =_\sigma f(\mathrm{fix}_\sigma(f)). \tag{3.48}$$

3.5.6. *Syntactic bottom.* The term $\bot_\sigma := \mathrm{fix}_\sigma(\lambda x^\sigma.\ x)$ acts as the least element with respect to the contextual preorder $\sqsubseteq_\sigma$:

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash \bot_\sigma \sqsubseteq_\sigma t. \tag{3.49}$$

To establish (3.49), we first observe that $\mathrm{unfold}^\tau(\mathrm{fold}^\tau(k))$ does not evaluate to any value. Indeed, there cannot be a minimal derivation for the evaluation

$$\mathrm{unfold}^\tau(\mathrm{fold}^\tau(k))\mathrm{fold}^\tau(k) \Downarrow v$$

for any value v since if there were one, it would follow from the ($\Downarrow$ unfold) rule and the fact that k is a λ-abstraction that

$$\frac{\mathrm{fold}(k) \Downarrow \mathrm{fold}(k) \quad k \Downarrow k}{\mathrm{unfold}(\mathrm{fold}(k)) \Downarrow k},$$

so the derivation of $\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k) \Downarrow v$ would be

$$\frac{\mathrm{unfold}(\mathrm{fold}(k)) \Downarrow k \qquad \dfrac{\lambda x^\sigma.\ x \Downarrow \lambda x^\sigma.\ x \quad \mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k) \Downarrow v}{(\lambda x^\sigma.\ x)(\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k)) \Downarrow v}}{\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k) \Downarrow v}$$

which contradicts the existence of a minimal derivation of

$$\mathrm{unfold}^\tau(\mathrm{fold}^\tau(k))\mathrm{fold}^\tau(k) \Downarrow v.$$

So, with respect to the Kleene preorder, $\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k)$ is the least element of Exp_σ vacuously. It then follows from Proposition 3.9 that

$$\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k)$$

is the least element of Exp_σ with respect to the contextual preorder. It then follows from the β-equality (3.30) that

$$\begin{aligned}\bot_\sigma &:= \mathrm{fix}\lambda x^\sigma.\ x\\ &=_\sigma (\lambda x^\sigma.\ x)(k\mathrm{fold}(k))\\ &=_\sigma k(\mathrm{fold}(k))\\ &=_\sigma (\lambda x^\sigma.\ x)(\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k))\\ &=_\sigma \mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k).\end{aligned}$$

Consequently, by the transitivity of $\sqsubseteq_\sigma$, it follows that $\bot_\sigma$ is the least element of Exp_σ with respect to the contextual preorder.

3.5.7. *Rational-chain completeness and continuity.* In addition to the fixed-point property (3.48), terms of the form $\mathrm{fix}_\sigma(f)$ enjoy the following least prefixed-point property.

If $f \in \mathrm{Exp}_{\sigma\to\sigma}$ and $t \in \mathrm{Exp}_\sigma$, then

$$f(t) \sqsubseteq_\sigma t \implies \mathrm{fix}_\sigma(f) \sqsubseteq_\sigma t. \tag{3.50}$$

In fact, this least prefixed-point property follows from a more general property, which we will now explain.

Definition 3.11. We define *rational chains* to be chains of the form

$$g(\bot_\sigma) \sqsubseteq_\tau gh(\bot_\sigma) \sqsubseteq_\tau gh^{(2)}(\bot_\sigma) \sqsubseteq_\tau \ldots,$$

where $g : \sigma \to \tau$ and $h : \sigma \to \sigma$ are some function-type FPC closed terms.

Crucially, the following *rational chain completeness* property holds.

Theorem 3.12. For any rational chain of the form

$$g(\bot_\sigma) \sqsubseteq_\tau gh(\bot_\sigma) \sqsubseteq_\tau gh^{(2)}(\bot_\sigma) \sqsubseteq_\tau \ldots,$$

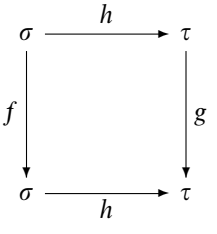
where $g : \sigma \to \tau$ and $h : \sigma \to \sigma$ are some function-type FPC closed terms, we have

$$\bigsqcup_n g(f^{(n)}(\bot_\sigma)) =_\tau g(\mathrm{fix}_\sigma(f)).$$

Proof. An operational proof of this is given in Ho (2006b, pages 81–91). □

A handy result that we will often use is Plotkin's uniformity principle.

Lemma 3.13 (Plotkin's uniformity principle). Let $f : \sigma \to \sigma$, $g : \tau \to \tau$ be FPC programs and $h : \sigma \to \tau$ be a strict program, that is, $h(\bot_\sigma) = \bot_\tau$, such that the diagram



commutes, that is, $g \circ h = h \circ f$. We then have

$$\mathrm{fix}(g) = h(\mathrm{fix}(f)).$$

Proof. Using rational-chain completeness, rational continuity and the identity

$$h \circ f = g \circ h$$

in turn, we get

$$\begin{aligned} h(\mathrm{fix}(f)) &= h\left(\bigsqcup_n f^{(n)}(\bot_\sigma)\right) \\ &= \bigsqcup_n h \circ f^{(n)}(\bot_\sigma) \\ &= \bigsqcup_n g^{(n)} \circ h(\bot_\sigma) \\ &= \bigsqcup_n g^{(n)}(\bot_\tau) \\ &= \mathrm{fix}(g). \end{aligned}$$

□

4. FPC considered as a category

The business proper of the paper starts here. We will begin by setting up an appropriate categorical framework within which we can carry out an operational domain-theoretic treatment of recursive types. In this section, we demonstrate how FPC types-in-context can be viewed as functors.

Definition 4.1. The objects of the category **FPC** are the closed FPC types and the morphisms are closed terms of function type (modulo contextual equivalence). Given closed type σ, the identity morphism id_σ is just the closed term $\lambda x^\sigma.\, x$, and the composition of two morphisms f and g is defined as $g \circ f := \lambda x.g(f(x))$. The category $\mathbf{FPC}_!$ is the subcategory whose morphisms are the strict **FPC**-morphisms. It is an easy consequence of the results in Section 3 that the unit and associativity laws are satisfied by this definition of the composition of morphisms in both **FPC** and $\mathbf{FPC}_!$.

4.1. *Realisable actions and functors*

The difficulties of defining type expressions as functors because of the presence of mixed variance are well known.

(1) Once the function-type $\to$ constructor is involved, we need to separate the covariant and contravariant variables. For instance, $X \to Y$ consists of X as a contravariant variable and Y as a covariant variable.
(2) A given type variable may occur as covariant in one position and contravariant in another within the same type expression. For example, the type variable X in $X \to X$ is contravariant in the first slot and covariant in the second.

The usual solution to this problem of mixed variance, following Freyd (1992), is to work with the *product category* $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$, but we will not take that approach in this section† but instead work with its full subcategory $\mathbf{FPC}_!{}^{\delta}$. We define $\mathbf{FPC}_!{}^{\delta}$, the *diagonal category*, to be the full subcategory of $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$ whose objects are those of $\mathbf{FPC}_!$ and whose morphisms are pairs of $\mathbf{FPC}_!$-morphisms, denoted by $u : \sigma \to \tau$ (or sometimes more explicitly as $\langle u^-, u^+ \rangle$), of the form

$$\sigma \underset{u^-}{\overset{u^+}{\rightleftarrows}} \tau$$

In $\mathbf{FPC}_!{}^{\delta}$, $u \circ v$, is defined as the pair $\langle v^- \circ u^-, u^+ \circ v^+ \rangle$.

Notation 4.2. To avoid excessive use of $+$, $-$ and $\leftrightarrows$, we write

$$f : R \to S \text{ for } f^+ : R \leftrightarrows S : f^-$$
$$\vec{f} : \vec{R} \to \vec{S} \text{ for } f_i^+ : R_i \leftrightarrows S_i : \vec{f_i^-}, \quad i = 1, 2, \ldots, n.$$

We define FPC type expressions as functors in two steps. The first step involves showing how FPC type expressions define syntactic operations called *actions*, and proving that actions preserve composition of $\mathbf{FPC}_!{}^{\delta}$-morphisms. The second step involves establishing an operational minimal invariance theorem, and proving that actions defined by FPC type expressions preserve identity morphisms and are thus functors. We will carry out the first step in this section, but because of the heavy operational machinery involved in proving the operational minimal invariance theorem, we will postpone the second step to Section 5.

Definition 4.3. Let $\mathbf{C}$ and $\mathbf{D}$ be categories. An *action* $F : \mathbf{C} \longrightarrow \mathbf{D}$ is an operation that:

(1) assigns:
   (a) an object $F(C) \in \mathbf{D}$ to an object $C \in \mathbf{C}$, and
   (b) a $\mathbf{D}$-morphism $F(f)$ to a $\mathbf{C}$-morphism $f : C \longrightarrow D$;

(2) preserves composition of $\mathbf{C}$-morphisms, that is, for any $\mathbf{C}$-morphisms

$$f : C_1 \longrightarrow C_2$$
$$g : C_2 \longrightarrow C_3,$$

we have

$$F(g \circ f) = F(g) \circ F(f).$$

An action is not obliged to preserve identity morphisms. When it does, it is actually a functor.

Definition 4.4. A *realisable action* $T : (\mathbf{FPC}_!{}^{\delta})^n \to \mathbf{FPC}_!{}^{\delta}$ is an action that is realised by:

(1) a type-in-context $\vec{X} \vdash \tau$; and

† We shall consider the product category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$ in Section 6.

(2) a pair of terms-in-context of the form

$$\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash t^{-} : \tau[\vec{S}/\vec{X}] \to \tau[\vec{R}/\vec{X}]$$
$$t^{+} : \tau[\vec{R}/\vec{X}] \to \tau[\vec{S}/\vec{X}]$$

such that for any $\vec{\sigma} \in (\mathbf{FPC}_{!}{}^{\delta})^{n}$,

$$T(\vec{\sigma}) = \tau[\vec{\sigma}/\vec{X}]$$

and for any $\vec{\rho},\vec{\sigma} \in (\mathbf{FPC}_{!}{}^{\delta})^{n}$ and any $\vec{u} \in (\mathbf{FPC}_{!}{}^{\delta})^{n}(\vec{\rho},\vec{\sigma})$,

$$T(\vec{u}) = t[\vec{u}/\vec{f}].$$

A realisable action that is also a functor is called a *realisable functor*.

Remark 4.5.

(1) Let $\vec{u},\vec{v} \in (\mathbf{FPC}_{!}{}^{\delta})^{n}(\vec{\rho},\vec{\sigma})$ be given and suppose $\vec{u} \sqsubseteq \vec{v}$. Then, by monotonicity (that is, Property (3.28)), any realisable action is *locally monotone* in the sense that $T(\vec{u}) \sqsubseteq T(\vec{v})$.
(2) Let $\vec{u}_k \in (\mathbf{FPC}_{!}{}^{\delta})^{n}(\vec{\rho},\vec{\sigma})$ be rational chains. Then by rational continuity, any realisable action is *locally continuous* in the sense that $T(\bigsqcup_k \vec{u}_k) = \bigsqcup_k T(\vec{u}_k)$.

Given an FPC type expression $\Theta \vdash \sigma$, we define a corresponding realisable action $F_{\Theta\vdash\sigma}$ by induction on the structure of type expressions:

— *Type variable:*
If $\vec{X} \vdash X_i$, the morphism part of $F_{\vec{X}\vdash X_i}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash f_i^{-} : S_i \to R_i, f_i^{+} : R_i \to S_i.$$

If $\vec{X} \vdash Y$ where $Y \notin \{X_1,\ldots,X_n\}$, we define $F_{\vec{X}\vdash Y} = \mathrm{id}_Y$ whose morphism part is realised by $\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash \mathrm{id}_Y : Y \leftrightarrows Y : \mathrm{id}_Y$. Clearly, $F_{\vec{X}\vdash X_i}$ and $F_{\vec{X}\vdash Y}$ preserve composition of $\mathbf{FPC}_{!}{}^{\delta}$-morphisms.

Next, we use a construction that is simultaneously associated with an inductive proof of correctness, that is, the subsequent syntactic operations described are indeed actions. To this end, we assume the following induction hypotheses:

(a) $F_{\vec{X}\vdash\tau_i}$ $(i = 1,2)$, $F_{\vec{X}\vdash\tau}$ and $F_{\vec{X},X\vdash\sigma}$ are the realisable actions corresponding to $\vec{X} \vdash \tau_i$ $(i = 1,2)$, $\vec{X} \vdash \tau$ and $\vec{X},X \vdash \sigma$, respectively. In particular, these realisable actions preserve composition.
(b) The morphism parts of these realisable actions are realised by
— $\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash t_i : \tau_i[\vec{R}/\vec{X}] \to \tau_i[\vec{S}/\vec{X}] \quad (i = 1,2)$
— $\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash t : \tau[\vec{R}/\vec{X}] \to \tau[\vec{S}/\vec{X}]$, and
— $\vec{R},R,\vec{S},S;\vec{f} : \vec{R} \to \vec{S},f : R \to S \vdash s : \sigma[\vec{R}/\vec{X},R/X] \to \sigma[\vec{S}/\vec{X},S/X]$.

— *Product type:*
The morphism part of $F_{\vec{X}\vdash\tau_1\times\tau_2}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R}\to\vec{S}\vdash t : (\tau_1\times\tau_2)[\vec{R}/\vec{X}]\to(\tau_1\times\tau_2)[\vec{S}/\vec{X}],$$

where

$$t^- := \lambda p : (\tau_1\times\tau_2)[\vec{S}/\vec{X}].\ (t_1^-(\mathrm{fst}(p)), t_2^-(\mathrm{snd}(p)))$$
$$t^+ := \lambda q : (\tau_1\times\tau_2)[\vec{R}/\vec{X}].\ (t_1^+(\mathrm{fst}(q)), t_2^+(\mathrm{snd}(q))).$$

Note that $F_{\vec{X}\vdash\tau_1\times\tau_2}$ preserves composition of $\mathbf{FPC}_!^{\ \delta}$-morphisms since the $F_{\vec{X}\vdash\tau_i}$ do by the induction hypothesis.
The proof that $F_{\vec{X}\vdash\tau_1\times\tau_2}$ preserves composition exploits some operational equalities from Section 3 (for example, Property (3.30)). This is also true, though implicit, for the proof of composition preservation for the various F's in the remaining cases below.

— *Sum type:*
The morphism part of $F_{\vec{X}\vdash\tau_1+\tau_2}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R}\to\vec{S}\vdash t : (\tau_1+\tau_2)[\vec{R}/\vec{X}]\to(\tau_1+\tau_2)[\vec{S}/\vec{X}],$$

where

$$t^- := \lambda z : (\tau_1+\tau_2)[\vec{S}/\vec{X}].\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{inl}(t_1^-(x))\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{inr}(t_2^-(y))$$
$$t^+ := \lambda z : (\tau_1+\tau_2)[\vec{R}/\vec{X}].\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{inl}(t_1^+(x))\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{inr}(t_2^+(y)).$$

Note that $F_{\vec{X}\vdash\tau_1+\tau_2}$ preserves composition of $\mathbf{FPC}_!^{\ \delta}$-morphisms since the $F_{\vec{X}\vdash\tau_i}$ do by the induction hypothesis.

— *Lifted type:*
The morphism part of $F_{\vec{X}\vdash\tau_\perp}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R}\to\vec{S}\vdash t' : \tau_\perp[\vec{R}/\vec{X}]\to\tau_\perp[\vec{S}/\vec{X}],$$

where

$$(t')^- := \lambda z.\tau_\perp[\vec{S}/\vec{X}].\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{up}(x).\ \mathrm{up}(t^-(x))$$
$$(t')^+ := \lambda z.\tau_\perp[\vec{R}/\vec{X}].\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{up}(x).\ \mathrm{up}(t^+(x)).$$

Note that $F_{\vec{X}\vdash\tau_\perp}$ preserves composition of $\mathbf{FPC}_!^{\ \delta}$-morphisms since $F_{\vec{X}\vdash\tau}$ does by the induction hypothesis.

— *Function type:*
The morphism part of $F_{\vec{X}\vdash\tau_1\to\tau_2}$ is given by

$$\vec{R},\vec{S};\vec{f} : \vec{R}\to\vec{S}\vdash t : (\tau_1\to\tau_2)[\vec{R}/\vec{X}]\to(\tau_1\to\tau_2)[\vec{S}/\vec{X}]$$

where

$$t^- := \lambda h : (\tau_1\to\tau_2)[\vec{S}/\vec{X}].\lambda x : \tau_1[\vec{R}/\vec{X}].\ t_2^-(h(t_1^+(x)))$$
$$t^+ := \lambda g : (\tau_1\to\tau_2)[\vec{R}/\vec{X}].\lambda y : \tau_1[\vec{S}/\vec{X}].\ t_2^+(g(t_1^-(y))).$$

Note that $F_{\vec{X}\vdash\tau_1\to\tau_2}$ preserves composition of $\mathbf{FPC}_!^{\ \delta}$-morphisms since the $F_{\vec{X}\vdash\tau_i}$ $(i=1,2)$ do by the induction hypothesis.

— *Recursive type:*
The morphism part of $F_{\vec{X}\vdash\mu X.\ \sigma}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash \mathrm{fix}(\lambda v.\ i^{-1}_{F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{S})} \circ s[v/f] \circ i_{F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{R})}),$$

where $i := \langle \mathrm{fold}, \mathrm{unfold}\rangle$.
By Lemma 4.7, $F_{\vec{X}\vdash\mu X.\ \sigma}$ preserves composition of $\mathbf{FPC}_!{}^{\delta}$-morphisms.

