Technical Report M2010-01
June 2010
Mathematics and Mathematics Education
National Institute of Education
Singapore

An Operational Domain-theoretic Treatment of Recursive Types

Weng Kin Ho
wengkin.ho@nie.edu.sg
Mathematics and Mathematics Education
National Institute of Education, Nanyang Technological University

June 30, 2010

Abstract

We develop an operational domain theory for treating recursive types with respect to contextual equivalence. The principal approach taken here deviates from classical domain theory in that we do not produce the recursive types via usual inverse limits constructions - we have it 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. In addition, we apply techniques developed therein to reason about FPC programs.

Keywords: Operational domain theory, recursive types, FPC, realisable functors, operational minimal invariance, algebraic compactness, generic approximation lemma, denotational semantics

1 Introduction

We develop a domain theory for treating recursive types with respect to contextual equivalence. The language we consider is sequential and has, in addition to recursive types, sum, product, function and lifted types. It is, by now, well accepted that the domain-theoretic model of such a language is *computationally adequate* but fails to be *fully abstract*, i.e., the denotational equality of two terms implies their contextual equivalence but not the converse (Fiore and Plotkin, 1994; Pitts, 1996).

In order to cope with this mismatch, we develop the operational counterpart of domain theory that deals with the solutions of recursive domain equations, i.e., via the functional programming language, FPC (Fixed Point Calculus). Such an enterprise may be seen as an extension of a similar programme in (Escardo and Ho, 2005) for the language PCF. Indeed several other authors have already exported domain-theory into the study of the operational order. Amongst these are (Mason et al., 1996), as well as (Birkedal and Harper, 1999) who, in particular, gave a relational interpretation to recursive types in an operational setting.

Our present study is founded on operational tools (which can be traced back to works like (Gordon, 1994, 1995; Howe, 1989, 1996; Mason et al., 1996)) developed in (Pitts, 1997) but in the setting of the language FPC. The principal approach taken here deviates from classical domain theory in that we do not produce recursive types by inverse limits constructions – we have it for free by directly with the recursive type declaration in FPC. Orthogonal to Birkedal and Harper's relational interpretation on recursive types (Birkedal and Harper, 1999), we focus on the connection between an operational version of P. Freyd's algebraic compactness and the underlying domain structure of the contextual order.

We work with two choices of syntactic categories. The first one is the diagonal category $\mathbf{FPC}_!{}^{\delta}$ built from closed FPC types $\mathbf{FPC}_!$ while the second is the product category $\mathbf{F\check{P}C}_!$. The bulk of the mathematical development, carried out in Section 3, involves the extension of formal type expressions to endofunctors on these categories. In order to establish the functoriality of type expressions, we rely on an operational version of the minimal invariance theorem – a landmark result in the solution of recursive domain equations. Crucially, the proof of this key property is based purely on operational arguments. Founded on such a development, we introduce an operational notion of (parameterised) algebraic compactness.

The main result of this paper (in Section 4) asserts that the syntactic categories considered (i.e., both $\mathbf{FPC}_!{}^{\delta}$ and $\mathbf{F\check{P}C}_!$) are (parameterised) algebraically compact operationally, the consequence of which are covered in the next two sections. In particular, every closed FPC type has a pre-deflationary structure from which one discovers an operational proof of the "generic approximation lemma" (Gibbons and Hutton, 2005). In Section 6, running examples taken from (Pitts, 1997) demonstrate the versatility of this proof method as compared to other such as bisimulation techniques (Gordon, 1994, 1995; Pitts, 1997) and program fusion (Gibbons and Hutton, 2005).

Throughout the discussion, we assume that the reader has some familiarity with a sequential functional language, for instance, PCF or `Haskell`. In addition, we omit the definitions of standard concepts such as η-rules, β-rules, contextual pre-order and equivalence, relying on the reader's pre-requisites. For the theory of recursive domain equations, the reader is invited to refer to (Abramsky and Jung, 1994; Gierz et al., 2003), and for category theory (MacLane, 1998).

2 The programming language FPC

We choose to work with a call-by-name version of FPC[1] (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. Let us now familiarise ourselves with the syntax and operational semantics of this sequential functional language. (Experts of FPC may skip this section entirely.)

[1] The interested reader may also find information on call-by-name FPC in McCusker (McCusker, 2000).

2.1 The syntax

We assume a set of *type variables* (ranged over by X, Y, etc.) and the *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, sum types, lifted types, recursive types and function types. A *closed type* is a type expression containing no free type variables, i.e., if any occurring type variable X is bound under the scope of a recursive type constructor μX. 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 following grammar, modulo α-equivalence (see Figure 1).

$t :=$	x	term variable
	$\mid (s, t)$	pairs
	$\mid \mathrm{fst}(p)$	first projection
	$\mid \mathrm{snd}(p)$	second projection
	$\mid \mathrm{inl}(t)$	separated sum
	$\mid \mathrm{inr}(t)$	separated sum
	$\mid \mathrm{case}(s)$ of $\mathrm{inl}(x).t$ or $\mathrm{inr}(y).t'$	case
	$\mid \mathrm{up}(t)$	lifting
	$\mid \mathrm{case}(s)$ 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

Figure 1: FPC syntax

Terms containing no free variables are called *closed terms*. Otherwise, they are known as *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 $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$ where $\Theta \vdash \sigma_i$ $(i = 1, \ldots, n)$ are well-formed types-in-context. When there is no confusion, we omit the type context Θ. The typing rules in FPC are given in Figure 2.

Convention: We use Θ to range over type contexts; X, Y, R, S over type variables; $\vec{X}, \vec{Y}$ over sequences of type variables; ρ, σ, τ over type expressions; Γ over term contexts; x, y, z, f, g, h over terms variables; $\vec{f}, \vec{g}$ over sequences of term variables, and s, t, u, v 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 also abbreviate the term context $x_1 : \sigma_1, \ldots, x_n : \sigma_n$ as $\vec{x} : \vec{\sigma}$.

The following is routine:

Lemma 2.1. *(1) If* $\Gamma \vdash t : \sigma$, *then* $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$.

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

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

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

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

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

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

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

Figure 2: Rules for type assignments in FPC

(2) If $\Gamma \vdash t:\sigma$ *and* $x \notin \mathrm{dom}(\Gamma)$*, then* $\Gamma, x:\tau \vdash t:\sigma$ *for any* τ*.*

(3) If $\Gamma, \Gamma' \vdash t:\sigma$ *and* $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$*, 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* $\Gamma \vdash s[\vec{t}/\vec{x}]:\sigma$*.*

Let $\mathrm{Exp}_{\sigma}(\Gamma)$ denote the set of FPC terms that can be assigned the closed type σ, given Γ, i.e., $\mathrm{Exp}_{\sigma}(\Gamma) := \{t | \Gamma \vdash t : \sigma\}$. We simply write Exp_{σ} for $\mathrm{Exp}_{\sigma}(\emptyset)$.

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:

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

A closed term v generated by the above grammar is called a *canonical value*. Let Val_{σ} denote the set of canonical values of the closed type σ, i.e.,

$$\mathrm{Val}_{\sigma} := \{v | \emptyset \vdash v : \sigma\}.$$

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

Evaluation is deterministic and preserves typing, i.e.,

Proposition 2.2. *(1) If* $t \Downarrow v$ *and* $t \Downarrow v'$*, then* $v \equiv v'$*.*

(2) If $t \Downarrow v$ *and* $t \in \mathrm{Exp}_{\sigma}$*, then* $v \in \mathrm{Exp}_{\sigma}$*.*

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

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

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

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

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

Figure 3: Rules for evaluating FPC terms

2.3 Fixed point operator

Like in existing works 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 $k := \lambda x^\tau.f(\text{unfold}^\tau(x)x)$.

Readers who are familiar with call-by-name PCF may wish to note that this fixed point operator evaluation rule **does not** hold:

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

2.4 Some notations

In this section, we shall gather at one place the notations which we use regarding the syntax of FPC.

To begin with, there are three special closed types 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 canonical values. Lifting the type 1 produces the *Sierpinski type*, $1_\bot$, which we denote by Σ. The non-divergent element of Σ, $\text{up}(\bot)$, is denoted by $\top$. We shall be exploiting Σ to make program observations.

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

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

Notice 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:

$$0 := \bot_{\overline{\omega}} \text{ and } n + 1 = \text{fold}(\text{up}(n)).$$

We define $n - 1 := \text{case}(\text{unfold}(n))\ \text{of}\ \text{up}(x).x$ and $\infty := \text{fix}(+1)$ where

$(+1) := \lambda x.x + 1$. The Σ-valued convergence test

$$(> 0) := \lambda x^{\overline{\omega}}.\text{case}(\text{unfold}(x)) \text{ of } \text{up}(y).\top$$

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

Some of our programs in Section 7 makes use of the *lazy natural numbers type*, which we now introduce. We define the lazy natural number data type to be the recursive type

$$\texttt{Nat} := \mu X.1 + X.$$

The data type `Nat` has canonical values given by:

$$\begin{array}{lcllcl} 0 & := & \text{fold}(\text{inl}(\bot_1)) & \overline{0} & := & \text{fold}(\text{inr}(\bot_{\texttt{Nat}})) \\ n+1 & := & \text{succ}(n) & \overline{n+1} & := & \text{succ}(\overline{n}) \\ & & & \infty & := & \text{fix}(\text{succ}) \end{array}$$

where $\text{succ} := \text{fold} \circ \text{inr}$. For our programs, we are only interested in computations with canonical values of the form n, which we call the *natural numbers*.

Using the operationally based theory of equivalence in works such as (Pitts, 1997; Ho, 2006b), it can shown that the contextual orders of 1, Σ, $\overline{\omega}$ and `Nat` are the usual ones interpreted by the Scott model. These are illustrated in Figure 4 below:

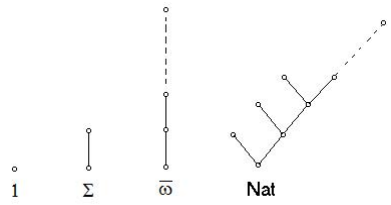


Figure 4: 1, Σ, $\overline{\omega}$, `Nat`

Remark 2.3. Because we want to work with a single evaluation strategy (i.e., call-by-name) throughout this paper, our version of FPC does not have flat natural numbers type and hence does not subsume PCF. If one wishes to have the flat natural numbers type in the language, one can always introduce an infinitary sum constructor or a distinguished flat natural numbers type.

2.5 FPC contexts

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

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

where p ranges over a fixed set of *parameters* (or holes). The details of the defining grammar is spelt out in Figure 5.

$C :=$	x	term variables
	$\mid (S, T)$	pairs
	$\mid \mathrm{fst}(P)$	first projection
	$\mid \mathrm{snd}(P)$	second projection
	$\mid \mathrm{inl}(T)$	separated sum
	$\mid \mathrm{inr}(T)$	separated sum
	$\mid \mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T\ \mathrm{or}\ \mathrm{inr}(y).T'$	case
	$\mid \mathrm{up}(T)$	liftings
	$\mid \mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T$	case up
	$\mid \mathrm{fold}(T)$	fold
	$\mid \mathrm{unfold}(T)$	unfold
	$\mid \lambda x.T$	function abstraction
	$\mid S(T)$	function application
	$\mid$ p	parameter (or hole)

Figure 5: FPC contexts

Convention. 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 contexts which are typable. The relation

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

assigning a closed type σ to a context C given the term Γ, is inductively generated by axioms and rules in Figure 6. 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 $\mathrm{Ctx}_\sigma(\emptyset)$.

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*, 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*. Notice that we have chosen the ground type Σ to be the type on which program observations are based.

Remark 2.4. Let $s, t : \sigma$ be closed terms. Then $s \sqsubseteq_\sigma t$ iff

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash S : \sigma_1 + \sigma_2 \quad \Gamma, x : \sigma_1 \vdash T_1 : \tau \quad \Gamma, y : \sigma_2 \vdash T_2 : \tau}{\Gamma \vdash \mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2 : \tau}\ (\text{case})$$

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

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

$$\frac{}{\Gamma \vdash -_{\sigma} : \sigma}\ (\text{par})$$

Figure 6: Typing rules for FPC contexts

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

3 Foundations

Our theory reported here builds upon the following foundations.

1. Every closed FPC type is *rationally-chain complete*. We apply A.M. Pitt's operationally based theories of program equivalence (Pitts, 1997) to FPC where contextual equivalence is taken with respect to the unit type Σ. Most importantly, each closed type σ is (pre)ordered contextually (denoted by $\sqsubseteq_\sigma$) is closed under the formation of rational chains:

$$\bot_\sigma \sqsubseteq_\sigma f(\bot_\sigma) \sqsubseteq_\sigma f^{(2)}(\bot_\sigma) \sqsubseteq_\sigma \ldots \sqsubseteq_\sigma f^{(n)}(\bot_\sigma) \sqsubseteq_\sigma \ldots$$

   where $\bot_\sigma := \mathrm{fix}(\lambda x^\sigma.x)$ and $f : \sigma \to \sigma$. Moreover, $\bigsqcup_n f^{(n)}(\bot_\sigma) = \mathrm{fix}(f)$. The crucial point here is that rational-chain completeness is proven by purely operational means and is *independent* of the properties of recursive type expressions reported herein.

2. Every program of function type is *rationally continuous*, i.e., for any $h : \sigma \to \tau$ and $f : \sigma \to \sigma$ it holds that

$$h(\bigsqcup_n f^{(n)}(\bot_\sigma)) = \bigsqcup_n h \circ f^{(n)}(\bot_\sigma).$$

   In addition, every program of function type is *monotone* with respect to the contextual preorder.

3. The operators fold and unfold are mutually inverse (modulo contextual equivalence). This can be established by involving one of the many η- and β-rules enjoyed by the contextual equivalence, namely:

$$\mathrm{fold} \circ \mathrm{unfold} = \mathrm{id}\ (\eta\text{-rule})\ \&\ \mathrm{unfold} \circ \mathrm{fold} = \mathrm{id}\ (\beta\text{-rule}).$$

4. Inequational logic and extensionality properties concerning the contextual preorder are properties so natural that they are frequently invoked without explicit mention. For the convenience of our readers, these properties are listed in Figures 7 and 8 below:

5. It is sometimes useful to know if two programs are *Kleene equivalent*. For each closed type σ, define the *Kleene preorder* and *Kleene equivalence* as follows:

$$t \sqsubseteq_\sigma^{kl} t' \iff \forall v.(t \Downarrow v \implies t' \Downarrow v)$$

   and

$$t \cong_\sigma^{kl} t' \iff (t \sqsubseteq_\sigma^{kl} t') \wedge (t' \sqsubseteq_\sigma^{kl} t).$$

   An important operational result is that any two Kleene equivalent programs are contextually equivalent.

The proofs of the above foundational facts are labour-intensive re-workings of those developed in (Pitts, 1997) in the present language of FPC and are thus omitted from this paper. Readers who are interested in pursuing such details may wish to refer to (Ho, 2006b), Chapters 7 and 8.

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash t \sqsubseteq_\sigma t \quad (1)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \vdash t' \sqsubseteq_\sigma t'') \implies \Gamma \vdash t \sqsubseteq_\sigma t'' \quad (2)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \vdash t' \sqsubseteq_\sigma t) \iff \Gamma \vdash t =_\sigma t' \quad (3)$$

$$\Gamma, x : \sigma \vdash t \sqsubseteq_\tau t' \implies \Gamma \vdash \lambda x.t \sqsubseteq_{\sigma\to\tau} \lambda x.t' \quad (4)$$

$$(\Gamma \vdash s \sqsubseteq_{\sigma_\perp} s' \wedge \Gamma, x : \sigma \vdash t \sqsubseteq_\rho t') \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).t' \quad (5)$$

$$(\Gamma \vdash s \sqsubseteq_{\sigma+\tau} s' \wedge \Gamma, x : \sigma \vdash t_1 \sqsubseteq_\rho t_1' \wedge \Gamma, y : \tau \vdash t_2 \sqsubseteq_\rho t_2') \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t_1'\ \mathrm{or}\ \mathrm{inr}(y).t_2' \quad (6)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \subseteq \Gamma') \implies \Gamma' \vdash t \sqsubseteq_\sigma t' \quad (7)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma, x : \sigma \vdash s : \tau) \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s[t'/x] \quad (8)$$

$$(\Gamma \vdash t : \sigma \wedge \Gamma, x : \sigma \vdash s \sqsubseteq_\tau s') \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s'[t/x] \quad (9)$$

Figure 7: Inequational logic

4 FPC considered as a category

Business starts proper here: we now set up an appropriate categorical framework upon which an operational domain-theoretic treatment of recursive types can be carried out. In this section, we show how FPC types-in-context can be viewed as realisable functors. In order to achieve this, we prove operational versions of the Plotkin's uniformity principle (also known as the Plotkin's axiom) and the minimal invariance property.

The first category we consider, called $\mathbf{FPC}_!$, allows us to interpret types-in-context $X_1, \ldots, X_n \vdash \sigma$ as functors $\mathbf{FPC}_!^n \to \mathbf{FPC}_!$, provided type recursion in σ does not occur in contravariant positions (Sections 4.1 and 4.2). The second category, which is constructed out of $\mathbf{FPC}_!$, called $\mathbf{FPC}_!{}^\delta$, allows us to remove this restriction (Section 4.3).

4.1 The category of FPC types

We now give an account of the categorical framework within which our theory is organised. Our approach, largely adapted from Abadi & Fiore (Abadi and Fiore, 1996), turns out to be a convenient option amongst others. We carefully explain this in two stages:

(i) understand the basic type expressions (i.e., type expressions in which type recursion does not occur in contravariant positions) as functors, and then

(ii) consider those built from all possible type constructors.

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

$$\vec{x} : \vec{\sigma} \vdash s \sqsubseteq_\sigma s' \iff \forall t_i \in \mathrm{Exp}_{\sigma_i}(i = 1, \ldots, n). \; (s[\vec{t}/\vec{x}] \sqsubseteq_\sigma s'[\vec{t}/\vec{x}]). \tag{10}$$

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

$$s \sqsubseteq_\Sigma s' \iff (s \Downarrow \top \implies s' \Downarrow \top). \tag{11}$$

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

$$p \sqsubseteq_{\sigma\times\tau} p' \iff (\mathrm{fst}(p) \sqsubseteq_\sigma \mathrm{fst}(p') \wedge \mathrm{snd}(p) \sqsubseteq_\tau \mathrm{snd}(p')). \tag{12}$$

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

$$\begin{aligned} s \sqsubseteq_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma. \forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma. s' \Downarrow \mathrm{inl}(a') \wedge a \sqsubseteq_\sigma a') \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau. s' \Downarrow \mathrm{inr}(b') \wedge b \sqsubseteq_\tau b'). \end{aligned} \tag{13}$$

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

$$\begin{aligned} t \sqsubseteq_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s'. t' \Downarrow \mathrm{up}(s') \wedge s \sqsubseteq_\sigma s'). \end{aligned} \tag{14}$$

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

$$t \sqsubseteq_{\mu X.\sigma} t' \iff \mathrm{unfold}(t) \sqsubseteq_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t'). \tag{15}$$

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

$$f \sqsubseteq_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma. (f(t) \sqsubseteq_\tau f'(t)). \tag{16}$$

Figure 8: Extensionality properties

The objects of the category **FPC** are the closed FPC types (i.e., type expressions with no free variables) 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 of **FPC** whose morphisms are the strict **FPC**-morphisms.

We make use of the following notations:

$\vec{\sigma}$	for a sequence of closed types $\sigma_1, \ldots, \sigma_n$;
$\vec{t}$	for a sequence of closed terms $t_1, \ldots, t_n$;
$\vec{X}$	for a sequence of type variables $X_1, \ldots, X_n$;
$\vec{x}$	for a sequence of term variables $x_1, \ldots, x_n$;
$\vec{\sigma}/\vec{X}$	for the substitutions $\sigma_1/X_1, \ldots, \sigma_n/X_n$;
$\vec{t}/\vec{x}$	for the substitutions $t_1/x_1, \ldots, t_n/x_n$;
$\vec{f} : \vec{R} \to \vec{S}$	for $f_1 : R_1 \to S_1, \ldots, f_n : R_n \to S_n$.

