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, 1, Nanyang Walk, Singapore 637616, Singapore.

Received 26 April 2011; Revised 31 July 2012

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, of which a purely operational proof is given.

Contents

1 Introduction 2
2 The programming language FPC 3
   2.1 The syntax 3
   2.2 Operational semantics 5
   2.3 Fixed point operator 6
   2.4 Some notations 6
   2.5 FPC contexts 7
3 Foundations 8
   3.1 FPC (bi)simulation and (bi)similarity 8
   3.2 Operational extensionality theorem 9
   3.3 Kleene preorder and equivalence 11
   3.4 Contextual orders of 1, Σ and ω 12
   3.5 Properties of FPC contextual preorder and equivalence 12
4 FPC considered as a category 17
   4.1 Realisable actions and functors 18

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

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

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

The principal approach taken in this paper deviates from classical domain theory in that we do not produce recursive types by inverse limits constructions – we have it for free by working directly with the recursive type declaration in FPC. By considering suitable syntactic categories, we extend type expressions to functors. Our approach is similar to an earlier work by (Abadi and Fiore, 1996) in that we too work with the diagonal category $\mathbf{FPC}^\delta$. The major differences are as follows. Firstly, we do not derive the functoriality using the domains model. Secondly, we are not constrained by the syntax to work in the diagonal category $\mathbf{FPC}^\delta$ alone (c.f. (Abadí and Fiore, 1996), p.5). More precisely, we show that it is possible to work with the product category $\mathbb{FPC}$. To establish functoriality of type expressions, we rely on an operational version of the minimal invariance theorem. An important contribution of this paper is its purely operational proof of this minimal invariance theorem. Similar results and proofs already appeared in (Birkedal and Harper, 1999) and (Lassen, 1998a). However, the languages they considered have only
one top-level recursive type. Our work here is more general since FPC has the facility
to deal with nested recursion for types. Furthermore, we develop an operational version
of P. Freyd’s algebraic compactness (see [Freyd, 1991]) founded on the preceding cate-
gorical constructions. This paper culminates with a new result that relates the algebraic
compactness established earlier for $\text{FPC}_\delta$ and $\bar{\text{FPC}}_\delta$, respectively.

Our subsequent discourse is organised as follows. Section 2 introduces the language
FPC, together with operational notions relevant to our study (e.g., contextual pre-order
and equivalence). Section 3 contains the operational domain-theoretic toolkit in the style
of [Pitts, 1997]. Not wanting to burden the reader with technical details that are not the
foci of this paper, we omit proofs and instead give precise references to where these can
be found. Section 4 deals with the categorical constructions. The bulk of the development
here involves the extension of formal type expressions to endofunctors on these categories.
The minimal invariance theorem, proven in a purely operational manner, appears in
Section 5. Section 6 expounds on (parameterised) algebraic compactness of the syntactic
categories constructed in the preceding section.

Throughout the discussion, the reader’s familiarity with a sequential functional lan-
guage (e.g., PCF or Haskell) is assumed. For the theory of recursive domain equations,
the reader is referred to [Abramsky and Jung, 1994] [Gierz et al., 2003], and for category
theory [Mac Lane, 1998] [Poigné, 1992].

2. The programming language FPC

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

2.1. The syntax

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

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

For type expressions, we have type variables, product types, separated sum types, lifted
types, recursive types and function types. Our choice of the type constructors is natural
in the context of call-by-name, and similar choices have also been made in existing works
such as [McCusker, 2000] [Rohr, 2002]. A closed type is a type expression containing no
free type variables, i.e., if every occurring type variable $X$ is bound under the scope of a
recursive type constructor $\mu X$.

The set of all closed types is denoted by Type. A type context is a list of distinct type

\[\text{Interested readers can also read about call-by-name FPC in McCusker, 2000.}\]
variables (which may be empty). We write \( \Theta \vdash \sigma \) for the type \( \sigma \) in context \( \Theta \), indicating that the set of free type variables occurring in \( \sigma \) is a subset of the type context \( \Theta \).

The raw FPC terms are given by the syntax trees generated by the following grammar, modulo \( \alpha \)-equivalence (see Figure 1). Terms of the form \( s(t) \) are called applications, while those of the form \( \lambda x. t \) are called abstractions. For variables bound by \( \lambda \)'s, we employ the usual convention of \( \alpha \)-conversion according to which terms are considered as equal (denoted by \( s \equiv t \)) if they can be obtained from each other by an appropriate renaming of bound variables. The set of free variables appearing in a term \( t \) is denoted by \( \text{fv}(t) \). 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 \( \Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n \) where \( \Theta \vdash \sigma_i \) if they can be obtained from each other by an appropriate renaming of bound variables. The set of free variables appearing in a term \( t \) is denoted by \( \text{fv}(t) \). Terms containing no free variables are called closed terms. Otherwise, they are known as open terms.

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

The following is routine:

**Lemma 2.1.**

1. If \( \Gamma \vdash t : \sigma \), then \( \text{fv}(t) \subseteq \text{dom}(\Gamma) \).
2. If \( \Gamma \vdash t : \sigma \) and \( x \not\in \text{dom}(\Gamma) \), then \( \Gamma, x : \tau \vdash t : \sigma \) for any \( \tau \).
Fig. 2. Typing rules of FPC

(3) If $\Gamma, \Gamma' \vdash t : \sigma$ and $\text{fv}(t) \subseteq \text{dom}(\Gamma)$, then $\Gamma \vdash t : \sigma$.

(4) If $\Gamma \vdash t_i : \sigma_i$ for $i = 1, \ldots, n$ and $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash s : \sigma$, then $\Gamma \vdash s[t_i/x_i] : \sigma$.

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

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:

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

A closed term $v$ generated by the above grammar is called a value. Let $\text{Val}_\sigma$ denote the set of values of the closed type $\sigma$, i.e.,

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

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

Proposition 2.2. Evaluation is deterministic and preserves typing, i.e.,

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

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


2.3. Fixed point operator

Like in existing works such as [Rohr, 2002], we can define a fixed point operator using the recursive types. This is done as follows:

\[ \text{fix}_\sigma := \lambda f : (\sigma \rightarrow \sigma). k(\text{fold}^\top(k)) \]

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

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

\[ f(\text{fix}_\sigma(f)) \downarrow v \]

What turns out to be true about this fixed point operator is the following operational property: \( f(\text{fix}_\sigma(f)) \) is contextually equivalent to \( \text{fix}_\sigma(f) \) (see Property (3.48) in Section 3). The concept of contextual equivalence will be defined in Section 2.5.

2.4. Some notations

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

For each FPC closed type \( \sigma \), define the syntactic bottom of type \( \sigma \) to be the term

\[ \bot_\sigma := \text{fix}_\sigma(\lambda x^\sigma. x). \]

There are three special closed types worth mentioning:

\[ 1 := \mu X. X, \quad \Sigma := 1_\bot, \quad \Sigma := \mu X. (X_\bot) \]

The type 1 is called the void type and contains no values. Lifting the type 1 produces the unit type, \( 1_\bot \), which we denote by \( \Sigma \). The non-divergent element of \( \Sigma \), \( \text{up}(\bot_1) \), is denoted by \( \top \). We shall be exploiting \( \Sigma \) 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 $\mathbb{N}$ has elements $0, 1, \ldots, \infty$ which can be encoded by defining:

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

where $(+1) := \lambda x. \ x + 1$. We define $n - 1 := \text{case}(\text{unfold}(n))$ of $\text{up}(x) \cdot x$. The $\Sigma$-valued convergence test $(> 0) := \lambda x. \mathbb{N}$, if $\text{unfold}(x)$ then $\top$ iff $x$ evaluates to $n + 1$ for some $n : \mathbb{N}$.

### 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:

$$C ::= \ldots | p$$

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

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 $\_ \sigma$ has closed type $\sigma$. We restrict ourselves to FPC contexts which are typable.