Remark 4.6. Arguments concerning actions defined by FPC type expressions are often best carried out with the help of commutative diagrams. To illustrate this, consider the action $F_{\vec{X}\vdash\mu X.\ \sigma}$ associated with $\vec{X} \vdash \mu X.\ \sigma$. Given any $\vec{\rho},\ \vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})$ and any $\vec{u} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\rho},\vec{\sigma})$, we get that $F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{u})$ is, by case (6) in the above inductive definition, the least morphism v such that the following diagram commutes:

$$\begin{array}{ccc}
F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\rho}) & \xrightarrow{\quad v \quad} & F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\sigma}) \\
\Big\downarrow{\scriptstyle i_{F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\rho})}} & & \Big\downarrow{\scriptstyle i_{F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\sigma})}} \\
F_{\vec{X},X\vdash\sigma}(\vec{\rho}, F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\rho})) & \xrightarrow[F_{\vec{X},X\vdash\sigma}(\vec{u},v)]{} & F_{\vec{X},X\vdash\sigma}(\vec{\sigma}, F_{\vec{X}\vdash\mu X.\ \sigma}(\vec{\sigma}))
\end{array}$$

Lemma 4.7. The action $F_{\vec{X}\vdash\mu X.\ \sigma}$ preserves composition of morphisms.

Proof. We use S and T to denote $F_{\vec{X}\vdash\mu X.\ \sigma}$ and $F_{\vec{X},X\vdash\sigma}$, respectively. To show that S preserves composition, we must prove that for any morphism pairs

$$\vec{u} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\rho},\vec{\sigma})$$
$$\vec{v} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\sigma},\vec{\tau}),$$

we have

$$S(\vec{v}) \circ S(\vec{u}) = S(\vec{v} \circ \vec{u}).$$

We denote $\langle \mathrm{fold}, \mathrm{unfold}\rangle$ by i and define two programs as follows:

$$\Psi_1 \ : \ (S(\vec{\tau}) \to S(\vec{\sigma})) \times (S(\vec{\sigma}) \to S(\vec{\tau})) \longrightarrow (S(\vec{\tau}) \to S(\vec{\sigma})) \times (S(\vec{\sigma}) \to S(\vec{\tau}))$$
$$\Psi_1 \ := \ \lambda(a,b).\ i^{-1}_{S(\vec{\tau})} \circ T(\vec{v},(a,b)) \circ i_{S(\vec{\sigma})}$$

$$\Psi_2 \ : \ (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau})) \longrightarrow (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau}))$$
$$\Psi_2 \ := \ \lambda(c,d).\ i^{-1}_{S(\vec{\tau})} \circ T(\vec{v} \circ \vec{u},(c,d)) \circ i_{S(\vec{\rho})}.$$

The diagram

$$\begin{array}{ccc} (S(\vec{\tau}) \to S(\vec{\sigma})) \times (S(\vec{\sigma}) \to S(\vec{\tau})) & \xrightarrow{- \circ S(\vec{u})} & (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau})) \\ \Big\downarrow \Psi_1 & & \Big\downarrow \Psi_2 \\ (S(\vec{\tau}) \to S(\vec{\sigma})) \times (S(\vec{\sigma}) \to S(\vec{\tau})) & \xrightarrow[- \circ S(\vec{u})]{} & (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau})) \end{array}$$

then commutes since for all $a : S(\vec{\tau}) \to S(\vec{\sigma})$ and $b : S(\vec{\sigma}) \to S(\vec{\tau})$, we have

$$\begin{aligned} \Psi_1(a,b) \circ S(\vec{u}) &= i^{-1}_{S(\vec{\tau})} \circ T(\vec{v}, (a,b)) \circ i_{S(\vec{\sigma})} \circ i^{-1}_{S(\vec{\sigma})} \circ T(\vec{u}, S(\vec{u})) \circ i_{S(\vec{\rho})} \\ &= i^{-1}_{S(\vec{\tau})} \circ T(\vec{v} \circ \vec{u}, (a,b) \circ S(\vec{u})) \circ i_{S(\vec{\rho})} \\ &= \Psi_2((a,b) \circ S(\vec{u})). \end{aligned}$$

Moreover, because $S(\vec{u})^-$ is strict, the program

$$- \circ S(\vec{u}) := \lambda(a,b).(S(\vec{u})^- \circ a, b \circ S(\vec{u})^+)$$

is strict. Therefore, by Lemma 3.13, we have

$$\mathrm{fix}(\Psi_2) = \mathrm{fix}(\Psi_1) \circ S(\vec{u})$$

that is, $S(\vec{v}) \circ S(\vec{u}) = S(\vec{v} \circ \vec{u})$. □

5. Operational minimal invariance

The following theorem is a crucial result for proving that realisable actions arising from types-in-context are actually functors (in that they preserve identity $\mathbf{FPC}_!{}^{\delta}$-morphisms).

Theorem 5.1 (operational minimal invariance for realisable functors). We let $T\colon (\mathbf{FPC}_!{}^{\delta})^{n+1} \to \mathbf{FPC}_!{}^{\delta}$ be a realisable functor, let $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$ and write $S(\vec{\sigma})$ for $\mu X.\ T(\vec{\sigma}, X)$. Then the least $\mathbf{FPC}_!{}^{\delta}$-endomorphism

$$e : S(\vec{\sigma}) \to S(\vec{\sigma})$$

for which the diagram

$$\begin{array}{ccc} S(\vec{\sigma}) & \xrightarrow{e} & S(\vec{\sigma}) \\ \Big\downarrow i_{S(\vec{\sigma})} & & \Big\downarrow i_{S(\vec{\sigma})} \\ T(\vec{\sigma}, S(\vec{\sigma})) & \xrightarrow[T(\mathrm{id}_{\vec{\sigma}}, e)]{} & T(\vec{\sigma}, S(\vec{\sigma})) \end{array}$$

commutes is the identity morphism $\langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle$, where $i := \langle \mathrm{fold}, \mathrm{unfold} \rangle$. Moreover, the identity is the only such endomorphism. Consequently, S preserves identity morphisms, that is,

$$S(\langle \mathrm{id}_{\vec{\sigma}}, \mathrm{id}_{\vec{\sigma}} \rangle) = \langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle.$$

In this section, we supply a purely operational proof of this main theorem. Admittedly, this proof is very technical, and we will need to develop some new machinery. The operational techniques we develop here are different from those employed in existing work such as Birkedal and Harper (1999) and Lassen (1998b) – in particular, the concept of the associates of a closed type is specifically developed to cope with nested recursion for types, which is not present in these two papers.

5.1. *Twin morphisms*

First we need the following definition.

Definition 5.2. An $\mathbf{FPC}_!^{\delta}$-morphism is said to be *twin* if it is of the form

$$u : \sigma \leftrightarrows \sigma : u,$$

that is, $u^- = u^+ = u$.

The next lemma guarantees that twins are preserved by realisable actions.

Lemma 5.3. Let $\vec{X} \vdash \tau$ be a type-in-context and $F_{\vec{X}\vdash\tau}$ be the realisable action associated with it. Then, for any $\vec{\sigma} \in (\mathbf{FPC}_!^{\delta})^n$ and any sequence of twin morphisms

$$\vec{u} \in (\mathbf{FPC}_!^{\delta})^n(\vec{\sigma}, \vec{\sigma})$$

(that is, $u_i : \sigma_i \leftrightarrows \sigma_i : u_i$ $(i = 1, \ldots, n)$), the morphism $F_{\vec{X}\vdash\tau}(\vec{u})$ is again twin.

Proof. The proof is by induction on the structure of $\vec{X} \vdash \sigma$.

We will just prove the only interesting case, which is the recursive type. Let $\vec{X}, X \vdash \tau$ be given and $T = F_{\vec{X},X\vdash\tau}$ be the realisable action defined by it. The induction hypothesis states that for any twin morphism $\vec{v} \in (\mathbf{FPC}_!^{\delta})^{n+1}(\vec{\rho}, \vec{\rho})$, the morphism $T(\vec{v})$ is again twin.

We want to prove for every twin morphism $\vec{u} \in (\mathbf{FPC}_!^{\delta})^n(\vec{\sigma}, \vec{\sigma})$, that $F_{\vec{X}\vdash\mu X.\,\tau}(\vec{u})$ is again twin. As usual, we abbreviate $F_{\vec{X}\vdash\mu X.\,\tau}$ by S. By definition, $F_{\vec{X}\vdash\mu X.\,\tau}(\vec{u})$ is the least $t : S(\vec{\sigma}) \to S(\vec{\sigma})$ such that the diagram

$$\begin{array}{ccc} S(\vec{\sigma}) & \xrightarrow{\;t\;} & S(\vec{\sigma}) \\ {\scriptstyle i_{S(\vec{\sigma})}}\downarrow & & \downarrow{\scriptstyle i_{S(\vec{\sigma})}} \\ T(\vec{\sigma}, S(\vec{\sigma})) & \xrightarrow[T(\vec{u},t)]{} & T(\vec{\sigma}, S(\vec{\sigma})) \end{array}$$

commutes. Here, we use i to denote $\langle \mathrm{fold}, \mathrm{unfold} \rangle$. Let $\phi := \lambda t.i^{-1} \circ T(\vec{u}, t) \circ i$. Then, on the one hand, by the definition of $S(\vec{u})$, we have $t = \mathrm{fix}(\phi)$. On the other hand,

$$\mathrm{fix}(\phi) = \bigsqcup_n \phi^{(n)}(\bot, \bot)$$

by rational completeness. A further induction on n then shows that $\phi^{(n)}(\bot, \bot)$ is twin for every $n \in \mathbb{N}$. The proof is easy. For $n = 0$, we have the trivial twin $(\bot, \bot)$. We now assume

that $\phi^{(n)}(\bot,\bot)$ is twin for $n \in \mathbb{N}$. We wish to prove that $\phi^{(n+1)}(\bot,\bot)$ is twin. Since $\vec{u}$ and $\phi^{(n)}(\bot,\bot)$ are twin, the pairing

$$\vec{v} := (\vec{u}, \phi^{(n)}(\bot,\bot))$$

is twin. Invoking the earlier induction hypothesis that T preserves twin, it is clear that $T(\vec{u}, \phi^{(n)}(\bot,\bot))$ is twin. We denote this morphism by $\langle f, f\rangle$. Thus, the morphism

$$\phi^{(n+1)}(\bot,\bot) = i^{-1} \circ \langle f, f\rangle \circ i,$$

which can be illustrated by

$$S(\vec{\sigma}) \underset{\text{fold}}{\overset{\text{unfold}}{\rightleftarrows}} T(\vec{\sigma}, S(\vec{\sigma})) \underset{f}{\overset{f}{\rightleftarrows}} T(\vec{\sigma}, S(\vec{\sigma})) \underset{\text{unfold}}{\overset{\text{fold}}{\rightleftarrows}} S(\vec{\sigma}),$$

must be twin. Consequently, $\text{fix}(\phi)$ is twin and the proof is complete. □

Corollary 5.4. Let F be a realisable action and σ be a closed type. Then, $F(\langle \text{id}_\sigma, \text{id}_\sigma\rangle)$ is twin.

We have not yet shown that realisable actions preserve identity morphisms, but the following holds.

Lemma 5.5. Let $F_{\Theta\vdash\tau}$ be the realisable action associated with $\Theta \vdash \tau$. Then, for any sequence of n closed types $\vec{\sigma}$,

$$F_{\Theta\vdash\tau}(\text{id}_{\vec{\sigma}} : \vec{\sigma} \leftrightarrows \vec{\sigma} : \text{id}_{\vec{\sigma}}) \sqsubseteq (\text{id}_{F_{\Theta\vdash\tau}(\vec{\sigma})}, \text{id}_{F_{\Theta\vdash\tau}(\vec{\sigma})}).$$

Proof. The proof is by a straightforward induction on the structure of $\Theta \vdash \sigma$. □

5.2. *Fischer–Ladner closure*

The next concept we need is that of the *associates* of an FPC closed type. To define this concept, we appeal to an analogue of the Fischer–Ladner closure of PDL (Fischer and Ladner 1979), which is applicable to FPC closed types; though our style is closer to that found in Kozen (1983, page 333).

A type expression τ is called a *μ-expression* (or *recursive type*) if it is of the form $\mu X.\ \rho$. A type expression σ is *μ-free* if it does not contain any μ-subexpressions. If X is a bound type variable of an FPC closed type σ, there is a unique μ-subexpression $\mu X.\ \rho$ of σ in which X is bound. We denote this subexpression by μX. The type variable X is called a *μ-variable* if $\mu X \equiv \mu X.\ \rho$.

Definition 5.6. Let σ be an FPC closed type. We define $\sigma \preccurlyeq \tau$ if τ appears as a subexpression of σ, and $\sigma \prec \tau$ if τ appears as a proper subexpression of σ. For $\sigma \preccurlyeq \tau$, we define $V_\tau = X_1, \ldots, X_n$ to be the sequence of all type variables X_i's such that $\mu X_i \prec \tau$ (for $i = 1, \ldots, n$), taken in the order

$$\mu X_1 \prec \cdots \prec \mu X_n \prec \tau.$$

A μ-subexpression τ of σ is said to be *maximal* if no such sequence exists.

Definition 5.7. If $V_\tau = X_1, \ldots, X_n$, let μV_τ denote the sequence $\mu X_1, \ldots, \mu X_n$. We define the map e on subexpressions of σ by

$$e(\tau) := \tau[\mu V_\tau / V_\tau],$$

where $\sigma \preccurlyeq \tau$.

The *Fischer–Ladner closure* (or just *FL-closure* for short) of σ is the range of e, that is,

$$\mathrm{CL}(\sigma) := \{e(\tau) \mid \sigma \preccurlyeq \tau\}.$$

Those elements of $\mathrm{CL}(\sigma)$ that are μ-expressions (that is, types of the form $\mu X.\rho$) are called the *associates* of σ.

Clearly, $e(\tau)$ is a closed FPC type since if X occurs free in τ, then $\mu X \preccurlyeq \tau$. The following proposition is obvious from Definition 5.7.

Proposition 5.8. $\mathrm{CL}(\sigma)$ is a finite set.

The FL-closure of an FPC closed type can be characterised by the following proposition.

Proposition 5.9. Let σ be an FPC closed type. Then $\mathrm{CL}(\sigma)$ is the smallest set containing σ and satisfying:

(1) If $\mu X.\ \tau \in \mathrm{CL}(\sigma)$, then $\tau[\mu X.\ \tau/X] \in \mathrm{CL}(\sigma)$.
(2) If any of $\tau_1 \to \tau_2$, $\tau_1 \times \tau_2$, $\tau_1 + \tau_2$ is in $\mathrm{CL}(\sigma)$, then so are τ_1 and τ_2.
(3) If $\tau_\perp \in \mathrm{CL}(\sigma)$, then $\tau \in \mathrm{CL}(\sigma)$.

Example 5.10. Given the closed type

$$\sigma = (\mu X_1.\ \mu X_2.\ (X_1 \times X_2)) \to (\mu X_3.\ (X_3 + \mu X_4.\ (X_4 + X_3))),$$

the elements of $\mathrm{CL}(\sigma)$ are:

$$\begin{aligned}
\rho_0 &:= \sigma \\
\rho_1 &:= \mu X_1.\ \mu X_2.\ (X_1 \times X_2) \\
\rho_3 &:= \mu X_3.\ (X_3 + \mu X_4.\ (X_4 + X_3)) \\
\rho_2 &:= \mu X_2.\ (\rho_1 \times X_2) \\
\rho_4 &:= \mu X_4.\ (X_4 + \rho_3) \\
\rho_1 \times \rho_2 &= (\rho_1 \times X_2)[\rho_2/X_2] \\
\rho_3 + \rho_4 &= (X_3 + \mu X_4.\ (X_4 + X_3))[\rho_3/X_3] \\
\rho_4 + \rho_3 &= (X_4 + \rho_3)[\rho_4/X_4].
\end{aligned}$$

The associates of σ are those μ-expressions in the preceding list, that is, ρ_i $(i = 1, 2, 3, 4)$.

The following theorem is a direct consequence of Proposition 5.8.

Theorem 5.11 (mutual definitions theorem). Let σ be an FPC closed type and $\rho_1, \ldots, \rho_n$ be its associates. Then for every associate

$$\rho_i := \mu X_i.\sigma_i \qquad (i = 1, \ldots, n),$$

there exists a μ-free type-in-context $Y_1, \ldots, Y_n \vdash \beta_i$ such that

$$\sigma_i[\rho_i/X_i] \equiv \beta_i[\vec{\rho}/\vec{Y}].$$

5.3. *Canonical pre-deflations and deflations*

In this section, we define two type-indexed families of endofunctions that will be useful in the operational proof of the minimal invariance property.

Definition 5.12. A *pre-deflation* on a type σ is an element of type $(\sigma \to \sigma)$ that is:

(i) idempotent; and
(ii) below the identity.

A *deflation* on a type σ is a pre-deflation with the additional property of having a finite image modulo contextual equivalence.

A *rational pre-deflationary structure* (respectively, *rational deflationary structure*) on a closed FPC type σ is a rational chain id_n^σ of idempotent pre-deflations (respectively, deflations) with $\bigsqcup_n \mathrm{id}_n^\sigma = \mathrm{id}_\sigma$.

Note that every type has a trivial pre-deflationary structure given by the constantly identity chain.

In the following, we define for each type, in parallel, a non-trivial pre-deflationary structure and a deflationary structure.

Recall that we define the vertical natural numbers $\overline{\omega}$ (*cf.* Section 2.4) by

$$\overline{\omega} = \mu X.\ X_\perp.$$

Using $\overline{\omega}$, we first define the programs $\mathrm{e} : \overline{\omega} \to (\sigma \to \sigma)$ by induction on σ as follows:

$$\begin{aligned}
\mathrm{e}^{\sigma\times\tau}(n)(p) &= (\mathrm{e}^\sigma(n)(\mathrm{fst}(p)), \mathrm{e}^\tau(n)(\mathrm{snd}(p))) \\
\mathrm{e}^{\sigma+\tau}(n)(z) &= \mathrm{case}(z)\ \text{of}\ \mathrm{inl}(x).\ \mathrm{e}^\sigma(n)(x)\ \text{or}\ \mathrm{inr}(y).\ \mathrm{e}^\tau(n)(y) \\
\mathrm{e}^{\sigma_\perp}(n)(z) &= \mathrm{case}(z)\ \text{of}\ \mathrm{up}(x).\ \mathrm{up}(\mathrm{e}^\sigma(n)(x)) \\
\mathrm{e}^{\sigma\to\tau}(n)(f) &= \mathrm{e}^\tau(n) \circ f \circ \mathrm{e}^\sigma(n).
\end{aligned}$$

For the recursive type $\mu X.\ \sigma$, the program $\mathrm{e}^{\mu X.\ \sigma}(n)$ is defined as follows.