When we write $\vec{X}, X$, it is understood that X does not appear in $\vec{X}$.

4.2 Basic functors

FPC type expressions are called *basic* if they are generated by the following fragment of the grammar:

$$\mathtt{B} := \mathtt{C} \mid X \mid \mathtt{B} \times \mathtt{B} \mid \mathtt{B} + \mathtt{B} \mid \mathtt{B}_\perp \mid \mu X.\mathtt{B} \mid \mathtt{C} \to \mathtt{B}$$

where $\mathtt{C}$ ranges over closed types. Note that the set of basic type expressions is a proper subset of those type expressions in which recursion does not occur in the contravariant positions. For instance, $\mu X.((X \to \mathtt{C}) \to \mathtt{C}$ is not basic.

A *basic functor* $T : \mathbf{FPC}_!^n \to \mathbf{FPC}_!$ is one realised by:

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

(2) a term-in-context

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

such that for any $\vec{\sigma} \in \mathbf{FPC}_!^n$, it holds that

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

and for any $\vec{\rho}$ and $\vec{\sigma}$, and any $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, it holds that

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

Now we show how basic type expressions define basic functors. We first present the construction and then prove functoriality. For a basic type expression $\mathtt{B}$ in context $\Theta \equiv \vec{X}$, we define, by induction on the structure of types, an associated functor $S_{\Theta \vdash \mathtt{B}} : \mathbf{FPC}_!^n \to \mathbf{FPC}_!$ (or simply S) as follows:

(1) Closed type.
Let $\Theta \vdash \mathtt{C}$.
For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = \mathtt{C}$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v}) = \mathrm{id}_{\mathtt{C}}$.

(2) Type variable.
Let $\Theta \vdash X_i$ $(i \in \{1, \ldots, n\})$.
For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = \sigma_i$.

For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v}) = v_i$.

Let $\Theta \vdash \mathtt{B}_1, \mathtt{B}_2$ be given. Assume that T_j $(j = 1, 2)$ is the basic functor associated with $\Theta \vdash \mathtt{B}_j$, whose morphism part is realised by

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash t_j : \mathtt{B}_j[\vec{R}/\vec{X}] \to \mathtt{B}_j[\vec{S}/\vec{X}].$$

(3) Product type.
For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = T_1(\vec{\sigma}) \times T_2(\vec{\sigma})$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the unique morphism h such that the following diagram

$$\begin{array}{ccc} S(\vec{\rho}) & \xrightarrow{\pi_j} & T_j(\vec{\rho}) \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle T_j(\vec{v})} \\ S(\vec{\sigma}) & \xrightarrow[\pi_j]{} & T_j(\vec{\sigma}) \end{array}$$

commutes $(j = 1, 2)$. The morphism part of S is realised by

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \lambda z.(t_1(\pi_1 z).t_2(\pi_2 z)).$$

(4) Sum type.
For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = T_1(\vec{\sigma}) + T_2(\vec{\sigma})$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the unique morphism h which makes the diagrams

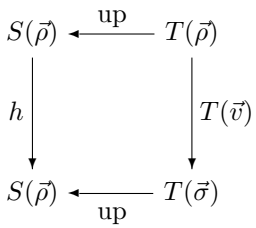
$$\begin{array}{ccc} S(\vec{\rho}) & \xleftarrow{\text{inl}} & T_1(\vec{\rho}) \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle T_1(\vec{v})} \\ S(\vec{\sigma}) & \xleftarrow[\text{inl}]{} & T_1(\vec{\sigma}) \end{array} \qquad \begin{array}{ccc} S(\vec{\rho}) & \xleftarrow{\text{inr}} & T_2(\vec{\rho}) \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle T_2(\vec{v})} \\ S(\vec{\sigma}) & \xleftarrow[\text{inr}]{} & T_2(\vec{\sigma}) \end{array}$$

commute. The morphism part of S is realised by

$$\vec{R}; \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \lambda z.\text{case}(z)\text{of} \begin{cases} \text{inl}(x).\text{inl}(t_1(x)) \\ \\ \text{inr}(y).\text{inr}(t_2(y)) \end{cases}$$

(5) Lifted type.
Let $\Theta \vdash \mathtt{B}$ be given and T its associated basic functor.
For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = (T(\vec{\sigma}))_\perp$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the unique morphism h

which makes the diagram



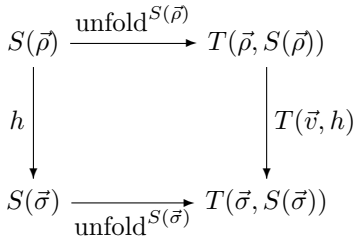
commute. The morphism part of S is realised by

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \lambda z.\text{case}(z) \text{ of } \text{up}(x).\text{up}(t(x))$$

where the morphism part of T is realised by

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

(6) Recursive type.
Let $\Theta, X \vdash \mathtt{B}$ ($X \notin \{X_1, \ldots, X_n\}$) and T the associated basic functor.
For object $\vec{\sigma} \in \mathbf{FPC}^n_!$, define $S(\vec{\sigma}) = \mu X.T(\vec{\sigma}, X)$. We write $T(\vec{\sigma}, S(\vec{\sigma}))$ for $\mathtt{B}[\vec{\sigma}/\vec{X}, S(\vec{\sigma})/X]$.
For the morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the least morphism h that makes the diagram



commute. The morphism part of S is realised by

$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \text{fix}(\lambda g.\text{fold} \circ t[g/f] \circ \text{unfold})$$

where the morphism part of T is realised by

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

(7) Restricted function type.
Let $\Theta \vdash \mathtt{B}$ be given and T the associated functor. Let $\mathtt{C}$ be a closed type.
We want to define the functor S which is associated to $\Theta \vdash \mathtt{C} \to \mathtt{B}$.
For object $\vec{\sigma} \in \mathbf{FPC}^n_!$, define $S(\vec{\sigma}) = \mathtt{C} \to T(\vec{\sigma})$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be

$$\lambda g : (\mathtt{C} \to T(\vec{\rho})) \to (\mathtt{C} \to T(\vec{\sigma})).T(\vec{v}) \circ g.$$

The morphism part of S is realised by

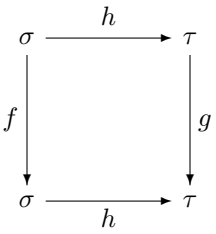
$$\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash \lambda g.t \circ g$$

where the morphism part of T is realised by

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

Functoriality relies on the following two key lemmas.

Lemma 4.1. (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, i.e., $h(\bot_\sigma) = \bot_\tau$, such that the following diagram



commutes, i.e., $g \circ h = h \circ f$. Then it holds that

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

Proof. Using rational-chain completeness, rational continuity, $h \circ f = g \circ h$ in turn, it follows that

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

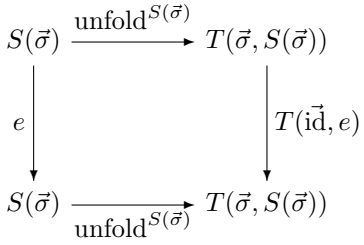
□

Remark 4.2. Notice that this uniformity of least fixed point relies on the rational-chain completeness enjoyed by FPC types and rational continuity of function-type programs. Both these facts which have already been highlighted Section 3 (see Facts (1) and (2)) are fully justified by Theorem 7.6.6 of (Ho, 2006b).

The functoriality of type expressions relies crucially on the following lemma, which is applied to establish preservation of identity morphisms.

Lemma 4.3. (Operational minimal invariance for basic functors)
Let $T : \mathbf{FPC}_!^{n+1} \to \mathbf{FPC}_!$ be a basic functor and $\vec{\sigma}$ any sequence of closed types. Write $S(\vec{\sigma})$ for $\mu X.T(\vec{\sigma}, X)$. Then the least endomorphism $e : S(\vec{\sigma}) \to S(\vec{\sigma})$ for

which the diagram



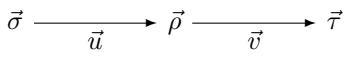
commutes must be $\text{id}_{S(\vec{\sigma})}$.

An operational proof of this is developed in Section 5 below. For the moment, we observe that, although the Scott model is not fully abstract, there is a quick denotational proof: The interpretation of e in the Scott model, denoted by $[\![e]\!]$, makes the corresponding diagram commute. But it is well-known in Domain Theory that the only such endomorphism on $[\![S(\vec{\sigma})]\!]$ must be $\text{id}_{[\![S(\vec{\sigma})]\!]}$. Now, by Lemma 4.4 below, it follows that $e = \text{id}_{S(\vec{\sigma})}$, and hence Lemma 4.3 is proved.

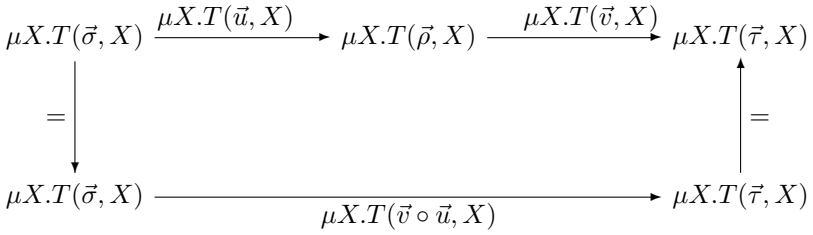
Lemma 4.4. *Let $e : \tau \to \tau$ be a closed term such that $[\![e]\!] = \text{id}_{[\![\tau]\!]}$ in the Scott-model. Then $e = \text{id}_\tau$.*

Proof. Notice that $[\![e]\!] = \text{id}_{[\![\tau]\!]} = [\![\text{id}_\tau]\!]$. By computational adequacy of the Scott model, it follows that $e = \text{id}_\tau$. □

We are now ready to prove functoriality. First we prove the preservation of composition of morphisms. Consider the following composition of morphisms:



It is easy to see that type expressions which are of the following forms: type variables, sum types, product types and lifted types, preserve composition as the corresponding constituent functors do. Thus, it remains to verify that constructors of the form $\mu X.T(\vec{X}, X)$ do preserve the above composition, i.e., the following diagram commutes:



Let us abbreviate $\mu X.T(\vec{\sigma}, X)$ by $S(\vec{\sigma})$ as before.

Consider the following diagram:

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

where $\Phi = \lambda h.\mathrm{fold}^{S(\vec{\tau})} \circ T(\vec{v}, h) \circ \mathrm{unfold}^{S(\vec{\rho})}$ and
$\Psi = \lambda f.\mathrm{fold}^{S(\vec{\tau})} \circ T(\vec{v} \circ \vec{u}, f) \circ \mathrm{unfold}^{S(\vec{\sigma})}$.
The diagram commutes because for any $h : S(\vec{\rho}) \to S(\vec{\tau})$,

$$\begin{array}{ll} & \Psi(h \circ S(\vec{v})) \\ = & \mathrm{fold}^{S(\vec{\tau})} \circ T(\vec{v} \circ \vec{u}, h \circ S(\vec{u})) \circ \mathrm{unfold}^{S(\vec{\sigma})} \\ = & \mathrm{fold}^{S(\vec{\tau})} \circ T(\vec{v}, h) \circ \mathrm{unfold}^{S(\vec{\rho})} \circ \mathrm{fold}^{S(\vec{\rho})} \circ T(\vec{u}, S(\vec{u})) \circ \mathrm{unfold}^{S(\vec{\sigma})} \\ = & \Phi(h) \circ S(\vec{u}) \end{array}$$

Because $- \circ S(\vec{u})$ is a strict program, by Lemma 4.1 it follows that

$$S(\vec{v} \circ \vec{u}) = \mathrm{fix}(\Psi) = \mathrm{fix}(\Phi) \circ S(\vec{u}) = S(\vec{v}) \circ S(\vec{u}).$$

Next we prove the preservation of identity morphisms. By definition $S(\mathrm{id}_{\vec{\sigma}})$ is the least solution e of the equation $e = \mathrm{fold}^{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, e) \circ \mathrm{unfold}^{S(\vec{\sigma})}$. But Lemma 4.3 already asserts that $e = \mathrm{id}_{S(\vec{\sigma})}$.

4.3 Realisable functors

An unrestricted FPC type expression is more problematic.

(1) Once the function-type $\to$ constructor is involved, one needs to separate the covariant and the contravariant variables. For instance, $X \to Y$ consists of X as a contravariant variable and Y as a contravariant variable.

(2) A particular type variable may be covariant and contravariant. 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 category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$. In this section, we do not do so[2] but instead work with a full subcategory, $\mathbf{FPC}_!{}^{\delta}$, of this. 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 morphisms being pairs of $\mathbf{FPC}_!$-morphisms, denoted by $u : \sigma \to \tau$ (or $\langle u^-, u^+ \rangle$), of the form:

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

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

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

The reader should note the use of the following notations.
Notations. In order to avoid excessive use of $+$, $-$ and $\leftrightarrows$, we write

$$\begin{array}{lll} f : R \to S & \text{for} & f^+ : R \leftrightarrows S : f^-, \\ \vec{f} : \vec{R} \to \vec{S} & \text{for} & \vec{f^+} : \vec{R} \leftrightarrows \vec{S} : \vec{f^-}. \end{array}$$

Definition 4.5. A *realisable functor* $T : (\mathbf{FPC}_!{}^\delta)^n \to \mathbf{FPC}_!{}^\delta$ is a functor which is realised by:

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

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

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

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

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

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

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

Remark 4.6. (1) Let $\vec{u}, \vec{v} \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$ be given and suppose that $\vec{u} \sqsubseteq \vec{v}$. Then by monotonicity, any realisable functor 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 functor is *locally continuous* in the sense that $T(\bigsqcup_k \vec{u}_k) = \bigsqcup_k T(\vec{u}_k)$.

Definition 4.7. A type expression is *functional* if it is of the form $\tau_1 \to \tau_2$ for some types-in-context $\Theta \vdash \tau_1, \tau_2$.

We show how FPC type expressions define realisable functors. Again we proceed by induction on the structure of types. The expert reader can choose to skip the details for the non-functional type expressions and read only those of the functional ones. This is because the cases for the non-functional type expressions are similar to those found in the construction of the basic functors, i.e., one merely upgrades these to functors typed $(\mathbf{FPC}_!{}^\delta)^n \to \mathbf{FPC}_!{}^\delta$ by adding the obvious dual arrow when defining the morphism part. However, for the sake of completeness, we do include details of these constructions as well.

(1) Functional type expressions.
Let $\vec{X} \vdash \tau_1, \tau_2$ be given. By induction hypothesis, there are functors T_1 and T_2 associated to these whose morphism parts can be realised by the following terms-in-context ($j = 1, 2$):

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

We now define the functor T associated to $\vec{X} \vdash \tau_1 \to \tau_2$ as follows:
For any $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$, define $T(\vec{\sigma}) = T_1(\vec{\sigma}) \to T_2(\vec{\sigma})$.
For any $\vec{u} \in (\mathbf{FPC}_!^{\ \delta})^n(\vec{\rho}, \vec{\sigma})$, define $T(\vec{u})$ to be v where

$$\begin{aligned} v^- &:= \lambda h : T_1(\vec{\sigma}) \to T_2(\vec{\sigma}).\lambda x : T_1(\vec{\rho}).\Pi_1 T_2(\vec{u}) \circ h \circ \Pi_2 T_1(\vec{u})(x) \\ v^+ &:= \lambda g : T_1(\vec{\rho}) \to T_2(\vec{\rho}).\lambda y : T_1(\vec{\sigma}).\Pi_2 T_2(\vec{u}) \circ g \circ \Pi_1 T_1(\vec{u})(y) \end{aligned}$$

The morphism part of T 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

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

Note that T preserves composition and identities since T_j's do.

(2) Non-functional type expressions.

(a) Type variable.
Let $X_1, \ldots, X_n \vdash X_i$ be given.
For any $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$, define $T(\vec{\sigma}) = \sigma_i$.
For any $\vec{\rho}, \vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$ and any $\vec{u} \in (\mathbf{FPC}_!^{\ \delta})^n(\vec{\rho}, \vec{\sigma})$, define $T(\vec{u})$ to be $u_i : \rho_i \to \sigma_i$. The morphism part of T is realised by

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

Note that T preserves composition and identities.

For the purpose of cases (b) and (c), let us suppose we are given $\vec{X} \vdash \tau_1, \tau_2$. By induction hypothesis, there are associated realisable functors T_j $(j = 1, 2)$ whose morphism parts are realised by

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

(b) Product type.
We want to define the functor T associated to $\vec{X} \vdash \tau_1 \times \tau_2$.
For any $\vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$, define $T(\vec{\sigma}) = T_1(\vec{\sigma}) \times T_2(\vec{\sigma})$.
For any $\vec{\rho}, \vec{\sigma} \in (\mathbf{FPC}_!^{\ \delta})^n$ and any $\vec{u} \in (\mathbf{FPC}_!^{\ \delta})^n(\vec{\rho}, \vec{\sigma})$, define $T(\vec{u})$ to be v where

$$\begin{aligned} v^- &:= \lambda p : T_1(\vec{\sigma}) \times T_2(\vec{\sigma}).(\Pi_1 T_1(\vec{u})(\mathrm{fst}(p)), \Pi_1 T_2(\vec{u})(\mathrm{snd}(p))) \\ v^+ &:= \lambda q : T_1(\vec{\rho}) \times T_2(\vec{\rho}).(\Pi_2 T_1(\vec{u})(\mathrm{fst}(q)), \Pi_2 T_2(\vec{u})(\mathrm{snd}(q))). \end{aligned}$$

The morphism part of T are 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

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

Note that T preserves composition and identities since T_j's do.

(c) Sum type.
We want to define the functor T associated to $\vec{X} \vdash \tau_1 + \tau_2$.
For any $\vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, define $T(\vec{\sigma}) = T_1(\vec{\sigma}) + T_2(\vec{\sigma})$.
For any $\vec{\rho}, \vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$ and any $\vec{u} \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$, define $T(\vec{u})$ to be v where

$$\begin{aligned} v^- &:= \lambda z.T_1(\vec{\sigma}) + T_2(\vec{\sigma}).\mathrm{case}(z)\mathrm{of} \begin{cases} \mathrm{inl}(x).\mathrm{inl}(\Pi_1 T_1(\vec{u})(x)) \\ \\ \mathrm{inr}(y).\mathrm{inr}(\Pi_1 T_2(\vec{u})(y)) \end{cases} \\ v^+ &:= \lambda w.T_1(\vec{\rho}) + T_2(\vec{\rho}).\mathrm{case}(z)\mathrm{of} \begin{cases} \mathrm{inl}(x).\mathrm{inl}(\Pi_2 T_1(\vec{u})(x)) \\ \\ \mathrm{inr}(y).\mathrm{inr}(\Pi_2 T_2(\vec{u})(y)). \end{cases} \end{aligned}$$

The morphism part of T 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

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

Again T preserves composition and identities since T_j's do.

(d) Lifted type.
Let $\vec{X} \vdash \tau$ be given and by induction hypothesis there is an associated realisable functor T. We want to define a realisable functor $T_\perp$ which is associated to $\vec{X} \vdash \tau_\perp$.
For any $\vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, define $T_\perp(\sigma) = T(\vec{\sigma})_\perp$.
For any $\vec{\rho}, \vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, and any $\vec{u} \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$, define $T(\vec{u})$ to be v where

$$\begin{aligned} v^- &:= \lambda z : (T(\vec{\sigma}))_\perp.\mathrm{case}(z)\ \mathrm{of}\ \mathrm{up}(x).\mathrm{up}(\Pi_1 T(\vec{u})(x)) \\ v^+ &:= \lambda w : (T(\vec{\rho}))_\perp.\mathrm{case}(w)\ \mathrm{of}\ \mathrm{up}(x).\mathrm{up}(\Pi_2 T(\vec{u})(x)). \end{aligned}$$

If the morphism part of T is realised by

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

then the morphism part of $T_\perp$ is realised by

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

where

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

Note that $T_\perp$ preserves composition and identities since T does.