The relation

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

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

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

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

We define

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

to be the set of FPC contexts that can be assigned to the closed type $\sigma$, given $\Gamma$. We write $\text{Ctx}_\sigma$ for $\text{Ctx}_\sigma(\emptyset)$.

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

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

but not in the FPC context

$$C_2[\_ : \sigma] := (\lambda x. \_ : \sigma)(\text{if } \_ : \sigma \text{ then } 1 \text{ else } 2).$$
Let $\Gamma \vdash s, t : \sigma$ be two FPC terms-in-context. We write

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

to mean that for all ground contexts $C[\sigma] \in \text{Ctx}_\Sigma$ with $\Gamma$ 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 $\equiv$. For a given term $\sigma$, the order induced by the preorder $\sqsubseteq$ on the set of equivalence classes of closed terms of type $\sigma$ is called the contextual order. Notice that we have chosen the ground type $\Sigma$ to be the type on which program observations are based.

**Remark 2.3.** Let $s, t : \sigma$ be closed terms. Then $s \sqsubseteq_\sigma t$ iff

$$\forall p : \sigma \rightarrow \Sigma. \ (p(s) \downarrow \top \implies p(t) \downarrow \top).$$

**Proof.**

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

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

3. Foundations

In this section, one can find the operational toolkit which we shall use from time to time during the ensuing treatment of recursive types. The operational machineries reported here is a reworking of [Pitts, 1997] to suit our needs in FPC. Crucially, we rely on them to reason about program equivalence without appeal to any denotational model. The operational proofs mentioned in this section (whether shown or cited) are independent of the properties reported in the rest of the paper. Due to limited space, we omit proofs of many technical lemmas in Section 3.1 and give, instead, precise references whenever such omissions occur.

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

As FPC (bi)simulation and (bi)similarity are the main operational tools by which we derive many properties concerning contextual preorder and equivalence, we devote a short section for them.

Let $\mathcal{R} = \{\mathcal{R}_\sigma | \sigma \in \text{Type}\}$ be an type-indexed family of binary relations $\mathcal{R}_\sigma$ between closed FPC terms of type $\sigma$. Given $\mathcal{R}$, the definitions of $\langle \mathcal{R} \rangle$ and $[\mathcal{R}]$ are given in Figure 4. Because the operators $\mathcal{R} \mapsto \langle \mathcal{R} \rangle$ and $\mathcal{R} \mapsto [\mathcal{R}]$ are monotone on the set of all type-indexed families of binary relations between closed FPC terms, by the Tarski-Knaster’s Fixed Point Theorem they have greatest (post-)fixed points.
Definition 3.1. A type-indexed family $S$ of binary relations $S_\sigma$ between the closed FPC terms of closed type $\sigma$ which satisfies $S_\sigma \subseteq \langle S \rangle$ (respectively, $S_\sigma \subseteq [S]$) is called an FPC simulation (respectively, FPC bisimulation); the greatest such is called an FPC similarity (respectively, FPC bisimilarity) and is denoted by $\preceq$ (respectively, $\simeq$).

We invite the reader to compare the definition of $\langle R \rangle$ with the extensionality properties (3.36)–(3.42) which we claim to hold. The idea is to first define $\langle R \rangle$ (respectively, $[R]$) in such a way as to ‘model’ after the extensionality properties we have in mind and once we have established that the contextual preorder is a bisimulation, then it is immediate that it satisfies these extensionality properties.

A powerful proof principle we often invoke is the following:

Proposition 3.2. (Co-induction principle for $\preceq$ and $\simeq$)

Given $s, t : \sigma$, to prove that $s \simeq_\sigma t$ (respectively, $s \preceq_\sigma t$), it suffices to find an FPC bisimulation $B$ (respectively, an FPC simulation $S$) such that $s \ B \sigma t$ (respectively, $s \ S \sigma t$).

Proof. If $B \subseteq [B]$, then $B \subseteq \simeq$ since $\simeq$ is the greatest post-fixed point of $[-]$, so that $B_\sigma \subseteq \simeq_\sigma$. Thus, if $s \ B_\sigma t$, then $s \simeq_\sigma t$. \qed

Once we have established that FPC bisimilarity and contextual equivalence coincide, the co-induction principle will provide a powerful tool for proving contextual equivalence.

The following facts about (bi)similarity are a direct consequence of the co-induction principle.

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

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

Note that property 3 of the preceding proposition holds because the evaluation relation is deterministic.

3.2. Operational extensionality theorem

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

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

$\Gamma \vdash s \ R^\sigma_\sigma s' \iff \forall t_1 \in \text{Exp}_{\sigma_1}, \ldots, t_n \in \text{Exp}_{\sigma_n}. (s[t_1/x] R_\sigma s'[t_1/x])$.

We call $R^\sigma$ the open extension of $\mathcal{R}$. Applying this construction to $\preceq$ and $\simeq$, we obtain the relations $\preceq^\sigma$ and $\simeq^\sigma$ on open terms, which we can call open similarity and open bisimilarity respectively.
Fig. 4. Definitions of $\langle R \rangle$ and $[R]$ in FPC
Proposition 3.5. FPC open similarity is a preorder and FPC open bisimilarity is the equivalence relation induced by it.

Proof. Because \(\preceq^\circ\) is defined via an extension of \(\preceq\) by considering closed instantiation of open terms, one just relies on the closed analogue in Proposition 3.3.

FPC open similarity \(\preceq^\circ\) respects substitution in the following sense:

Lemma 3.6. If \(\Gamma \vdash t : \sigma\) and \(\Gamma, x : \sigma \vdash s \preceq^\circ s'\) (where \(x \not\in \text{dom}(\Gamma)\)), then
\[
\Gamma \vdash s[t/x] \preceq^\circ s'[t/x].
\]

Proof. Let \(\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n\) and suppose that \(t_i \in \text{Exp}_{\sigma_i}\) (for \(i = 1, \ldots, n\)). Since \(\Gamma, x : \sigma \vdash s \preceq^\circ s'\), it follows that \(s[t/x][\bar{t}/\bar{x}] \preceq s'[t/x][\bar{t}/\bar{x}]\). But \((s[t/x])[\bar{t}/\bar{x}] \equiv s[t/x, \bar{t}/\bar{x}]\) as \(x \not\in \text{dom}(\Gamma)\), it follows that \((s[t/x])[\bar{t}/\bar{x}] \preceq (s'[t/x])[\bar{t}/\bar{x}]\) as required.

We now state (without proof) the main result of this subsection.

Theorem 3.7. (Operational extensionality theorem for FPC)
Contextual preorder (respectively, equivalence) coincides with similarity (respectively, bisimilarity):
\[
\Gamma \vdash t \sqsubseteq_\sigma t' \iff \Gamma \vdash t \preceq^\circ t'
\]
and
\[
\Gamma \vdash t =_\sigma t' \iff \Gamma \vdash t \simeq^\circ t'.
\]
In particular, the following coinduction principle for contextual equivalence holds: To prove that two closed FPC terms are contextually equivalent, it suffices to to find an FPC bisimulation which relates them.

Proof. The proof of the above result is technical. Firstly, the notions of FPC precongruence and congruence need to be defined. Secondly, one defines an auxiliary relation \(\preceq^\ast\) in terms of \(\preceq^\circ\), and establishes that \(\preceq^\circ\) and \(\preceq^\ast\) are equivalent. Exploiting this equivalence, it can be shown that \(\preceq^\circ\) is an FPC precongruence and as a consequence, is contained in the contextual preorder. This special technique is an adaptation of Howe’s method (Howe, 1989; Howe, 1996). Lastly, it is to be verified that the contextual preorder, when restricted to closed terms, is an FPC simulation. Consequently, by the co-induction principle, it follows that the contextual preorder is contained in the FPC similarity. The proof will finally be complete once it is shown that the above required implications can be extended from the closed terms to the open terms. For details of the proof, see Chapter 8 of (Ho, 2006b), pp. 92–117.