Recall that $F_{X\vdash\sigma} : \mathbf{FPC}_!{}^\delta \to \mathbf{FPC}_!{}^\delta$ is the realisable action associated with the type-in-context $X \vdash \sigma$. By Lemma 5.3, we may abuse notation by writing

$$(F_{X\vdash\sigma}(\mathrm{e}^{\mu X.\ \sigma}(n), \mathrm{e}^{\mu X.\ \sigma}(n)))^+$$

as

$$F_{X\vdash\sigma}(\mathrm{e}^{\mu X.\ \sigma}(n)).$$

We define

$$\mathrm{e}^{\mu X.\ \sigma}(n)(x) := \mathrm{if}\ \ (n > 0)\ \ \mathrm{then}\ \ \mathrm{fold} \circ F_{X\vdash\sigma}(\mathrm{e}^{\mu X.\ \sigma}(n-1)) \circ \mathrm{unfold}(x).$$

Note that the definition of $\mathrm{e}^{\mu X.\ \sigma}$ is a simple recursive definition given explicitly by using fix. Clearly, $\mathrm{e}^{\mu X.\ \sigma}$ satisfies the equations

$$\mathrm{e}^{\mu X.\ \sigma}(0) = \bot_{\mu X.\ \sigma \to \mu X.\ \sigma} \tag{5.1}$$

$$\mathrm{e}^{\mu X.\ \sigma}(n+1) = \mathrm{fold} \circ F_{X \vdash \sigma}(\mathrm{e}^{\mu X.\ \sigma}(n)) \circ \mathrm{unfold}. \tag{5.2}$$

Relying on $\overline{\omega}$ again, we next define the programs $\mathrm{d}^{\sigma} : \overline{\omega} \to (\sigma \to \sigma)$ by induction on σ as follows:

$$\begin{aligned}
\mathrm{d}^{\sigma \times \tau}(n)(p) &= (\mathrm{d}^{\sigma}(n)(\mathrm{fst}(p)), \mathrm{d}^{\tau}(n)(\mathrm{snd}(p))) \\
\mathrm{d}^{\sigma + \tau}(n)(z) &= \mathrm{case}(z)\ \text{of}\ \mathrm{inl}(x).\ \mathrm{d}^{\sigma}(n)(x)\ \text{or}\ \mathrm{inr}(y).\ \mathrm{d}^{\tau}(n)(y) \\
\mathrm{d}^{\sigma_{\bot}}(n)(z) &= \mathrm{case}(z)\ \text{of}\ \mathrm{up}(x).\ \mathrm{up}(\mathrm{d}^{\sigma}(n)(x)) \\
\mathrm{d}^{\sigma \to \tau}(n)(f) &= \mathrm{d}^{\tau}(n) \circ f \circ \mathrm{d}^{\sigma}(n),
\end{aligned}$$

and for the recursive type $\mu X.\ \sigma$, the program $\mathrm{d}^{\mu X.\ \sigma}(n)$ is defined by

$$\mathrm{d}^{\mu X.\ \sigma}(n) := \text{if}\ n > 0\ \text{then}\ \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(n-1) \circ \mathrm{unfold}.$$

Note that $\mathrm{d}^{\mu X.\ \sigma}$ satisfies the equations

$$\mathrm{d}^{\mu X.\ \sigma}(0) = \bot_{\mu X.\ \sigma \to \mu X.\ \sigma} \tag{5.3}$$

$$\mathrm{d}^{\mu X.\ \sigma}(n+1) = \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(n) \circ \mathrm{unfold}. \tag{5.4}$$

At first glance it looks as if the definition of $\mathrm{d}^{\mu X.\ \sigma}$ is not recursive, that is, it is not of the form $\mathrm{fix}(t)$ for some term t, because the type-index $\sigma[\mu X.\ \sigma/X]$ appears in the right-hand term of Equation (5.4). However, by Theorem 5.11, since a single unfolding of each associate can be obtained using an application of (a realisable action arising from) a μ-free type-in-context to all the associates, $\mathrm{d}^{\mu X.\ \sigma}$ can then be defined using a finite system of mutual recursions. We make this precise in the following lemma.

Lemma 5.13. Let $\mu X_1.\ \sigma_1$ be a given closed type whose associates are denoted by

$$\rho_i = \mu X_i.\ \sigma_i (i = 1, \ldots, n).$$

We can then define $\mathrm{d}^{\mu X_1.\ \sigma_1}(n)$ by a finite system of mutual recursions involving some $\mathrm{d}^{\rho_i}(j)$'s, where $0 \leqslant j \leqslant n$.

Proof. Let $\rho_1, \ldots, \rho_n$ be the associates of $\varnothing \vdash \tau_1$, setting $\rho_1 := \tau_1$. By Theorem 5.11, for each $i = 1, \ldots, n$,

$$\sigma_i[\rho_i/X_i] = \beta_i[\vec{\rho}/\vec{Y}]$$

for some μ-free type-in-context $\vec{Y} = Y_1, \ldots, Y_n \vdash \beta_i$. It then follows that

$$\mathrm{d}^{\rho_i}(k) = \text{if}\ k > 0\ \text{then}\ \mathrm{fold} \circ (F_{\vec{Y} \vdash \beta_i}(\mathrm{d}^{\rho_1}(k), \ldots, \mathrm{d}^{\rho_n}(k)))^{+} \circ \mathrm{unfold}. \qquad \square$$

We can now revisit Example 5.10 in the light of the preceding lemma.

Example 5.14. Consider again

$$\varnothing \vdash \tau_0 := (\mu X_1.\ \mu X_2.\ (X_1 \times X_2)) \to (\mu X_3.\ X_3 + \mu X_4.\ (X_4 + X_3))$$

as before. Then the deflations of the associates derived from τ_0 can be mutually defined using simple recursion:

$$\begin{aligned}
\mathrm{d}^{\rho_1}(n) &= \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\rho_2}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\rho_2}(n) &= \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\rho_1 \times \rho_2}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\rho_3}(n) &= \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\rho_3 + \rho_4}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\rho_4}(n) &= \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\rho_4 + \rho_3}(n-1) \circ \mathrm{unfold}.
\end{aligned}$$

Proposition 5.15. For any closed recursive type $\mu X.\ \sigma$,

$$\mathrm{d}^{\mu X.\ \sigma}(\infty) = \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(\infty) \circ \mathrm{unfold}.$$

Proof. The result is obvious since $\infty =_{\overline{\omega}} \infty + 1$. □

We now establish the order-theoretic relation between the families of functions d^σ and e^σ as follows.

Theorem 5.16. For every finite $n \in \overline{\omega}$ and every closed FPC types σ, we have

$$\mathrm{d}^\sigma(n) \sqsubseteq \mathrm{e}^\sigma(n) \sqsubseteq \mathrm{id}_\sigma,$$

where $\sqsubseteq$ is the contextual preorder on $(\sigma \to \sigma)$.

Proof. For the base case $n = 0$, we have

$$\mathrm{d}^\sigma(0) = \mathrm{e}^\sigma(0) = \bot_{\sigma \to \sigma} \sqsubseteq \mathrm{id}_\sigma.$$

We proceed by induction on the structure of closed types as follows. For brevity, we will just verify the case for sum types (the other non-recursive types are proved similarly):

— *Sum type* $\sigma + \tau$:
By definition,

$$\begin{aligned}
\mathrm{d}^{\sigma+\tau}(0)(z) &= \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{d}^\sigma(0)(x)\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{d}^\tau(0)(y) && \text{(definition of } \mathrm{d}^{\sigma+\tau}) \\
&= \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \bot_{\sigma\to\sigma}(x)\ \mathrm{or}\ \mathrm{inr}(y).\ \bot_{\tau\to\tau}(y) && \text{(induction hypothesis)} \\
&= \bot_{(\sigma+\tau)\to(\sigma+\tau)}.
\end{aligned}$$

$\mathrm{e}^{\sigma+\tau}(0)(z)$ is treated similarly.

We now focus on the recursive types:

— *Recursive type* $\mu X.\ \sigma$:
From Equations (5.1–5.2), it follows immediately that

$$\mathrm{d}^{\mu X.\ \sigma}(0) = \bot_{\mu X.\ \sigma \to \mu X.\ \sigma} = \mathrm{e}^{\mu X.\ \sigma}(0) \sqsubseteq \mathrm{id}_{\mu X.\ \sigma}.$$

So the base case for $n = 0$ is established.

For the induction step, we assume that there is a $k \in \mathbb{N}$ such that for all $n \leqslant k$,

$$\text{for all closed types } \sigma,\ \mathrm{d}^\sigma(n) \sqsubseteq \mathrm{e}^\sigma(n) \sqsubseteq \mathrm{id}_\sigma$$

always holds, and need to prove that

$$\text{for all closed types } \sigma,\ \mathrm{d}^{\sigma}(k+1) \sqsubseteq \mathrm{e}^{\sigma}(k+1) \sqsubseteq \mathrm{id}_{\sigma}.$$

We again proceed by induction on the structure of types.

For the non-recursive types, we will again just show the proof for one type, the product type, as an example (the other non-recursive types are just as easy and thus omitted).

— *Product type* $\sigma = \sigma_1 \times \sigma_2$:
Note that

$$\mathrm{d}^{\sigma}(k+1) = (\mathrm{d}^{\sigma_1}(k+1), \mathrm{d}^{\sigma_2}(k+1)).$$

By the induction hypotheses, since

$$\mathrm{d}^{\sigma_1}(k+1) \sqsubseteq \mathrm{e}^{\sigma_1}(k+1) \sqsubseteq \mathrm{id}_{\sigma_1}$$
$$\mathrm{d}^{\sigma_2}(k+1) \sqsubseteq \mathrm{e}^{\sigma_2}(k+1) \sqsubseteq \mathrm{id}_{\sigma_2},$$

it follows that

$$\begin{aligned}
\mathrm{d}^{\sigma}(k+1) &= (\mathrm{d}^{\sigma_1}(k+1), \mathrm{d}^{\sigma_2}(k+1)) \\
&\sqsubseteq (\mathrm{e}^{\sigma_1}(k+1), \mathrm{e}^{\sigma_2}(k+1)) \\
&\sqsubseteq \mathrm{e}^{\sigma}(k+1) \\
&= (\mathrm{e}^{\sigma_1}(k+1), \mathrm{e}^{\sigma_2}(k+1)) \\
&\sqsubseteq (\mathrm{id}_{\sigma_1}, \mathrm{id}_{\sigma_2}) \\
&\sqsubseteq \mathrm{id}_{\sigma}.
\end{aligned}$$

We now turn to the recursive types. To proceed with the proof by induction on the structure of τ, we suppose that $\sigma = \mu X.\ \tau$ for some type-in-context $X \vdash \tau$.

— Case I(1): *Type variable* $\tau = X$:
The proof is straightforward as follows:

$$\begin{aligned}
\mathrm{d}^{\mu X.\ X}(k+1) &= \mathrm{fold} \circ \mathrm{d}^{\mu X.\ X}(k) \circ \mathrm{unfold} && \text{(definition of } \mathrm{d}^{\mu X.\ X}\text{)} \\
&\sqsubseteq \mathrm{fold} \circ \mathrm{e}^{\mu X.\ X}(k) \circ \mathrm{unfold} && \text{(induction hypothesis)} \\
&= \mathrm{e}^{\mu X.\ X}(k+1). && \text{(definition of } \mathrm{e}^{\mu X.\ X}\text{)}
\end{aligned}$$

— Case I(2): *Product type* $\tau = \tau_1 \times \tau_2$:
The proof proceeds as follows:

$$\begin{aligned}
\mathrm{d}^{\mu X.\ \tau}(k+1) &= \mathrm{d}^{\mu X.\ \tau_1 \times \tau_2}(k+1) \\
&= \mathrm{fold} \circ \mathrm{d}^{(\tau_1 \times \tau_2)[\tau/X]}(k) \circ \mathrm{unfold} && \text{(definition of } \mathrm{d}^{\mu X.\ \tau_1 \times \tau_2}\text{)} \\
&= \mathrm{fold} \circ (\mathrm{d}^{\tau_1[\tau/X]}(k), \mathrm{d}^{\tau_2[\tau/X]}(k)) \circ \mathrm{unfold} && \text{(definition of } \mathrm{d}^{(\tau_1 \times \tau_2)[\tau/X]}\text{)} \\
&\sqsubseteq \mathrm{fold} \circ (\mathrm{e}^{\tau_1[\tau/X]}(k), \mathrm{e}^{\tau_2[\tau/X]}(k)) \circ \mathrm{unfold} && \text{(induction hypothesis)} \\
&= \mathrm{fold} \circ \mathrm{e}^{(\tau_1 \times \tau_2)[\tau/X]}(k) \circ \mathrm{unfold} && \text{(definition of } \mathrm{e}^{(\tau_1 \times \tau_2)[\tau/X]}\text{)} \\
&= \mathrm{e}^{\mu X.\ \tau}(k+1)
\end{aligned}$$

It is easy to establish using similar reasoning that the statement holds for the other non-recursive cases.

— Case II: *Recursive type* $\tau = \mu Y.\rho$:
Without loss of generality, we can assume that ρ is not recursive (otherwise, we can apply the following reasoning to the type $\tau = \mu Y_1.\mu Y_2.\ldots.\mu Y_m.\rho'$ where ρ' is not recursive).
To show that

$$\mathrm{d}^{\mu X.\ \mu Y.\rho}(k+1) \sqsubseteq \mathrm{e}^{\mu X.\ \mu Y.\rho}(k+1),$$

it will be sufficient to establish

$$\mathrm{d}^{\mu Y.\rho[\tau_0/X]}(k) \sqsubseteq F_{X\vdash\mu Y.\rho}(\mathrm{e}^{\tau_0}(k))$$

where $\tau_0 := \mu X.\ \mu Y.\rho$.
To this end, we will first prove that

$$\mathrm{d}^{\mu Y.\rho[\tau_0/X]}(k) \sqsubseteq F_{X\vdash\mu Y.\rho}(\mathrm{d}^{\tau_0}(k)).$$

We now consider the cases where $X, Y \vdash \rho$ is either μ-free or not μ-free:

– $X, Y \vdash \rho$ is μ-free:
We define

$$\tau_1 := \mu Y.\rho[\tau_0/X]$$
$$\tau_2 := \rho[\tau_0/X, \tau_1/X].$$

Since $X, Y \vdash \rho$ is μ-free, it follows that

$$\mathrm{d}^{\rho[\tau_0/X,\tau_1/Y]}(k-1) = F_{X,Y\vdash\rho}(\mathrm{d}^{\tau_0}(k-1), \mathrm{d}^{\tau_1}(k-1)).$$

Because $\tau_1 = \mu Y.\rho[\tau_0/X]$, by unfolding $k-1$ times, we have the following nestings:

$$R(\mathrm{d}^{\tau_0}(k-1), \mathrm{fold} \circ R(\mathrm{d}^{\tau_0}(k-2), \ldots, \mathrm{fold} \circ R(\mathrm{d}^{\tau_0}(0), \mathrm{d}^{\tau_1}(0)) \circ \mathrm{unfold}) \circ \mathrm{unfold}) \ldots \circ \mathrm{unfold})$$

where $R := F_{X,Y\vdash\rho}$. By the monotonicity of realisable actions and extensionality properties, it follows from

$$\mathrm{d}^{\tau_0}(j) \sqsubseteq \mathrm{d}^{\tau_0}(k) \quad (j = 0, 1, \ldots, k-2)$$

that the above term is below

$$s := R(\mathrm{d}^{\tau_0}(k), \mathrm{fold} \circ R(\mathrm{d}^{\tau_0}(k), \ldots, \mathrm{fold} \circ R(\mathrm{d}^{\tau_0}(0), \mathrm{d}^{\tau_1}(0)) \circ \mathrm{unfold}) \circ \mathrm{unfold}) \ldots \circ \mathrm{unfold})$$

with respect to the contextual pre-order. Thus,

$$\mathrm{fold} \circ \mathrm{d}^{\rho[\tau_0/X,\tau_1/Y]}(k-1) \circ \mathrm{unfold} \sqsubseteq \mathrm{fold} \circ s \circ \mathrm{unfold} \sqsubseteq \bigsqcup_{j\geqslant 0} \Phi^j(\mathrm{d}^{\tau_1}(0))$$

where $\Phi := \lambda u.\mathrm{fold} \circ R(\mathrm{d}^{\tau_0}(k), u) \circ \mathrm{unfold}$. Consequently, we have

$$\mathrm{d}^{\mu Y.\rho[\tau_0/X]}(k) \sqsubseteq F_{X\vdash\mu Y.\rho}(\mathrm{d}^{\tau_0}(k)).$$

– $X, Y \vdash \rho$ is not μ-free:
Since we have assumed that ρ is not recursive, by Lemma 5.13, there exist a unique μ-free type-in-context $\vec{X}, X, Y \vdash \delta$ and maximal recursive types $\alpha_i := \mu X_i.\ \beta_i$ such that

$$\mathrm{d}^{\rho[\tau_0/X,\tau_1/Y]}(k-1) = F_{\vec{X},X,Y\vdash\delta}(\mathrm{d}^{\alpha_1}(k-1),\ldots,\mathrm{d}^{\alpha_n}(k-1),\mathrm{d}^{\tau_0}(k-1),\mathrm{d}^{\tau_1}(k-1)).$$

By the induction hypothesis, we assume that $\mathrm{d}^{\alpha_j}(k-1) \sqsubseteq \mathrm{id}_{\alpha_j}$ for all $j = 1,\ldots,n$. Since

$$F_{\vec{X},X,Y\vdash\delta}(\mathrm{id}_{\vec{\alpha}},\mathrm{d}^{\tau_0}(k-1),\mathrm{d}^{\tau_1}(k-1)) \sqsubseteq F_{X,Y\vdash\rho}(\mathrm{d}^{\tau_0}(k-1),\mathrm{d}^{\tau_1}(k-1))$$

and $F_{\vec{X},X,Y\vdash\delta}$ is monotone, we have

$$\mathrm{d}^{\rho[\tau_0/X,\tau_1/Y]}(k-1) \sqsubseteq F_{X,Y\vdash\rho}(\mathrm{d}^{\tau_0}(k-1),\mathrm{d}^{\tau_1}(k-1)).$$

Using the same argument as in the previous case, we deduce by transitivity of $\sqsubseteq$ that

$$\mathrm{d}^{\mu Y.\ \rho[\tau_0/X]}(k) \sqsubseteq F_{X\vdash\mu Y.\ \rho}(\mathrm{d}^{\tau_0}(k)),$$

noting that this argument does not depend on whether the action $R_{X,Y\vdash\rho}$ is μ-free (in fact, it is not in this case).