(e) Recursive type.
Let $\vec{X}, X \vdash \tau$ be given. The induction hypothesis asserts that there is a realisable functor $T : (\mathbf{FPC}_!{}^\delta)^{n+1} \to \mathbf{FPC}_!{}^\delta$ associated to $\vec{X}, X \vdash \tau$. Since T is realisable, there is a pair of terms-in-context

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

which realises the morphism part of it.
We want to define a realisable functor

$$S : (\mathbf{FPC}_!{}^\delta)^n \to \mathbf{FPC}_!{}^\delta$$

associated to $\vec{X} \vdash \mu X.\tau$.
For any $\vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, define $S(\vec{\sigma}) = \mu X.\tau[\vec{\sigma}/\vec{X}]$.
For any $\vec{\rho}, \vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, and any $\vec{u} \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$, define $S(\vec{u})$ to be the least morphism v such that the following diagram commute:

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

where $i := \langle \mathrm{fold}, \mathrm{unfold} \rangle$.
Equivalently, $S(\vec{u}) := \mathrm{fix}(\Phi)$ where Φ is defined as:

$$\lambda v. i^{-1}_{S(\vec{\sigma})} \circ T(\vec{u}, v) \circ i_{S(\vec{\rho})}.$$

The morphism part of S is realised by:

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

By Lemma 4.8 below, S preserves composition. That S preserves identities is due to *operational minimal invariance*, i.e., Theorem 4.9 – whose proof we shall present in the next section.

Lemma 4.8. *S preserves composition of morphisms.*

Proof. The proof strategy[3] used here is the same as that used for establishing

[3]That least homomorphisms compose can also be found in Lemma 5.3.1 of (Abramsky and

that basic type expressions (as functors) do preserve compositions. To show that S preserves, we must prove that for any morphism pairs $\vec{u} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\rho}, \vec{\sigma})$ and $\vec{v} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\sigma}, \vec{\tau})$, it holds that

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

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

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

$$\begin{array}{lcl} \Psi_2 & : & (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau})) \longrightarrow (S(\vec{\tau}) \to S(\vec{\rho})) \times (S(\vec{\rho}) \to S(\vec{\tau})) \\ \Psi_1 & := & \lambda(c,d).i^{-1}_{S(\vec{\tau})} \circ T(\vec{v} \circ \vec{u}, c, d) \circ i_{S(\vec{\rho})} \end{array}$$

Then the following diagram

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

commutes since for all $a : S(\vec{\tau}) \to S(\vec{\sigma})$ and $b : S(\vec{\sigma}) \to S(\vec{\tau})$, it holds that

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

Moreover, because $\Pi_1 S(\vec{u})$ is strict, the program

$$- \circ S(\vec{u}) := \lambda(a,b).(\Pi_1 S(\vec{u}) \circ a, b \circ \Pi_2 S(\vec{u}))$$

is strict. Therefore, by Plotkin's uniformity Lemma 4.1, we have

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

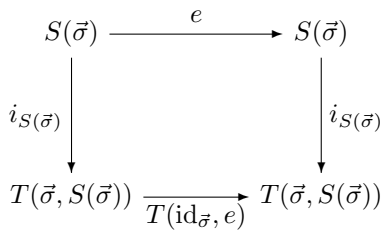
i.e., $S(\vec{v}) \circ S(\vec{u}) = S(\vec{v} \circ \vec{u})$. □

Theorem 4.9. (Operational minimal invariance for realisable functors)
Let $T : (\mathbf{FPC}_!{}^{\delta})^{n+1} \to \mathbf{FPC}_!{}^{\delta}$ be a realisable functor and $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$. As usual, we write $S(\vec{\sigma})$ for $\mu X.T(\vec{\sigma}, X)$. Then the least $\mathbf{FPC}_!{}^{\delta}$-endomorphism

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

Jung, 1994).

for which the following commutes



must be the identity morphism $\langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle$. *Moreover, the identity is the only such endomorphism. Consequently,* S *preserves identity morphisms, i.e.,*

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

5 Operational minimal invariance

In this section, we give an operational proof of the minimal invariance theorem stated in the previous section. For this purpose, we do not assume that type expressions preserve identity morphisms. However, we have seen that they do preserve composition of morphisms and for convenience we call these categorical gadgets that arises out of FPC type expressions *realisable semi-functors* (because they just fall short of being identity-preserving).

More precisely, given a type-in-context $\Theta \vdash \tau$, the operator $T_{\Theta \vdash \tau}$ as defined in the previous section is called the *realisable semi-functor associated to* $\Theta \vdash \tau$. Notice that such a definition serves a very transient purpose since we shall no longer need it once the operational minimal invariance theorem is established by the end of this section.

5.1 Twin morphisms

First, we need the following definition:

Definition 5.1. An $\mathbf{FPC}_!{}^{\delta}$-morphism is said to be *twin* if it is of the form

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

i.e., $u^- = u^+ = u$.

The following lemma guarantees that twins are preserved by realisable semi-functors.

Lemma 5.2. *Let* $\vec{X} \vdash \tau$ *be a type-in-context and* $T_{\vec{X} \vdash \tau}$ *as defined in the construction. Then for any* $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$ *and for any sequence of twin morphisms* $\vec{u} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\sigma}, \vec{\sigma})$ *(i.e.,* $u_i : \sigma_i \leftrightarrows \sigma_i : u_i$ *(*$i = 1, \ldots, n$*)), the morphism* $T(\vec{u})$ *is again twin.*

In particular, if $T : (\mathbf{FPC}_!{}^{\delta})^{n+1} \to \mathbf{FPC}_!{}^{\delta}$ *is a realisable semi-functor and* $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$ *and* $S(\vec{\sigma})$ *is, as usual,* $\mu X.T(\vec{\sigma}, X)$*, then the least* $\mathbf{FPC}_!{}^{\delta}$*-endomorphism*

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

for which the following commutes

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

is twin.

Proof. By induction on the structure of $\vec{X} \vdash \sigma$.
The only interesting case is the recursive type $\vec{X} \vdash \mu X.\tau$ which we prove below. Let $\vec{X}, X \vdash \sigma$ be given. We want to prove that for every twin morphism $\vec{u} \in (\mathbf{FPC}_!{}^{\delta})^n(\vec{\sigma}, \vec{\sigma})$, it holds that $S_{\vec{X} \vdash \mu X.\tau}(\vec{u})$ is again twin. By definition, $S_{\vec{X} \vdash \mu X.\tau}(\vec{u})$ is the least $t : S(\vec{\sigma}) \to S(\vec{\sigma})$ such that the diagram

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

commutes. Here we denote $\langle \mathrm{fold}, \mathrm{unfold} \rangle$ by i. Let $\phi := \lambda t.i^{-1} \circ T(\vec{u}, t) \circ i$. Then on 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 $(\bot, \bot)$. For the inductive step, it follows from the two induction hypotheses that $\phi^{(n+1)}(\bot, \bot) = i^{-1} \circ T(\vec{u}, \phi^{(n)}(\bot, \bot)) \circ i$ must be twin. Finally, by taking the first and second projections, one easily has that $\mathrm{fix}(\phi)$ is twin and the proof is complete. □

The above lemma ensures that the action of realisable semi-functors on identity morphisms in the diagonal category $\mathbf{FPC}_!{}^{\delta}$ (i.e., pair of identities on closed types) is again a pair of equal morphisms, i.e., twin.

Although we have yet shown that realisable semi-functors preserve identity morphisms, we do have the following result.

Lemma 5.3. *For every type-in-context $\Theta \vdash \tau$, the realisable semi-functor $T_{\Theta \vdash \tau}$ associated to it satisfies the following property:*
For every sequence of closed types $\vec{\sigma}$, it holds that

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

Proof. By a straightforward induction on the structure of $\Theta \vdash \sigma$. □

5.2 Canonical unfolding of FPC closed types

We introduce here a technical jargon that will enable a fluent discourse later.

Definition 5.4. Recall that a type-in-context of the form $\Theta \vdash \mu X.\sigma$ is termed as *recursive*. If there is no confusion, we conveniently drop the context Θ and say that the type expression $\mu X.\sigma$ is recursive. A type-in-context (or simply type expression) $\Theta \vdash \sigma$ is *μ-free* if σ does not contain any recursive sub-expressions.

Examples 5.5. 1. $Y \vdash \mu X.X + Y$ and $\emptyset \vdash \mu Y.\mu Z.Y \to Z$ are recursive type expressions while $Y \vdash (\mu X.X + Y) \to (\mu Y.\mu Z.Y \to Z)$ is not.

2. $X, Y, Z \vdash X + (Y \to Z)$ and $X, Y, Z \vdash (X \to Y) \to Z$ are μ-free while $Y \vdash \mu X.X + Y$ and $X \vdash X \times \mu Y.(X + Y)$ are not.

Definition 5.6. A recursive subexpression $\Theta \vdash \mu X.\sigma$ of a type expression $\Theta, \vec{X} \vdash \tau$ is said to be *recursively maximal* in τ if it is not a subexpression of any recursive subexpression of $\Theta, \vec{X} \vdash \tau$.

Example 5.7. Consider the type expression $\tau := X, Y \vdash (\mu U.(U \times X)) \to (\mu Z.\mu W.Y \to (Z + W))$. Then both $\mu U.(U \times X)$ and $\mu Z.\mu W.Y \to (Z + W)$ are recursively maximal in τ while $\mu W.Y \to (Z + W)$ is not.

For a closed FPC type τ_0, we perform the so-called *canonical unfolding* which is a certain procedure of generating some new closed types associated to it. We describe this unfolding below:

1. (i) If τ_0 is recursive, i.e., it is of the form $\mu X_0.\sigma$, then define $\tau_1 := \sigma[\tau_0/X_0]$. Note that when we replace in every occurrence of X_0 the symbol τ_0, we do not replace τ_0 by $\mu X_0.\sigma$ any more. Thus once we have completed the replacement, we do not look into variable X_0 in τ_0 again.

   (ii) Otherwise τ_0 is not recursive. Then there exists a unique μ-free type-in-context $X_1, \ldots, X_n \vdash \rho$ where X_i's are the bound type variables of subexpressions $\mu X_i.\sigma_i$ recursively maximal in τ_0 such that

   $$\tau_0 \equiv \rho[\tau_1/X_1, \ldots, \tau_n/X_n]$$

   where each τ_i stands for $\mu X_i.\sigma_i$. As in (i), once the type variables have been substituted for the symbols τ_i's, the type variables X_i's are not looked into any more.

2. For each τ_i's $(i = 1, \ldots, n)$, stop the procedure if there are no more type variables. Otherwise, repeat for each τ_i's the above steps.

Definition 5.8. For any given closed FPC type τ_0, we call all the τ_j's generated from it via the above procedure its *associates*.

It is time for an example:

Example 5.9. Consider the $\tau_0 := (\mu U.\mu V.(U \times V)) \to (\mu X.X + \mu Y.(Y + X))$.

(1) The μ-free type in context is $A, B \vdash A \to B$.
Define $\tau_1 = \mu U.\mu V.(U \times V)$ and $\tau_2 := \mu X.X + \mu Y.(Y + X)$.
So $\tau_0 = \tau_1 \to \tau_2$.

(2) For τ_1, define $\tau_3 := \mu V.(\tau_1 \times V)$.

(3) For τ_3, define $\tau_4 := \tau_1 \times \tau_3$.

(4) For τ_2, define $\tau_5 := \tau_2 + \mu Y.(Y + \tau_2)$.

(5) For τ_5, define $\tau_6 := \mu Y.(Y + \tau_2)$.

(6) For τ_6, define $\tau_7 := \tau_6 + \tau_2$.

As long as the type variables are not 'exhausted', we can record the procedure by associating (denoted by $\rightsquigarrow$) a closed type with either (a) its single unfolding (if it is a recursive closed type) or (b) its first μ-free decomposition (if it is not μ-free). This is presented by the following system of 'equations':

$$\begin{aligned}
\tau_0 &\rightsquigarrow (\tau_1 \to \tau_2) \\
\tau_1 &\rightsquigarrow \tau_3 \\
\tau_3 &\rightsquigarrow \tau_4 \\
\tau_4 &\rightsquigarrow (\tau_1 \times \tau_3) \\
\tau_2 &\rightsquigarrow \tau_5 \\
\tau_5 &\rightsquigarrow (\tau_2 + \tau_6) \\
\tau_6 &\rightsquigarrow \tau_7 \\
\tau_7 &\rightsquigarrow (\tau_6 + \tau_2)
\end{aligned}$$

We now establish a useful lemma.

Lemma 5.10. *Let τ_0 be a closed FPC type. Then there is a unique (up to renaming) set of associates of τ_0, denoted by τ_j's ($j = 1, 2, \ldots, m$). Moreover, the system of 'equations' presented by these associates has a finite number of 'equations', each of which is in exactly one of the following forms:*

$$\tau_j \equiv \mu X_j.\sigma_j \rightsquigarrow \sigma_j[\tau_j/X_j] \equiv \tau_{j'}$$

arising from the single unfolding of a recursive type τ_j, or

$$\tau_j \rightsquigarrow \rho[\vec{\tau_{jk}}/\vec{X_k}]$$

for some unique μ-free type-in-context ρ and τ_{jk}'s are all recursive types, where τ_j is not recursive and $\rightsquigarrow$ is actually a syntactic equality.

Proof. By a simple induction on the number of type variables X_i's appearing in a closed FPC type, i.e., the number of μX_i's appearing in it. □

5.3 Canonical pre-deflations and deflations

In this subsection, we define two type-indexed families of endofunctions instrumental in the operational proof of the minimal invariance property.

Definition 5.11. 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 (resp. deflationary) structure* on a closed FPC type σ is a rational chain id^σ_n of idempotent pre-deflations (resp. deflations) with $\bigsqcup_n \mathrm{id}^\sigma_n = \mathrm{id}_\sigma$.

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

In what follows, 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. Subsection 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:

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

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

Let $S : \mathcal{C}^\delta \to \mathcal{C}^\delta$ be the realisable functor associated to the type-in-context $X \vdash \sigma$. By Lemma 5.2, we may abuse notation by writing $\Pi_2 S(\mathrm{e}^{\mu X.\sigma}(n), \mathrm{e}^{\mu X.\sigma}(n))$ as $S(\mathrm{e}^{\mu X.\sigma}(n))$. Define

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

Then $\mathrm{e}^{\mu X.\sigma}$ satisfies the following equations:

$$\begin{array}{rcl}
\mathrm{e}^{\mu X.\sigma}(0) & = & \perp_{\mu X.\sigma \to \mu X.\sigma} \\
\mathrm{e}^{\mu X.\sigma}(n+1) & = & \mathrm{fold} \circ S(\mathrm{e}^{\mu X.\sigma}(n)) \circ \mathrm{unfold}.
\end{array}$$

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:

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

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

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

At the first glance, it looks as if the definition of $\mathrm{d}^{\mu X.\sigma}$ is not recursive one, i.e., in the sense that it is of the form $\mathrm{fix}(t)$ for some term t since it involves the type-index $\sigma[\mu X.\sigma/X]$ in the right hand term. However, by finitely unfolding $\mu X.\sigma$ in obtaining its (unique) finite set of associates as in Lemma 5.10, it turns out that $\mathrm{d}^{\mu X.\sigma}$ can indeed be defined via a finite system of mutual recursions. We make this precise here:

Lemma 5.12. *Let $\tau_0 \equiv \mu X.\sigma$ be a given recursive closed type. Then the above $\mathrm{d}^{\mu X.\sigma}(n)$ can be defined by a finite system of mutual recursion, where the first*

equation is given by

$$\mathrm{d}^{\tau_0}(n) = \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\tau_1}(n) \circ \mathrm{unfold}, \qquad \tau_1 \equiv \sigma[\tau_0/X]$$

and the remaining equations, each indexed by the associates of τ_0 obtained via the canonical unfolding, are either of the form

$$\mathrm{d}^{\tau_j}(n) = \text{if } n > 0 \text{ then } \mathrm{fold} \circ \mathrm{d}^{\tau_j[\sigma_j/X_j]}(n-1) \circ \mathrm{unfold}$$

arising from unfolding of a recursive type $\tau_j \equiv \mu X_j.\sigma_j$, or

$$\mathrm{d}^{\tau_j}(n) = \text{if } n > 0 \text{ then } \rho[\mathrm{d}^{\vec{\tau}_{jk}}(n)]$$

for a unique μ-free type-in-context ρ so that τ_{jk}'s are recursive and are associates of τ_0, this case arising from a non-recursive τ_j.

Proof. This is a direct consequence of Lemma 5.10 and the definition of $\mathrm{d}^{\mu X.\sigma}$. □

At this juncture, an example should serve good pedagogical purpose:

Example 5.13. Let $\tau_0 := (\mu U.\mu V.(U \times V)) \to (\mu X.X + \mu Y.(Y + X))$. We write down the following system of equations which arise from the definition of d^{τ_0} as follows:

$$\begin{aligned}
\mathrm{d}^{\tau_0}(n) &= \lambda f.\mathrm{d}^{\tau_2}(n) \circ f \circ \mathrm{d}^{\tau_1}(n) \\
\mathrm{d}^{\tau_1}(n) &= \mathrm{fold} \circ \mathrm{d}^{\tau_3}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\tau_3}(n) &= \mathrm{fold} \circ \mathrm{d}^{\tau_4}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\tau_4}(n) &= (\mathrm{d}^{\tau_1}(n), \mathrm{d}^{\tau_3}(n)) \\
\mathrm{d}^{\tau_2}(n) &= \mathrm{fold} \circ \mathrm{d}^{\tau_5}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\tau_5}(n) &= \mathrm{case}(z) \text{ of } \mathrm{inl}(x).\mathrm{d}^{\tau_2}(n)(x) \text{ or } \mathrm{inr}(y).\mathrm{d}^{\tau_6}(n)(y) \\
\mathrm{d}^{\tau_6}(n) &= \mathrm{fold} \circ \mathrm{d}^{\tau_7}(n-1) \circ \mathrm{unfold} \\
\mathrm{d}^{\tau_7}(n) &= \mathrm{case}(z) \text{ of } \mathrm{inl}(x).\mathrm{d}^{\tau_6}(n)(x) \text{ or } \mathrm{inr}(y).\mathrm{d}^{\tau_2}(n)(y)
\end{aligned}$$

Proposition 5.14. *For any closed recursive type $\mu X.\sigma$, it holds that*

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

Proof. 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 5.15. *For every $n \in \overline{\omega}$, for every closed FPC types σ, it holds that*

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

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

Proof. For the base case $n = 0$, we proceed by induction on the structure of closed types as follows. For convenience, we just verify the case for sum types as follows:

(Sum type) $\sigma + \tau$:
By definition,

$$\begin{array}{lll} \mathrm{d}^{\sigma+\tau}(0)(z) & = \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\mathrm{d}^{\sigma}(0)(x)\ \mathrm{or}\ \mathrm{inr}(y).\mathrm{d}^{\tau}(0)(y) & (\text{defn. of } \mathrm{d}^{\sigma+\tau}) \\ & \sqsubseteq \mathrm{case}(z)\ \mathrm{of}\ \mathrm{inl}(x).\mathrm{e}^{\sigma}(0)(x)\ \mathrm{or}\ \mathrm{inr}(y).\mathrm{e}^{\tau}(0)(y) & (\text{Ind. hyp. \& Fig. 5 (6)}) \\ & = \mathrm{e}^{\sigma+\tau}(0) & (\text{defn. of } \mathrm{e}^{\sigma+\tau}) \\ & \sqsubseteq \mathrm{id}_{\sigma+\tau} & (\text{Ind. hyp.}). \end{array}$$

The other non-recursive types are proven similarly relying on the extensionality properties.
Now we focus on the recursive types:
(Recursive type) $\mu X.\sigma$:
From the recursive clauses in the definitions of $\mathrm{d}^{\mu X.\sigma}$ and $\mathrm{e}^{\mu X.\sigma}$, 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.
Assuming that there is a $k \in \mathbb{N}$ such that for all $n \leq k$, the following always holds:

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

We want to show that

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

Again, we proceed by induction on the structure of types.
We show one case of non-recursive type, e.g., the product type. For this purpose, we assume $\sigma = \sigma_1 \times \sigma_2$.
Note that $\mathrm{d}^{\sigma}(k+1) = (\mathrm{d}^{\sigma_1}(k+1), \mathrm{d}^{\sigma_2}(k+1))$. By induction hypotheses, since $\mathrm{d}^{\sigma_1}(k+1) \sqsubseteq \mathrm{e}^{\sigma_1}(k+1) \sqsubseteq \mathrm{id}_{\sigma_1}$ and $\mathrm{d}^{\sigma_2}(k+1) \sqsubseteq \mathrm{e}^{\sigma_2}(k+1) \sqsubseteq \mathrm{id}_{\sigma_2}$, it follows that

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

and similarly,

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

The rest of the non-recursive types are just as easy and thus omitted.