3.3. Kleene preorder and equivalence

In this section, we look at a very useful notion of program equivalence called Kleene equivalence which turns out to be an FPC bismulation. In certain instances, establishing Kleene equivalence is a convenient way of proving contextual equivalence in the light of Theorem 3.7.
Definition 3.8. For each closed type $\sigma$, consider the following binary relations on $\text{Exp}_\sigma$:

$$t \preceq^k_\sigma t' \iff \forall v \in \text{Val}_\sigma. (t \Downarrow v \implies t' \Downarrow v)$$

and

$$t \simeq^k_\sigma t' \iff (t \preceq^k_\sigma t') \land (t' \preceq^k_\sigma t).$$

The relation $\preceq^k$ is called the Kleene preorder. If $t \simeq^k_\sigma t'$, we say that $t$ and $t'$ are Kleene equivalent.

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

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

Proposition 3.9. For any closed type $\sigma$ and any $t, t' \in \text{Exp}_\sigma$, it holds that

1. $t \preceq^k_\sigma t' \implies t \preceq_\sigma t'$.
2. $t \simeq^k_\sigma t' \implies t \simeq_\sigma t'$.

Proof. Because $\simeq$ is the symmetrization of $\preceq$, once (1) is established (2) will follow. To prove (1), the reader can easily check that the relation $\preceq^k$ is an FPC simulation and subsequently invoke Theorem 3.7. \qed

3.4. Contextual orders of $1$, $\Sigma$ and $\omega$

Define the type-indexed relation $\mathcal{I}$ on the FPC closed terms by enforcing $\forall s, s': 1. s \mathcal{I}_1 s'$, and replacing all occurrences of $[R]$ in equations (3.8)–(3.13) by $\mathcal{I}$. It is trivial that $\mathcal{I}$ is an FPC bisimulation. So, for any $t : 1$, it holds that $t \mathcal{I}_1 \bot_1$ which then implies, by the co-induction principle, that $t =_1 \bot_1$. This proves that the contextual order of the void type is the trivial singleton order. Similarly, by choosing suitable FPC bisimulations and applying the co-induction principle, it can be shown that the contextual orders of the unit type $\Sigma$ and the ordinal type $\omega$ are the usual ones interpreted by the Scott model (see (Ho, 2006b), p.66 for details). The contextual orders of 1, unit and $\omega$ are given below:

3.5. Properties of FPC contextual preorder and equivalence

With the operational machinery developed in the earlier subsection, we are now ready to state and prove the following groups of properties concerning FPC contextual preorder and equivalence.
Fig. 5. 1, Σ, ⊨

3.5.1. Inequational logic

\[ \Gamma \vdash t : \sigma \implies \Gamma \vdash t \sqsubseteq t \]  
(3.21)

\[ \Gamma \vdash t \sqsubseteq t' \land \Gamma \vdash t'' \implies \Gamma \vdash t \sqsubseteq t'' \]  
(3.22)

\[ \Gamma \vdash t \sqsubseteq t' \land \Gamma \vdash t'' \sqsubseteq t \implies \Gamma \vdash t =_{\sigma} t' \]  
(3.23)

\[ \Gamma, x : \sigma \vdash t \sqsubseteq t' \implies \Gamma \vdash \lambda x. t \sqsubseteq_{\sigma \rightarrow \tau} \lambda x. t' \]  
(3.24)

\[ \Gamma \vdash s \sqsubseteq_{\sigma + \tau} s' \land \Gamma, x : \sigma \vdash t_1 \sqsubseteq_{\rho} t_1' \implies \sqsubseteq_{\rho} \text{case}(s') \text{ of } \text{up}(x). t_1' \]  
(3.25)

\[ \Gamma \vdash s \sqsubseteq_{\sigma + \tau} s' \land \Gamma, y : \tau \vdash t_2 \sqsubseteq_{\rho} t_2' \implies \sqsubseteq_{\rho} \text{case}(s') \text{ of } \text{inl}(x). t_1' \text{ or } \text{inr}(y). t_2' \]  
(3.26)

\[ \Gamma \vdash t \sqsubseteq_{\sigma} t' \land \Gamma \sqsubseteq_{\sigma} \Gamma' \implies \Gamma' \vdash t \sqsubseteq t' \]  
(3.27)

\[ \Gamma \vdash t \sqsubseteq t' \land \Gamma, x : \sigma \vdash s : \tau \implies \Gamma \vdash s[t/x] \sqsubseteq_{\tau} s[t'/x] \]  
(3.28)

\[ \Gamma \vdash t : \sigma \land \Gamma, x : \sigma \vdash s \sqsubseteq_{\tau} s' \implies \Gamma \vdash s[t/x] \sqsubseteq_{\tau} s'[t/x] \]  
(3.29)

Properties (3.21)–(3.28) (known as inequational logic) are direct consequences of the definitions of \( \sqsubseteq_{\sigma} \) and \( =_{\sigma} \). By contrast, (3.29) cannot be immediately established as the operation \( s \mapsto s[t/x] \) is not necessarily of the form \( s \mapsto C[s] \) for some FPC context \( C[-] \). Instead, one can readily establish that

\[ \Gamma, x : \sigma \vdash s \sqsubseteq_{\tau} s' \implies \Gamma \vdash (\lambda x. s)t \sqsubseteq_{\tau} (\lambda x. s')t \]

using the definition of contextual preorder directly. Then, one relies on the \( \beta \)-equality (3.30) to achieve the desired result.

3.5.2. \( \beta \)-equalities

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

\[ \Gamma \vdash s : \sigma \land \Gamma \vdash t : \tau \implies \Gamma \vdash \text{fst}(s, t) =_{\tau} s \land \]  
(3.31)

\[ \Gamma \vdash \text{snd}(s, t) =_{\tau} t \]
\[ \Gamma \vdash t : \sigma \implies \Gamma \vdash \text{case}(t) \text{ of } \text{up}(x).\ s =_{\sigma} s[t/x] \tag{3.32} \]
\[ \Gamma \vdash t : \sigma \implies \forall \Gamma \vdash s : \tau, \ \Gamma \vdash s' : \tau. \tag{3.33} \]
\[(\Gamma \vdash \text{case}(\text{inl}(t)) \text{ of } \text{inl}(x).\ s \text{ or inr}(y).\ s' =_{\sigma} s[t/x]) \]
\[ \Gamma \vdash t : \sigma \implies \forall \Gamma \vdash s : \tau, \ \Gamma \vdash s' : \tau. \tag{3.34} \]
\[(\Gamma \vdash \text{case}(\text{inr}(t)) \text{ of } \text{inl}(x).\ s \text{ or inr}(y).\ s' =_{\sigma} s'[t/y]) \]
\[ \Gamma \vdash t : \sigma[X. \sigma/X] =_{\sigma[X. \sigma/X]} t \tag{3.35} \]

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

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

For all $s, s' \in \text{Exp}_{\Sigma}$,
\[ s \sqsubseteq_{\Sigma} s' \iff (s \not\downarrow \top \implies s' \not\downarrow \top). \tag{3.37} \]

For all $p, p' \in \text{Exp}_{\sigma \times \tau}$,
\[ p \sqsubseteq_{\sigma \times \tau} p' \iff (\text{fst}(p) \sqsubseteq_{\sigma} \text{fst}(p') \land \text{snd}(p) \sqsubseteq_{\tau} \text{snd}(p')). \tag{3.38} \]

For all $s, s' \in \text{Exp}_{\sigma + \tau}$,
\[ s \sqsubseteq_{\sigma + \tau} s' \iff \forall a \in \text{Exp}_{\sigma}, \forall b \in \text{Exp}_{\tau}. \tag{3.39} \]
\[ (s \not\downarrow \text{inl}(a) \implies \exists a' \in \text{Exp}_{\sigma}. s' \not\downarrow \text{inl}(a') \land a \sqsubseteq_{\sigma} a') \lor \]
\[ (s \not\downarrow \text{inr}(b) \implies \exists b' \in \text{Exp}_{\tau}. s' \not\downarrow \text{inr}(b') \land a \sqsubseteq_{\sigma} a'). \]