To complete the proof, we now apply the monotonicity of the realisable action $F_{X\vdash\mu Y.\rho}$ and the induction hypothesis $\mathrm{d}^{\tau_0}(k) \sqsubseteq \mathrm{e}^{\tau_0}(k) \sqsubseteq \mathrm{id}_{\tau_0}$ to obtain the desired result:

$$\begin{aligned}
\mathrm{d}^{\mu X.\ \mu Y.\rho}(k) &\sqsubseteq \mathrm{fold} \circ F_{X\vdash\mu Y.\ \rho}(\mathrm{d}^{\tau_0}(k)) \circ \mathrm{unfold} \\
&\sqsubseteq \mathrm{fold} \circ F_{X\vdash\mu Y.\ \rho}(\mathrm{e}^{\tau_0}(k)) \circ \mathrm{unfold} \\
&\sqsubseteq \mathrm{fold} \circ F_{X\vdash\mu Y.\ \rho}(\mathrm{id}_{\tau_0}) \circ \mathrm{unfold} \\
&\sqsubseteq \mathrm{fold} \circ \mathrm{id}_{\mu Y.\ \rho[\tau_0/X]} \circ \mathrm{unfold} \\
&\sqsubseteq \mathrm{id}_{\mu X.\ \mu Y.\rho},
\end{aligned}$$

where the fourth inequality holds by virtue of Lemma 5.5. □

Corollary 5.17. For any closed type σ, we have

$$\mathrm{d}^{\sigma}(\infty) \sqsubseteq \mathrm{e}^{\sigma}(\infty) \sqsubseteq \mathrm{id}_{\sigma}.$$

Proof. The statement follows from rational-chain completeness ($\bigsqcup_{n<\omega} n = \infty$), local continuity of realisable actions and Theorem 5.16. □

5.4. *Compilation and canonical deflationary structure*

In this section we will prove that the family d^{σ} (defined in Section 5.3) induces a canonical deflationary structure on each closed FPC type σ. This means that in addition to showing that for each $n \in \overline{\omega}$ and $n < \infty$,

(1) $\mathrm{d}^{\sigma}(n)$ is idempotent,
(2) $\mathrm{d}^{\sigma}(n) \sqsubseteq \mathrm{id}_{\sigma}$ and
(3) $\mathrm{d}^{\sigma}(n)$ has finite image modulo contextual equivalence,

we must also prove that

$$\mathrm{d}^{\sigma}(\infty) = \mathrm{id}_{\sigma}.$$

Note that property (2) has already been shown in Theorem 5.16, and that property (1) will be essential in the proof of minimal invariance – we will not need to use property (3).

As a consequence of $\mathrm{d}^{\sigma}(\infty) = \mathrm{id}_{\sigma}$ and Theorem 5.16, we have

$$\mathrm{d}^{\sigma}(\infty) = \mathrm{e}^{\sigma}(\infty) = \mathrm{id}_{\sigma},$$

and this is crucial in establishing the operational minimal invariance theorem, as we shall explain later.

The following proposition is easy to establish.

Lemma 5.18. For any closed FPC type σ and for each $n \in \overline{\omega}$ and $n < \infty$:

(1) $\mathrm{d}^{\sigma}(n)$ is idempotent,
(2) $\mathrm{d}^{\sigma}(n) \sqsubseteq \mathrm{id}_{\sigma}$; and
(3) $\mathrm{d}^{\sigma}(n)$ has finite image modulo contextual equivalence.

In addition, $\mathrm{d}^{\sigma}(\infty)$ satisfies (1) and (2).

Proof. Property (2) is immediate from Theorem 5.16, and properties (1) and (3) can be established by induction on the structure of closed type σ. In particular, the proof of property (1) is straightforward. We will only give a proof sketch for property (3) since this property will not be used later. The only interesting bit lies in the recursive type. In order to show that $\mathrm{d}^{\mu X.\,\sigma}(n)$ has finite image modulo contextual equivalence, we use Lemma 5.13 to define $\mathrm{d}^{\mu X.\,\sigma}(n)$ in terms of the deflations on the associates of $\mu X.\,\sigma$ (through mutual recursion) and then use the induction hypothesis saying that all these associated deflations have finite images modulo contextual equivalence. □

In order to prove that d^{σ} does indeed define a deflationary structure on each closed FPC type σ, we make essential use of the compilation of terms and contexts. This technical consideration first appeared in Birkedal and Harper (1999, Theorem 3.66) in their operational proof of the 'syntactic minimal invariance' in the form of a certain compilation relation $\Rightarrow$.

In this section, we will define and prove several elementary properties of this relation.

The *compilation relation* on $\mathrm{Exp}_{\sigma}(\Gamma)$ is defined by induction on the derivation of $\Gamma \vdash t : \sigma$ given by the axioms and rules in Figure 7. It is intended that the compilation relation applies canonical deflations $\mathrm{d}(\infty)$ to a given FPC term t (only when an introduction rule is invoked) in a uniform way, and *compiles* t to such a resulting term. Motivated by this idea, it comes as no surprise that the compilation relation $\Rightarrow$ is a function.

Proposition 5.19. If $\Gamma \vdash t : \sigma$, then $\Gamma \vdash t : \sigma \Rightarrow |t|$ for some unique $|t| \in \mathrm{Exp}_{\sigma}(\Gamma)$.

Proof. The proof is by induction on the derivation of $\Gamma \vdash t : \sigma$. □

Lemma 5.20. If $\Gamma \vdash t : \sigma \Rightarrow |t|$, then $\Gamma \vdash \mathrm{d}^{\sigma}(\infty)(|t|) =_{\sigma} |t|$.

Proof. The proof is by induction on the derivation of $\Gamma \vdash t : \sigma \Rightarrow |t|$.

$$\Gamma \vdash x : \sigma \Rightarrow \mathrm{d}^{\sigma}(\infty)(x)(\text{if } x \in \mathrm{dom}(\Gamma)) \qquad (\Rightarrow \text{ var})$$

$$\frac{\Gamma \vdash s : \sigma \Rightarrow |s| \quad \Gamma \vdash t : \tau \Rightarrow |t|}{\Gamma \vdash (s,t) : \sigma \times \tau \Rightarrow \mathrm{d}^{\sigma\times\tau}(\infty)(|s|,|t|)} \qquad (\Rightarrow \text{ pair})$$

$$\frac{\Gamma \vdash p : \sigma \times \tau \Rightarrow |p|}{\Gamma \vdash \mathrm{fst}(p) : \sigma \Rightarrow \mathrm{fst}(|p|)} \qquad (\Rightarrow \text{ fst})$$

$$\frac{\Gamma \vdash p : \sigma \times \tau \Rightarrow |p|}{\Gamma \vdash \mathrm{snd}(p) : \tau \Rightarrow \mathrm{snd}(|p|)} \qquad (\Rightarrow \text{ snd})$$

$$\frac{\Gamma \vdash s : \sigma \Rightarrow |s|}{\Gamma \vdash \mathrm{inl}(s) : \sigma + \tau \Rightarrow \mathrm{d}^{\sigma+\tau}(\infty)(\mathrm{inl}(|s|))} \qquad (\Rightarrow \text{ inl})$$

$$\frac{\Gamma \vdash s : \tau \Rightarrow |s|}{\Gamma \vdash \mathrm{inr}(s) : \sigma + \tau \Rightarrow \mathrm{d}^{\sigma+\tau}(\infty)(\mathrm{inr}(|s|))} \qquad (\Rightarrow \text{ inl})$$

$$\frac{\Gamma \vdash s : \sigma + \tau \Rightarrow |s| \quad \Gamma, x : \sigma \vdash t_1 : \rho \Rightarrow |t_1| \quad \Gamma, y : \tau \vdash t_2 : \rho \Rightarrow |t_2|}{\Gamma \vdash \mathrm{case}(s) \text{ of } \mathrm{inl}(x).\ t_1 \text{ or } \mathrm{inr}(y).\ t_2 : \rho \Rightarrow \mathrm{case}(|s|) \text{ of } \mathrm{inl}(x).\ |t_1| \text{ or } \mathrm{inr}(y).\ |t_2|} (\Rightarrow \text{ case})$$

$$\frac{\Gamma \vdash s : \sigma \to \tau \Rightarrow |s| \quad \Gamma \vdash t : \sigma \Rightarrow |t|}{\Gamma \vdash s(t) : \tau \Rightarrow |s|(|t|)} \qquad (\Rightarrow \text{ app})$$

$$\frac{\Gamma, x : \sigma \vdash t : \tau \Rightarrow |t|}{\Gamma \vdash \lambda x^{\sigma}.t : \sigma \to \tau \Rightarrow \mathrm{d}^{\sigma\to\tau}(\infty)(\lambda x.\ |t|)} \qquad (\Rightarrow \text{ abs})$$

$$\frac{\Gamma \vdash t : \sigma \Rightarrow |t|}{\Gamma \vdash \mathrm{up}(t) : \sigma_{\perp} \Rightarrow \mathrm{d}^{\sigma_{\perp}}(\infty)(|t|)} \qquad (\Rightarrow \text{ up})$$

$$\frac{\Gamma \vdash s : \sigma_{\perp} \Rightarrow |s| \quad \Gamma, x : \sigma \vdash t : \rho \Rightarrow |t|}{\Gamma \vdash \mathrm{case}(s) \text{ of } \mathrm{up}(x).\ t : \rho \Rightarrow \mathrm{case}(|s|) \text{ of } \mathrm{up}(x).\ |t|} (\Rightarrow \text{ case up})$$

$$\frac{\Gamma \vdash t : \mu X.\ \sigma \Rightarrow |t|}{\Gamma \vdash \mathrm{unfold}(t) : \sigma[\mu X.\ \sigma/X] \Rightarrow \mathrm{unfold}(|t|)} \qquad (\Rightarrow \text{ unfold})$$

$$\frac{\Gamma \vdash t : \sigma[\mu X.\ \sigma/X] \Rightarrow |t|}{\Gamma \vdash \mathrm{fold}(t) : \mu X.\ \sigma \Rightarrow \mathrm{d}^{\mu X.\ \sigma}(\infty)(\mathrm{fold}(|t|))} \qquad (\Rightarrow \text{ fold})$$

Fig. 7. Definition of $\Gamma \vdash t : \sigma \Rightarrow |t|$.

The cases for (⇒ var), (⇒ pair), (⇒ inl), (⇒ inr), (⇒ abs), (⇒ up) and (⇒ fold) rely on the idempotence of $\mathrm{d}(\infty)$ (*cf.* Lemma 5.18) without having to invoke the induction hypothesis. We will just show the case for (⇒ var) here.

— (⇒ var):
Given that $\Gamma \vdash x : \sigma \Rightarrow |x|$. By definition, $|x| = \mathrm{d}^\sigma(\infty)(x)$. We need to show that

$$\Gamma \vdash \mathrm{d}^\sigma(\infty)|x| = |x|.$$

But this follows from the idempotence of $\mathrm{d}^\sigma(\infty)$, that is,

$$\Gamma \vdash \mathrm{d}^\sigma(\infty)|x| =_\sigma \mathrm{d}^\sigma(\infty)(\mathrm{d}^\sigma(\infty)(x)) =_\sigma \mathrm{d}^\sigma(\infty)(x) = |x|.$$

The other cases mentioned above are fairly routine.

We now show the case (⇒ unfold), which requires the expansion of $\mathrm{d}^{\mu X.\ \sigma}$.

— (⇒ unfold):
Let $\Gamma \vdash \mathrm{unfold}(t) : \sigma[\mu X.\ \sigma/X] \Rightarrow |\mathrm{unfold}(t)|$ be given. We must show that

$$\Gamma \vdash \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(\infty)|\mathrm{unfold}(t)| =_{\sigma[\mu X.\ \sigma/X]} |\mathrm{unfold}(t)|.$$

The (⇒ unfold) inference rule

$$\frac{\Gamma \vdash t : \mu X.\ \sigma \Rightarrow |t|}{\Gamma \vdash \mathrm{unfold}(t) : \sigma[\mu X.\ \sigma/X] \Rightarrow \mathrm{unfold}(|t|)}$$

guarantees that $|\mathrm{unfold}(t)| \equiv \mathrm{unfold}(|t|)$. The induction hypothesis asserts that

$$\Gamma \vdash \mathrm{d}^{\mu X.\ \sigma}(|t|) =_{\mu X.\ \sigma} |t|.$$

It then follows that

$$\begin{aligned}
\Gamma \vdash \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(\infty)|\mathrm{unfold}(t)| &\equiv \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(\infty)(\mathrm{unfold}(|t|)) \\
&= \mathrm{unfold} \circ \mathrm{d}^{\mu X.\ \sigma}(\infty)(|t|) && \text{(definition of } \mathrm{d}^{\mu X.\ \sigma(\infty)}\text{)} \\
&= \mathrm{unfold}(|t|) && \text{(induction hypothesis)} \\
&= |\mathrm{unfold}(t)|.
\end{aligned}$$

□

Lemma 5.21. If $\Gamma \vdash t : \sigma \Rightarrow |t|$, then $\Gamma \vdash |t| \sqsubseteq_\sigma t$.

Proof. The proof is by induction on $\Gamma \vdash t : \sigma \Rightarrow |t|$. The proof for each case typically involves unwinding the definition of $\mathrm{d}^\sigma(\infty)$ according to the type structure of σ, followed by an application of the previous lemma to the induction hypothesis/hypotheses, and then the use of inequational logic (3.28). □

5.5. *Compilation of a context*

One final technical gadget we will use is to compile a context $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$. For a given context $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$, we define a compiled context $|C|[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ using the axioms and rules similar to those for defining $\Gamma \vdash t : \sigma \Rightarrow |t|$. The axioms and rules for defining $|C|$ are an extension of those in Figure 7 by:

— appending a parameter axiom (par)

$$\Gamma \vdash -_\sigma \Rightarrow \mathrm{d}^\sigma(\infty)(-_\sigma) \quad (\Rightarrow \text{ par});$$

— retaining the rule (var); and
— for each of the remaining rules in Figure 7, replacing every occurrence of a term by a context.

For example, the rules ($\Rightarrow$ inr) and ($\Rightarrow$ unfold) are given by

$$\frac{\Gamma \vdash S : \tau \Rightarrow |S|}{\Gamma \vdash \mathrm{inr}(S) : \sigma + \tau \Rightarrow \mathrm{d}^{\sigma+\tau}(\infty)(\mathrm{inr}(|S|))} \quad (\Rightarrow \text{ inr})$$

$$\frac{\Gamma \vdash T : \mu X.\ \sigma \Rightarrow |T|}{\Gamma \vdash \mathrm{unfold}(T) : \sigma[\mu X.\ \sigma/X] \Rightarrow \mathrm{unfold}(|T|)} \quad (\Rightarrow \text{ unfold})$$

Lemma 5.22. If $\Gamma \vdash t : \sigma \Rightarrow |t|$ and $C[-_\sigma] \in \mathrm{Ctx}(\Gamma)$, then

$$\Gamma \vdash |C[t]| =_\tau |C|[|t|].$$

Proof. The proof is by induction on the structure of $C[-_\sigma]$ and Lemma 5.20. □

Lemma 5.23. Let $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ and $t \in \mathrm{Exp}_\sigma$. Then

$$|C|[t] \sqsubseteq_\tau C[t].$$

Proof. The proof is by induction on the structure of $C[-_\sigma]$ and Lemma 5.21. □

5.6. *A crucial lemma*

Lemma 5.24.

$$(\varnothing \vdash t : \sigma \Rightarrow |t| \ \wedge t \Downarrow v) \implies \varnothing \vdash |t| =_\sigma |v|.$$

Proof. The proof is by induction on the derivation of $t \Downarrow v$:

— ($\Downarrow$ can):
This case is trivial.
— ($\Downarrow$ fst,snd):
Given that $\varnothing \vdash \mathrm{fst}(p) : \sigma \Rightarrow |\mathrm{fst}(p)|$ and $\mathrm{fst}(p) \Downarrow v$, we must show that

$$\varnothing \vdash |\mathrm{fst}(p)| = |v|.$$

The premise of the only evaluation rule ($\Downarrow$ fst) matching $\mathrm{fst}(p) \Downarrow v$ consists of

$$p \Downarrow (s, t) \qquad s \Downarrow v.$$

The induction hypothesis asserts that

$$\varnothing \vdash |p| =_{\sigma\times\tau} |(s, t)|$$
$$\varnothing \vdash |s| =_\sigma |v|.$$

Based on these, we have

$$\begin{aligned}
\varnothing \vdash |\mathrm{fst}(p)| &\equiv \mathrm{fst}(|p|) && \text{(definition of } |\mathrm{fst}(p)|) \\
&=_\sigma \mathrm{fst}(|(s,t)|) && \text{(induction hypothesis)} \\
&=_\sigma \mathrm{fst}(\mathrm{d}^\sigma(\infty)(|s|), \mathrm{d}^\tau(\infty)(|t|)) && \text{(definition of } |(s,t)|) \\
&=_\sigma \mathrm{d}^\sigma(\infty)(|s|) && (\beta\text{-rule (3.31)}) \\
&=_\sigma |s| && \text{(Lemma 5.20)} \\
&=_\sigma |v|. && \text{(induction hypothesis)}
\end{aligned}$$

The case for ($\Downarrow$ snd) is similar.