We now turn to the recursive type. For this case, we suppose that $\sigma = \mu X.\tau$ for some type-in-context $X \vdash \tau$. To proceed with the proof by induction on the structure of τ.
Case I(1): Type variable.
This case concerns $\tau = X$.

The proof is straightforward as follows:

$$\begin{array}{lll} \mathrm{d}^{\mu X.X}(k+1) & = \mathrm{fold} \circ \mathrm{d}^{\mu X.X}(k) \circ \mathrm{unfold} & (\text{defn. of } \mathrm{d}^{\mu X.X}) \\ & \sqsubseteq \mathrm{fold} \circ \mathrm{e}^{\mu X.X}(k) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\ & = \mathrm{e}^{\mu X.X}(k+1) & (\text{defn. of } \mathrm{e}^{\mu X.X}) \end{array}$$

Case I(2): Product type.
This case concerns $\tau = \tau_1 \times \tau_2$.
The proof proceeds as follows:

$$\begin{array}{lll} \mathrm{d}^{\mu X.\tau}(k+1) & = \mathrm{d}^{\mu X.\tau_1 \times \tau_2}(k+1) & \\ & = \mathrm{fold} \circ \mathrm{d}^{(\tau_1 \times \tau_2)[\tau/X]}(k) \circ \mathrm{unfold} & (\text{defn. of } \mathrm{d}^{\mu X.\tau_1 \times \tau_2}) \\ & = \mathrm{fold} \circ (\mathrm{d}^{\tau_1[\tau/X]}(k), \mathrm{d}^{\tau_2[\tau/X]}(k)) \circ \mathrm{unfold} & (\text{defn. of } \mathrm{d}^{(\tau_1 \times \tau_2)[\tau/X]}) \\ & \sqsubseteq \mathrm{fold} \circ (\mathrm{e}^{\tau_1[\tau/X]}(k), \mathrm{e}^{\tau_2[\tau/X]}(k)) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\ & = \mathrm{fold} \circ \mathrm{e}^{(\tau_1 \times \tau_2)[\tau/X]}(k) \circ \mathrm{unfold} & (\text{defn. of } \mathrm{e}^{(\tau_1 \times \tau_2)[\tau/X]}) \\ & = \mathrm{e}^{\mu X.\tau}(k+1) & \end{array}$$

So similarly, one can easily establish that the statement holds for the rest of the non-recursive cases.
Case II: Recursive type.
This case concerns $\tau = \mu Y.\rho$. Without loss of generality, it can be assumed that ρ is not recursive. (Otherwise, apply the same reasoning that follows 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 if we can establish

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

where $\tau_0 := \mu X.\mu Y.\rho$.

To this end, we first prove that

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

Then either $X, Y \vdash \rho$ is μ-free or it is not.
Assume that $X, Y \vdash \rho$ is μ-free. Define $\tau_1 := \mu Y.\rho[\tau_0/X]$ and $\tau_2 := \rho[\tau_0/X, \tau_1/X]$. Since $X, Y \vdash \rho$ is μ-free, it follows that

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

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 $R := R_{X,Y \vdash \rho}$. By monotonicity of realisable semi-functors and extensionality properties, it follows from $\mathrm{d}^{\tau_0}(j) \sqsubseteq \mathrm{d}^{\tau_0}(k)$ $(j = 0, 1, \ldots, k-2)$ that the above term must be 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,

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

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

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

It now remains to settle the case when $X, Y \vdash \rho$ is not μ-free. Since we have assumed that ρ is not recursive, by Lemma 5.12 there exist a unique μ-free type expression $\vec{X}, X, Y \vdash \delta$ (with corresponding realisable functor denoted by D) and recursive types $\alpha_i := \mu X_i.\beta_i$ (where each β_j may or may not contain τ_0, τ_1 as a subexpression) such that

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

By induction hypothesis, we may assume that $\mathrm{d}^{\alpha_j}(k-1) \sqsubseteq \mathrm{id}_{\alpha_j}$ for all $j = 1, \ldots, n$. Since D is monotone and $D_{\vec{X},X,Y \vdash \rho}(\mathrm{id}_{\alpha_1}, \ldots, \mathrm{id}_{\alpha_n}, X, Y) = R_{X,Y \vdash \rho}(X, Y)$, it holds that

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

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

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

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

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

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

where the fourth inequality holds by virtue of Lemma 5.3. □

5.4 Compilation and canonical deflationary structure

The focus of this subsection is to prove that the family d^σ (defined in Subsection 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 $n < \omega$,

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 prove that

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

As a consequence of this theorem and the preceding Theorem 5.15, we have that

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

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

The following proposition is easy to establish:

Lemma 5.16. *For any closed FPC type σ and for each $n \in \overline{\omega}$ and $n < \omega$, the following holds for $\mathrm{d}^{\sigma}(n)$:*

1. *is idempotent,*
2. *$\sqsubseteq \mathrm{id}_{\sigma}$ and*
3. *has finite image modulo contextual equivalence.*

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

Proof. Note that (2) is a consequence of Theorem 5.15. (1) and (3) can be established by induction on the structure of closed type σ. In particular, (1) is straightforward. For (3), the interesting bit lies in the recursive type. In order to show that $\mathrm{d}^{\mu X.\sigma}(n)$ has finite image modulo contextual equivalence, one invokes Lemma 5.12 to define $\mathrm{d}^{\mu X.\sigma}(n)$ in terms of the deflations on the associates of $\mu X.\sigma$ (via mutual recursion) and then uses the induction hypothesis that all these associated deflations have finite image 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. These technical consideration first appeared in Birkedal & Harper's work (See Theorem 3.66 of (Birkedal and Harper, 1999)) in their operational proof of the 'syntactic minimal invariance' in the form of a certain compilation relation $\Rightarrow$.

In this section, we define and prove several elementary properties regarding 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 9.

The compilation relation $\Rightarrow$ turns out to be a function.

Proposition 5.17. *If $\Gamma \vdash t : \sigma$, then $\Gamma \vdash t : \sigma \Rightarrow |t|$ for some unique $|t| \in \mathrm{Exp}_{\sigma}(\Gamma)$.*

Proof. By induction on the derivation of $\Gamma \vdash t : \sigma$. □

Lemma 5.18. *If $\Gamma \vdash t : \sigma \Rightarrow |t|$, then $\Gamma \vdash \mathrm{d}^{\sigma}(\infty)(|t|) =_{\sigma} |t|$.*

Proof. By induction on the derivation of $\Gamma \vdash t : \sigma \Rightarrow |t|$.
The cases for ($\Rightarrow$ var), ($\Rightarrow$ pair), ($\Rightarrow$ inl), ($\Rightarrow$ inr), ($\Rightarrow$ abs), ($\Rightarrow$ up) and ($\Rightarrow$ fold) rely on the idempotence of $\mathrm{d}(\infty)$ (cf. Lemma 5.16) without having to invoke the induction hypothesis. We show the case for ($\Rightarrow$ var) here.
Given that $\Gamma \vdash x : \sigma \Rightarrow |x|$. By definition, $|x| = \mathrm{d}^{\sigma}(\infty)(x)$. We are to show that $\Gamma \vdash \mathrm{d}^{\sigma}(\infty)|x| = |x|$. But this follows from the idempotence of $\mathrm{d}^{\sigma}(\infty)$, i.e.,

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 9: Definition of $\Gamma \vdash t : \sigma \Rightarrow |t|$

$\Gamma \vdash \mathrm{d}^\sigma(\infty)|x| =_\sigma \mathrm{d}^\sigma(\mathrm{d}^\sigma(x)) =_\sigma \mathrm{d}^\sigma(x) = |x|$.
The rest of the cases are fairly routine except for the case ($\Rightarrow$ unfold) which we now show.
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 inference rule ($\Rightarrow$ unfold)

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

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

$$\begin{array}{lll} & \Gamma \vdash \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)|\mathrm{unfold}(t)| & \\ \equiv & \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)(\mathrm{unfold}(|t|)) & \\ = & \mathrm{unfold} \circ \mathrm{d}^{\mu X.\sigma}(\infty)(|t|) & (\text{def. of } \mathrm{d}^{\mu X.\sigma(\infty)}) \\ = & \mathrm{unfold}(|t|) & (\text{Ind. hyp.}) \\ = & |\mathrm{unfold}(t)|. & \end{array}$$

□

Lemma 5.19. *If* $\Gamma \vdash t : \sigma \Rightarrow |t|$, *then* $\Gamma \vdash |t| \sqsubseteq_\sigma t$.

Proof. By induction on $\Gamma \vdash t : \sigma \Rightarrow |t|$, using the previous lemma. □

5.5 Compilation of a context

One last technical gadget 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 $|C|$ is given in Figure 10.

Lemma 5.20. *If* $\Gamma \vdash t : \sigma \Rightarrow |t|$ *and* $C[-_\sigma] \in \mathrm{Ctx}(\Gamma)$, *then*

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

Proof. By induction on the structure of $C[-_\sigma]$. □

Lemma 5.21. *Let* $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ *and* $t \in \mathrm{Exp}_\sigma$. *Then*

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

Proof. By induction on the structure of $C[-_\sigma]$. □

5.6 A crucial lemma

Lemma 5.22.

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

Proof. By induction on the derivation of $t \Downarrow v$.

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash S : \sigma + \tau \Rightarrow |S| \quad \Gamma, x : \sigma \vdash T_1 : \rho \Rightarrow |T_1| \quad \Gamma, y : \tau \vdash T_2 : \rho \Rightarrow |T_2|}{\Gamma \vdash \mathrm{case}(S) \text{ of } \mathrm{inl}(x).T_1 \text{ or } \mathrm{inr}(y).T_2 : \rho \Rightarrow \mathrm{case}(|S|) \text{ of } \mathrm{inl}(x).|T_1| \text{ or } \mathrm{inr}(y).|T_2|} (\Rightarrow \text{ case})$$

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

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

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

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

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

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

Figure 10: Definition of $\Gamma \vdash C[-_\sigma] : \tau \Rightarrow |C|[-_\sigma]$

(1) ($\Downarrow$ can): Trivial.

(2) ($\Downarrow$ fst,snd):
Given that $\emptyset \vdash \mathrm{fst}(p) : \sigma \Rightarrow |\mathrm{fst}(p)|$ and $\mathrm{fst}(p) \Downarrow v$. We must show that $\emptyset \vdash |\mathrm{fst}(p)| = |v|$. The premise of the only evaluation rule ($\Downarrow$ fst) which matches $\mathrm{fst}(p) \Downarrow v$ consists of

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

The induction hypothesis asserts that $\emptyset \vdash |p| =_{\sigma\times\tau} |(s,t)|$ and $\emptyset \vdash |s| =_\sigma |v|$. Based on these, one deduces that

$$\begin{array}{lcll}
\emptyset \vdash |\mathrm{fst}(p)| & \equiv & \mathrm{fst}(|p|) & (\text{def. of } |\mathrm{fst}(p)|) \\
& =_\sigma & \mathrm{fst}(|(s,t)|) & (\text{Ind. hyp.}) \\
& =_\sigma & \mathrm{fst}(\mathrm{d}^\sigma(\infty)(|s|), \mathrm{d}^\tau(\infty)(|t|)) & (\text{def. of } |(s,t)|) \\
& =_\sigma & \mathrm{d}^\sigma(\infty)(|s|) & (\beta\text{-rule}) \\
& =_\sigma & |s| & (\text{Lemma 5.18}) \\
& =_\sigma & |v|. & (\text{Ind. hyp.})
\end{array}$$

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

(3) ($\Downarrow$ app):
Given that $\emptyset \vdash s(t) \Rightarrow |s(t)|$ and $s(t) \Downarrow v$. We must show that $\emptyset \vdash |s(t)| =_\tau |v|$. The only derivation of $s(t) \Downarrow v$ is via 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 $\emptyset \vdash |s| =_{\sigma\to\tau} |\lambda x.r|$ and $\emptyset \vdash r[t/x] =_\sigma |v|$. Then the desired result follows from:

$$\begin{array}{lcll}
\emptyset \vdash |s(t)| & \equiv & |s|(|t|) & (\text{def. of } |s(t)|) \\
& =_\tau & |\lambda x.r|(|t|) & (\text{Ind. hyp.}) \\
& \equiv & (\mathrm{d}^{\sigma\to\tau}(\infty)(\lambda x.|r|))(|t|) & (\text{def. of } |\lambda x.r|) \\
& \equiv & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r| \circ \mathrm{d}^\sigma(\infty))(|t|) & (\text{def. of } \mathrm{d}^{\sigma\to\tau}) \\
& =_\tau & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r|)(\mathrm{d}^\sigma(\infty)(|t|)) & \\
& =_\tau & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r|)(|t|) & (\text{Lemma 5.18}) \\
& =_\tau & \mathrm{d}^\tau(\infty)(|r|[|t|/x]) & (\beta\text{-rule}) \\
& =_\tau & \mathrm{d}^\tau(\infty)(|r[t/x]|) & (\text{Lemma 5.20}) \\
& =_\tau & \mathrm{d}^\tau(\infty)(|v|) & (\text{Ind. hyp.}) \\
& =_\tau & |v|. & (\text{Lemma 5.18})
\end{array}$$

(4) ($\Downarrow$ case):
Given that

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

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

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

W.l.o.g., let us assume that the following evaluation rule ($\Downarrow$ case inl) de-

rives 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

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

It then follows that

	$\emptyset \vdash \lvert\mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2\rvert$	
$\equiv$	$\mathrm{case}(\lvert s\rvert)\ \mathrm{of}\ \mathrm{inl}(x).\lvert t_1\rvert\ \mathrm{or}\ \mathrm{inr}(y).\lvert t_2\rvert$	(by def.)
$=_\rho$	$\mathrm{case}(\mathrm{inl}(\lvert t\rvert))\ \mathrm{of}\ \mathrm{inl}(x).\lvert t_1\rvert\ \mathrm{or}\ \mathrm{inr}(y).\lvert t_2\rvert$	(Ind. hyp.)
$=_\rho$	$\lvert t_1\rvert[\lvert t\rvert/x]$	(Kleene equivalence)
$=_\rho$	$\lvert t_1[t/x]\rvert$	(Lemma 5.20)
$=_\rho$	$\lvert v\rvert$.	(Ind. hyp.)

(5) ($\Downarrow$ case up):
Given that $\emptyset \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \Rightarrow |\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t|$ and $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \Downarrow v$. We want to prove that

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

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

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

The induction hypothesis asserts that

$$|s| =_{\sigma_\perp} |\mathrm{up}(t')| \text{ and } |t[t'/x]| =_\rho |v|.$$

The desired result then follows from

	$\emptyset \vdash \lvert\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t\rvert$	
$\equiv$	$\mathrm{case}(\lvert s\rvert)\ \mathrm{of}\ \mathrm{up}(x).\lvert t\rvert$	(def. of $\lvert\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t\rvert$)
$=_\rho$	$\mathrm{case}(\lvert\mathrm{up}(t')\rvert)\ \mathrm{of}\ \mathrm{up}(x).\lvert t\rvert$	(Ind. hyp.)
$=_\rho$	$\mathrm{case}(\mathrm{up}(\lvert t'\rvert))\ \mathrm{of}\ \mathrm{up}(x).\lvert t\rvert$	(def. of $\lvert\mathrm{up}(t')\rvert$)
$=_\rho$	$\lvert t\rvert[\lvert t'\rvert/x]$	(Kleene equivalence)
$=_\rho$	$\lvert t[t'/x]\rvert$	(Lemma 5.20)
$=_\rho$	$\lvert v\rvert$.	(Ind. hyp.)

(6) ($\Downarrow$ unfold):
Given that $\emptyset \vdash \mathrm{unfold}(t) \Rightarrow |\mathrm{unfold}(t)|$ and $\mathrm{unfold}(t) \Downarrow v$. We must show that

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

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

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

The induction hypothesis asserts that

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

The desired result follows from

	$\emptyset \vdash \lvert\mathrm{unfold}(t)\rvert$	
$\equiv$	$\mathrm{unfold}(\lvert t\rvert)$	(def. of $\lvert\mathrm{unfold}(t)\rvert$)
$=_{\sigma[\mu X.\sigma/X]}$	$\mathrm{unfold}(\lvert\mathrm{fold}(s)\rvert)$	(Ind. hyp.)
$\equiv$	$\mathrm{unfold}(\mathrm{d}^{\mu X.\sigma}(\infty)(\mathrm{fold}(\lvert s\rvert)))$	(def. of $\lvert\mathrm{fold}(s)\rvert$)
$=_{\sigma[\mu X.\sigma/X]}$	$\mathrm{unfold} \circ \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)(\lvert s\rvert)$	(Proposition 5.14)
$=_{\sigma[\mu X.\sigma/X]}$	$\mathrm{unfold}(\mathrm{fold}(\lvert s\rvert))$	(Lemma 5.18)
$=_{\sigma[\mu X.\sigma/X]}$	$\lvert s\rvert$	(β-rule)
$=_{\sigma[\mu X.\sigma/X]}$	$\lvert v\rvert$.	(Ind. hyp.)

□

5.7 Proof of functoriality

We are now ready to present an operational proof of Lemma 4.9.

Lemma 5.23. *Let $f, g \in \mathrm{Exp}_{\mu X.\sigma \to \mu X.\sigma}$ be given. Suppose that for all $t \in \mathrm{Exp}_{\sigma[\mu X.\sigma/X]}$ and for all contexts of the form $C[-_{\mu X.\sigma \to \mu X.\sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$ it holds that*

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

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

Proof. By the extensionality property (16), in order to prove that $f \sqsubseteq g$, it suffices to prove that for all $s \in \mathrm{Exp}_{\mu X.\sigma}$, $f(s) \sqsubseteq_{\mu X.\sigma} g(s)$ holds. Let $s \in \mathrm{Exp}_{\mu X.\sigma}$ be given and suppose $C[-_{\mu X.\sigma}] \in \mathrm{Ctx}_\Sigma$ is such that $C[f(s)] \Downarrow \top$. Because of η-rule, 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 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 η-rule and the definition of $\sqsubseteq$, we have that $C[g(s)] \Downarrow \top$, as required. □

Lemma 5.24. *For any type-in-context of the form $X \vdash \sigma$, we have*

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

Proof. By Lemma 5.23, it suffices to show that for any $t \in \mathrm{Exp}_{\sigma[\mu X.\sigma/X]}$ and for any context $C[-_{\mu X.\sigma \to \mu X.\sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$, it holds that

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

Let $C[-_{\mu X.\sigma \to \mu X.\sigma}](\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$ be arbitrary. Since $\mathrm{id}_{\mu X.\sigma}(\mathrm{fold}(t)) =_{\mu X.\sigma} \mathrm{fold}(t)$ holds (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 5.19, it suffices to show that

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

But by Lemma 5.18, it suffices to show that

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

By Lemma 5.22, $C[\text{fold}(t)] \Downarrow \top$ implies that $|C[\text{fold}(t)]| = |\top| =_\Sigma \top$. It then follows that

$$\begin{array}{rcll} C[\text{fold}(t)] \Downarrow \top & \Longrightarrow & |C[\text{fold}(t)]| \Downarrow \top & \text{(Lemma 5.22)} \\ & \Longrightarrow & |C[|\text{fold}(t)|] \Downarrow \top & \text{(Lemma 5.20)} \\ & \Longrightarrow & C[|\text{fold}(t)|] \Downarrow \top. & \text{(Lemma 5.21)} \end{array}$$

which is what we aim to show. □

As remarked in the beginning of Section 5.4, we have by Theorem 5.15 that

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

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

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

Now as a consequence of Lemmata 5.24 and 5.16, $\text{d}^\sigma(\infty) = \text{id}_\sigma$. Thus it follows that

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

This means

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

and thus $\text{id}_{\mu X.\tau}$ is the (only) least endomorphism e on $\mu X.\tau$ which satisfies the equation

$$e = \text{fold} \circ S_{X\vdash\tau}(e) \circ \text{unfold}.$$

Consequently, for the more general case, the least endomorphism e on $S(\vec{\sigma})$ for which the following diagram

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

commutes must be the identity morphism $\langle \text{id}_{S(\vec{\sigma})}, \text{id}_{S(\vec{\sigma})} \rangle$. Thus the proof that all realisable semi-functors preserve identity morphisms is complete and so func-

toriality of type expressions is now established.