For all $t, t' \in \text{Exp}_{\sigma \bot}$,
\[ t \sqsubseteq_{\sigma \bot} t' \iff \forall s \in \text{Exp}_{\sigma}. \tag{3.40} \]
\[ (t \not\downarrow \text{up}(s) \implies \exists s' \in \text{Exp}_{\sigma}. t' \not\downarrow \text{up}(s') \land s \sqsubseteq_{\sigma} s'). \]

For all $t, t' \in \text{Exp}_{\mu \sigma}$,
\[ t \sqsubseteq_{\mu \sigma} t' \iff \text{unfold}(t) \sqsubseteq_{\sigma[X. \sigma/X]} \text{unfold}(t'). \tag{3.41} \]

For all $f, f' \in \text{Exp}_{\sigma \rightarrow \tau}$,
\[ f \sqsubseteq_{\sigma \rightarrow \tau} f' \iff \forall t \in \text{Exp}_{\sigma}. (f(t) \sqsubseteq_{\tau} f'(t)). \tag{3.42} \]

Extensionality properties analogous to the above equations (3.37)–(3.42) (see Figure 4(1)) hold by construction for the notion of FPC similarity defined in Section 3.1.
Thus, equations (3.37)–(3.42) follow immediately from the coincidence of FPC bisimilarity with contextual equivalence (Theorem 3.7). Note that by virtue of Theorem 3.7, (3.36) then follows from Lemma 3.6.

3.5.4. $\eta$-equalities The following $\eta$-equalities follow by combining the extensionality properties with the corresponding $\beta$-equality:

$$
\Gamma \vdash f : \sigma \to \tau \land x \notin \text{dom} (\Gamma) \implies \Gamma \vdash f =_{\sigma \to \tau} \lambda x. f(x) \quad (3.43)
$$

$$
\Gamma \vdash p : \sigma \times \tau \implies \Gamma \vdash p =_{\sigma \times \tau} (\text{lst}(p), \text{snd}(p)) \quad (3.44)
$$

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

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

For instance, to prove (3.47), it suffices, by virtue of (3.41), to show that

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

But the $\beta$-equality (3.30) guarantees that the above contextual equivalence. Hence (3.47) holds. Notice that properties (3.35) and (3.47) together imply that:

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

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

3.5.5. Fixed-point operator Recall that

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

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

Using the $\beta$-equality (3.30) we have

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

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

$$
\Gamma \vdash \text{fix}_\sigma(f) \equiv (\lambda x^\tau. f(\text{unfold}^\tau(x)x))(\text{fold}^\tau(k))
$$

$$
\equiv_{\sigma} f(\text{unfold}^\tau(\text{fold}^\tau(\text{fold}^\tau(k))))
$$

$$
\equiv_{\sigma} f(k(\text{fold}^\tau(k))).
$$

Thus, we have:

$$
\Gamma \vdash f : \sigma \to \sigma \implies \text{fix}(f) =_{\sigma} f(\text{fix}_\sigma(f)) \quad (3.48)
$$

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

$$
\Gamma \vdash t : \sigma \implies \Gamma \vdash \bot_\sigma \subseteq_{\sigma} t \quad (3.49)
$$
To establish (3.49), we reason as follows. First of all, we observe that unfold\(\tau\)(fold\(\tau\)(k)) does not evaluate to any value. Indeed, there cannot be a minimal derivation for the evaluation unfold\(\tau\)(fold\(\tau\)(k))fold\(\tau\)(k) \(\Downarrow\) v for any value v. Suppose otherwise, it follows from the \((\Downarrow \ unfold)\) rule and the fact that k is a \(\lambda\)-abstraction that

\[
\begin{align*}
\text{fold}(k) & \Downarrow \text{fold}(k) \quad k \Downarrow k \\
\text{unfold}(\text{fold}(k)) & \Downarrow k
\end{align*}
\]

Thus, the derivation of unfold(fold(k))fold(k) \(\Downarrow\) v must be

\[
\begin{align*}
\text{unfold}(\text{fold}(k)) & \Downarrow k \\
\lambda x^\sigma. x & \Downarrow \lambda x^\sigma. x \\
\text{unfold}(\text{fold}(k)) & \Downarrow v
\end{align*}
\]

which contradicts the existence of a minimal derivation of

\[
\text{unfold}^{\tau}(\text{fold}^{\tau}(k))\text{fold}^{\tau}(k) \Downarrow v.
\]

So, with respect to the Kleene preorder, unfold(fold(k))fold(k) is the least element of Exp\(_\sigma\) vacuously. It then follows from Proposition 3.9 that unfold(fold(k))fold(k) is the least element of Exp\(_\sigma\) with respect to the contextual preorder. It then follows from the \(\beta\)-equality (3.30) that

\[
\perp_\sigma := \text{fix} \lambda x^\sigma. x
\]

\[
=\sigma (\lambda x^\sigma. x)(k\text{fold}(k))
\]

\[
=\sigma k(\text{fold}(k))
\]

\[
=\sigma (\lambda x^\sigma. x)(\text{unfold}(\text{fold}(k))\text{fold}(k))
\]

\[
=\sigma \text{unfold}(\text{fold}(k))\text{fold}(k).
\]

Consequently, by the transitivity of \(\subseteq_\sigma\), it follows that \(\perp_\sigma\) is the least element of Exp\(_\sigma\) with respect to the contextual preorder.

### 3.5.7. Rational-chain completeness and continuity

In addition to the fixed-point property (3.48), terms of the form fix\(_\sigma\)(f) enjoy a least prefixed-point property below.

If \(f \in \text{Exp}_{\sigma \rightarrow \sigma}\) and \(t \in \text{Exp}_{\sigma}\), then

\[
f(t) \subseteq_\sigma t \quad \Rightarrow \quad \text{fix}_\sigma(f) \subseteq_\sigma t. \quad (3.50)
\]

In fact, the above least prefixed-point property follows from a more general property which we explain below.

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

\[
g(\perp_\sigma) \subseteq_\tau gh(\perp_\sigma) \subseteq_\tau gh^{(2)}(\perp_\sigma) \subseteq_\tau \ldots,
\]

where \(g : \sigma \rightarrow \tau\) and \(h : \tau \rightarrow \tau\) are some function-type FPC closed terms.

Crucially, the following rational chain completeness property holds:
Theorem 3.12. For any rational chain of the form
\[ g(\bot_\sigma) \sqsubseteq_\tau gh(\bot_\sigma) \sqsubseteq_\tau gh^{(2)}(\bot_\sigma) \sqsubseteq_\tau \ldots, \]
where \( g : \sigma \to \tau \) and \( h : \sigma \to \sigma \) are some function-type FPC closed terms, it holds that
\[ \bigsqcap_n g(f^{(n)}(\bot_\sigma)) = \sigma g(\text{fix}_\sigma(f)). \]

Proof. An operational proof of this can be found on pp. 81–91 of [Ho, 2006b].

A handy result we often make use of is Plotkin’s uniformity principle.

Lemma 3.13. (Plotkin’s uniformity principle)
Let \( f : \sigma \to \sigma \), \( g : \tau \to \tau \) be FPC programs and \( h : \sigma \to \tau \) be a strict program, i.e., \( h(\bot_\sigma) = \bot_\tau \), such that the following diagram
\[
\begin{array}{ccc}
\sigma & \xrightarrow{h} & \tau \\
\downarrow{f} & & \downarrow{g} \\
\sigma & \xrightarrow{h} & \tau \\
\end{array}
\]
commutes, i.e., \( g \circ h = h \circ f \). Then it holds that
\[ \text{fix}(g) = h(\text{fix}(f)). \]

Proof. Using rational-chain completeness, rational continuity, and the identity \( h \circ f = g \circ h \) in turn, it follows that
\[
h(\text{fix}(f)) = h(\bigsqcap_n f^{(n)}(\bot_\sigma)) = \bigsqcap_n h \circ f^{(n)}(\bot_\sigma) = \bigsqcap_n g^{(n)} \circ h(\bot_\sigma) = \bigsqcap_n g^{(n)}(\bot_\tau) = \text{fix}(g).
\]

4. FPC considered as a category
Business proper starts 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 demonstrate how FPC types-in-context can be viewed as functors.