— ($\Downarrow$ app):
Given that $\varnothing \vdash s(t) \Rightarrow |s(t)|$ and $s(t) \Downarrow v$, we must show that

$$\varnothing \vdash |s(t)| =_\tau |v|.$$

The only derivation of $s(t) \Downarrow v$ is through an application of the evaluation rule ($\Downarrow$ app) whose premise is given by

$$s \Downarrow \lambda x.\ r \qquad r[t/x] \Downarrow v.$$

The induction hypothesis asserts that

$$\begin{aligned}
\varnothing \vdash |s| &=_{\sigma\to\tau} |\lambda x.\ r| \\
\varnothing \vdash |r[t/x]| &=_\sigma |v|.
\end{aligned}$$

The desired result then follows from

$$\begin{aligned}
\varnothing \vdash |s(t)| &\equiv |s|(|t|) && \text{(definition of } |s(t)|) \\
&=_\tau |\lambda x.\ r|(|t|) && \text{(induction hypothesis)} \\
&\equiv (\mathrm{d}^{\sigma\to\tau}(\infty)(\lambda x.\ |r|))(|t|) && \text{(definition of } |\lambda x.\ r|) \\
&\equiv (\lambda x.\ \mathrm{d}^\tau(\infty) \circ |r| \circ \mathrm{d}^\sigma(\infty))(|t|) && \text{(definition of } \mathrm{d}^{\sigma\to\tau}) \\
&=_\tau (\lambda x.\ \mathrm{d}^\tau(\infty) \circ |r|)(\mathrm{d}^\sigma(\infty)(|t|)) \\
&=_\tau (\lambda x.\ \mathrm{d}^\tau(\infty) \circ |r|)(|t|) && \text{(Lemma 5.20)} \\
&=_\tau \mathrm{d}^\tau(\infty)(|r|[|t|/x]) && (\beta\text{-rule (3.30)}) \\
&=_\tau \mathrm{d}^\tau(\infty)(|r[t/x]|) && \text{(Lemma 5.22)} \\
&=_\tau \mathrm{d}^\tau(\infty)(|v|) && \text{(induction hypothesis)} \\
&=_\tau |v|. && \text{(Lemma 5.20)}
\end{aligned}$$

— ($\Downarrow$ case):
Given that

$$\varnothing \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2 \Rightarrow |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2|$$

and $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2 \Downarrow v$, we want to prove that

$$\varnothing \vdash |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2| =_\rho |v|.$$

Without loss of generality, we are able assume that the following evaluation rule ($\Downarrow$ case inl) derives the given evaluation:

$$\frac{s \Downarrow \mathrm{inl}(t) \quad t_1[t/x] \Downarrow v}{\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2 \Downarrow v}.$$

The induction hypothesis asserts that

$$\varnothing \vdash |s| =_{\sigma+\tau} |\mathrm{inl}(t)|$$
$$\varnothing \vdash |t_1[t/x]| \Downarrow v.$$

It then follows that

$$\begin{aligned}
\varnothing \vdash |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).\ t_1\ \mathrm{or}\ \mathrm{inr}(y).\ t_2| &\equiv \mathrm{case}(|s|)\ \mathrm{of}\ \mathrm{inl}(x).\ |t_1|\ \mathrm{or}\ \mathrm{inr}(y).\ |t_2| \\
&\qquad \text{(by definition)} \\
&=_\rho \mathrm{case}(\mathrm{inl}(|t|))\ \mathrm{of}\ \mathrm{inl}(x).\ |t_1|\ \mathrm{or}\ \mathrm{inr}(y).\ |t_2| \\
&\qquad \text{(induction hypothesis)} \\
&=_\rho |t_1|[|t|/x] \qquad (\beta\text{-rule (3.33)}) \\
&=_\rho |t_1[t/x]| \qquad \text{(Lemma 5.22)} \\
&=_\rho |v|. \qquad \text{(induction hypothesis)}
\end{aligned}$$

— ($\Downarrow$ case up):
Given that $\varnothing \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t \Rightarrow |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t|$ and $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t \Downarrow v$, we want to prove that

$$\varnothing \vdash |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t| =_\rho |v|.$$

The premise of the evaluation rule deriving $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t \Downarrow v$ consists of

$$s \Downarrow \mathrm{up}(t') \qquad t[t'/x] \Downarrow v.$$

The induction hypothesis asserts that

$$\begin{aligned}
|s| &=_{\sigma_\perp} |\mathrm{up}(t')| \\
|t[t'/x]| &=_\rho |v|.
\end{aligned}$$

The desired result then follows from

$$\begin{aligned}
\varnothing \vdash |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t| &\equiv \mathrm{case}(|s|)\ \mathrm{of}\ \mathrm{up}(x).\ |t| \\
&\qquad \text{(definition of } |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).\ t|) \\
&=_\rho \mathrm{case}(|\mathrm{up}(t')|)\ \mathrm{of}\ \mathrm{up}(x).\ |t| \qquad \text{(induction hypothesis)} \\
&=_\rho \mathrm{case}(\mathrm{up}(|t'|))\ \mathrm{of}\ \mathrm{up}(x).\ |t| \qquad \text{(definition of } |\mathrm{up}(t')|) \\
&=_\rho |t|[|t'|/x] \qquad (\beta\text{-rule (3.32)}) \\
&=_\rho |t[t'/x]| \qquad \text{(Lemma 5.22)} \\
&=_\rho |v|. \qquad \text{(induction hypothesis)}
\end{aligned}$$

— ($\Downarrow$ unfold):
Given that $\varnothing \vdash \mathrm{unfold}(t) \Rightarrow |\mathrm{unfold}(t)|$ and $\mathrm{unfold}(t) \Downarrow v$, we must show that

$$\varnothing \vdash |\mathrm{unfold}(t)| =_{\sigma[\mu X.\ \sigma/X]} |v|.$$

The premise of the evaluation rule deriving $\mathrm{unfold}(t) \Downarrow v$ consists of

$$t \Downarrow \mathrm{fold}(s) \qquad s \Downarrow v.$$

The induction hypothesis asserts that

$$\varnothing \vdash |t| =_{\mu X.\ \sigma} |\mathrm{fold}(s)|$$
$$\varnothing \vdash |s| =_{\sigma[\mu X.\ \sigma/X]} |v|.$$

The desired result follows from

$$\begin{aligned}
\varnothing \vdash |\mathrm{unfold}(t)| &\equiv && \mathrm{unfold}(|t|) && \text{(definition of } |\mathrm{unfold}(t)|) \\
&=_{\sigma[\mu X.\ \sigma/X]} && \mathrm{unfold}(|\mathrm{fold}(s)|) && \text{(induction hypothesis)} \\
&\equiv && \mathrm{unfold}(\mathrm{d}^{\mu X.\ \sigma}(\infty)(\mathrm{fold}(|s|))) && \text{(definition of } |\mathrm{fold}(s)|) \\
&=_{\sigma[\mu X.\ \sigma/X]} && \mathrm{unfold} \circ \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\ \sigma/X]}(\infty)(|s|) && \text{(Proposition 5.15)} \\
&=_{\sigma[\mu X.\ \sigma/X]} && \mathrm{unfold}(\mathrm{fold}(|s|))5.20) && \text{(Lemma)} \\
&=_{\sigma[\mu X.\ \sigma/X]} && |s| && (\beta\text{-rule (3.35)}) \\
&=_{\sigma[\mu X.\ \sigma/X]} && |v|. && \text{(induction hypothesis)}
\end{aligned}$$

□

5.7. *Proof of functoriality*

We are now ready to present an operational proof of Theorem 5.1.

Lemma 5.25. We let $f, g \in \mathrm{Exp}_{\mu X.\ \sigma \to \mu X.\ \sigma}$ be given and suppose that for all $t \in \mathrm{Exp}_{\sigma[\mu X.\ \sigma/X]}$ and all contexts of the form

$$C[-_{\mu X.\ \sigma \to \mu X.\ \sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma,$$

we have

$$C[f(\mathrm{fold}(t))] \sqsubseteq_\Sigma C[g(\mathrm{fold}(t))].$$

Then $f \sqsubseteq_{\mu X.\ \sigma \to \mu X.\ \sigma} g$.

Proof. By the extensionality property (3.42), in order to prove that $f \sqsubseteq g$, it suffices to prove that for all $s \in \mathrm{Exp}_{\mu X.\ \sigma}$, we have $f(s) \sqsubseteq_{\mu X.\ \sigma} g(s)$. We assume $s \in \mathrm{Exp}_{\mu X.\ \sigma}$ is given and suppose $C[-_{\mu X.\ \sigma}] \in \mathrm{Ctx}_\Sigma$ is such that $C[f(s)] \Downarrow \top$. Because of the η-rule (3.47), it follows from the definition of $\sqsubseteq$ that

$$C[f(s)] \Downarrow \top \iff C[f(\mathrm{fold}(\mathrm{unfold}(s)))] \Downarrow \top.$$

Thus, by the assumption that

$$C[f(\mathrm{fold}(t))] \sqsubseteq_\Sigma C[g(\mathrm{fold}(t))]$$

for all $t \in \mathrm{Exp}_{\sigma[\mu X.\ \sigma/X]}$, it follows (by defining $t := \mathrm{unfold}(s)$) that

$$C[g(\mathrm{fold}(\mathrm{unfold}(s)))] \Downarrow \top.$$

Again invoking the η-rule (3.47) and the definition of $\sqsubseteq$, we have $C[g(s)] \Downarrow \top$, as required. □

Lemma 5.26. For any type-in-context of the form $X \vdash \sigma$, we have

$$\varnothing \vdash \mathrm{id}_{\mu X.\ \sigma} \sqsubseteq \mathrm{d}^{\mu X.\ \sigma}(\infty).$$

Proof. By Lemma 5.25, it suffices to show that for any $t \in \mathrm{Exp}_{\sigma[\mu X.\ \sigma/X]}$ and any context

$$C[-_{\mu X.\ \sigma \to \mu X.\ \sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma,$$

we have

$$C[\mathrm{id}_{\mu X.\ \sigma}(\mathrm{fold}(t))] \sqsubseteq_\Sigma C[\mathrm{d}^{\mu X.\ \sigma}(\infty)(\mathrm{fold}(t))].$$

Let

$$C[-_{\mu X.\ \sigma \to \mu X.\ \sigma}](\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$$

be arbitrary. Since

$$\mathrm{id}_{\mu X.\ \sigma}(\mathrm{fold}(t)) =_{\mu X.\ \sigma} \mathrm{fold}(t)$$

(this is an instance of Kleene equivalence), it suffices to prove that

$$C[\mathrm{fold}(t)] \sqsubseteq_\Sigma C[\mathrm{d}^{\mu X.\ \sigma}(\infty)(\mathrm{fold}(t))].$$

By Lemma 5.21, it suffices to show that

$$C[\mathrm{fold}(t)] \sqsubseteq_\Sigma C[\mathrm{d}^{\mu X.\ \sigma}(\infty)(|\mathrm{fold}(t)|)].$$

But by Lemma 5.20, it suffices to show that

$$C[\mathrm{fold}(t)] \sqsubseteq_\Sigma C[|\mathrm{fold}(t)|].$$

By Lemma 5.24, $C[\mathrm{fold}(t)] \Downarrow \top$ implies that

$$|C[\mathrm{fold}(t)]| = |\top| =_\Sigma \top.$$

It then follows that

$$\begin{aligned} C[\mathrm{fold}(t)] \Downarrow \top &\Longrightarrow |C[\mathrm{fold}(t)]| \Downarrow \top && \text{(Lemma 5.24)} \\ &\Longrightarrow |C|[|\mathrm{fold}(t)|] \Downarrow \top && \text{(Lemma 5.22)} \\ &\Longrightarrow C[|\mathrm{fold}(t)|] \Downarrow \top, && \text{(Lemma 5.23)} \end{aligned}$$

which is what we aimed to show. □

We can now give the proof of Theorem 5.1.

Proof of Theorem 5.1. As noted at the beginning of Section 5.4, we have by Theorem 5.16 that

$$\mathrm{d}^\sigma(n) \sqsubseteq \mathrm{e}^\sigma(n) \sqsubseteq \mathrm{id}_\sigma$$

for all closed types σ and all $n \in \overline{\omega}$. In particular, we must have

$$\mathrm{d}^{\sigma}(\infty) \sqsubseteq \mathrm{e}^{\sigma}(\infty) \sqsubseteq \mathrm{id}_{\sigma}.$$

Now, as a consequence of Lemmata 5.26 and 5.18, $\mathrm{d}^{\sigma}(\infty) = \mathrm{id}_{\sigma}$. Thus it follows that

$$\mathrm{e}^{\sigma}(\infty) = \mathrm{id}_{\sigma}.$$

This means

$$\begin{aligned}\mathrm{id}_{\mu X.\ \tau} &= \mathrm{e}^{\mu X.\ \tau}(\infty)\\ &= \mathrm{fold} \circ F_{X \vdash \tau}(\mathrm{e}^{\mu X.\ \tau}(\infty - 1)) \circ \mathrm{unfold}\\ &= \mathrm{fold} \circ F_{X \vdash \tau}(\mathrm{e}^{\mu X.\ \tau}(\infty)) \circ \mathrm{unfold}\\ &= \mathrm{fold} \circ F_{X \vdash \tau}(\mathrm{id}_{\mu X.\ \tau}) \circ \mathrm{unfold}\end{aligned}$$

and thus $\mathrm{id}_{\mu X.\ \tau}$ is the least endomorphism e on $\mu X.\ \tau$ satisfying the equation

$$e = \mathrm{fold} \circ F_{X \vdash \tau}(e) \circ \mathrm{unfold}.$$

Consequently, for the more general case, the least endomorphism e on

$$S(\vec{\sigma}) := \mu X.\ T(\vec{\sigma}, X)$$

for which the diagram

$$\begin{array}{ccc} S(\vec{\sigma}) & \xrightarrow{\quad e \quad} & S(\vec{\sigma}) \\ \Big\downarrow{\scriptstyle i_{S(\vec{\sigma})}} & & \Big\downarrow{\scriptstyle i_{S(\vec{\sigma})}} \\ T(\vec{\sigma}, S(\vec{\sigma})) & \xrightarrow[T(\mathrm{id}_{\vec{\sigma}}, e)]{} & T(\vec{\sigma}, S(\vec{\sigma})) \end{array}$$

commutes is the identity morphism $\langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle$.

Now suppose an $\mathbf{FPC}_!$-morphism $h : S(\vec{\sigma}) \to S(\vec{\sigma})$ defines an endomorphism such that

$$h \circ i_{S(\vec{\sigma})} = i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{S(\vec{\sigma})}, h).$$

We apply Plotkin's uniformity principle, that is, Lemma 3.13, to the diagram (where $\phi := \lambda g.\ i_{S(\vec{\sigma})} \circ T(\vec{\sigma}, g) \circ i^{-1}_{S(\vec{\sigma})}$) and get

$$\begin{array}{ccc} (S(\vec{\sigma}) \to S(\vec{\sigma})) & \xrightarrow{H} & (S(\vec{\sigma}) \to S(\vec{\sigma})) \\ \Big\downarrow{\scriptstyle \phi} & & \Big\downarrow{\scriptstyle \phi} \\ (S(\vec{\sigma}) \to S(\vec{\sigma})) & \xrightarrow{H} & (S(\vec{\sigma}) \to S(\vec{\sigma})) \end{array}$$

where $H := \lambda g.\ h \circ g$. This is a strict program since h is assumed to be strict. The above diagram commutes since

$$\begin{aligned} H \circ \phi(g) &= H(i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, g) \circ i^{-1}_{S(\vec{\sigma})}) \\ &= h \circ i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, g) \circ i^{-1}_{S(\vec{\sigma})} \\ &= i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, h) \circ T(\mathrm{id}_{\vec{\sigma}}, g) \circ i^{-1}_{S(\vec{\sigma})} \\ &= i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, h \circ g) \circ i^{-1}_{S(\vec{\sigma})} \\ &= i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, H(g)) \circ i^{-1}_{S(\vec{\sigma})} \\ &= \phi \circ H(g). \end{aligned}$$

By Lemma 3.13, it follows that

$$\mathrm{id}_{S(\vec{\sigma})} = \mathrm{fix}(\phi) = H(\mathrm{fix}(\phi)) = h \circ \mathrm{id}_{S(\vec{\sigma})} = h.$$

So, the identity is the least endomorphism such that

$$h \circ i_{S(\vec{\sigma})} = i_{S(\vec{\sigma})} \circ T(\mathrm{id}_{S(\vec{\sigma})}, h).$$

□

Theorem 5.27. Let $\Theta \vdash \sigma$ be a type-in-context. Then, the realisable action $F_{\Theta \vdash \sigma}$ (whose construction first appears immediately after Remark 4.5) is in fact a realisable functor.