6 Operational algebraic compactness

In (Freyd, 1991), P.J. Freyd 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[4]) 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 one to cope well with the mixed-variant functors - making the study of recursive domain equations complete. Now proving that $\mathbf{DCPO}_{\perp !}$ is algebraically compact is no easy feat as one inevitably has to switch to the category of embeddings and projections, together with a bilimit construction. Using the operational machinery developed so far, we shall establish operational algebraic compactness with respect to the class of realisable functors.

In this chapter, 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 $\check{\mathbf{FPC}}_! := \mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$, and show that this is also parametrised algebraically compact. We then briefly compare the two approaches.

The reader should note that we rely on uniformity (cf. Lemma 4.1) in establishing the algebraic compactness results in Sections 6.1 - 6.2. Such a proof technique was probably first done in (Simpson, 1992) for a more general setting of Kleisli-categories.

6.1 Operational algebraic compactness

Theorem 6.1. (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 which realises F. Denote $\mu X.\tau$ by D and $(\mathrm{unfold}, \mathrm{fold})^{\mu X.\tau}$ by i. We claim that (D, i) is an initial F-algebra. For that purpose, 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

[4]If non-strict maps are considered then the identity functor does not have an initial algebra.

begin by defining k to be the least homomorphism for which the diagram

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

commute. In other words, 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, suppose that k' is another morphism which makes the above diagram commute. Then we consider the following diagram:

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

where the programs Φ, Ψ and G are defined as follows.

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

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

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

Note that $\mathrm{fix}(\Phi) = (\mathrm{id}_D, \mathrm{id}_D)$ by Lemma 4.9. Since G is strict, it follows from Lemma 4.1 that

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

Thus, the uniqueness of k is established. □

Theorem 6.2. (Operational algebraic compactness I)
Let $F : \mathbf{FPC}_!^{\ \delta} \to \mathbf{FPC}_!^{\ \delta}$ be a realisable endofunctor. Then every initial F-algebra is bifree, i.e., its inverse is also a final coalgebra.

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

Proof. W.l.o.g., we may consider the initial F-algebra

$$i : F(D) \to D$$

where (D, i) is as defined in the proof of Theorem 6.1. Note that $i^{-1} = (\text{fold}, \text{unfold})^D$ so that

$$i^{-1} : D \to F(D)$$

is an F-coalgebra. Using the arguments similar to those for reestablishing initiality, it is evident that (D, i^{-1}) is a final F-coalgebra. □

Theorem 6.3. (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. 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 $(H(Q), i_Q \circ F(f, H(Q)))$, i.e., $H(f)$ is the unique morphism g for which the diagram

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

commutes. By the universal property of initial algebras, H is a functor and by construction, i is a natural transformation. Moreover, Theorem 6.2 ensures that $(H(P), i_P)$ is a bifree $F(P, \_)$-algebra. □

6.2 Alternative choice of category

The classical theory of recursive domain equations centres 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 implied by Corollary 5.6 of (Fiore and Plotkin, 1994).

Let $\mathbf{F\breve{P}C}_!$ denote the product category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$ where $\mathbf{FPC}_!$ is defined in Section 4.1. The natural question to ask is whether the category $\mathbf{F\breve{P}C}_!$ is algebraically compact. In order that this question makes sense, one has 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 $\mathbf{F\breve{P}C}_!$ is parametrised algebraically compact with respect to $\mathcal{F}$, i.e., for every $\mathcal{F}$-functor $T : (\mathbf{F\breve{P}C}_!)^{n+1} \to \mathbf{F\breve{P}C}_!$, there exists an $\mathcal{F}$-functor $H : (\mathbf{F\breve{P}C}_!)^n \to (\mathbf{F\breve{P}C}_!)$ 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 $T(\vec{\sigma}, -, +) : \mathbf{F\breve{P}C}_! \to \mathbf{F\breve{P}C}_!$.

In the framework of the product category $\mathbf{F\breve{P}C}_!$, it is mandatory 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. We use the following notations:

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

Sometimes, we also use P and Q to denote objects in $\mathbf{F\breve{P}C}_!$, and u for morphisms in $\mathbf{F\breve{P}C}_!$.

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

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

A seemingly reasonable choice is the class of *syntactic functors* (originally used by A. Rohr in his Ph.D. thesis (Rohr, 2002)) which is defined as follows.

A syntactic functor $T : (\mathbf{F\breve{P}C}_!)^n \to \mathbf{FPC}_!$ is a functor which 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 $\vec{\sigma} \in \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!{}^n$, it holds that

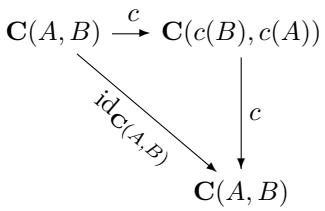
$$T(\vec{\sigma}) = \tau[\vec{\sigma}/\vec{X}]$$

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

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

However, there are some problems with this definition. Firstly, syntactic functors aren't functors of type $\mathbf{F}\breve{\mathbf{P}}\mathbf{C}_! \to \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!$ and so it does not immediately make sense to study parametrised algebraic compactness with respect to this class of functors. The first problem is superficial and can be easily overcome as follows. For a given syntactic functor $F : (\mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{FPC}_!$, there is a standard way of turning it to an endofunctor $\breve{F} : (\mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!$.

Here we help the reader recall what involutory categories are. An *involutory category* is a category $\mathbf{C}$ with an involution $c : \mathbf{C} \to \mathbf{C}^{\mathrm{op}}$, i.e., for all $C \in \mathbf{C}$, it holds that $c^2(C) = C$ and for any object $A,\ B \in \mathbf{C}$, the diagram



commutes in **Set** (c.f. (Fiore and Plotkin, 1994)). The category of involutary categories, **InvCat**, has as objects the involutory categories $(\mathbf{C}, c)$ and as morphisms those functors $F : (\mathbf{C}, c) \to (\mathbf{D}, d)$ between involutory categories such that $F^{\mathrm{op}} \circ c = d \circ F$.

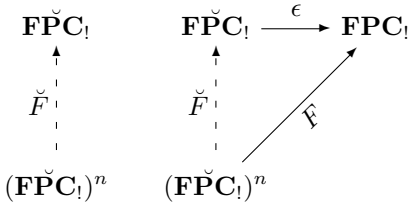
There is a well-known adjunction between the following categories:

$$\mathbf{InvCat} \underset{G_1}{\overset{U}{\rightleftarrows}} \mathbf{Cat}$$

where U is the forgetful functor and $G_1 : \mathbf{C} \mapsto (\breve{\mathbf{C}}, (-)^{\S})$ where

$$\begin{aligned} \breve{\mathbf{C}} &= \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \\ (C^-, C^+)^{\S} &= (C^+, C^-) \\ (f^-, f^+)^{\S} &= (f^+, f^-). \end{aligned}$$

We now exploit this adjunction. Via the adjunction, there corresponds a unique functor $\breve{F} : (\mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!$ such that the following triangle



commutes. The explicit definition of $\breve{F}$ is given by:

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

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

However, if we work with this definition, a serious problem[5] arises. As we shall see in Theorem 6.7, the parametrised initial algebra of such functors are not of the form $\breve{H}$ for some functor H.

This can be fixed by working with our official definition:

Definition 6.4. An n-ary functor $F : (\mathbf{F\breve{P}C}_!)^n \to \mathbf{F\breve{P}C}_!$ 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 $\vec{\sigma} \in \mathbf{F\breve{P}C}_!{}^n$,

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

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

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

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

Example 6.5. (1) Consider the type-in-context $X \vdash X \to X$. The object part part of the syntactic functor $T_{\vec{X} \vdash X \to X}$ is realised by the types-in-context

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

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

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

where

$$\begin{aligned} t^- &:= \lambda g : (S^+ \to S^-).f^- \circ g \circ f^+ \\ t^+ &:= \lambda h : (R^- \to R^+).f^+ \circ h \circ f^-. \end{aligned}$$

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

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

Remark 6.6. Crucially, Example 6.5(2) indicates that if a minimal invariance for $X_2, X_1 \vdash X_1 \to X_2$ were to exist then it cannot be simply given by $X_2 \vdash \mu X_1.(X_1 \to X_2)$. Theorems 6.7 and 6.9 below provide us with a way to calculate the minimal invariance.

Adapting the proof of Freyd's Product Theorem in the operational setting, we establish the following.

Theorem 6.7. (Operational algebraic completeness II)
Every syntactic functor

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

has an initial algebra.

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

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

$$T^- : \mathbf{F\breve{P}C}_! \to \mathbf{FPC}_!^{\mathrm{op}} \text{ and } T^+ : \mathbf{F\breve{P}C}_! \to \mathbf{FPC}_!$$

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 want to construct an initial F-algebra in stages.

(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. One need not look very far for one:

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

For convenience, denote $\mathrm{unfold}^{\mathrm{op}}$ by $f^-_{\sigma^+}$ and $\mu X^-.T^-(X^-, \sigma^+)$ by $F^-(\sigma^+)$. Rewriting gives:

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

To prove that this is an initial $T^-(\_, \sigma^+)$-algebra in $\mathbf{FPC}_!^{\mathrm{op}}$, suppose we are given another $T^-(\_, \sigma^+)$-algebra $a^{\mathrm{op}} : T^-(\tau, \sigma^+) \to \tau$. We need to show that there is a unique morphism $h^{\mathrm{op}} : F^-(\sigma^+) \to \tau$ such that the following diagram commute in $\mathbf{FPC}_!$:

$$\begin{array}{ccc} T^-(F^-(\sigma^+), \sigma^+) & \xleftarrow{\mathrm{unfold}} & F^-(\sigma^+) \\ \uparrow{\scriptstyle T^-(h, \mathrm{id}_{\sigma^+})} & & \uparrow{\scriptstyle h} \\ T^-(\tau, \sigma^+) & \xleftarrow{a} & \tau \end{array}$$

For existence, we define h to be the least fixed point of the program

$$\begin{array}{lcl} \Psi & : & (\tau \to F^-(\sigma^+)) \to (\tau \to F^-(\sigma^+)) \\ \Psi & = & \lambda h.\mathrm{fold} \circ T^-(h, \mathrm{id}_{\sigma^+}) \circ a \end{array}$$

Since $\Psi(h) = \Psi(\mathrm{fix}(\Psi)) = \mathrm{fix}(\Psi) = h$, it follows that h fits into the above commutative diagram. It therefore remains to establish its uniqueness. For that purpose, we consider the program

$$\begin{array}{lcl} \Phi & : & F^-(\sigma^+) \to F^-(\sigma^+) \\ \Phi & = & \lambda k.\mathrm{fold} \circ T^-(k, \mathrm{id}_{\sigma^+}) \circ \mathrm{unfold} \end{array}$$

We now show that $\mathrm{fix}(\Phi) = \mathrm{id}_{F^-(\sigma^+)}$. Note that because T^- is syntactic, so is $T^-(\_, \sigma^+)$. Denote $\mathrm{fix}(\Phi)$ by k. Again appealing to minimal invariance (cf. Lemma 4.9), it follows that $k = \mathrm{id}_{F^-(\sigma^+)}$.

In order to show that h is the unique morphism which fits into the diagram, we suppose that there is another such morphism h'. Consider the following diagram:

$$\begin{array}{ccc} (F^-(\sigma^+) \to F^-(\sigma^+)) & \xrightarrow{- \circ h'} & (\tau \to F^-(\sigma^+)) \\ \Big\downarrow \Phi & & \Big\downarrow \Psi \\ (F^-(\sigma^+) \to F^-(\sigma^+)) & \xrightarrow{- \circ h'} & (\tau \to F^-(\sigma^+)) \end{array}$$

This diagram commutes since for every $k : F^-(\sigma^+) \to F^-(\sigma^+)$ it holds that

$$\begin{array}{lcl} \Phi(k) \circ h' & = & \mathrm{fold} \circ T^-(k, \mathrm{id}_{\sigma^+}) \circ \mathrm{unfold} \circ h' \\ & = & \mathrm{fold} \circ T^-(k, \mathrm{id}_{\sigma^+}) \circ \mathrm{unfold} \circ \mathrm{fold} \circ T^-(h', \mathrm{id}_{\sigma^+}) \circ a \\ & = & \mathrm{fold} \circ T^-(k \circ h', \mathrm{id}_{\sigma^+}) \circ a \\ & = & \Psi(k \circ h') \end{array}$$

Note that $- \circ h'$ is always strict. Invoking Lemma 4.1, we conclude that

$$h = \mathrm{fix}(\Psi) = \mathrm{fix}(\Phi) \circ h' = \mathrm{id}_{F^-(\sigma^+)} \circ h' = h'.$$

Thus we have established that $f^-_{\sigma^+} : T^-(F^-(\sigma^+), \sigma^+) \to F^-(\sigma^+)$ is an initial $T^-(\_, \sigma^+)$-algebra in $\mathbf{FPC}^{\mathrm{op}}_!$.

(2) We now extend F^- to be a functor $\mathbf{FPC}_! \to \mathbf{FPC}^{\mathrm{op}}_!$. For that, we define the morphism part of F^-. Let $w^+ : \rho^+ \to \sigma^+$ be a $\mathbf{FPC}_!$-morphism. Using the initiality of $F^-(\rho^+)$, define $F^-(w^+)$ to be the unique morphism

which makes the following diagram commute in $\mathbf{FPC}_!^{\mathrm{op}}$:

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

Rediagramming a little gives:

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

Notice that the functoriality of F^- derives from the initiality of $F^-(\rho^+)$.

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

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

In a similar way, 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 notations δ^+ for $\mu X^+.G(X^+)$ and d^+ for $\mathrm{fold}^{\mu X^+.G(X^+)}$. Rewriting, we have the initial G-algebra given by

$$d^+ : T^+(F^-(\delta^+),\delta^+) \to \delta^+ \tag{17}$$

We further define $\delta^- := F^-(\delta^+)$ and denote the initial $T^-(\_,\delta^+)$-algebra $f^-_{\delta^+} : T^-(F^-(\delta^+),\delta^+) \to F^-(\delta^+)$ by

$$d^- : T^-(\delta^-,\delta^+) \to \delta^- \tag{18}$$

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

(4) We aim to show that

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

is an initial (T^-,T^+)-algebra. So suppose that we have another (T^-,T^+)-algebra

$$(t^-,t^+) : (T^-(\tau^-,\tau^+),T^+(\tau^-,\tau^+)) \to (\tau^-,\tau^+).$$

We want to show that there is a unique algebra homomorphism from $(d^-,d^+) : (T^-(\delta^-,\delta^+),T^+(\delta^-,\delta^+)) \to (\delta^-,\delta^+)$ to this. In this stage,

we show the existence. By the initiality of $F^-(\tau^+)$, there is a unique morphism $v^- : F^-(\tau^+) \to \tau^-$ in $\mathbf{FPC}_!^{\mathrm{op}}$ so that the following diagram in $\mathbf{FPC}_!^{\mathrm{op}}$:

$$\begin{array}{ccc} T^-(F^-(\tau^+), \tau^+) & \xrightarrow{T^-(v^-, \mathrm{id}_{\tau^+})} & T^-(\tau^-, \tau^+) \\ \Big\downarrow{\scriptstyle f^-_{\tau^+}} & & \Big\downarrow{\scriptstyle t^-} \\ F^-(\tau^+) & \xrightarrow[v^-]{} & \tau^- \end{array}$$

Next we use the initiality of δ^+ to define $u^+ : \delta^+ \to \tau^+$ to be the unique $\mathbf{FPC}_!$-morphism so that the Diagram (1) commutes in $\mathbf{FPC}_!$.

$$\begin{array}{ccc} T^+(F^-(\delta^+), \delta^+) & \xrightarrow{T^+(F^-(u^+), u^+)} & T^+(F^-(\tau^+), \tau^+) \\ & & \Big\downarrow{\scriptstyle T^+(v^-, \mathrm{id}_{\tau^+})} \\ \Big\downarrow{\scriptstyle d^+} & (1) & T^+(\tau^-, \tau^+) \\ & & \Big\downarrow{\scriptstyle t^+} \\ \delta^+ & \xrightarrow[u^+]{} & \tau^+ \end{array}$$

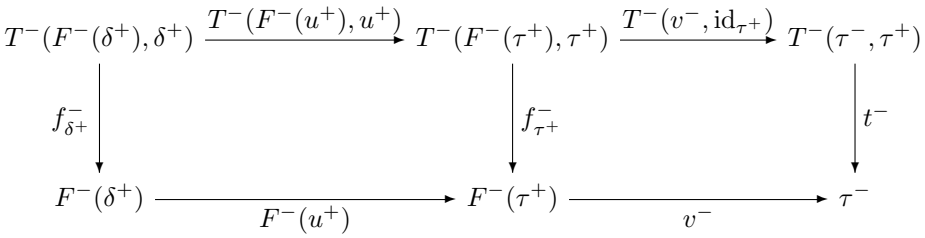
Now define $u^- := v^- \circ F^-(u^+)$ in $\mathbf{FPC}_!^{\mathrm{op}}$ and redraw Diagram (1) (still in $\mathbf{FPC}_!$) as Diagram (2).

$$\begin{array}{ccc} T^+(\delta^-, \delta^+) & \xrightarrow{T^+(u^-, u^+)} & T^+(\tau^-, \tau^+) \\ \Big\downarrow{\scriptstyle d^+} & (2) & \Big\downarrow{\scriptstyle t^+} \\ \delta^+ & \xrightarrow[u^+]{} & \tau^+ \end{array}$$

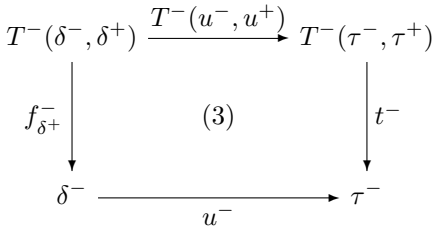
Apply the functor F^- to the morphism $u^+ : \delta^+ \to \tau^+$ so that we get the following diagram in $\mathbf{FPC}_!^{\mathrm{op}}$:

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

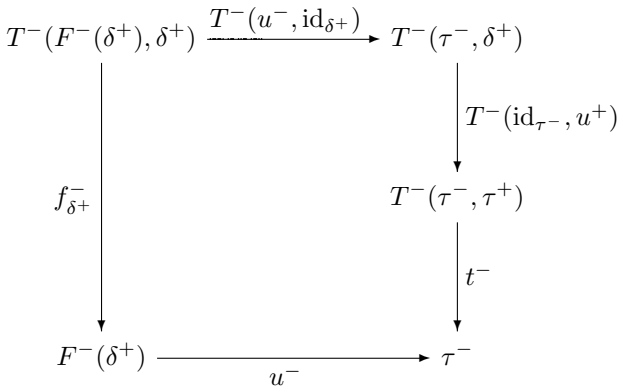
Pasting the unnumbered diagrams, we obtain the following in $\mathbf{FPC}_!^{\mathrm{op}}$:



Finally making use of the definitions of δ^- and u^-, we reduce the outer-quadrangle of the above diagram to the following in $\mathbf{FPC}_!^{\mathrm{op}}$:



(5) Now it remains to show that (u^-, u^+) is unique. Suppose that we are given the Diagrams (2) and (3). Notice that for any $u^+ : \delta^+ \to \tau^+$, there is a unique u^- such that Diagram (3) commute as can be seen by considering the following diagram:



On the other hand, we know from an earlier part of the proof that for any u^+, we may take $u^- = v^- \circ F^-(u^+)$ in $\mathbf{FPC}_!^{\mathrm{op}}$ to obtain Diagram (3). Hence we can conclude that $u^- = v^- \circ F^-(u^+)$ in $\mathbf{FPC}_!^{\mathrm{op}}$. Consequently, putting these into Diagram (2) yields the commutativity of Diagram (1). Now by the initiality of δ^+, we can conclude that u^+ is unique.