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

4.1. Realisable actions and functors

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

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

(2) A given type variable may occur as covariant in one position and contravariant in another within the same type expression. For example, the type variable $X$ in $X \rightarrow X$ is contravariant in the first slot and covariant in the second.

The usual solution to this problem of mixed variance, following Freyd (1992), is to work with the product category $\text{FPC}_i^{\text{op}} \times \text{FPC}_i$. In this section, we do not do so but instead work with a full subcategory, $\text{FPC}_i^{\delta}$, of this. Define $\text{FPC}_i^{\delta}$, the diagonal category, to be the full subcategory of $\text{FPC}_i^{\text{op}} \times \text{FPC}_i$ whose objects are those of $\text{FPC}_i$ and whose morphisms are pairs of $\text{FPC}_i$-morphisms, denoted by $u : \sigma \rightarrow \tau$ (or sometimes more explicitly as $\langle u^-, u^+ \rangle$), of the form:

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

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

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

$$f : R \rightarrow S \quad \text{for} \quad f^+ : R \Rightarrow S : f^-;$$
$$\bar{f} : \bar{R} \rightarrow \bar{S} \quad \text{for} \quad f_i^+ : R_i \Rightarrow S_i : \bar{f}_i, \quad i = 1, 2, \ldots, n.$$ We define $\text{FPC}$ type expressions as functors in two steps. The first step involves showing how $\text{FPC}$ type expressions define syntactic operations called actions and proving that actions preserve composition of $\text{FPC}_i^{\delta}$-morphisms. The second step involves establishing an operational minimal invariance theorem, and proving that actions defined by $\text{FPC}$ type expressions do preserve identity morphisms and are thus functors. We carry out the first step in this section. Because of heavy operational machineries involved in proving the operational minimal invariance theorem, we choose to postpone the second step to the next section.

Definition 4.2. Let $C$ and $D$ be categories. An action $F : C \rightarrow D$ is an operation which assigns

(1) an object $F(C) \in D$ to an object $C \in C$, and

\[\tag{4.2}\]

\[\overset{\text{†}}{\text{We shall consider the product category } \text{FPC}_i^{\text{pp}} \times \text{FPC}_i \text{ in Section 6.}}\]
(2) a $D$-morphism $F(f)$ to a $C$-morphism $f : C \rightarrow D$
and preserves composition of $C$-morphisms, i.e., for any $C$-morphisms $f : C_1 \rightarrow C_2$ and 
g : $C_2 \rightarrow C_3$, it holds that

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

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

**Definition 4.3.** A realisable action $T : (\text{FPC}^\delta)^n \rightarrow \text{FPC}^\delta$ is an action which is
realised by:

1. a type-in-context $\vec{X} \vdash \tau$; and
2. a pair of terms-in-context of the form:

$\vec{R}, \vec{S} ; \vec{f} : \vec{R} \rightarrow \vec{S} | t^\rightarrow : \tau[\vec{S}/\vec{X}] \rightarrow \tau[\vec{R}/\vec{X}]$ and $t^\rightarrow : \tau[\vec{R}/\vec{X}] \rightarrow \tau[\vec{S}/\vec{X}]$

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

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

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

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

A realisable action which is also a functor is called a realisable functor.

**Remark 4.4.**

1. Let $\vec{u}, \vec{v} \in (\text{FPC}^\delta)^n(\vec{\rho}, \vec{\sigma})$ be given and suppose that $\vec{u} \sqsubseteq \vec{v}$. Then by monotonicity
   (i.e., Property (3.28) in Section 3), any realisable action is locally monotone in the sense that
   $T(\vec{u}) \sqsubseteq T(\vec{v})$.
2. Let $\vec{u}_k \in (\text{FPC}^\delta)^n(\vec{\rho}, \vec{\sigma})$ be rational chains. Then by rational continuity, any realisable action is
   locally continuous in the sense that $T(\bigsqcup_k \vec{u}_k) = \bigsqcup_k T(\vec{u}_k)$.

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

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

$\vec{R}, \vec{S} ; \vec{f} : \vec{R} \rightarrow \vec{S} | f_i^\rightarrow : S_i \rightarrow R_i, f_i^\rightarrow : R_i \rightarrow S_i$.

If $\vec{X} \vdash Y$ where $Y \not\in \{X_1, \ldots, X_n\}$, define $F_{\vec{X} \vdash Y} = \text{id}_Y$ whose morphism part is
realised by $\vec{R}, \vec{S} ; \vec{f} : \vec{R} \rightarrow \vec{S} | \text{id}_Y : Y \equiv Y : \text{id}_Y$. Clearly, $F_{\vec{X} \vdash X_i}$ and $F_{\vec{X} \vdash Y}$
preserve composition of $\text{FPC}^\delta$-morphisms.

Next, we perform a construction that is simultaneously associated to an inductive proof
of correctness, i.e., the subsequent syntactic operations described are indeed actions. For
this purpose, we assume the following induction hypotheses:

(a) $F_{\vec{X} \vdash \tau_i}$ ($i = 1, 2$), $F_{\vec{X} \vdash \tau}$ and $F_{\vec{X}, \vec{X} \vdash \sigma}$ are the realisable actions corresponding to $\vec{X} \vdash \tau_i$
($i = 1, 2$), $\vec{X} \vdash \tau$ and $\vec{X}, \vec{X} \vdash \sigma$ respectively. In particular, these realisable actions
preserve composition.
(b) The morphism parts of these realisable actions are realised by

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t_i : \tau_i[\overline{R}/\overline{X}] \rightarrow \tau_i[\overline{S}/\overline{X}] \quad (i = 1, 2) \]

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t : \tau[\overline{R}/\overline{X}] \rightarrow \tau[\overline{S}/\overline{X}] \text{, and} \]

\[ \overline{R}, \overline{R}, \overline{S}, \overline{f} : \overline{R} \rightarrow \overline{S}, \overline{f} : \overline{R} \rightarrow \overline{S} \vdash s : \sigma[\overline{R}/\overline{X}, R/X] \rightarrow \sigma[\overline{S}/\overline{X}, S/X]. \]

(2) Product type.

The morphism part of \( F \) is realised by

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t : (\tau_1 \times \tau_2)[i\overline{R}/\overline{X}] \rightarrow (\tau_1 \times \tau_2)[i\overline{S}/\overline{X}], \]

where

\[ t^- := \lambda p : (\tau_1 \times \tau_2)[\overline{S}/\overline{X}], \text{ case}(x) \text{ of } \text{inl}(x). \text{inl}(t^-_1(x)) \text{ or } \text{inr}(t^-_2(y)) \]

\[ t^+ := \lambda q : (\tau_1 \times \tau_2)[\overline{R}/\overline{X}], \text{ case}(x) \text{ of } \text{inl}(x). \text{inl}(t^+_1(x)) \text{ or } \text{inr}(t^+_2(y)). \]

Note that \( F_{X^{\downarrow} \times Y^{\downarrow}} \) preserves composition of \( \text{FPC}_{\downarrow} \)-morphisms since \( F_{X^{\downarrow}} \)'s do by induction hypothesis.

The proof that \( F_{X^{\downarrow} \times Y^{\downarrow}} \) preserves composition exploits some operational equalities (e.g., Property (3.30)) from Section 3. This is also true, though kept implicit, of the proof of preservation of composition for the various \( F \)'s in the remaining cases below.

(3) Sum type.