Proof. The construction following Remark 4.5 ensures that realisable actions $F_{\Theta \vdash \sigma}$ preserve composition of morphisms, so it just remains to show that they preserve identity morphisms. The proof proceeds by induction on the structure of $\Theta \vdash \sigma$:

— *Type variables:*
If $\vec{X} \vdash X_i$, the morphism part of $F_{\vec{X} \vdash X_i}$ is realised by

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash f_i : R_i \to S_i.$$

If $\vec{X} \vdash Y$ where $Y \neq X_1, \ldots, X_n$, we define

$$F_{\vec{X} \vdash Y} = \mathrm{id}_Y.$$

Clearly, $F_{\vec{X} \vdash X_i}$ and $F_{\vec{X} \vdash Y}$ preserve identity morphisms.

Next, we perform a construction that is simultaneously associated with an inductive proof of correctness, that is, the subsequent syntactic operations described do indeed preserve identity morphisms.

To this end, we assume the following induction hypotheses:

(a) $F_{\vec{X} \vdash \tau_i}$ $(i = 1, 2)$, $F_{\vec{X} \vdash \tau}$ and $F_{\vec{X}, X \vdash \sigma}$ are the realisable actions corresponding to $\vec{X} \vdash \tau_i$ $(i = 1, 2)$, $\vec{X} \vdash \tau$ and $\vec{X}, X \vdash \sigma$ respectively, and they do preserve identity morphisms.
(b) The morphism parts of these realisable actions are respectively realised by
— $\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash t_i : \tau_i[\vec{R}/\vec{X}] \to \tau_i[\vec{S}/\vec{X}] \quad (i = 1, 2)$;
— $\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash t : \tau[\vec{R}/\vec{X}] \to \tau[\vec{S}/\vec{X}]$; and
— $\vec{R}, R, \vec{S}, S; \vec{f} : \vec{R} \to \vec{S}, f : R \to S \vdash s : \sigma[\vec{R}/\vec{X}, R/X] \to \sigma[\vec{R}/\vec{X}, R/X]$.

— *Non-recursive types:*
The proofs for non-recursive types are straightforward and we will just give the proof for the case of product types as an example.
The morphism part of $F_{\vec{X}\vdash\tau_1\times\tau_2}$ is realised by

$$\vec{R},\vec{S};\vec{f} : \vec{R} \to \vec{S} \vdash t : (\tau_1 \times \tau_2)[\vec{R}/\vec{X}] \to (\tau_1 \times \tau_2)[\vec{S}/\vec{X}],$$

where

$$t^- := \lambda p : (\tau_1 \times \tau_2)[\vec{S}/\vec{X}].(t_1^-(\mathrm{fst}(p)), t_2^-(\mathrm{snd}(p)))$$
$$t^+ := \lambda q : (\tau_1 \times \tau_2)[\vec{R}/\vec{X}].(t_1^+(\mathrm{fst}(q)), t_2^+(\mathrm{snd}(q))).$$

Note that $F_{\vec{X}\vdash\tau_1\times\tau_2}$ preserves identity morphisms because the $F_{\vec{X}\vdash\tau_i}$ do by the induction hypotheses.
— *Recursive types:*
For recursive types, the fact that $F_{\vec{X}\vdash\mu X.\,\sigma}$ preserves identity morphisms is just Theorem 5.1. □

6. Operational algebraic compactness

Freyd (1991) introduced the notion of algebraic compactness to capture the bifree nature of the canonical solution to the domain equation

$$X = FX$$

in that 'every endofunctor (on cpo-enriched categories, for example, $\mathbf{DCPO}_{\perp!}$, the category of pointed cpos and strict maps†) has an initial algebra and a final co-algebra, and they are canonically isomorphic'. In the same reference, Freyd proved the *Product Theorem*, which asserts that algebraic compactness is closed under finite products. Crucially, this implies that $\mathbf{DCPO}_{\perp!} \times \mathbf{DCPO}_{\perp!}^{\mathrm{op}}$ is algebraically compact (since its components are), and thus allows us to cope well with the mixed-variant functors, which makes the study of recursive domain equations complete. On the other hand, proving that $\mathbf{DCPO}_{\perp!}$ is algebraically compact is no easy feat if we just use the usual proof technique of switching to the category of embeddings and projections, and using a bilimit construction. However, we shall use the operational machinery developed so far to establish operational algebraic compactness with respect to the class of realisable functors.

In this section, we establish that the diagonal category $\mathbf{FPC}_!^{\delta}$ is parametrised algebraically compact. We also consider an alternative choice of categorical framework, namely the product category $\breve{\mathbf{FPC}}_! := \mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$, and show that this is also parametrised algebraically compact. We then briefly compare the two approaches.

Note that we will rely on uniformity (*cf.* Lemma 3.13) in establishing the algebraic compactness results in Sections 6.1–6.2 (the proof technique we use was probably first employed in Simpson (1992) for Kleisli-categories).

To begin, we will need to put in place a few definitions.

† If non-strict maps are considered, the identity functor does not have an initial algebra.

Definition 6.1. Let F be an endofunctor on a category $\mathcal{C}$. An *F-algebra* is an object A together with a morphism $f : FA \to A$, denoted by (A, f). An F-algebra homomorphism from (A, f) to (A', f') is a $\mathcal{C}$-morphism $g : A \to A'$ such that the diagram

$$\begin{array}{ccc} FA & \xrightarrow{Fg} & FA' \\ {\scriptstyle f_A}\downarrow & & \downarrow{\scriptstyle f_{A'}} \\ A & \xrightarrow[g]{} & A' \end{array}$$

commutes. The category of F-algebras, denoted by $\mathcal{C}^F$, consists of F-algebras as objects and F-algebra homomorphisms as morphisms. The initial F-algebras are precisely the initial objects in $\mathcal{C}^F$. Dually, we can also define the F-coalgebras and the final F-coalgebras.

6.1. *Operational algebraic compactness*

Theorem 6.2 (operational algebraic completeness I). Every realisable endofunctor

$$F : \mathbf{FPC}_!^{\ \delta} \to \mathbf{FPC}_!^{\ \delta}$$

has an initial algebra.

We say that the category $\mathbf{FPC}_!^{\ \delta}$ is *operationally algebraically complete* with respect to the class of realisable functors.

Proof. Let $X \vdash \tau$ be the type-in-context that realises F. We use D to denote $\mu X.\ \tau$ and i to denote $(\mathrm{unfold}, \mathrm{fold})^{\mu X.\ \tau}$. We claim that (D, i) is an initial F-algebra. To show this, we suppose (D', i') is another F-algebra. We must show that there is a unique F-algebra homomorphism $k = (k^-, k^+)$ from (D, i) to (D', i'). We begin by defining k to be the least homomorphism for which the diagram

$$\begin{array}{ccc} FD & \xrightarrow{i} & D \\ {\scriptstyle Fk}\downarrow & & \downarrow{\scriptstyle k} \\ FD' & \xrightarrow[i']{} & D' \end{array}$$

commute. In other words, we define k to be the least solution of the recursive equation

$$k = i' \circ F(k) \circ i^{-1}.$$

Of course, k fits into the above commutative diagram. It remains to show that k is unique. To achieve this, we suppose k' is another morphism that makes the above diagram

commute. Then we consider the diagram

$$\begin{array}{ccc} (D \to D) \times (D \to D) & \xrightarrow{G} & (D' \to D) \times (D \to D') \\ \Big\downarrow \Phi & & \Big\downarrow \Psi \\ (D \to D) \times (D \to D) & \xrightarrow[G]{} & (D' \to D) \times (D \to D') \end{array}$$

where the programs Φ, Ψ and G are defined by

$$\begin{aligned} \Phi &:= \lambda h : (D \to D) \times (D \to D).i \circ F(h) \circ i^{-1} \\ \Psi &:= \lambda k : (D' \to D) \times (D \to D').i' \circ F(k) \circ i^{-1} \\ G &:= \lambda h : (D \to D) \times (D \to D).k' \circ h. \end{aligned}$$

Note that from the definition of k, we have $\mathrm{fix}(\Psi) = k$. The above diagram commutes because for any $h : (D \to D) \times (D \to D)$, we have

$$\begin{aligned} G(\Phi(h)) &= k' \circ \Phi(h) \\ &= k' \circ i \circ F(h) \circ i^{-1} && \text{(definition of } \Phi) \\ &= i' \circ F(k') \circ i^{-1} \circ i \circ F(h) \circ i^{-1} && (k' = i' \circ F(k') \circ i^{-1}) \\ &= i' \circ F(k') \circ F(h) \circ i^{-1} && (\text{unfold} = \text{fold}^{-1}) \\ &= i' \circ F(k' \circ h) \circ i^{-1} && (F \text{ is a functor}) \\ &= \Psi(k' \circ h) && (\text{definition of } \Psi) \\ &= \Psi(G(h)). \end{aligned}$$

Note that $\mathrm{fix}(\Phi) = (\mathrm{id}_D, \mathrm{id}_D)$ by Lemma 5.1. Since G is strict (because k' is), it follows from Lemma 3.13 that

$$k = \mathrm{fix}(\Psi) = k' \circ \mathrm{fix}(\Phi) = k' \circ (\mathrm{id}_D, \mathrm{id}_D) = k'.$$

Thus we have established the uniqueness of k. □

Definition 6.3. Let $F : \mathbf{C} \to \mathbf{C}$ be an endofunctor. An initial F-algebra is called a *bifree algebra* if its inverse is also a final co-algebra.

Theorem 6.4 (operational algebraic compactness I). Let $F : \mathbf{FPC}_!^{\delta} \to \mathbf{FPC}_!^{\delta}$ be a realisable endofunctor. Then, every initial F-algebra is bifree.

We say that the category $\mathbf{FPC}_!^{\delta}$ is *operationally algebraically compact* with respect to the class of realisable functors.

Proof. Consider the initial F-algebra $i : F(D) \to D$ where (D, i) is as defined in the proof of Theorem 6.2. Note that $i^{-1} = (\mathrm{fold}, \mathrm{unfold})^D$, so $i^{-1} : D \to F(D)$ is an F-coalgebra. Using arguments similar to those for establishing initiality, it is straightforward to show that (D, i^{-1}) is a final F-coalgebra. □

Theorem 6.5 (operational parametrised algebraic compactness I). Let $F : (\mathbf{FPC}_!^{\ \delta})^{n+1} \to \mathbf{FPC}_!^{\ \delta}$ be a realisable functor. Then there exists a realisable functor $H : (\mathbf{FPC}_!^{\ \delta})^n \to \mathbf{FPC}_!^{\ \delta}$ and a natural isomorphism i such that for all sequences of closed types P in $(\mathbf{FPC}_!^{\ \delta})^n$, we have

$$i_P : F(P, H(P)) \cong H(P).$$

Moreover, $(H(P), i_P)$ is a bifree algebra for the endofunctor

$$F(P, \_) : \mathbf{FPC}_!^{\ \delta} \to \mathbf{FPC}_!^{\ \delta}.$$

We say that the category $\mathbf{FPC}_!^{\ \delta}$ is *operationally parametrised algebraically compact* with respect to the class of realisable functors.

Proof. Assume that the functor F is realised by a type-in-context $\vec{X}, X \vdash \tau$ and a pair of terms-in-context of the form

$$\vec{R}, R, \vec{S}, S; \vec{f}, f : \vec{R}, R \to \vec{S}, S \vdash t : \tau[\vec{R}/\vec{X}, R/X] \to \tau[\vec{S}/\vec{X}, S/X].$$

It is clear that every $P \in (\mathbf{FPC}_!^{\ \delta})^n$ induces a realisable endofunctor

$$F(P, \_) : \mathbf{FPC}_!^{\ \delta} \to \mathbf{FPC}_!^{\ \delta},$$

and by operational algebraic completeness of $\mathbf{FPC}_!^{\ \delta}$, we always have an initial $F(P, \_)$-algebra, which we denote by $(H(P), i_P)$. Next we extend the action of H to morphisms. For every $f : P \to Q$, let $H(f)$ be the unique $F(P, \_)$-algebra homomorphism from $(H(P), i_P)$ to $(H(Q), i_Q \circ F(f, H(Q)))$, that is, $H(f)$ is the unique morphism g for which the diagram

$$\begin{array}{ccccc} F(P, H(P)) & & \xrightarrow{\quad i_P \quad} & & H(P) \\ \Big\downarrow{\scriptstyle F(P,g)} & & & & \Big\downarrow{\scriptstyle g} \\ F(P, H(Q)) & \xrightarrow[F(f, H(Q))]{} & F(Q, H(Q)) & \xrightarrow[i_Q]{} & H(Q) \end{array}$$

commutes. By the universal property of initial algebras, H is a functor and by construction, i is a natural transformation. Moreover, Theorem 6.4 ensures that $(H(P), i_P)$ is a bifree $F(P, \_)$-algebra. Finally, note that H is realised by the type-in-context $\vec{X} \vdash \mu X.\ \tau$ and the pair of terms-in-context

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \mathrm{fix}(\lambda v. i^{-1}_{H(\vec{R})} \circ t[v/f] \circ i_{H(\vec{S})}),$$

where $i := \langle \mathrm{fold}, \mathrm{unfold} \rangle$. □

6.2. *Alternative choice of category*

The theory of recursive domain equations since the work of Freyd, Fiore and Plotkin in the early 1990's has been centred around functors of the form

$$F : (\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!})^{n+1} \to (\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}).$$

As noted before, $\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}$ is algebraically compact. But more generally, $\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}$ is parameterised algebraically compact – a result that is implied by Fiore and Plotkin (1994, Corollary 5.6).

Let $\mathbf{F\check{P}C}_!$ denote the product category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$, where $\mathbf{FPC}_!$ was defined in Definition 4.1. The natural question to ask is whether the category $\mathbf{F\check{P}C}_!$ is algebraically compact. In order for this question to make sense, we have to identify an appropriate class of functors, $\mathcal{F}$, with respect to which algebraic compactness is defined. In this section, we show that, with a suitable choice of $\mathcal{F}$, the category $\mathbf{F\check{P}C}_!$ is parametrised algebraically compact with respect to $\mathcal{F}$, that is, for every $\mathcal{F}$-functor $T : (\mathbf{F\check{P}C}_!)^{n+1} \to \mathbf{F\check{P}C}_!$, there exists an $\mathcal{F}$-functor $H : (\mathbf{F\check{P}C}_!)^n \to (\mathbf{F\check{P}C}_!)$ and a natural isomorphism i such that for every sequence of closed types $\vec{\sigma} := \sigma_1^-, \sigma_1^+, \ldots, \sigma_n^-, \sigma_n^+$, the pair $(H(\vec{\sigma}), i_{\vec{\sigma}})$ is a bifree algebra of the endofunctor $T(\vec{\sigma}, -, +) : \mathbf{F\check{P}C}_! \to \mathbf{F\check{P}C}_!$.

In the framework of the product category $\mathbf{F\check{P}C}_!$, we are required to enforce a separation of positive and negative occurrences of variables. An occurrence of X in a type expression is positive (respectively, negative) if it is hereditarily to the left of an even (respectively, odd) number of function space constructors. For example, for the type expression $X + (X \to X)$, separation yields $X^+ + (X^- \to X^+)$.

Notation 6.6. We use the following notation:

$$\begin{aligned}
\vec{X} &:= X_1^-, X_1^+, \ldots, X_n^-, X_n^+ \\
\vec{X}^{\pm} &:= X_1^+, X_1^-, \ldots, X_n^+, X_n^- \\
\vec{\sigma} &:= \sigma_1^-, \sigma_1^+, \ldots, \sigma_n^-, \sigma_n^+ \\
\vec{\sigma}^{\pm} &:= \sigma_1^+, \sigma_1^-, \ldots, \sigma_n^+, \sigma_n^- \\
\vec{f} : \vec{R} \to \vec{S} &:= \vec{f^+} : \vec{R^+} \to \vec{S^+}, \vec{f^-} : \vec{S^-} \to \vec{R^-}.
\end{aligned}$$

We will also sometimes use P and Q to denote objects in $\mathbf{F\check{P}C}_!$, and u for morphisms in $\mathbf{F\check{P}C}_!$.

We begin by considering an appropriate class of n-ary functors of type

$$(\mathbf{F\check{P}C}_!)^n \to \mathbf{FPC}_!.$$

A seemingly reasonable choice is the class of *syntactic functors*[†], which is defined as follows.

A syntactic functor $T : (\mathbf{F\check{P}C}_!)^n \to \mathbf{FPC}_!$ is a functor that is realised by

(1) a type-in-context $\vec{X} \vdash \tau$; and
(2) a term-in-context of the form

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash t : \tau[\vec{R}/\vec{X}] \to \tau[\vec{S}/\vec{X}]$$

such that for any $\vec{\sigma} \in \mathbf{F\check{P}C}_!^{\,n}$, we have

$$T(\vec{\sigma}) = \tau[\vec{\sigma}/\vec{X}]$$

[†] This class was originally used by A. Rohr in his Ph.D. thesis Rohr (2002).

and for any $\vec{\rho}, \vec{\sigma} \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}$, and any $\vec{u} \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}(\vec{\rho}, \vec{\sigma})$, we have

$$T(\vec{u}) = t[\vec{u}/\vec{f}].$$

However, there are some problems with this definition. First, syntactic functors are not functors of type $\mathbf{F}\check{\mathbf{P}}\mathbf{C}_! \to \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!$, so it does not immediately make sense to study parametrised algebraic compactness with respect to this class of functors. However, this problem is superficial and is easy to overcome in the following way. For a given syntactic functor $F : (\mathbf{F}\check{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{FPC}_!$, there is a standard way of turning it into an endofunctor $\check{F} : (\mathbf{F}\check{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!$ by defining $\check{F}$ by

$$\check{F}(\vec{\sigma}) = (F(\vec{\sigma}^{\pm}), F(\vec{\sigma})).$$

So we might consider defining a functor $G : \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n} \to \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!$ to be syntactic if there exists a syntactic functor $F : \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n} \to \mathbf{FPC}_!$ such that $G = \check{F}$.

However, if we work with this definition, a serious problem[†] arises. As we shall see in Theorem 6.10, the parametrised initial algebra of such functors are not of the form $\check{H}$ for some functor H.

This can be fixed by working with our official definition as follows.