□

Theorem 6.8. (Operational algebraic compactness II)
Let $F : \mathbf{F\breve{P}C}_! \to \mathbf{F\breve{P}C}_!$ 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 $\mathbf{F\breve{P}C}_!$ is *operationally algebraically compact* with respect to the class of syntactic functors.

Proof. Walking through the stages of the proof of Theorem 6.7, one can check at each stage that a final coalgebra results when each initial algebra structure map is inverted. Notice this works even for the definition of F^- in stage (2). □

Theorem 6.9. (Operational parametrised algebraic compactness II)
Let $F : (\mathbf{F\breve{P}C}_!)^{n+1} \to \mathbf{F\breve{P}C}_!$ be a syntactic functor. Then there exists a syntactic functor $H : \mathbf{F\breve{P}C}_!{}^n \to \mathbf{F\breve{P}C}_!$ and a natural isomorphism i such that for all sequence of closed types P in $(\mathbf{F\breve{P}C}_!)^n$ we have

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

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

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

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

Proof. For each $P \in (\mathbf{F\breve{P}C}_!)^n$, we have that $F(P, \_) : \mathbf{F\breve{P}C}_! \to \mathbf{F\breve{P}C}_!$ is a syntactic endofunctor. So we can set $(H(P), i_P)$ to be an initial algebra $F(P, \_)$-algebra. To extend the action of H to morphisms, for every $f : P \to Q$ in $(\mathbf{F\breve{P}C}_!)^n$, let $H(f) : H(P) \to H(Q)$ to be the unique $F(P, \_)$-algebra morphism h from $(H(P), i_P)$ to $(H(Q), i_Q \circ F(f, H(Q)))$, i.e., the following commutes:

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

Notice that this unique h is also the least map for which the diagram commutes because initiality is derived from least fixed point construction (cf. Stage (1) of the proof of Theorem 6.7). By the universal property of initial algebras, H is a functor $(\mathbf{F\breve{P}C}_!)^n \to \mathbf{F\breve{P}C}_!$, and, by construction, i is a natural transformation. Moreover, it is clear that H is syntactic. Finally, the bifreeness of $(H(P), i_P)$ derives directly from Theorem 6.8. □

Definition 6.10. In Theorem 6.9, the functor $H : (\mathbf{F\breve{P}C}_!)^n \to \mathbf{F\breve{P}C}_!$ is constructed out of the functor $F : (\mathbf{F\breve{P}C}_!)^{n+1} \to \mathbf{F\breve{P}C}_!$ as a minimal invariant in the last argument pair X^-, X^+. 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:

$$\begin{aligned} H^-(P) &= \mu X^-.T^-(X^-, H^+(P)) \\ H^+(P) &= \mu X^+.T^+(\mu X^-.T^-(X^-, X^+), X^+). \end{aligned}$$

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

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

commutes.

Examples 6.11. The syntactic functor acting as minimal X_1-invariant for $X_1 \to X_2$ is given by

$$H(X_1^-, X_1^+) = (\mu X_2^-.X_1^+ \to X_2^-, \mu X_2^+.X_1^- \to X_2^+).$$

Remark 6.12. In general, the functor $H := \mu F$ is not symmetric. But we expect that symmetry can be achieved in the form of an operational analogue of Fiore's diagonalisation technique (cf. p.124 of (Fiore, 1996)).

6.3 On the choice of categorical frameworks

In this section, we compare the two approaches via 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. We show how this can be done by induction on the structure of $\Theta \vdash \sigma$. We denote the syntactic functor associated to $\Theta \vdash \sigma$ by $F_{\Theta \vdash \sigma}$, or simply F.

(1) Type variable.
Let $\Theta \vdash X_i$ be given. Define the functor $F : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$ as follows.
For object $P \in \mathbf{F\check{P}C}_!^{\,n}$, define $T(P) := P_i$.
For morphism $u \in \mathbf{F\check{P}C}_!^{\,n}(P, Q)$, define $T(u) := u_i$.

Let $\Theta \vdash \sigma_1, \sigma_2$ be given and $F_1, F_2 : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$ be their associated realisable functors. For a given syntactic functor $F : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{F\check{P}C}_!$, we write $F^- : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{FPC}_!^{\mathrm{op}}$ and $F^+ : \mathbf{F\check{P}C}_!^{\,n} \to \mathbf{FPC}_!$ for its two component functors.

(2) Product type.
For object P, define

$$F(P) := (F_1^-(P) \times F_2^-(P), F_1^+(P) \times F_2^+(P)$$

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

$$F(u) := (F_1^-(u) \times F_2^-(u), F_1^+(u) \times F_2^+(u)).$$

where the component morphisms are defined as follows:

$$\begin{array}{rcl} F_1^-(u) \times F_2^-(u) & = & \lambda p.(F_1^-(u)(\mathrm{fst}(p)), F_2^-(u)(\mathrm{snd}(p))) \\ F_1^+(u) \times F_2^+(u) & = & \lambda q.(F_1^2(u)(\mathrm{fst}(q)), F_2^2(u)(\mathrm{snd}(q))). \end{array}$$

(3) Sum type.
For object P, define

$$F(P) := (F_1^-(P) + F_2^-(P), F_1^+(P) + F_2^+(P)$$

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

$$F(u) := (F_1^-(u) + F_2^-(u), F_1^+(u) + F_2^+(u)).$$

where the component morphisms are defined as follows:

$$\begin{array}{rcl} F_1^-(u) + F_2^-(u) & = & \lambda w.\mathrm{case}(w) \text{ of } \begin{cases} \mathrm{inl}(x).\mathrm{inl}(F_1^-(u)(x)) \\ \mathrm{inr}(y).\mathrm{inr}(F_2^-(u)(y)) \end{cases} \\ F_1^+(u) + F_2^+(u) & = & \lambda z.\mathrm{case}(z) \text{ of } \begin{cases} \mathrm{inl}(x).\mathrm{inl}(F_1^-(u)(x)) \\ \mathrm{inr}(y).\mathrm{inr}(F_2^-(u)(y)) \end{cases} \end{array}$$

(4) Function type.
For object $P \in \mathbf{F}\check{\mathbf{P}}\mathbf{C}_!^{\ n}$, define

$$F(P) := (F_1^+(P) \to F_2^-(P), F_1^-(P) \to F_2^+(P))$$

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

$$F(u) := (F_1^+(u) \to F_2^-(u), F_1^-(u) \to F_2^+(u))$$

where the component morphisms are defined as follows:

$$\begin{array}{rcl} F_1^+(u) \to F_2^-(u) & = & \lambda g : F_1^+(Q) \to F_2^-(Q).F_2^-(u) \circ g \circ F_1^+(u) \\ F_1^-(u) \to F_2^+(u) & = & \lambda h : F_1^-(Q) \to F_2^+(Q).F_2^+(u) \circ h \circ F_1^-(u). \end{array}$$

(5) Lifted type.
Given the realisable functor $F_{\Theta \vdash \sigma}$, we want to define $F_{\Theta \vdash \sigma_\perp}$.
For object P, define

$$F_{\Theta \vdash \sigma_\perp}(P) := ((F^-_{\Theta \vdash \sigma}(P))_\perp, (F^+_{\Theta \vdash \sigma}(P))_\perp)$$

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

$$F_{\Theta \vdash \sigma_\perp}(u) := (F^-_\perp(u), F^+_\perp(u))$$

where the component morphisms are defined as follows:

$$\begin{aligned} F^-_\perp(u) &= \lambda w.\mathrm{case}(w)\ \text{of}\ \mathrm{up}(x).\mathrm{up}(F^-_{\Theta\vdash\sigma}(u)(x)) \\ F^+_\perp(u) &= \lambda z.\mathrm{case}(z)\ \text{of}\ \mathrm{up}(x).\mathrm{up}(F^+_{\Theta\vdash\sigma}(u)(x)). \end{aligned}$$

(6) Recursive type.
Let $\Theta, X \vdash \sigma$ be given and F the syntactic functor realising it. Define $F_{\Theta\vdash\mu X.\sigma}$ to be μF as in Definition 6.10.

Notation. The syntactic functor associated to the type-in-context $\Theta \vdash \sigma$ is denoted by $F_{\Theta\vdash\sigma}$.

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

Proposition 6.13. *For every type-in-context $\Theta \vdash \sigma$, the realisable functor $S_{\Theta\vdash\sigma}$ restricts and co-restricts to the syntactic functor $F_{\Theta\vdash\sigma}$, i.e., the diagram*

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

commutes up to natural isomorphism.

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

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

(1) Type variable.
Let $\Theta \vdash X_i$ be given. Define $\eta : F_{\Theta\vdash X_i} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta\vdash X_i}$ 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 induction hypothesis, there are natural isomorphisms

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

for $j = 1, 2$. We write $\eta_j = \langle \eta_j^-, \eta_j^+ \rangle$.

(2) Product type.
We define $\eta : F_{\Theta\vdash\tau_1\times\tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta\vdash\tau_1\times\tau_2}$ 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

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

(3) Sum type.
We define $\eta : F_{\Theta \vdash \tau_1 + \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_1 + \tau_2}$ 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

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

(4) Function type.
We define $\eta : F_{\Theta \vdash \tau_1 \to \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_1 \to \tau_2}$ 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

$$\begin{aligned}(\eta_1^+ \to \eta_2^-) &:= \lambda g.\eta_2^- \circ g \circ \eta_1^+ \\ (\eta_1^- \to \eta_2^+) &:= \lambda h.\eta_2^+ \circ h \circ \eta_1^-.\end{aligned}$$

(5) Lifted type.
Let $\Theta \vdash \tau$ be given. The induction hypothesis asserts that there is a natural isomorphism

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

We define a natural isomorphism

$$\eta_\perp : F_{\Theta \vdash \tau_\perp} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_\perp}$$

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

$$(\eta_\perp)_{\vec{\sigma}} := \mathrm{case}(z)\ \mathrm{of}\ \mathrm{up}(x).\mathrm{up}(\eta(x)).$$

(6) Recursive type.
Let $\Theta, X \vdash \tau$ be given. The induction hypothesis asserts that there is a natural isomorphism

$$\zeta : F_{\Theta, X \vdash \tau} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta, X \vdash \tau}.$$

We define a natural isomorphism

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

as follows. For every $\vec{\sigma} \in (\mathbf{FPC}_!{}^\delta)^n$, define $\eta_{\vec{\sigma}}$ to be the unique map h

which fits into the commutative diagram:

$$\begin{array}{ccccc}
F(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) & & \xrightarrow{\quad i_{\mathrm{Inj}^n(\vec{\sigma})} \quad} & & H \circ \mathrm{Inj}^n(\vec{\sigma}) \\
\Big\downarrow{\scriptstyle F(\mathrm{Inj}^n(\vec{\sigma}), h)} & & & & \Big\downarrow{\scriptstyle h} \\
F \circ \mathrm{Inj}^n(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\zeta_{\vec{\sigma}, S_{\mu X.\tau}(\vec{\sigma})}]{} & \mathrm{Inj} \circ S_{\Theta, X \vdash \tau}(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\langle \mathrm{unfold}, \mathrm{fold} \rangle]{} & \mathrm{Inj} \circ S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})
\end{array}$$

where $H := \mu F$. Note that the existence and uniqueness of h is guaranteed by the initiality of $i_{\mathrm{Inj}^n(\vec{\sigma})} : F(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) \to H \circ \mathrm{Inj}^n(\vec{\sigma})$. Since ζ, i and $\langle \mathrm{unfold}, \mathrm{fold} \rangle$ are natural, so is h. It remains to show that h is an isomorphism. For this purpose, we have to define the inverse of h. Now since $i^{-1}_{\mathrm{Inj}^n(\vec{\sigma})} : H \circ \mathrm{Inj}^n(\vec{\sigma}) \to F(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma}))$ is a final coalgebra and ζ is an isomorphism, there exists a unique g which fits into the following commutative diagram:

$$\begin{array}{ccccc}
F(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) & & \xleftarrow{\quad i^{-1}_{\mathrm{Inj}^n(\vec{\sigma})} \quad} & & H \circ \mathrm{Inj}^n(\vec{\sigma}) \\
\Big\uparrow{\scriptstyle F(\mathrm{Inj}^n(\vec{\sigma}), g)} & & & & \Big\uparrow{\scriptstyle g} \\
F \circ \mathrm{Inj}^n(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xleftarrow[\zeta^{-1}_{\vec{\sigma}, S_{\mu X.\tau}(\vec{\sigma})}]{} & \mathrm{Inj} \circ S_{\Theta, X \vdash \tau}(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xleftarrow[\langle \mathrm{fold}, \mathrm{unfold} \rangle]{} & \mathrm{Inj} \circ S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})
\end{array}$$

We claim that $g \circ h = \mathrm{id}_{H \circ \mathrm{Inj}^n(\vec{\sigma})}$ and $h \circ g = \mathrm{id}_{\mathrm{Inj} \circ S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})}$. To prove the first equation, notice that $g \circ h$ is an $F(\mathrm{Inj}^n(\vec{\sigma}), \_)$-algebra endomorphism on $H \circ \mathrm{Inj}^n(\vec{\sigma})$. Thus by initiality of

$$i_{\mathrm{Inj}^n(\vec{\sigma})} : F(\mathrm{Inj}^n(\vec{\sigma}), H \circ \mathrm{Inj}^n(\vec{\sigma})) \to H \circ \mathrm{Inj}^n(\vec{\sigma}),$$

it must be 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 above two diagrams: unique g which fits into the following commutative diagram:

$$\begin{array}{ccccc}
F \circ \mathrm{Inj}^n(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\zeta_{\vec{\sigma}, S_{\mu X.\tau}(\vec{\sigma})}]{} & \mathrm{Inj} \circ S_{\Theta, X \vdash \tau}(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\langle \mathrm{unfold}, \mathrm{fold} \rangle]{} & \mathrm{Inj} \circ S_{\Theta \vdash \mu X.\tau}(\vec{\sigma}) \\
\Big\downarrow{\scriptstyle F(\mathrm{Inj}^n(\vec{\sigma}), h \circ g)} & & \vdots & & \Big\downarrow{\scriptstyle h \circ g} \\
F \circ \mathrm{Inj}^n(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\zeta_{\vec{\sigma}, S_{\mu X.\tau}(\vec{\sigma})}]{} & \mathrm{Inj} \circ S_{\Theta, X \vdash \tau}(\vec{\sigma}, S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})) & \xrightarrow[\langle \mathrm{unfold}, \mathrm{fold} \rangle]{} & \mathrm{Inj} \circ S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})
\end{array}$$

where the dotted arrow is the morphism

$$\langle S_{\Theta, X \vdash \tau}(\vec{\sigma}, (h \circ g)^-), S_{\Theta, X \vdash \tau}(\vec{\sigma}, (h \circ g)^+) \rangle.$$

So for the second quadrangle, $(h \circ g)^-$ and $(h \circ g)^+$ are both endomorphisms

on $S_{\Theta\vdash\mu X.\tau}(\vec{\sigma})$. By the initiality of

$$\text{fold} : S_{\Theta,X\vdash\tau}(\vec{\sigma}, S_{\Theta\vdash\mu X.\tau}(\vec{\sigma})) \to S_{\Theta\vdash\mu X.\tau}(\vec{\sigma}),$$

we conclude that $h \circ g = \text{id}_{\text{Inj}\circ S_{\Theta\vdash\mu X.\tau}(\vec{\sigma})}$.

□

Within the framework of $\mathbf{F\breve{P}C}_!$, the treatment of recursive types can be described schematically as follows.

(1) Perform a separation of type variables for a given type expression, i.e., into the positive and negative occurrences.

(2) Carry out treatment (i.e. the investigation in question), e.g. calculating the minimal invariance of some syntactic functors.

(3) Perform a diagonalisation to derive the relevant conclusion regarding the original type expression.

In view of Proposition 6.13, 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 which 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 explicitly distinguish 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 little categorical overhead.

However, as a mathematical theory for treating recursive types, the approach via the product category, $\mathbf{F\breve{P}C}_!$, is general and can cope with mathematical notions, such as di-algebras (cf. (Freyd, 1991)), which the diagonal category cannot.

7 The Generic Approximation Lemma

In this section, we study the canonical pre-deflationary structure on the closed FPC types, defined earlier in Section 5.3 and develop, as a consequence of this, a powerful proof technique known as the *Generic Approximation Lemma*[6]. This lemma was first proposed by G. Hutton and J. Gibbons in (Hutton and Gibbons, 2001) in which it was established, via denotational semantics, for *polynomial types* (i.e., types built only from unit, sums and products). In that same reference, the authors have suggested that it is possible to generalise the lemma "to mutually recursive, parameterised, exponential and nested datatypes" (cf. p.4 of (Hutton and Gibbons, 2001)). Here we confirm this by providing a proof based on the operational domain theory we developed in Section 4. Also we use some running examples from (Pitts, 1997) and (Gibbons and Hutton, 2005) to demonstrate the power of the Generic Approximation Lemma as a proof technique for establishing program equivalence, where previously many other more complex techniques had been employed.

[6]This is a generalisation of R. Bird's approximation lemma (Bird, 1998), which in turn generalises the well-known take lemma (Bird and Wadler, 1988).

7.1 The Generic Approximation Lemma

Theorem 7.1. *The rational-chain* $\mathrm{id}_n^\sigma := \mathrm{e}^\sigma(n)$ *defines a non-trivial rational pre-deflationary structure on* σ *for every closed type* σ*.*

Proof. By induction on σ. Here we present only the proof for the case of recursive types. Let S be the functor realising $X \vdash \sigma$. We now prove (1). (Base case) The case where $n = 0$ is trivially true.
(Inductive step) This is justified by the following calculations:

$$\begin{array}{lll} & \mathrm{id}_{n+1}^{\mu X.\sigma} \circ \mathrm{id}_{n+1}^{\mu X.\sigma} & \\ = & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} \circ \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{def. of } \mathrm{id}_{n+1}^{\mu X.\sigma}) \\ = & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (\beta\text{-rule}) \\ = & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma} \circ \mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (S \text{ is a functor.}) \\ = & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\ = & \mathrm{id}_{n+1}^{\mu X.\sigma}. & (\text{def. of } \mathrm{id}_{n+1}^{\mu X.\sigma}) \end{array}$$

For (2), we rely on the monotonicity of S as follows.

$$\begin{array}{lll} & \mathrm{id}_{n+1}^{\mu X.\sigma} & \\ = & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{def. of } \mathrm{id}_{n+1}^{\mu X.\sigma}) \\ \sqsubseteq & \mathrm{fold} \circ S(\mathrm{id}_{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\ = & \mathrm{fold} \circ \mathrm{id}_{\sigma[\mu X.\sigma/X]} \circ \mathrm{unfold} & (S\text{is a functor.}) \\ = & \mathrm{id}_{\mu X.\sigma}. & (\eta\text{-rule}) \end{array}$$

Because $\infty = \infty - 1$, the morphism $k := \mathrm{id}_\infty^{\mu X.\sigma}$ satisfies the recursive equation

$$k \quad = \quad \mathrm{fold} \circ S(k) \circ \mathrm{unfold}.$$

By Lemma 4.9, $\mathrm{id}_{\mu X.\sigma}$ is the least solution of the above equation and thus must be below $\mathrm{id}_\infty^{\mu X.\sigma}$. On the other hand, $\mathrm{id}_\infty^{\mu X.\sigma} = \bigsqcup_n \mathrm{id}_n^{\mu X.\sigma}$ so that $\mathrm{id}_\infty^{\mu X.\sigma} \sqsubseteq \mathrm{id}_{\mu X.\sigma}$. Hence $\mathrm{id}_\infty^{\mu X.\sigma} = \bigsqcup_n \mathrm{id}_n^{\mu X.\sigma} = \mathrm{id}_{\mu X.\sigma}$ and thus (3) holds. □

Notation. We write $x =_n y$ for $\mathrm{id}_n(x) = \mathrm{id}_n(y)$.