The morphism part of \( F_{X^{\downarrow} + Y^{\downarrow}} \) is realised by

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t : (\tau_1 + \tau_2)[i\overline{R}/\overline{X}] \rightarrow (\tau_1 + \tau_2)[i\overline{S}/\overline{X}], \]

where

\[ t^- := \lambda z : (\tau_1 + \tau_2)[\overline{S}/\overline{X}], \text{ case}(z) \text{ of } \text{inl}(x). \text{inl}(t^-_1(x)) \text{ or } \text{inr}(t^-_2(y)) \]

\[ t^+ := \lambda z : (\tau_1 + \tau_2)[\overline{R}/\overline{X}], \text{ case}(z) \text{ of } \text{inl}(x). \text{inl}(t^+_1(x)) \text{ or } \text{inr}(t^+_2(y)). \]

Note that \( F_{X^{\downarrow} + Y^{\downarrow}} \) preserves composition of \( \text{FPC}_{\downarrow} \)-morphisms since \( F_{X^{\downarrow}} \)'s do by induction hypothesis.

(4) Lifted type.

The morphism part of \( F_{X^{\downarrow} \parallel Y^{\downarrow}} \) is realised by

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t' : \tau_\parallel[\overline{R}/\overline{X}] \rightarrow \tau_\parallel[\overline{S}/\overline{X}], \]

where

\[ (t')^- := \lambda z.\tau_\parallel[\overline{S}/\overline{X}], \text{ case}(z) \text{ of } \text{up}(x). \text{up}(t^-_1(x)) \]

\[ (t')^+ := \lambda z.\tau_\parallel[\overline{R}/\overline{X}], \text{ case}(z) \text{ of } \text{up}(x). \text{up}(t^+_1(x)). \]

Note that \( F_{X^{\downarrow} \parallel Y^{\downarrow}} \) preserves composition of \( \text{FPC}_{\downarrow} \)-morphisms since \( F_{X^{\downarrow}} \) does by induction hypothesis.

(5) Function type.

The morphism part of \( F_{X^{\downarrow} \rightarrow Y^{\downarrow}} \) is given by:

\[ \overline{R}, \overline{S}; \overline{f} : \overline{R} \rightarrow \overline{S} \vdash t : (\tau_1 \rightarrow \tau_2)[\overline{R}/\overline{X}] \rightarrow (\tau_1 \rightarrow \tau_2)[\overline{S}/\overline{X}] \]
where

\[
\begin{align*}
  t^- & \defeq \lambda h : (\tau_1 \to \tau_2)[\bar{S}/\bar{X}], \lambda x : \tau_1[\bar{R}/\bar{X}], t_2^-(h(t_1^+(x))) \\
  t^+ & \defeq \lambda g : (\tau_1 \to \tau_2)[\bar{R}/\bar{X}], \lambda y : \tau_1[\bar{S}/\bar{X}], t_2^+(g(t_1^+(y))).
\end{align*}
\]

Note that \( F_{\bar{X}^{\rightarrow \tau_1 \rightarrow \tau_2}} \) preserves composition of \( \text{FPC}_{\delta} - \)morphisms since \( F_{\bar{X}^{\rightarrow \tau_i}} \) (\( i = 1, 2 \)) do by induction hypothesis.

(6) Recursive type.

The morphism part of \( F_{\bar{X}^{\rightarrow \mu X}.\sigma} \) is realised by

\[
\tilde{R}, \tilde{S} ; \tilde{f} : \tilde{R} \to \tilde{S} \vdash \text{fix}(\lambda v. i^{-1}_{\tilde{X}, \tilde{S}} \cdot s[v/f] \circ i_{\tilde{X}, \tilde{S}}),
\]

where \( i := (\text{fold}, \text{unfold}) \).

By Lemma 4.6 below, \( F_{\bar{X}^{\rightarrow \mu X}.\sigma} \) preserves composition of \( \text{FPC}_{\delta} - \)morphisms.

Remark 4.5. Very often, arguments concerning actions defined by FPC type expressions are best carried out with the help of commutative diagrams. To illustrate this, let us consider the action \( \text{FPC}_{\delta} \) and any \( \bar{u} \in (\text{FPC}_{\delta})^n(\bar{\rho}, \bar{\sigma}) \). The action \( F_{\bar{X}^{\rightarrow \mu X}.\sigma}(\bar{u}) \) is, by the above inductive definition (6), the least morphism \( v \) such that the following diagram commute:

\[
\begin{array}{ccc}
  F_{\bar{X}^{\rightarrow \mu X}.\sigma}(\bar{\rho}) & \xrightarrow{v} & F_{\bar{X}^{\rightarrow \mu X}.\sigma}(\bar{\sigma}) \\
  i_{\tilde{X}, \tilde{S}}(\bar{\rho}) & \xrightarrow{F_{\bar{X}^{\rightarrow \mu X}.\sigma}(\bar{u},v)} & i_{\tilde{X}, \tilde{S}}(\bar{\sigma})
\end{array}
\]

Lemma 4.6. The action \( F_{\bar{X}^{\rightarrow \mu X}.\sigma} \) preserves composition of morphisms.

Proof. Denote \( F_{\bar{X}^{\rightarrow \mu X}.\sigma} \) by \( S \) and \( F_{\bar{X}^{\rightarrow \mu X}.\sigma} \) by \( T \). To show that \( S \) preserves composition, we must prove that for any morphism pairs \( \bar{u} \in (\text{FPC}_{\delta})^n(\bar{\rho}, \bar{\sigma}) \) and \( \bar{v} \in (\text{FPC}_{\delta})^n(\bar{\sigma}, \bar{\tau}) \), it holds that

\[
S(\bar{u}) \circ S(\bar{v}) = S(\bar{v} \circ \bar{u}).
\]

We denote \( (\text{fold}, \text{unfold}) \) by \( i \). Define two programs as follows:

\[
\begin{align*}
  \Psi_1 & \defeq (S(\bar{\tau}) \to S(\bar{\sigma})) \times (S(\bar{\sigma}) \to S(\bar{\tau})) \to (S(\bar{\tau}) \to S(\bar{\sigma})) \times (S(\bar{\sigma}) \to S(\bar{\tau})) \\
  \Psi_1 & \defeq \lambda (a, b). i^{-1}_{S(\bar{\sigma})} \circ T(\bar{v}, (a, b)) \circ i_{S(\bar{\sigma})} \\
  \Psi_2 & \defeq (S(\bar{\tau}) \to S(\bar{\rho})) \times (S(\bar{\rho}) \to S(\bar{\tau})) \to (S(\bar{\tau}) \to S(\bar{\rho})) \times (S(\bar{\rho}) \to S(\bar{\tau})) \\
  \Psi_2 & \defeq \lambda (c, d). i^{-1}_{S(\bar{\rho})} \circ T(\bar{v} \circ \bar{u}, (c, d)) \circ i_{S(\bar{\rho})}
\end{align*}
\]
Then the following diagram

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

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

\[
\Psi_1(a, b) \circ S(\overrightarrow{u}) = i^{\overrightarrow{u}} - 1 \circ S(\overrightarrow{\sigma}) \circ i \circ S(\overrightarrow{\rho})
\]

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

\[- \circ S(\overrightarrow{u}) := \lambda(a, b). (S(\overrightarrow{u})^- \circ a, b \circ S(\overrightarrow{u})^+)\]

is strict. Therefore, by Lemma 3.13 we have

\[\text{fix}(\Psi_2) = \text{fix}(\Psi_1) \circ S(\overrightarrow{u})\]

i.e., \(S(\overrightarrow{v}) \circ S(\overrightarrow{u}) = S(\overrightarrow{v} \circ \overrightarrow{u})\).

5. Operational minimal invariance

To prove that realisable actions arising from types-in-context are actually functors (in that they preserve identity \(\text{FPC}_!^\delta\)-morphisms), the following theorem is a crucial result.