Definition 6.7. An n-ary functor $F : (\mathbf{F}\check{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!$ is said to be *syntactic* if it is given by

(i) a pair of types-in-context $\vec{X} \vdash \tau^-, \tau^+$, and
(ii) a pair of terms-in-context $\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash$

$$t^- : \tau^-[\vec{S}/\vec{X}] \to \tau^-[\vec{R}/\vec{X}], t^+ : \tau^+[\vec{R}/\vec{X}] \to \tau^+[\vec{S}/\vec{X}].$$

such that for any $\vec{\sigma} \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}$, we have

$$F(\vec{\sigma}) = (\tau^-[\vec{\sigma}/\vec{X}], \tau^+[\vec{\sigma}/\vec{X}]),$$

and for any $\vec{\rho}, \vec{\sigma} \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}$ and any $\vec{u} \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}(\vec{\rho}, \vec{\sigma})$, we have

$$F(\vec{u}) = \langle t^-, t^+ \rangle [\vec{u}/\vec{f}].$$

Before we establish operational algebraic completeness and compactness for the category $\mathbf{F}\check{\mathbf{P}}\mathbf{C}_!$, we will look at some examples.

[†] This problem was discovered by the author and T. Streicher during a private communication in January 2006.

Example 6.8.

(1) Consider the type-in-context $X \vdash X \to X$. The object part of the syntactic functor $T_{\check{X}\vdash X\to X}$ is realised by the types-in-context

$$X^-, X^+ \vdash (X^+ \to X^-), (X^- \to X^+).$$

The morphism part of the syntactic functor $T_{\check{X}\vdash X\to X}$ is realised by the term-in-context

$$R, S; f : R \to S \vdash \langle t^-, t^+ \rangle$$

where

$$t^- := \lambda g : (S^+ \to S^-).f^- \circ g \circ f^+$$
$$t^+ := \lambda h : (R^- \to R^+).f^+ \circ h \circ f^-.$$

(2) The type-in-context

$$X_2 \vdash \mu X_1.(X_1 \to X_2)$$

is not functorial in X_2 since one unfolding of

$$\mu X_1.(X_1 \to X_2)$$

yields

$$(\mu X_1.(X_1 \to X_2)) \to X_2$$

and the latter expression does not respect the variance of X_2. It seems clear that there is no syntactic functor whose object part is realised by the type-in-context

$$X_2 \vdash \mu X_1.(X_1 \to X_2).$$

Remark 6.9. Crucially, Example 6.8(2) indicates that if a bifree algebra for

$$X_2, X_1 \vdash X_1 \to X_2$$

were to exist, it cannot be given simply by

$$X_2 \vdash \mu X_1.(X_1 \to X_2).$$

Theorems 6.10 and 6.12 will provide us with a way to obtain the required bifree algebra.

Following the method of Freyd's proof of his Product Theorem (Freyd 1990), we will establish the following in our operational setting – we will only give the essential constructions and omit the detailed verifications.

Theorem 6.10 (operational algebraic completeness II). Every syntactic functor

$$F : \mathbf{F\check{P}C}_! \to \mathbf{F\check{P}C}_!$$

has an initial algebra.

We say that the category $\mathbf{F\check{P}C}_!$ is *operationally algebraically complete* with respect to the class of syntactic functors.

Proof. Recall that F can be resolved into its coordinate functors

$$T^- : \mathbf{F}\check{\mathbf{P}}\mathbf{C}_! \to \mathbf{FPC}_!^{\mathrm{op}}$$
$$T^+ : \mathbf{F}\check{\mathbf{P}}\mathbf{C}_! \to \mathbf{FPC}_!,$$

which are explicitly defined as follows. Note that T^- (respectively, T^+) is realised by a type-in-context $\vec{X} \vdash \tau^-$ (respectively τ^+) and a term-in-context

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash t^- : \tau^-[\vec{S}/\vec{X}] \to \tau^-[\vec{R}/\vec{X}]$$

(respectively, t^+).

We will construct an initial F-algebra using the following steps:

(1) For each σ^+ in $\mathbf{FPC}_!$, consider the endofunctor

$$T^-(\_, \sigma^+) : \mathbf{FPC}_!^{\mathrm{op}} \to \mathbf{FPC}_!^{\mathrm{op}}.$$

The initial algebra of this endofunctor will be one of the ingredients required in the proof. Thus we must prove that $T^-(\_, \sigma^+)$ has an initial algebra. We do not need to look very far to find

$$\mathrm{unfold}^{\mathrm{op}} : T^-(\mu X^-.T^-(X^-, \sigma^+), \sigma^+) \to \mu X^-.T^-(X^-, \sigma^+).$$

For convenience, we will use $f^-_{\sigma^+}$ to denote $\mathrm{unfold}^{\mathrm{op}}$ and $F^-(\sigma^+)$ to denote

$$\mu X^-.T^-(X^-, \sigma^+).$$

Rewriting using these gives

$$f^-_{\sigma^+} : T^-(F^-(\sigma^+), \sigma^+) \to F^-(\sigma^+).$$

We can then verify that this is an initial $T^-(\_, \sigma^+)$-algebra in $\mathbf{FPC}_!^{\mathrm{op}}$, using a proof similar to that in Theorem 6.2.

(2) We now extend F^- to be a functor $\mathbf{FPC}_! \to \mathbf{FPC}_!^{\mathrm{op}}$. Note that F^- is realised by the type-in-context

$$X^+ \vdash \mu X^-.\ \tau^-$$

and the term-in-context

$$R^+, S^+; f^+ : R^+ \to S^+ \vdash \mathrm{fix}(\lambda v.\ \mathrm{fold} \circ t^-[v/f^-] \circ \mathrm{unfold}),$$

and is thus syntactic. We now define the morphism part of F^-. Let $w^+ : \rho^+ \to \sigma^+$ be a $\mathbf{FPC}_!$-morphism. Using the initiality of $F^-(\rho^+)$, we define $F^-(w^+)$ to be the unique morphism that makes the following diagram commute in $\mathbf{FPC}_!^{\mathrm{op}}$:

$$\begin{array}{ccc} T^-(F^-(\rho^+), \rho^+) & \xrightarrow{f^-_{\rho^+}} & F^-(\rho^+) \\ \downarrow{\scriptstyle T^-(F^-(w^+), w^+)} & & \downarrow{\scriptstyle F^-(w^+)} \\ T^-(F^-(\sigma^+), \sigma^+) & \xrightarrow[f^-_{\sigma^+}]{} & F^-(\sigma^+) \end{array}$$

Note that the functoriality of F^- is derived from the initiality of $F^-(\rho^+)$.

(3) In this step we define an endofunctor $G : \mathbf{FPC}_! \to \mathbf{FPC}_!$ by

$$G(\sigma^+) := T^+(F^-(\sigma^+), \sigma^+),$$

which is syntactic since it is realised by the type-in-context

$$X^+ \vdash \tau^+[(\mu X^-.\ \tau^-)/X^-]$$

and the term-in-context

$$R^+, S^+, f^+ : R^+ \to S^+ \vdash t^+[\mathrm{fix}(\lambda v.\ \mathrm{fold} \circ t^-[v/f^-] \circ \mathrm{unfold})/f^-].$$

We have the initial algebra for G given by

$$\mathrm{fold}^{\mu X^+.G(X^+)} : T^+(F^-(\mu X^+.G(X^+)), \mu X^+.G(X^+)) \to \mu X^+.G(X^+).$$

We use the denotations δ^+ for $\mu X^+.G(X^+)$ and d^+ for $\mathrm{fold}^{\mu X^+.G(X^+)}$. Rewriting, we have the initial G-algebra given by

$$d^+ : T^+(F^-(\delta^+), \delta^+) \to \delta^+. \tag{6.1}$$

We also define $\delta^- := F^-(\delta^+)$ and denote the initial $T^-(\_, \delta^+)$-algebra

$$f^-_{\delta^+} : T^-(F^-(\delta^+), \delta^+) \to F^-(\delta^+)$$

by

$$d^- : T^-(\delta^-, \delta^+) \to \delta^-, \tag{6.2}$$

bearing in mind that this is a morphism in $\mathbf{FPC}_!^{\mathrm{op}}$.

(4) By closely following the proof method outlined in Freyd (1990), we can show that

$$(d^-, d^+) : (T^-(\delta^-, \delta^+), T^+(\delta^-, \delta^+)) \to (\delta^-, \delta^+)$$

is an initial (T^-, T^+)-algebra, that is, F-algebra, which is what we set out to find initially. The proof is now complete. □

Theorem 6.11 (operational algebraic compactness II). Let $F : \mathbf{F\check{P}C}_! \to \mathbf{F\check{P}C}_!$ be a syntactic functor. Then the initial algebra of F is bifree in the sense that the inverse

$$(d^-, d^+)^{-1} : (\delta^-, \delta^+) \to F(\delta^-, \delta^+)$$

is a final F-coalgebra.

We say that the category $\mathbf{F\check{P}C}_!$ is *operationally algebraically compact* with respect to the class of syntactic functors.

Proof. By walking through the steps of the proof of Theorem 6.10, we can check that at each step a final coalgebra results when each initial algebra structure map is inverted. Note that this even works for the definition of F^- in step (2). □

Theorem 6.12 (operational parametrised algebraic compactness II). Let $F : (\mathbf{F\check{P}C}_!)^{n+1} \to \mathbf{F\check{P}C}_!$ be a syntactic functor. Then there exists a syntactic functor $H : \mathbf{F\check{P}C}_!{}^n \to \mathbf{F\check{P}C}_!$ and a natural isomorphism i such that for all sequences of closed types P in $(\mathbf{F\check{P}C}_!)^n$ we

have

$$i_P : F(P, H(P)) \cong H(P).$$

Moreover, $(H(P), i_P)$ is a bifree algebra for the endofunctor

$$F(P, \_) : \mathbf{F\check{P}C}_! \to \mathbf{F\check{P}C}_!.$$

In other words, $\mathbf{F\check{P}C}_!$ is *parametrised operationally algebraically complete* with respect to the syntactic functors.

Proof. Since $F : (\mathbf{F\check{P}C}_!)^{n+1} \to \mathbf{F\check{P}C}_!$ is syntactic, it is realised by a pair of types-in-contexts $\vec{X}; X^-, X^+ \vdash \tau^-, \tau^+$ and a pair of terms-in-context

$$\vec{R}, R^-, R^+, \vec{S}, S^-, S^+; \vec{f} : \vec{R} \to \vec{S}, f^- : S^- \to R^-, f^+ : R^+ \to S^+ \vdash$$
$$t^- : \tau^-[\vec{S}/\vec{X}, S^{\mp}/X^{\mp}] \to \tau^-[\vec{R}/\vec{X}, R^{\mp}/X^{\mp}]$$
$$t^+ : \tau^+[\vec{R}/\vec{X}, R^{\mp}/X^{\mp}] \to \tau^+[\vec{S}/\vec{X}, S^{\mp}/X^{\mp}].$$

For each $P \in (\mathbf{F\check{P}C}_!)^n$, we have that $F(P, \_) : \mathbf{F\check{P}C}_! \to \mathbf{F\check{P}C}_!$ is a syntactic endofunctor whose component functors are $T^-(P, \_)$ and $T^+(P, \_)$. Invoking Theorem 6.10, we form the initial $F(P, \_)$-algebra, $(H(P), i_P)$, where

$$H^-(P) = \mu X^-.\ T^-(P, X^-, H^+(P)),$$
$$H^+(P) = \mu X^+.\ T^+(P, \mu X^-.\ T^-(P, X^-, X^+), X^+).$$

and

$$i_P = (d^-, d^+) = \left(\mathrm{unfold}^{H^-(P)}, \mathrm{fold}^{H^+(P)}\right).$$

Note that H is a syntactic functor realised by a pair of types-in-context:

$$\vec{X} \vdash \mu X^-.\ \tau^-[(\mu X^+.\ \tau^+[(\mu X^-.\ \tau^-)/X^-])/X^+]$$
$$\vec{X} \vdash \mu X^+.\ \tau^+[(\mu X^-.\ \tau^-)/X^-]$$

and a pair of terms-in-context

$$\vec{P}, \vec{Q}; \vec{f} : \vec{P} \to \vec{Q} \vdash$$
$$s^- := \mathrm{fix}(\lambda v.\ \mathrm{fold} \circ t^-[v/f^-] \circ \mathrm{unfold})[s^+/f^+],$$
$$s^+ := \mathrm{fix}(\lambda u.\ \mathrm{fold} \circ t^+[(\mathrm{fix}(\lambda v.\ \mathrm{fold} \circ t^-[v/f^-] \circ \mathrm{unfold}))/f^-] \circ \mathrm{unfold}).$$

Because of the repetitive nature of the task (*cf.* Theorem 6.5), we leave showing the action of H on morphisms $f : P \to Q$ in $(\mathbf{F\check{P}C}_!)^n$ as an exercise. □

Definition 6.13. In Theorem 6.12, the functor $H : (\mathbf{F\check{P}C}_!)^n \to \mathbf{F\check{P}C}_!$ is a bifree F-algebra, where $F : (\mathbf{F\check{P}C}_!)^{n+1} \to \mathbf{F\check{P}C}_!$ is a syntactic functor. To indicate this dependence, we write

$$H := \mu F.$$

To each $P \in (\mathbf{F\check{P}C}_!)^n$, H assigns the following pair of closed types:

$$H^-(P) = \mu X^-.\ T^-(P, X^-, H^+(P))$$
$$H^+(P) = \mu X^+.\ T^+(P, \mu X^-.\ T^-(P, X^-, X^+), X^+).$$

To each morphism $u \in \mathbf{F\check{P}C}_!^{\,n}(P, Q)$, the morphism $H(u)$ is the least morphism h for which the diagram

$$\begin{array}{ccc} F(P, H(P)) & \xrightarrow{i_P} & H(P) \\ \Big\downarrow{\scriptstyle F(u,h)} & & \Big\downarrow{\scriptstyle h} \\ F(Q, H(Q)) & \xrightarrow[i_Q]{} & H(Q) \end{array}$$

commutes.

Examples 6.14. Returning to Remark 6.8, we can now say that the correct bifree algebra of

$$X^-, X^+ \vdash (X^+ \to X^-), (X^- \to X^+)$$

should be

$$H^- = \mu X^-.\ H^+ \to X^-$$
$$H^+ = \mu X^+.\ ((\mu X^-.\ (X^+ \to X^-)) \to X^+).$$

Remark 6.15. The functor $H := \mu F$ is never symmetric, but we expect that symmetry can be achieved in the form of an operational analogue of Fiore's diagonalisation technique (*cf.* Fiore (1996, page 124)). Although we do not adopt this approach in this paper, we aim to establish an important result, Proposition 6.17, which can be seen as a restricted form of diagonalisation. We will develop this result in Section 6.3, which will establish the equivalence to a symmetric functor with respect to a restriction on the domain of the functor.

6.3. *On the choice of categorical frameworks*

In this section, we compare the two approaches using the diagonal category, $\mathbf{FPC}_!^{\,\delta}$, and the product category, $\mathbf{F\check{P}C}_!$.

In the framework of the product category $\mathbf{F\check{P}C}_!$, it is appropriate to study the class of syntactic functors because all FPC types-in-context can be viewed as syntactic functors. To avoid too much repetition (*cf.* the similar definition in Section 4), we will only show very briefly how this construction can be carried out by induction on the structure of $\Theta \vdash \sigma$ for type variables, function types and recursive types. We denote the syntactic functor associated with $\Theta \vdash \sigma$ by $G_{\Theta \vdash \sigma}$, or simply G.

— *Type variable:*
Let $\Theta \vdash X_i$ be given. We define the functor $G : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$ as follows:
  – For an object $P \in \mathbf{F\check{P}C}_!^{\,n}$, we define $G(P) := P_i$.
  – For a morphism $u \in \mathbf{F\check{P}C}_!^{\,n}(P, Q)$, we define $G(u) := u_i$.

— *Function type:*
Let $\Theta \vdash \sigma_1, \sigma_2$ be given and $G_1, G_2 : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$ be their associated realisable

functors. For a given syntactic functor $G : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$, we write

$$G^- : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{FPC}_!^{\mathrm{op}}$$
$$G^+ : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{FPC}_!$$

for its two component functors.
For an object $P \in \mathbf{F\check{P}C}_!^{\,n}$, we define

$$G(P) := (G_1^+(P) \to G_2^-(P), G_1^-(P) \to G_2^+(P)),$$

and for a morphism $u \in \mathbf{F\check{P}C}_!^{\,n}(P, Q)$, we define

$$G(u) := (G_1^+(u) \to G_2^-(u), G_1^-(u) \to G_2^+(u)),$$

where the component morphisms are defined by

$$G_1^+(u) \to G_2^-(u) = \lambda g : G_1^+(Q) \to G_2^-(Q).G_2^-(u) \circ g \circ G_1^+(u)$$
$$G_1^-(u) \to G_2^+(u) = \lambda h : G_1^-(Q) \to G_2^+(Q).G_2^+(u) \circ h \circ G_1^-(u).$$

— *Recursive type:*
Let $\Theta, X \vdash \sigma$ be given and F be the syntactic functor realising it. We define $G_{\Theta \vdash \mu X.\ \sigma}$ to be μF as in Definition 6.13.

Notation 6.16. From this point on, we denote the syntactic functor (respectively, the realisable functor) associated with the type-in-context $\Theta \vdash \sigma$ by $G_{\Theta \vdash \sigma}$ (respectively, $F_{\Theta \vdash \sigma}$). We will also use $\mathrm{Inj} : \mathbf{FPC}_!^{\,\delta} \to \mathbf{FPC}_!$ to denote the functor whose object part assigns to $\sigma \in \mathbf{FPC}_!^{\,\delta}$ a pair of types (σ^-, σ^+) with $\sigma^- = \sigma^+ = \sigma$, and whose morphism part assigns to $f^+ : \sigma \leftrightarrows \tau : f^-$ in $\mathbf{FPC}_!^{\,\delta}$ a pair of programs

$$f^- : \tau^- = \tau \to \sigma = \sigma^-$$
$$f^+ : \sigma^+ = \sigma \to \tau = \tau^+.$$

The following proposition reveals how the classes of realisable functors and syntactic functors are related.