Corollary 7.2. (The Generic Approximation Lemma)
Let σ *be a closed type and* $x, y : \sigma$*. Then*

$$x = y \iff \forall n \in \mathbb{N}.(x =_n y).$$

Proof. ($\Longrightarrow$) Trivial.
($\Longleftarrow$) $x = \bigsqcup_n \mathrm{id}_n(x) = \bigsqcup_n \mathrm{id}_n(y) = y$ by Theorem 7.1. □

7.2 Sample applications

In this section, we demonstrate the versatility of the generic approximation lemma by using some running examples of programs taken from (Pitts, 1997) and (Gibbons and Hutton, 2005). For each example, we compare the use of the Generic Approximation Lemma (Corollary 7.2) with an alternative method.

7.2.1 List type and some related notations

Let τ be a closed type. The closed type $[\tau] := \mu\alpha.1 + \tau \times \alpha$ is called the *lazy list type* associated to τ. An element of $[\tau]$ may be thought of as a (finite or infinite) list of elements in τ (which may include $\bot_\tau$).

In the course of our discussion, we make use of the following:

(1) $[\ ] := \text{fold}(\text{inl}(*))$

(2) $\text{cons} : \tau \to [\tau] \to [\tau]$
$\text{cons}\ x\ xs = \text{fold}(\text{inr}(x, xs))$.
We also write $\text{cons}\ x\ xs$ as $(x : xs)$.

(3) Let σ be a closed type.
A program $f : [\tau] \to \sigma$ defined by cases, i.e.,

$$f(l) = \text{case}(l)\ \text{of} \begin{cases} \text{inl}(x).s_1 \\ \\ \text{inr}(y).s_2 \end{cases}$$

is written in `Haskell` style:

$$\begin{array}{lcl} f\ [\ ] & = & s_1 \\ f\ (x : xs) & = & s_2. \end{array}$$

We shall omit from our writing the cases which produce divergence. For instance, the familiar head function $\text{hd} : [\tau] \to \tau$ and tail function $\text{tl} : [\tau] \to [\tau]$ are defined as follows:

$$\begin{array}{lcl} \text{hd}\ (x : xs) & = & x \\ \text{tl}\ (x : xs) & = & xs. \end{array}$$

(4) For programs $f : (\tau \to \tau) \to (\tau \to \tau)$, we write $h := \text{fix}(f)$ as

$$\begin{array}{lcl} h & : & \tau \to \tau \\ h & = & f\ h. \end{array}$$

All the examples covered here only involve the basic type constructors. So in fact one could have, for the sake of these examples, developed just the machineries for basic type expressions.

The following lemma comes handy whenever the Generic Approximation Lemma is applied to list types.

Lemma 7.3. *Let $n > 0$ be a natural number and τ be a closed type. Then the program $\text{id}_n : [\tau] \to [\tau]$ satisfies the following equations:*

$$\begin{array}{rcl} \text{id}_n[\ ] & = & [\ ] \\ \text{id}_n(x : xs) & = & (x : \text{id}_{n-1}(xs)). \end{array}$$

Proof. For the empty list $[\ ]$, we have

$$\begin{array}{ll} & \text{id}_n[\ ] \\ = & \text{fold} \circ (1 + \tau \times \text{id}_{n-1}) \circ \text{unfold}(\text{fold}(\text{inl}(*))) \\ = & \text{fold} \circ (1 + \tau \times \text{id}_{n-1})(\text{inl}(*)) \\ = & \text{fold}(\text{inl}(*)) \\ = & [\ ]. \end{array}$$

For the list $(x : xs)$, we have

$$\begin{array}{ll} & \mathrm{id}_n(x : xs) \\ = & \mathrm{fold} \circ (1 + \tau \times \mathrm{id}_{n-1}) \circ \mathrm{unfold}(\mathrm{fold}(\mathrm{inr}(x, xs))) \\ = & \mathrm{fold} \circ (1 + \tau \times \mathrm{id}_{n-1})(\mathrm{inr}(x, xs)) \\ = & \mathrm{fold}(\mathrm{inr}(x, \mathrm{id}_{n-1}(xs))) \\ = & (x : \mathrm{id}_{n-1}(xs)). \end{array}$$

□

7.2.2 The map-iterate property

We define two familiar functions map and iterate.

$$\begin{array}{l} \mathrm{map} : (\tau \to \tau) \to [\tau] \to [\tau] \\ \mathrm{map}\ f\ [\,] \quad = \quad [\,] \\ \mathrm{map}\ f\ (x : xs) \quad = \quad (f(x) : \mathrm{map}\ f\ xs) \end{array}$$

$$\begin{array}{l} \mathrm{iterate} : (\tau \to \tau) \to \tau \to [\tau] \\ \mathrm{iterate}\ f\ x \quad = \quad (x : \mathrm{iterate}\ f\ f(x)) \end{array}$$

Proposition 7.4. (The map-iterate property)
Let τ be a closed type. For any $f : (\tau \to \tau)$ and any $x : \tau$, it holds that

$$\mathrm{map}\ f\ (\mathrm{iterate}\ f\ x) = \mathrm{iterate}\ f\ f(x).$$

This property had been proven in (Gibbons and Hutton, 2005) using *program fusion*. We reproduce their proof here. For this method, one needs to define the program unfd as follows. Let σ and τ be given closed types. Define

$$\begin{array}{l} \mathrm{unfd} : (\sigma \to \mathtt{Bool}) \to (\sigma \to \tau) \to (\sigma \to \sigma) \to \sigma \to [\tau] \\ \mathrm{unfd}\ p\ h\ t\ x = \mathrm{if}\ p(x)\ \mathrm{then}\ [\,]\ \mathrm{else}\ h(x) : \mathrm{unfd}\ p\ h\ t\ tx \end{array}$$

The unfd function "encapsulates the natural basic pattern of co-recursive definition" (p.9 of (Gibbons and Hutton, 2005)). Because several familiar co-recursive functions on lists can be defined in terms of unfd, one can rely on a universal property (which we describe below) of unfd to generate a powerful proof method whenever such co-recursive programs are involved. For example, if we define

$$\begin{array}{l} \mathrm{F} : \sigma \to \mathtt{Bool} \\ \mathrm{F}\ x = \mathrm{F}\ (:= \mathrm{inl}(\bot)) \end{array}$$

then we can define the program iterate as follows:

$$\mathrm{iterate}\ f := \mathrm{unfd}\ \mathrm{F}\ \mathrm{id}\ f.$$

Likewise, if we define

$$\begin{array}{l} \mathrm{null} : [\tau] \to \mathtt{Bool} \\ \mathrm{null}\ [\,] \quad = \quad \mathrm{T}\ (:= \mathrm{inr}(\bot)) \\ \mathrm{null}\ (x : xs) \quad = \quad \mathrm{F}\ (:= \mathrm{inl}(\bot)) \end{array}$$

then the program map can be defined as follows:

$$\mathrm{map}\ f = \mathrm{unfd}\ \mathrm{null}\ (f \circ \mathrm{hd})\ \mathrm{tl}.$$

The proof method relies on a universal property enjoyed by unfd, which we now

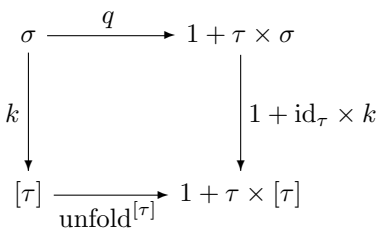
describe. Define $q : \sigma \to 1 + \tau \times \sigma$ by

$$q(x) = \text{if } p(x) \text{ then } \text{inl}(*) \text{ else } \text{inr}(h(x), t(x))$$

and $k : \sigma \to [\tau]$ by

$$k = \text{unfd } p\ h\ t.$$

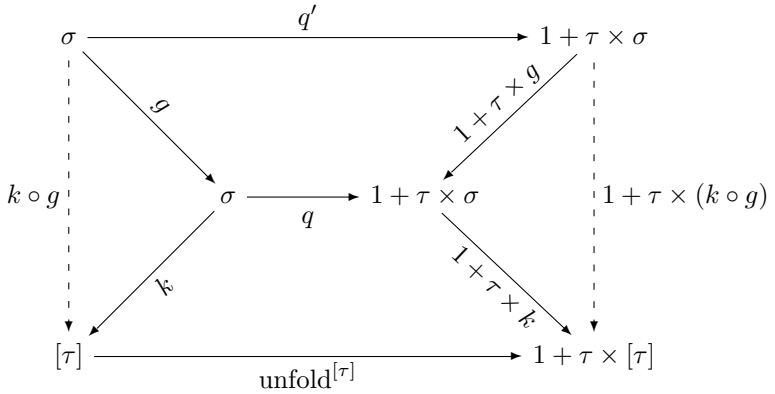
It then follows from the definitions of q and k that the diagram



commutes. Moreover $k = \text{unfd } p\ h\ t$ is the unique morphism making the above diagram commute since $\text{unfold}^{[\tau]} : [\tau] \to 1 + \tau \times [\tau]$ is a final $(1 + \tau \times -)$-coalgebra by Theorem 6.2. Further suppose that $p' : \sigma \to \texttt{Bool}, h' : \sigma \to \tau$ and $t' : \sigma \to \sigma$ are programs such that

$$p' = p \circ g,\ \ h' = h \circ g \text{ and } g \circ t' = t \circ g.$$

By defining $q'(x) = \text{if } p'(x) \text{ then } \text{inl}(*) \text{ else } \text{inr}(h'(x), t'(x))$, it follows that the upper quadrangle



commutes. Notice that the finality of $\text{unfold}^{[\tau]} : [\tau] \to 1 + \tau \times [\tau]$ guarantees that $k \circ g$ is the unique map such that the outer quadrangle of the above diagram commutes. Thus, the following inference rule holds:

$$\frac{p \circ g = p' \quad h \circ g = h' \quad t \circ g = g \circ t'}{(\text{unfd } p\ h\ t) \circ g = \text{unfold } p'\ h'\ t'}.$$

This rule states three conditions which together ensure that the composition of an unfd and a function can be *fused* together to give a single unfd.

It follows from the above inference rule that

$$(\text{unfd } p\ h\ t) \circ t \quad = \quad \text{unfd } (p \circ t)\ (h \circ t)\ t \tag{19}$$

$$\text{map } f \circ (\text{unfd } p\ h\ t) \quad = \quad \text{unfd } p\ (f \circ h)\ t \tag{20}$$

Proof.

	(iterate f) $\circ$ f	
=	(unfd F id f) $\circ$ f	(def. of iterate)
=	unfd (F $\circ$ f) (id $\circ$ f) f	(fusion (19))
=	unfd F ($f \circ$ id) f	(constant functions, composition)
=	map $f \circ$ unfd F id f	(fusion (20))
=	map $f \circ$ iterate f.	(def. of iterate)

□

Remark 7.5. Program fusion is a high-level method, i.e., it allows proofs to be performed in a purely equational way. However, it is too specialised a method in that programs involved must first be encoded using the unfd function.

We prove the Map-Iterate Proposition 7.4 using Corollary 7.2.

Proof. We prove by induction on n that for any $x : \tau$ and any $f : (\tau \to \tau)$ it holds that

$$\text{map } f\ (\text{iterate } f\ x) =_n \text{iterate } f\ f(x).$$

The base case is trivial and the inductive step is justified by:

	id_{n+1}(map f (iterate f x))	
=	id_{n+1}(map f (x : iterate f $f(x)$))	(def. of iterate)
=	$\text{id}_{n+1}(f(x)$: map f (iterate f $f(x)))$	(def. of map)
=	$f(x)$: id_n(map f (iterate f $f(x)$))	(Lemma 7.3)
=	$f(x)$: id_n(iterate f $f(f(x))$)	(Ind. hyp.)
=	$\text{id}_{n+1}(f(x)$: iterate f $f(f(x)))$	(Lemma 7.3)
=	id_{n+1}(iterate f $f(x)$)	(def. of iterate)

Thus the result holds by Corollary 7.2. □

7.2.3 Zipping two natural number lists

Let us define some programs.

zip : $[\sigma] \to [\tau] \to [\sigma \times \tau]$
zip [ ] l = [ ]
zip ($x : xs$) [ ] = [ ]
zip ($x : xs$) ($y : ys$) = (x, y) : zip xs ys
from : Nat $\to$ Nat $\to$ [Nat]
from x y = (x : from $(x + y)$ y)

succ : Nat $\to$ Nat
succ x = $x + 1$

We use the following notation:

$$\text{succ}^0 = \text{id}_{\texttt{Nat}} \text{ and } \text{succ}^{i+1} = \text{succ} \circ \text{succ}^i.$$

plus : (Nat $\times$ Nat) $\to$ Nat
plus (x, y) = if $x == 0$ then y else $1 + \text{plus}(x - 1, y)$

Note that $\text{plus}(x, y) = x + y$.

For each natural number k, define:
$\text{nats}_k : [\texttt{Nat}]$
$\text{nats}_k \quad = \quad (k : \text{map succ nats}_k)$

Proposition 7.6. *For any positive integer k, it holds that:*

$$\text{map plus (zip nats}_k \text{ nats}_k) = \text{from } 2k \text{ } 2.$$

In (Pitts, 1997), the above proposition is established using Kleene equivalence and list-bisimulations. Before we reproduce his proof, let us recall the definition of list-bisimulation (cf. (Pitts, 1997)), a technique used for proving contextual equivalence of lists.

Proposition 7.7. (List-bisimulation, Proposition 3.10 of (Pitts, 1997))
For any type τ, a binary relation $\mathcal{R} \subseteq \text{Exp}_{[\tau]} \times \text{Exp}_{[\tau]}$ is called a $[\tau]$-bisimulation if whenever $l \; \mathcal{R} \; l'$

$$\begin{aligned}
l \Downarrow [\,] \quad &\Longrightarrow \quad l' \Downarrow [\,] && (21)\\
l' \Downarrow [\,] \quad &\Longrightarrow \quad l \Downarrow [\,] && (22)\\
l \Downarrow (x : xs) \quad &\Longrightarrow \quad \exists x', xs'. && (23)\\
&\qquad\quad (l' \Downarrow (x' : xs') \wedge x =_\tau x' \wedge xs \; \mathcal{R} \; xs')\\
l' \Downarrow (x' : xs') \quad &\Longrightarrow \quad \exists x, xs. && (24)\\
&\qquad\quad (l \Downarrow (x : xs) \wedge x =_\tau x' \wedge xs \; \mathcal{R} \; xs').
\end{aligned}$$

Then for any $l, l' : [\tau]$,

$$l =_{[\tau]} l' \text{ iff } l \; \mathcal{R} \; l' \text{ for some } [\tau]\text{-bisimulation } \mathcal{R}\,.$$

Proof. Here we omit the proof which the reader can find in either (Pitts, 1997) or (Ho, 2006b). □

We reproduce Pitts' proof of Proposition 7.6 which uses Proposition 7.7.

Proof. Consider the following closed terms defined by induction on $n \in \mathbb{N}$:

$$\begin{array}{lllllllll}
n_0 & := & k & e_0 & := & 2k & l_0 & := & \text{nats}_k\\
n_{m+1} & := & \text{succ } n_m & e_{m+1} & := & e_m + 2 & l_{m+1} & := & \text{map succ } l_m
\end{array}$$

The definitions of from and e_m bear upon us to have:

$$\text{from } e_m \; 2 \Downarrow (e_m : \text{from } e_{m+1} \; 2) \tag{25}$$

From the definitions of map, nats_k, l_m and n_m, it follows by induction on m that

$$l_m \Downarrow (n_m : l_{m+1})$$

By applying the zip function, we have that

$$\text{zip } l_m \; l_m \Downarrow ((n_m, n_m) : \text{zip } l_{m+1} \; l_{m+1})$$

and from the definition of map that

$$\text{map plus } (\text{zip } l_m\ l_m) \Downarrow (\text{plus } (n_m, n_m) : \text{map plus } (\text{zip } l_{m+1}\ l_{m+1})) \tag{26}$$

One then establishes routinely by induction on m that

$$\text{plus } (n_m, n_m) \cong^{kl} e_m$$

and since Kleene equivalence is an FPC bisimulation, we have

$$\text{plus } (n_m, n_m) =_{\texttt{Nat}} e_m. \tag{27}$$

Now define $\mathcal{R} \subseteq [\texttt{Nat}] \times [\texttt{Nat}]$ by

$$\mathcal{R} := \{(\text{map plus } (\text{zip } l_m\ l_m), \text{from } e_m\ 2) \mid m \in \mathbb{N}\}.$$

Then properties (25) – (27) together imply that $\mathcal{R}$ satisfies all the conditions (21)-(24) and hence is a $[\texttt{Nat}]$-bisimulation. Thus the proof is complete by virtue of Proposition 7.7. □

Remark 7.8. Notice that in order to apply Proposition 7.7 one must come up with a suitable list-bisimulation.

Before using Corollary 7.2 to prove Proposition 7.6, we establish a useful property.

Proposition 7.9. *For any $i, k \in \mathbb{N}$, it holds that*

$$\text{map succ}^{i+1}\ \text{nats}_k = \text{map succ}^{i}\ \text{nats}_{k+1}.$$

Proof. We prove by induction on n that for all $i, k \in \mathbb{N}$,

$$\text{map succ}^{i+1}\ \text{nats}_k =_n \text{map succ}^{i}\ \text{nats}_{k+1}.$$

and the desired result follows from Corollary 7.2. The base case is trivial and the inductive step is justified by

$$\begin{aligned}
& \text{id}_{n+1}(\text{map succ}^{i+1}\ \text{nats}_k) \\
= \ & \text{id}_{n+1}(\text{map succ}^{i+1}\ (k : \text{map succ nats}_k)) \\
= \ & \text{id}_{n+1}(\text{succ}^{i+1}(k) : \text{map succ}^{i+1}(\text{map succ nats}_k)) \\
= \ & \text{id}_{n+1}(\text{succ}^{i+1}(k) : \text{map succ}^{i+2}\ \text{nats}_k) \\
= \ & (\text{succ}^{i+1}(k) : \text{id}_n(\text{map succ}^{i+2}\ \text{nats}_k)) \\
= \ & (\text{succ}^{i+1}(k) : \text{id}_n(\text{map succ}^{i+1}\ \text{nats}_{k+1})) \text{ (ind. hyp.)} \\
= \ & (\text{succ}^{i}(k+1) : \text{id}_n(\text{map succ}^{i}(\text{map succ nats}_{k+1}))) \\
= \ & \text{id}_{n+1}(\text{map succ}^{i}\ (k+1 : \text{map succ nats}_{k+1})) \\
= \ & \text{id}_{n+1}(\text{map succ}^{i}\ \text{nats}_{k+1})
\end{aligned}$$

□

We now prove Proposition 7.6 using Corollary 7.2.