**Theorem 5.1.** (Operational minimal invariance for realisable functors)

Let \(T : (\text{FPC}_!^\delta)^{n+1} \to \text{FPC}_!^\delta\) be a realisable functor and \(\overrightarrow{\sigma} \in (\text{FPC}_!^\delta)^n\). We write \(S(\overrightarrow{\sigma})\) for \(\mu X. T(\overrightarrow{\sigma}, X)\). Then, the least \(\text{FPC}_!^\delta\)-endomorphism

\[e : S(\overrightarrow{\sigma}) \to S(\overrightarrow{\sigma})\]

for which the following commutes

\[
\begin{array}{ccc}
S(\overrightarrow{\sigma}) & \xrightarrow{e} & S(\overrightarrow{\sigma}) \\
\downarrow i_S(\overrightarrow{\sigma}) & & \downarrow i_S(\overrightarrow{\sigma}) \\
T(\overrightarrow{\sigma}, S(\overrightarrow{\sigma})) & \xrightarrow{T(\text{id}_{\overrightarrow{\sigma}}, e)} & T(\overrightarrow{\sigma}, S(\overrightarrow{\sigma}))
\end{array}
\]
is the identity morphism \( \langle \text{id}_{S(\vec{\sigma})}, \text{id}_{S(\vec{\sigma})} \rangle \), where \( i := \langle \text{fold}, \text{unfold} \rangle \). Moreover, the identity is the only such endomorphism. Consequently, \( S \) preserves identity morphisms, i.e.,

\[
S(\langle \text{id}_{\vec{\sigma}}, \text{id}_{\vec{\sigma}} \rangle) = \langle \text{id}_{S(\vec{\sigma})}, \text{id}_{S(\vec{\sigma})} \rangle.
\]

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

5.1. Twin morphisms

First, we need the following definition:

**Definition 5.2.** An FPC\(^{\delta}\)-morphism is said to be twin if it is of the form

\[
u : \sigma \leftrightarrow \sigma : u,
\]
i.e., \( u^- = u^+ = u \).

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

**Lemma 5.3.** Let \( \vec{X} \vdash \tau \) be a type-in-context and \( F_{\vec{X} \vdash \tau} \) the realisable action associated to it. Then, for any \( \vec{\sigma} \in (\text{FPC}\^{\delta})^n \) and for any sequence of twin morphisms \( \vec{u} \in (\text{FPC}\^{\delta})^n(\vec{\sigma}, \vec{\sigma}) \) (i.e., \( u_i : \sigma_i \leftrightarrow \sigma_i : u_i \ (i = 1, \ldots, n) \)), the morphism \( F_{\vec{X} \vdash \tau}(\vec{u}) \) is again twin.

**Proof.** By induction on the structure of \( \vec{X} \vdash \sigma \).

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

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

\[
\begin{array}{c}
S(\vec{\sigma}) \xrightarrow{t} S(\vec{\sigma}) \\
\downarrow i_{S(\vec{\sigma})} \downarrow i_{S(\vec{\sigma})} \\
T(\vec{\sigma}, S(\vec{\sigma})) \rightarrow T(\vec{\sigma}, S(\vec{\sigma}))
\end{array}
\]

commutes. Here, we denote \( \langle \text{fold}, \text{unfold} \rangle \) 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 \( t = \text{fix}(\phi) \). On the other hand, \( \text{fix}(\phi) = \bigsqcup_n \phi^{(n)}(\perp, \perp) \) by rational completeness. A further induction on \( n \) then shows that \( \phi^{(n)}(\perp, \perp) \) is twin for every \( n \in \mathbb{N} \). The proof is easy. For \( n = 0 \), we have the trivial twin \( (\perp, \perp) \).
Assume that $\phi^{(n)}(\bot, \bot)$ is twin for $n \in \mathbb{N}$. We wish to prove that $\phi^{(n+1)}(\bot, \bot)$ is twin. Since $\bar{u}$ and $\phi^{(n)}(\bot, \bot)$ are twin, the pairing $\bar{v} := (\bar{u}, \phi^{(n)}(\bot, \bot))$ is twin. Invoking the earlier induction hypothesis that $T$ preserves twin, it is clear that $T(\bar{u}, \phi^{(n)}(\bot, \bot))$ is twin.

We denote this morphism by $\langle f, f \rangle$. Thus, the morphism $\phi^{(n+1)}(\bot, \bot) = i^{-1} \circ \langle f, f \rangle \circ i$, illustrated as follows,

\[
\begin{array}{c}
\text{S(\bar{\sigma})} \\
\text{unfold} \\
\text{T(\bar{\sigma}, S(\bar{\sigma}))} \\
\text{f} \\
\text{T(\bar{\sigma}, S(\bar{\sigma}))} \\
\text{fold} \\
\text{unfold} \\
\text{S(\bar{\sigma})}
\end{array}
\]

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

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

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

**Lemma 5.5.** Let $F_{\Theta \vdash \tau}$ be the realisable action associated to $\Theta \vdash \tau$. Then for any sequence of $n$ closed types $\bar{\sigma}$, it holds that

$F_{\Theta \vdash \tau}(\text{id}_\sigma : \bar{\sigma} \rightleftharpoons \bar{\sigma} : \text{id}_\sigma) \subseteq (\text{id}_{F_{\Theta \vdash \sigma}}, \text{id}_{F_{\Theta \vdash \sigma}})$.

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

### 5.2. Fischer-Ladner closure

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

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

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

A $\mu$-subexpression $\tau$ of $\sigma$ is said to be maximal if no such sequence above exists.

**Definition 5.7.** If $V_\tau = X_1, \cdots, X_n$, let $\mu V_\tau$ denote the sequence $\mu X_1, \cdots, \mu X_n$. Define the map $e$ on subexpressions of $\sigma$ by

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

An Operational Domain-theoretic Treatment of Recursive Types

where \( \sigma \ll \tau \).

The Fischer-Ladner closure (or simply, FL-closure) of \( \sigma \) is the range of \( e \), i.e.,

\[
\text{CL}(\sigma) := \{ e(\tau) \mid \sigma \ll \tau \}.
\]

Those elements of \( \text{CL}(\sigma) \) which are \( \mu \)-expressions (i.e., types of the form \( \mu X. \rho \)) are called the associates of \( \sigma \).

Clearly, \( e(\tau) \) is a closed FPC type since if \( X \) occurs free in \( \tau \), then \( \mu X \ll \tau \). From Definition 5.7, it is clear that

**Proposition 5.8.** \( \text{CL}(\sigma) \) is a finite set.

The FL-closure of an FPC closed type can be characterized by the following:

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

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

**Example 5.10.** Consider the closed type

\[
\sigma = (\mu X_1. \mu X_2. (X_1 \times X_2)) \rightarrow (\mu X_3. (X_3 + \mu X_4. (X_4 + X_3))).
\]

Then the elements of \( \text{CL}(\sigma) \) are given below:

1. \( \rho_0 := \sigma \)
2. \( \rho_1 := \mu X_1. \mu X_2. (X_1 \times X_2) \)
3. \( \rho_3 := \mu X_3. (X_3 + \mu X_4. (X_4 + X_3)) \)
4. \( \rho_2 := \mu X_2. (\rho_1 \times X_2) \)
5. \( \rho_4 := \mu X_4. (X_4 + \rho_3) \)
6. \( \rho_1 \times \rho_2 = (\rho_1 \times X_2)[\rho_2/X_2] \)
7. \( \rho_3 + \rho_4 = (X_3 + \mu X_4. (X_4 + X_3))[\rho_3/X_3] \)
8. \( \rho_4 + \rho_3 = (X_4 + \rho_3)[\rho_4/X_4] \)

The associates of \( \sigma \) are those \( \mu \)-expressions in the preceding list, i.e., \( \rho_i \) (i = 1, 2, 3, 4).