Proposition 6.17. For every type-in-context $\Theta \vdash \sigma$, the realisable functor $F_{\Theta \vdash \sigma}$ restricts and co-restricts to the syntactic functor $G_{\Theta \vdash \sigma}$, that is, the diagram

$$\begin{array}{ccc} (\mathbf{FPC}_!^{\,\delta})^n & \xrightarrow{F_{\Theta \vdash \sigma}} & \mathbf{FPC}_!^{\,\delta} \\ \downarrow{\scriptstyle \mathrm{Inj}^n} & & \downarrow{\scriptstyle \mathrm{Inj}} \\ (\mathbf{F\check{P}C}_!)^n & \xrightarrow[G_{\Theta \vdash \sigma}]{} & \mathbf{F\check{P}C}_! \end{array}$$

commutes up to natural isomorphism.

Proof. We will prove by induction on the structure of $\Theta \vdash \sigma$ that for every type-in-context $\Theta \vdash \sigma$, there is a natural isomorphism

$$\eta : G_{\Theta \vdash \sigma} \circ \mathrm{Inj}^n \cong \mathrm{Inj} \circ F_{\Theta \vdash \sigma}.$$

— *Type variable;*
Let $\Theta \vdash X_i$ be given. We define

$$\eta : G_{\Theta \vdash X_i} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta \vdash X_i}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$,

$$\eta_{\vec{\sigma}} := \langle \mathrm{id}_{\sigma_i}, \mathrm{id}_{\sigma_i} \rangle.$$

Let $\Theta \vdash \tau_1, \tau_2$ be given. By the induction hypothesis, there are natural isomorphisms

$$\eta_j : G_{\Theta \vdash \tau_j} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta \vdash \tau_j}$$

for $j = 1, 2$. We write $\eta_j = \langle \eta_j^-, \eta_j^+ \rangle$.
— *Product type:*
We define

$$\eta : G_{\Theta \vdash \tau_1 \times \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta \vdash \tau_1 \times \tau_2}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$,

$$\eta_{\vec{\sigma}} := \langle (\eta_1^- \times \eta_2^-)_{\vec{\sigma}}, (\eta_1^+ \times \eta_2^+)_{\vec{\sigma}} \rangle$$

where

$$(\eta_1^- \times \eta_2^-)_{\vec{\sigma}} = \lambda p.\ (\eta_1^-(\mathrm{fst}(p)), \eta_2^-(\mathrm{snd}(p)))$$
$$(\eta_1^+ \times \eta_2^+)_{\vec{\sigma}} = \lambda q.\ (\eta_1^+(\mathrm{fst}(q)), \eta_2^+(\mathrm{snd}(q))).$$

— *Sum type:*
We define

$$\eta : G_{\Theta \vdash \tau_1 + \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta \vdash \tau_1 + \tau_2}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$,

$$\eta_{\vec{\sigma}} := \langle (\eta_1^- + \eta_2^-)_{\vec{\sigma}}, (\eta_1^+ + \eta_2^+)_{\vec{\sigma}} \rangle$$

where

$$(\eta_1^- + \eta_2^-)_{\vec{\sigma}} = \lambda z.\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{inl}(\eta_1^-(x))\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{inr}(\eta_2^-(y))$$
$$(\eta_1^+ + \eta_2^+)_{\vec{\sigma}} = \lambda z.\ \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\ \mathrm{inl}(\eta_1^+(x))\ \mathrm{or}\ \mathrm{inr}(y).\ \mathrm{inr}(\eta_2^+(y)).$$

(4) *Function type:*
We define

$$\eta : G_{\Theta \vdash \tau_1 \to \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta \vdash \tau_1 \to \tau_2}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$,

$$\eta_{\vec{\sigma}} := \langle \eta_1^+ \to \eta_2^-, \eta_1^- \to \eta_2^+ \rangle$$

where

$$(\eta_1^+ \to \eta_2^-) = \lambda g.\eta_2^- \circ g \circ \eta_1^+$$
$$(\eta_1^- \to \eta_2^+) = \lambda h.\eta_2^+ \circ h \circ \eta_1^-.$$

— *Lifted type:*
Let $\Theta \vdash \tau$ be given. The induction hypothesis asserts that there is a natural isomorphism

$$\eta : G_{\Theta\vdash\tau} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta\vdash\tau}.$$

We define a natural isomorphism

$$\eta_\perp : G_{\Theta\vdash\tau_\perp} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta\vdash\tau_\perp}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$,

$$(\eta_\perp)_{\vec{\sigma}} := \mathrm{case}(z)\ \ \mathrm{of}\ \ \mathrm{up}(x).\ \mathrm{up}(\eta(x)).$$

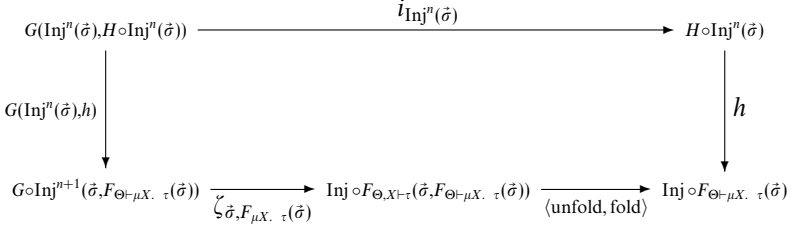
— *Recursive type:*
Let $\Theta, X \vdash \tau$ be given. The induction hypothesis asserts that there is a natural isomorphism

$$\zeta : G_{\Theta,X\vdash\tau} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta,X\vdash\tau}.$$

We define a natural isomorphism

$$\eta : G_{\Theta\vdash\mu X.\ \tau} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ F_{\Theta\vdash\mu X.\ \tau}$$

as follows:
For every $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$, we define $\eta_{\vec{\sigma}}$ to be the unique map h that fits into the commutative diagram:



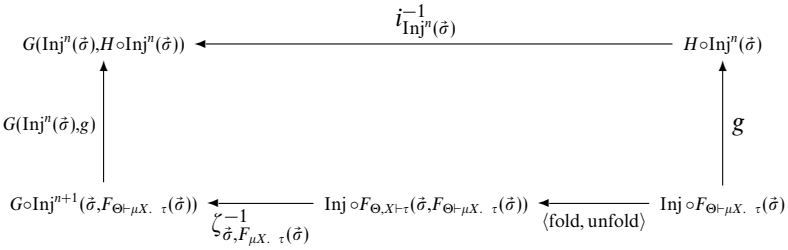
where $H := \mu G$. Note that the existence and uniqueness of h is guaranteed by the initiality of

$$i_{\mathrm{Inj}^n(\vec{\sigma})} : G(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) \to H \circ \mathrm{Inj}^n(\vec{\sigma}).$$

Since ζ, i and $\langle \mathrm{unfold}, \mathrm{fold} \rangle$ are natural, so is h. It remains to show that h is an isomorphism. To this end, we have to define the inverse of h. Now, since

$$i^{-1}_{\mathrm{Inj}^n(\vec{\sigma})} : H \circ \mathrm{Inj}^n(\vec{\sigma}) \to G(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma}))$$

is a final coalgebra and ζ is an isomorphism, there exists a unique g that fits into the following commutative diagram:



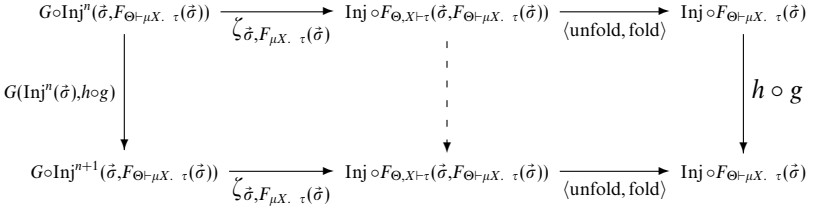
We claim that

$$g \circ h = \mathrm{id}_{H\circ\mathrm{Inj}^n(\vec{\sigma})}$$
$$h \circ g = \mathrm{id}_{\mathrm{Inj}\circ F_{\Theta\vdash\mu X.\ \tau}(\vec{\sigma})}.$$

To prove the first equation, note that $g \circ h$ is an $G(\mathrm{Inj}^n(\vec{\sigma}), \_)$-algebra endomorphism on $H \circ \mathrm{Inj}^n(\vec{\sigma})$. Thus, by the initiality of

$$i_{\mathrm{Inj}^n(\vec{\sigma})} : G(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) \to H \circ \mathrm{Inj}^n(\vec{\sigma}),$$

it must be the case that $g \circ h = \mathrm{id}_{H\circ\mathrm{Inj}^n(\vec{\sigma})}$. For the second equation, we consider the diagram below, which is obtained by pasting the two diagrams above:



where the dotted arrow is the morphism

$$\langle F_{\Theta, X\vdash\tau}(\vec{\sigma}, (h \circ g)^-), F_{\Theta, X\vdash\tau}(\vec{\sigma}, (h \circ g)^+)\rangle.$$

So for the second quadrangle, $(h \circ g)^-$ and $(h \circ g)^+$ are both endomorphisms on $F_{\Theta\vdash\mu X.\ \tau}(\vec{\sigma})$. By the initiality of

$$\mathrm{fold} : F_{\Theta, X\vdash\tau}(\vec{\sigma}, F_{\Theta\vdash\mu X.\ \tau}(\vec{\sigma})) \to F_{\Theta\vdash\mu X.\ \tau}(\vec{\sigma}),$$

we can conclude that

$$h \circ g = \mathrm{id}_{\mathrm{Inj}\circ F_{\Theta\vdash\mu X.\ \tau}(\vec{\sigma})}.$$

□

Within the framework of $\mathbf{F\check{P}C}_!$, the treatment of recursive types can be described schematically as follows:

(1) Perform a separation of type variables for a given type expression, that is, into the positive and negative occurrences, as described in the third paragraph of Section 6.2.
(2) Carry out the treatment (that is, the investigation in question), for example, the calculation of the bifree algebra of some syntactic functors as in Theorem 6.12.

(3) Perform a diagonalisation in the sense of Proposition 6.17 to derive the relevant conclusion regarding the original type expression.

In view of Proposition 6.17, this three-fold process can be carried out directly in the setting of the diagonal category. More precisely, for each closed type, there is a realisable functor that does the 'same' job as its syntactic counterpart restricted and co-restricted to the diagonal. Because realisable functors can cope with variances without having to distinguish explicitly between the positive and negative type variables, the theory developed from using the diagonal category, $\mathbf{FPC}_!^{\delta}$, is clean. For instance, the functoriality of recursive type expression $\mu X.\ \tau$ can be conveniently defined. This has a strong appeal to the programmer as it requires a relatively lighter categorical overhead.

However, as a mathematical theory for treating recursive types, the approach using the product category, $\check{\mathbf{FPC}}_!$, is general, and can cope with mathematical notions, such as di-algebras (*cf.* Freyd (1991)), that are beyond the diagonal category.

7. Conclusions and future work

The operational domain theory developed in this paper exploits the free algebra structure of the 'default' recursive construction offered by the syntax (and the operational semantics) of FPC. The categorical framework we have chosen facilitates a relatively clean theory on which convenient principles of reasoning about programs are based. The present work is a follow-up to the much earlier report Ho (2006a).

In future work, we aim to explore:

(1) the implication of Pitts' work (Pitts 1996) on relational properties of domains in our operational setting;
(2) the uses of our operational domain theory of recursive types in the field of exact real arithmetic (for example, in reasoning with streams of signed-digits in the representation of real numbers such as in Simpson (1998)); and
(3) the possibility of developing an operational domain-theory that caters for non-deterministic languages, such as in Hennessy and Ashcroft (1980).

Acknowledgements

Many people contributed to the birth and development of ideas in this paper. My special thanks go to M. H. Escardó, whose deep insight and gentle advice I have always benefited from as his student. I am grateful to P. B. Levy for first suggesting product categories as an alternative categorical framework. Following this suggestion, I was further reassured by M. P. Fiore that there should be no problems if I were to proceed with the use of product categories. T. Streicher's timely advice emerging from a series of email discussions eventually gave rise to the operational algebraic compactness result. My thanks are also due to M. H. Escardó and A. K. Simpson for encouraging me (time and again) to write this paper. I am also grateful to the anonymous referee for detailed proof-reading and many helpful comments, which tremendously improved the style of this paper. Finally, my special thanks go to my NIE colleagues for their support, which enabled me to focus on the task of writing during term time.

References

Abadi, M. and Fiore, M. (1996) Syntactic considerations on recursive types. In: *Proceedings of the 11th Annual IEEE Symposium on Logic In Computer Science*, IEEE Computer Society Press 242–252.

Abramsky, S. and Jung, A. (1994) Domain Theory. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of logic in computer science, volume 3: semantic structures*, Clarendon Press 1–168.

Birkedal, L. and Harper, R. (1999) Relational Interpretations of Recursive Types in an Operational Setting. *Information and Computation* **155** 3–63.

Escardó, M. H. and Ho, W. K. (2009) Operational domain theory and topology of sequential programming languages. *Information and Computation* **207** (3) 411–437.

Fiore, M. P. (1996) *Axiomatic Domain Theory in Categories of Partial Maps*, Ph.D. thesis, University of Edinburgh. Published in 2004 as: *Distinguished Dissertations in Computer Science* **14**, Cambridge University Press.

Fiore, M. P. and Plotkin, G. D. (1994) An Axiomatisation of Computationally Adequate Domain-Theoretic Models of FPC. In: *Proceedings of the 10th Annual IEEE Symposium on Logic In Computer Science*, IEEE Computer Society Press 92–102.

Fischer, M. and Ladner, R. (1979) Propositional dynamic logic of regular programs. *Journal of Computer System Science* **18** (2) 194–211.

Freyd, P. J. (1990) Recursive types reduced to inductive types. In: *Proceedings of the 5th Annual IEEE Symposium on Logic In Computer Science*, IEEE Computer Society Press 498–507.

Freyd, P. J. (1991) Algebraically complete categories. In: Smyth, B. and Cunningham, P. (eds.) Advances in Case-Based Reasoning, Proceedings 4th European Workshop, EWCBR-98. *Springer-Verlag Lecture Notes in Computer Science* **1488** 95–104.

Freyd, P. J. (1992) Remarks on algebraically compact categories. In: Fourman, M. P., Johnstone, P. T. and Pitts, A. M. (eds.) *Applications of Categories in Computer Science*, London Mathematical Society Lecture Note Series **177**, Cambridge University Press.

Gierz, G., Hofmann, K. H., Keimel, K., Lawson, J. D., Mislove, M. W. and Scott, D. S. (2003) *Continuous Lattices and Domains*, Encyclopedia of Mathematics and its Applications **93**, Cambridge University Press.

Hennessy, M. C. B. and Ashcroft, E. A. (1980) A mathematical semantics for a nondeterministic typed lambda-calculus. *Theoretical Computer Science* **11** (3) 227–245.

Ho, W. K. (2006a) An Operational Domain-theoretic Treatment of Recursive Types. In: Mislove, M. and Brookes, S. (eds.) Proceedings of the 22nd Conference on Mathematical Foundations in Programming Semantics. *Electronic Notes in Theoretical Computer Science* **158** 237–259.

Ho, W. K. (2006b) *An operational domain theory and topology of sequential functional languages*, Ph.D. thesis, The University of Brimingham.

Howe, D. J. (1989) Equality in lazy computation systems. In: *Proceedings of the 4th Annual Symposium on Logic In Computer Science*, IEEE Computer Society Press 198–203.

Howe, D. J. (1996) Proving congruence of bisimulation in functional programming languages. *Information and Computation* **124** (2) 103–112.

Kozen, D. (1983) Results on the propositional μ-calculus. *Theoretical Computer Science* **27** 333–354.

Lassen, S. B. (1998a) Relational Reasoning about Contexts. In: Gordon, A. D. and Pitts, A. M. (eds.) *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute series, Cambridge University Press 91–135.

Lassen, S. B. (1998b) *Relational Reasoning about Functions and Nondeterminism*, Ph.D. thesis, University of Aarhus.

Mac Lane, S. (1998) *Categories for the Working Mathematician*, 2nd edition, Springer-Verlag.

Mason, I. A., Smith, S. F. and Talcott, C. L. (1996) From operational semantics to domain theory. *Information and Computation* **128** (1) 26–47.

McCusker, G. (2000) Games and Full Abstraction for FPC. *Information and Computation* **160** 1–61.

Pitts, A. M. (1996) Relational Properties of Domains. *Information and Computation* **127** 66–90.

Pitts, A. M. (1997) Operationally-based theories of program equivalence. In: Dybjer, P. and Pitts, A. (eds.) *Semantics and Logics of Computation*, Publications of the Newton Institute series, Cambridge University Press 241–298.

Plotkin, G. D. (1977) LCF considered as a programming language. *Theoretical Computer Science* **5** (1) 223–255.

Plotkin, G. D. (1985) Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Languages and Information, Stanford.

Poigné, A. (1992) Basic category theory. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of logic in computer science, volume 1: Background – Mathematical Structures*, Oxford University Press 413–640.

Rohr, A. (2002) *A Universal Realizability Model for Sequential Functional Computation*, Ph.D. thesis, Technischen Universitat Darmstadt.

Simpson, A. K. (1992) Recursive Types in Kleisli Categories. Available from `http://homepages.inf.ed.ac.uk/als/Research`.

Simpson, A. K. (1998) Lazy functional algorithms for exact real functionals. In: Brim, L., Gruska, J. and Zlatuška, J. (eds.) Mathematical Foundations of Computer Science 1998: Proceedings 23rd International Symposium, MFCS'98. *Springer-Verlag Lecture Notes in Computer Science* **1450** 323–342.