Proof. We prove by induction on n that for all $k \in \mathbb{N}$, it holds that

$$\text{map plus } (\text{zip nats}_k\ \text{nats}_k) =_n \text{from } 2k\ 2.$$

The base case is trivial and the inductive step is justified by:

$$\begin{array}{ll} & \mathrm{id}_{n+1}(\text{map plus (zip nats}_k\ \text{nats}_k)) \\ = & \mathrm{id}_{n+1}(\text{map plus} \\ & ((k,k) : \text{zip (map succ nats}_k)\ \text{(map succ nats}_k))) \\ = & \mathrm{id}_{n+1}(2k : \text{map plus} \\ & \text{(zip (map succ nats}_k)\ \text{(map succ nats}_k))) \\ = & \mathrm{id}_{n+1}(2k : \text{map plus (zip nats}_{k+1}\ \text{nats}_{k+1})) \\ = & (2k : \mathrm{id}_n(\text{map plus (zip nats}_{k+1}\ \text{nats}_{k+1}))) \\ = & (2k : \mathrm{id}_n(\text{from } 2k+2\ 2)) \\ = & \mathrm{id}_{n+1}(2k : \text{from } 2k+2\ 2) \\ = & \mathrm{id}_{n+1}(\text{from } 2k\ 2) \end{array}$$

The desired result then follows from Corollary 7.2. □

7.2.4 The 'take' lemma

Let us now define the take function of (Bird and Wadler, 1988).

$$\begin{array}{l} \text{take} : \texttt{Nat} \to [\tau] \to [\tau] \\ \begin{array}{lcl} \text{take } 0\ l & = & [\,] \\ \text{take } n\ [\,] & = & [\,] \\ \text{take } n\ (x : xs) & = & (x : \text{take } n-1\ xs) \end{array} \end{array}$$

Proposition 7.10. (The 'take' lemma)
Let τ be a closed type and $l, l' : [\tau]$.

$$\forall n \in \mathbb{N}.(\text{take } n\ l =_{[\tau]} \text{take } n\ l') \implies l =_{[\tau]} l'.$$

We reproduce Pitts' proof (cf. (Pitts, 1997)) of Proposition 7.10 which uses Proposition 7.7.

Proof. For a given type τ, define $\mathcal{R} \subseteq \mathrm{Exp}_{[\tau]} \times \mathrm{Exp}_{[\tau]}$ by:

$$\mathcal{R} := \{(l, l') | \forall n \in \mathbb{N}(\text{take } n\ l =_{[\tau]} \text{take } n\ l'\}.$$

We prove that $\mathcal{R}$ satisfies conditions (21) - (24). First of all, by the evaluation rules and Kleene equivalence, the following properties hold: For all $n \in \mathbb{N}, x : \tau$ and $l, xs : [\tau]$,

(a) $\text{take } n+1\ l \Downarrow [\,] \iff l \Downarrow [\,]$.

(b) $\text{take } n+1\ l \Downarrow (x : xs) \iff \exists xs'.(l \Downarrow (x : xs') \wedge xs =_{[\tau]} \text{take } n\ xs')$.

Now suppose that $l\ \mathcal{R}\ l'$, i.e., $\forall n \in \mathbb{N}.\text{take } n\ l = \text{take } n\ l'$.

(1) To establish condition (21), we suppose that $l \Downarrow [\,]$. Then (a) implies that $\text{take } 1\ l \Downarrow [\,]$. Since $l\ \mathcal{R}\ l'$, by definition of $\mathcal{R}$, $\text{take } 1\ l =_{[\tau]} \text{take } 1\ l'$. Since contextual equivalence is an FPC bisimulation, it follows that $\text{take } 1\ l' \Downarrow [\,]$. Hence by (a) again, it holds that $l' \Downarrow [\,]$.

(2) A symmetrical argument shows that $\mathcal{R}$ satisfies condition (22).

(3) To see that it satisfies condition (23), suppose $l \Downarrow (x : xs)$. Then by (b) for any $n \in \mathbb{N}$ we have $\text{take } n+1\ l \Downarrow (x : \text{take } n\ xs)$. Since $l\ \mathcal{R}\ l'$, by definition

of $\mathcal{R}$, take $n+1\ l =_{[\tau]}$ take $n+1\ l'$. So since contextual equivalence is an FPC bisimulation, it follows that there are terms x' and xs'' with

$$\text{take } n+1\ l' \Downarrow (x' : xs'') \wedge x =_\tau x' \wedge \text{take } n\ xs =_{[\tau]} xs''.$$

By (b) again, $l' \Downarrow (x' : xs')$ for some xs' with $xs'' =_{[\tau]}$ take $n\ xs'$. We need finally to verify that $xs\ \mathcal{R}\ xs'$. But note that for all n we have take $n\ xs =_{[\tau]} xs'' =_{[\tau]}$ take $n\ xs'$. Thus we conclude that

$$\forall n \in \mathbb{N}.(\text{take } n\ xs =_{[\tau]} \text{take } n\ xs').$$

(4) A symmetrical argument shows that $\mathcal{R}$ also satisfies condition (24).

Thus $\mathcal{R}$ is a $[\tau]$-bisimulation. In particular, we have that $\mathcal{R}$ is a bisimulation and so the required contextual equivalence is obtained. □

Let us now provide an alternative proof of Proposition 7.10 by using Corollary 7.2.

Proof. We prove by induction on m that for all $l, l' \in [\tau]$, it holds that

$$\forall n \in \mathbb{N}.(\text{take } n\ l =_{[\tau]} \text{take } n\ l') \implies l =_m l'.$$

The base case is trivial and we proceed to the induction step.
Assume that the statement holds for the natural number m, we want to prove that it holds for $m+1$.
Case 1: $l =_{[\tau]} [\ ]$
Since take $1\ l =_{[\tau]}$ take $1\ l' =_{[\tau]} [\ ]$, it follows that $l' =_{[\tau]} [\ ]$, for otherwise if $l' =_{[\tau]} (x : xs)$ it would have been the case that take $1\ l' =_{[\tau]} (x : [\ ]) \neq_{[\tau]} [\ ]$. Thus we have $l =_{m+1} l'$ trivially.
Case 2: $l =_{[\tau]} (x : xs)$
In that case, $l' =_{[\tau]} (y : ys)$ for some terms y and ys. Again by applying take 1 to both the list, we have that $x =_\tau y$. Now assume for the purpose of induction that $l =_m l'$. Note that

$$\begin{array}{lll} \mathrm{id}_{m+1}(l) & =_{[\tau]} & \mathrm{id}_{m+1}(x : xs) \\ & =_{[\tau]} & (x : \mathrm{id}_m(xs)) \\ & =_{[\tau]} & (y : \mathrm{id}_m(xs)). \end{array}$$

Since $l =_{[\tau]} (x : xs)$ and $l' =_{[\tau]} (y : ys)$, it holds that

$$\forall n \in \mathbb{N}.\text{take } n\ (x : xs) =_{[\tau]} \text{take } n\ (y : ys).$$

This implies that

$$\forall n \in \mathbb{N}.\text{take } n\ xs =_{[\tau]} \text{take } n\ ys.$$

The induction hypothesis then asserts that $xs =_m ys$. Thus $\mathrm{id}_{m+1}(l) =_{[\tau]} (y : \mathrm{id}_m(xs)) =_{[\tau]} (y : \mathrm{id}_m(ys)) =_{[\tau]} \mathrm{id}_{m+1}(l')$, i.e., $l =_{m+1} l'$. □

7.2.5 The filter-map property

The next sample application involves the filter function, which we define below.
filter : $(\tau \to \texttt{Bool}) \to ([\tau] \to [\tau])$

$$\begin{array}{lcl} \text{filter } u\ [\,] & = & [\,] \\ \text{filter } u\ (x : xs) & = & \text{if } u(x) \text{ then } (x : \text{filter } u\ xs) \text{ else filter } u\ xs \end{array}$$

Proposition 7.11. *For any $u : (\tau \to \mathtt{Bool})$, $v : (\tau \to \tau)$ and $l : [\tau]$, it holds that*

$$\text{filter } u\ (\text{map } v\ l) =_{[\tau]} \text{map } v\ (\text{filter } (u \circ v)\ l).$$

This proposition was established in (Pitts, 1997) based on an induction on the *depths of proofs of evaluation*. Here we elaborate. Define the nth level evaluation relation $\Downarrow^n$ (written as $x \Downarrow^n v$) as follows. Replace in the axioms and rule regarding $\Downarrow$ (see Figure 3) each occurrence of $\Downarrow$ by $\Downarrow^n$ in an axiom or the premise of a rule and replacing $\Downarrow$ by $\Downarrow^{n+1}$ in the conclusion of each rule. Then of course we have:

$$x \Downarrow v \Leftrightarrow \exists n \in \mathbb{N}.(x \Downarrow^n v) \tag{28}$$

It suffices to show that there is a list bisimulation that relates filter u (map v l) and map v (filter $(u \circ v)$ l). Usually it is Hobson's choice.

Proof. Define

$$\mathcal{R} := \{(\text{filter } u\ (\text{map } v\ l), \text{map } v\ (\text{filter } (u \circ v)\ l)) | l : [\tau]\}.$$

Instead of proving the three conditions of a list bisimulation directly, we deduce them via (28), using the properties of $\Downarrow^n$:

(1) $\forall l.(\text{filter } u\ (\text{map } v\ l) \Downarrow^n [\,] \implies \text{map } v\ (\text{filter } (u \circ v)\ l \Downarrow [\,])$.

(2) $\forall l.(\text{map } v\ (\text{filter } (u \circ v)\ l \Downarrow^n [\,] \implies \text{filter } u\ (\text{map } v\ l) \Downarrow [\,])$.

(3) $\forall l, x, xs.(\text{filter } u\ (\text{map } v\ l) \Downarrow^n (x : xs) \implies$
$\exists xs'.(\text{map } v\ (\text{filter } (u \circ v)\ l) \Downarrow (x : xs') \wedge xs\ \mathcal{R}\ xs')$.

(4) $\forall l, x, xs'.(\text{map } v\ (\text{filter } (u \circ v)\ l) \Downarrow^n (x : xs') \implies$
$\exists xs.(\text{filter } u\ (\text{map } v\ l) \Downarrow (x : xs)\ \&\ xs\ \mathcal{R}\ xs')$.

The proofs of (1) - (4) are by induction on n. □

We now prove Proposition 7.11 by using Corollary 7.2.

Proof. Given any $l : [\tau]$, we have two possibilities:

(1) There is $n \in \mathbb{N}$ such that $\text{tl}^{(n)}(l) =_{[\tau]} [\,]$.

(2) For all $n \in \mathbb{N}$, $\text{tl}^{(n)}(l) \neq_{[\tau]} [\,]$.

Here $\text{tl}^{(0)}(l) := l$ and $\text{tl}^{(n+1)}(l) := \text{tl}(\text{tl}^{(n)}(l))$.
Those lists which satisfy (1) are called finite lists. For a finite list, we define its length to be $n \in \mathbb{N}$ for which $\text{tl}^{(n)}(l) =_{[\tau]} [\,]$. Those lists which satisfy (2) are called infinite lists.
We prove Proposition 7.11 for each of these cases.

(1) Finite lists
We prove by induction on the length of finite lists that

$$\text{filter } u\ (\text{map } v\ l) =_{[\tau]} \text{map } v\ (\text{filter } (u \circ v)\ l).$$

Base case: $n = 0$.
In this case, $l =_{[\tau]} [\,]$. On one hand, we have:

$$\begin{aligned} \text{filter } u \text{ (map } v \; l) &\equiv \text{filter } u \text{ (map } v \; [\,]) \\ &=_{[\tau]} \text{filter } u \; [\,] \\ &=_{[\tau]} [\,] \end{aligned}$$

On the other hand, we have:

$$\begin{aligned} \text{map } v \text{ (filter } (u \circ v) \; l) &\equiv \text{map } v \text{ (filter } (u \circ v) \; [\,]) \\ &=_{[\tau]} \text{map } v \; [\,] \\ &=_{[\tau]} [\,] \end{aligned}$$

Hence the statement holds.
Inductive step:
Assume that the statement holds for all finite lists of length n. We want to prove that the statement holds for all finite lists of length $n + 1$. We write $l = (x : xs)$.

$$\begin{aligned} \text{filter } u \text{ (map } v \; l) &\equiv \text{filter } u \text{ (map } v \text{ } (x : xs)) \\ &=_{[\tau]} \text{filter } u \text{ } (v(x) : \text{map } v \; xs) \\ &=_{[\tau]} \begin{cases} \text{filter } u \text{ (map } v \; xs) & \text{if } u \circ v(x) = \text{F} \\ (v(x) : \text{filter } u \text{ (map } v \; xs)) & \text{if } u \circ v(x) = \text{T} \end{cases} \\ &\overset{\text{Ind. hyp.}}{=_{[\tau]}} \begin{cases} \text{map } v \text{ (filter } (u \circ v) \; xs) & \text{if } u \circ v(x) = \text{F} \\ (v(x) : \text{map } v \text{ (filter } (u \circ v) \; xs)) & \text{if } u \circ v(x) = \text{T} \end{cases} \end{aligned}$$

(2) Infinite lists
We prove by induction on m that for all infinite lists l

$$\text{filter } u \text{ (map } v \; l) =_m \text{map } v \text{ (filter } (u \circ v) \; l).$$

Base case: $m = 0$. This holds trivially.
Inductive step: There are two possibilities:
(i) $u \circ v(\text{hd}(\text{tl}^{(n)}(l))) =_{[\tau]} \text{F}$ for all $n \in \mathbb{N}$.
Since the evaluations of filter u (map v l) and map v (filter $(u \circ v)$ l) involve infinite unfoldings, it follows that both diverges. Hence the statement holds.
(ii) There is a minimum $n \in \mathbb{N}$ such that $u \circ v(\text{hd}(\text{tl}^{(n)}(l))) =_{[\tau]} \text{T}$.
Then we have:

$$
\begin{array}{lll}
 & \mathrm{id}_{m+1}(\text{filter } u \text{ (map } v \ l)) & \\
=_{[\tau]} & \mathrm{id}_{m+1}(\text{filter } u \text{ (map } v \ \mathrm{tl}^{(n)}(l))) & \text{(Kleene equiv.)} \\
=_{[\tau]} & \mathrm{id}_{m+1}(v(\mathrm{hd}(\mathrm{tl}^{(n)}(l))) : \text{filter } u \text{ (map } v \ \mathrm{tl}^{n+1}(l))) & \text{(map \& filter)} \\
=_{[\tau]} & (v(\mathrm{hd}(\mathrm{tl}^{(n)}(l))) : \mathrm{id}_m(\text{filter } u \text{ (map } v \ \mathrm{tl}^{n+1}(l)))) & \text{(Lemma 7.3)} \\
=_{[\tau]} & (v(\mathrm{hd}(\mathrm{tl}^{(n)}(l))) : \mathrm{id}_m(\text{map } v \text{ (filter } (u \circ v) \ \mathrm{tl}^{n+1}(l)))) & \text{(Ind. hyp.)} \\
=_{[\tau]} & \mathrm{id}_{m+1}(v(\mathrm{hd}(\mathrm{tl}^{(n)}(l))) : \text{map } v \text{ (filter } (u \circ v) \ \mathrm{tl}^{n+1}(l))) & \text{(Lemma 7.3)} \\
=_{[\tau]} & \mathrm{id}_{m+1}(\text{map } v \text{ filter } (u \circ v) \ \mathrm{tl}^{(n)}(l)) & \text{(map \& filter)} \\
=_{[\tau]} & \mathrm{id}_{m+1}(\text{map } v \text{ (filter } (u \circ v) \ l)). & \text{(Kleene equiv.)}
\end{array}
$$

The desired result then follows from Corollary 7.2.

□

Remark 7.12. We have investigated the reasoning powers derived from the non-trivial pre-deflationary structure e^{σ}, which leads one to wonder if those derived from the deflationary structure d^{σ} are even more powerful. This should be investigated in our future works.

8 Conclusion and future work

The operational domain theory developed herein exploits the free algebra structure of the "default" recursive construction offered by the syntax (and the operational semantics) of FPC. The categorical framework we chose facilitates a relatively clean theory on which convenient principles of reasoning about programs are based. The present work is a follow-up on a much earlier report (Ho, 2006a).

In our future work, we shall explore:

1. the implication of Pitts' work (Pitts, 1996) on relational properties of domains in our operational setting; and

2. the possibility of developing an operational domain-theory that caters for non-deterministic languages, such as (Hennessy and Ashcroft, 1980).

Acknowledgements. Many people have contributed to the birth and development of ideas in this paper. Special thanks go to my PhD supervisor, M.H. Escardó, whose deep insight and gentle advice I have always benefited from. 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. Fiore that there should be no technical issues in proceeding with the use of product categories. T. Streicher's his timely advice emerging from a series of email-discussions eventually gave rise to the operational algebraic compactness result. I would also wish to acknowledge my colleagues in the Mathematics and Mathematics Education Academic Group (NIE), particularly D. Zhao and T.Y. Lee, concerning the operational proof of the minimal invariance property of realisable functors.

References

M. Abadi and M.P. Fiore. Syntactic considerations on recursive types. In *Proceedings of the 11th Annual IEEE Symposium on Logic In Computer Science*, pages 242–252. IEEE Computer Society Press, 1996.

S. Abramsky and A. Jung. *Domain Theory*, volume 3 of *Handbook of Logic in Computer Science*. Clarendon Press, Oxford, 1994.

R. Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall International, 2nd edition, 1998.

R. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice-Hall, 1988.

L. Birkedal and R. Harper. Relational Interpretations of Recursive Types in an Operational Setting. *Information and Computation*, (155):3 – 63, 1999.

M.H. Escardo and W.K. Ho. Operational domain theory and topology of a sequential language. In *Proceedings of the 20th Annual IEEE Symposium on Logic In Computer Science*, pages 427 – 436. IEEE Computer Society Press, 2005.

M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1996. Distinguished Dissertations in Computer Science.

M.P. Fiore and G.D. Plotkin. An Axiomatisation of Computationally Adequate Domain-Theoretic Models of FPC. In *Proceedings of the 10th Annual IEEE Symposium on Logic In Computer Science*, pages 92 – 102. IEEE Computer Society Press, 1994.

P.J. Freyd. Algebraically complete categories. In *Lecture Notes in Mathematics*, volume 1488, pages 95 – 104. Springer Verlag, 1991.

P.J. Freyd. Remarks on algebraically compact categories. In *Applications of Categories in Computer Science*, volume 177, pages 95 – 106. Cambridge University Press, 1992. Lecture Notes in Mathematics.

J. Gibbons and G. Hutton. Proof Methods for Corecursive Programs. *Fundamentae Informaticae*, 20:1 – 14, 2005.

G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M.W. Mislove, and D.S. Scott. *Continuous Lattices and Domains*. Number 93 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 2003.

A. D. Gordon. Functional programming and input/output. In *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1994.

A.D. Gordon. Bisimilarity as a theory of functional programming. *Notes Series: BRICS-NS-95-3, BRICS*, 1995. Department of Computer Science, University of Aarhus.

M.C.B. Hennessy and E.A. Ashcroft. A mathematical semantics for a non-deterministic typed lambda-calculus. *Theoretical Computer Science*, 11(3): 227–245, July 1980.

W.K. Ho. An Operational Domain-theoretic Treatment of Recursive Types. In M. Mislove and S. Brookes, editors, *Proceedings of the 22nd Conference on Mathematical Foundations in Programming Semantics*, number 158 in Electronic Notes in Theoretic Computer Science, pages 237–259, 2006a.

W.K. Ho. *An operational domain theory and topology of sequential functional languages*. PhD thesis, The University of Brimingham, October 2006b.

D.J. Howe. Equality in lazy computation systems. In *Proceedings of the 4th Annual Symposium on Logic In Computer Science*, pages 198–203. IEEE Computer Society Press, 1989.

D.J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, February 1996.

G. Hutton and J. Gibbons. The Generic Approximation Lemma. *Information Processing Letters*, 79(4):197 – 201, 2001.

S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 2nd edition, 1998.

I.A. Mason, S.F. Smith, and C.L. Talcott. From operational semantics to domain theory. *Information and Computation*, 128(1):26–47, 1996.

G. McCusker. Games and Full Abstraction for FPC. *Information and Computation*, (160):1–61, 2000.

A.M. Pitts. Relational Properties of Domains. *Information and Computation*, 127:66–90, 1996.

A.M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A.M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

G.D. Plotkin. Lectures on predomains and partial functions, 1985. Notes for a course given at the Center for the Study of Languages and Information, Stanford.

A. Rohr. *A Universal Realizability Model for Sequential Functional Computation*. PhD thesis, Technischen Universitat Darmstadt, July 2002.

A.K. Simpson. Recursive Types in Kleisli Categories. Available from `http://homepages.inf.ed.ac.uk/als/Research`, 1992.

Contents