A direct consequence of Proposition 5.8 is the following:

**Theorem 5.11.** (Mutual definitions theorem)

Let \( \sigma \) be an FPC closed type and \( \rho_1, \ldots, \rho_n \) its associates. Then for every associate \( \rho_i := \mu X_i. \sigma_i \) (i = 1, \ldots, n), there exists a \( \mu \)-free type-in-context \( Y_1, \ldots, Y_n \vdash \beta_i \) such that

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

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 5.12.** A pre-deflation on a type \( \sigma \) is an element of type \( (\sigma \to \sigma) \) that is (i) idempotent and (ii) below the identity. A deflation on a type \( \sigma \) 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 \( \sigma \) is a rational chain \( \iden_n^\sigma \) of idempotent pre-deflations (resp. deflations) with \( \bigcup_n \iden_n^\sigma = \iden_\sigma \).

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 \( \varpi \) (cf. Subsection 2.4) by \( \varpi = \mu X. X \). Using \( \varpi \), we first define the programs \( e : \varpi \to (\sigma \to \sigma) \) by induction on \( \sigma \) as follows:

\[
\begin{align*}
\sigma^X_\tau(n)(p) &= (\sigma(n)(\text{fst}(p)), \sigma(n)(\text{snd}(p))) \\
\sigma^{+\tau}(n)(z) &= \text{case}(z) \text{ of inl}(x). \sigma(n)(x) \text{ or inr}(y). \sigma(n)(y) \\
\sigma^\bot(n)(z) &= \text{case}(z) \text{ of up}(x). \text{up}(\sigma(n)(x)) \\
\sigma^{+\top}(n)(f) &= \sigma(n) \circ f \circ \sigma(n)
\end{align*}
\]

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

Recall that \( F_{\mu X. \sigma} : \text{FPC}^\varpi \to \text{FPC}^\varpi \) is the realisable action associated to the type-in-context \( X \vdash \sigma \). By Lemma 5.3, we may abuse notation by writing \( (F_{\mu X. \sigma}(e^{\mu X. \sigma}(n)), e^{\mu X. \sigma}(n)))^+ \) as \( F_{\mu X. \sigma}(e^{\mu X. \sigma}(n)) \). Define

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

Notice that the definition of \( e^{\mu X. \sigma} \) is a simple recursive definition given explicitly by using fix. Clearly, \( e^{\mu X. \sigma} \) satisfies the following equations:

\[
\begin{align*}
e^{\mu X. \sigma}(0) &= \bot_{\mu X. \sigma} \quad (5.1) \\
e^{\mu X. \sigma}(n + 1) &= \text{fold} \circ F_{\mu X. \sigma}(e^{\mu X. \sigma}(n)) \circ \text{unfold}. \quad (5.2)
\end{align*}
\]

Relying on \( \varpi \) again, we next define the programs \( d^\sigma : \varpi \to (\sigma \to \sigma) \) by induction on \( \sigma \) as follows:

\[
\begin{align*}
d^X_\tau(n)(p) &= (d^\sigma(n)(\text{fst}(p)), d^\sigma(n)(\text{snd}(p))) \\
d^{+\tau}(n)(z) &= \text{case}(z) \text{ of inl}(x). d^\sigma(n)(x) \text{ or inr}(y). d^\sigma(n)(y) \\
d^\bot(n)(z) &= \text{case}(z) \text{ of up}(x). \text{up}(d^\sigma(n)(x)) \\
d^{+\top}(n)(f) &= d^\sigma(n) \circ f \circ d^\sigma(n)
\end{align*}
\]

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

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

Note that \( d^{\mu X. \sigma} \) satisfies the following equations:

\[
\begin{align*}
d^{\mu X. \sigma}(0) &= \bot_{\mu X. \sigma} \\
d^{\mu X. \sigma}(n + 1) &= \text{fold} \circ d^{\sigma^{\mu X. \sigma/X}(n)} \circ \text{unfold} . \quad (5.3)
\end{align*}
\]
At first glance, it looks as if the definition of $d_{\mu X. \sigma}$ is not a recursive one, i.e., it is not of the form $\text{fix}(t)$ for some term $t$ because the type-index $\sigma[\mu X. \sigma/X]$ appears in the right hand term of Equation (5.4). However, by Theorem 5.11 since a single unfolding of each associate can be obtained via an application of (a realisable action arising from) a $\mu$-free type-in-context to all the associates, $d_{\mu X. \sigma}$ can then be defined using a finite system of mutual recursions. We make this precise here:

Lemma 5.13. Let $\mu X_1. \sigma_1$ be a given closed type whose associates are denoted by $\rho_i = \mu X_i. \sigma_i$ ($i = 1, \ldots, n$). Then, $d_{\mu X_1. \sigma_1}(n)$ can be defined by a finite system of mutual recursions involving some $d_{\rho_i}(j)$’s, where $0 \leq j \leq n$.

Proof. Let $\rho_1, \ldots, \rho_n$ be the associates of $\emptyset \vdash \tau_1$, setting $\rho_1 := \tau_1$. By Theorem 5.11, for each $i = 1, \ldots, n$, $\sigma_i[\rho_i/X_i] = \beta_i[\vec{\rho}/\vec{Y}]$ for some $\mu$-free type-in-context $\vec{Y} = Y_1, \ldots, Y_n \vdash \beta_i$. It follows that

\[
\begin{align*}
d_{\rho_1}(k) &= \text{if } k > 0 \text{ then } \text{fold} \circ (F_{\vec{Y}, \beta_1}(d_{\rho_1}(k), \ldots, d_{\rho_n}(k))) + \circ \text{unfold}, \\
d_{\rho_2}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_1 \times \rho_2}(n - 1) \circ \text{unfold}, \\
d_{\rho_3}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_3 + \rho_4}(n - 1) \circ \text{unfold}, \\
d_{\rho_4}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_4 + \rho_3}(n - 1) \circ \text{unfold}.
\end{align*}
\]

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

Example 5.14. Consider again $\emptyset \vdash \tau_0 := (\mu X_1. \mu X_2. (X_1 \times X_2)) \rightarrow (\mu X_3. X_3 + \mu X_4. (X_4 + X_3))$ as before. Then the deflations of the associates derived from $\tau_0$ can be mutually defined via simple recursion:

\[
\begin{align*}
d_{\rho_1}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_2}(n - 1) \circ \text{unfold}, \\
d_{\rho_2}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_1 \times \rho_2}(n - 1) \circ \text{unfold}, \\
d_{\rho_3}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_3 + \rho_4}(n - 1) \circ \text{unfold}, \\
d_{\rho_4}(n) &= \text{if } n > 0 \text{ then } \text{fold} \circ d_{\rho_4 + \rho_3}(n - 1) \circ \text{unfold}.
\end{align*}
\]

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

\[
d_{\mu X. \sigma}(\infty) = \text{fold} \circ d_{\sigma}[\mu X. \sigma/X](\infty) \circ \text{unfold}.
\]

Proof. Obvious since $\infty = \sigma \infty + 1$.

We now establish the order-theoretic relation between the families of functions $d_{\sigma}$ and $e_{\sigma}$ as follows:

Theorem 5.16. For every finite $n \in \mathbb{N}$, for every closed FPC types $\sigma$, it holds that

\[
d_{\sigma}(n) \sqsubseteq e_{\sigma}(n) \sqsubseteq \text{id}_\sigma
\]

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

Proof. For the base case $n = 0$, it is in fact the case that

\[
d_{\sigma}(0) = e_{\sigma}(0) = \bot_{\sigma \rightarrow \sigma} \sqsubseteq \text{id}_\sigma.
\]
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,

\[
\text{d}^{\sigma + \tau}(0)(z) = \text{case}(z) \text{ of inl}(x). \text{d}^{\sigma}(0)(x) \text{ or inr}(y). \text{d}^{\tau}(0)(y) \quad \text{(defn. of d}^{\sigma + \tau})
\]

\[
= \text{case}(z) \text{ of inl}(x). \bot_{\sigma \rightarrow \sigma}(x) \text{ or inr}(y). \bot_{\tau \rightarrow \tau}(y) \quad \text{(Ind. hyp.)}
\]

Similarly, for $e^{\sigma + \tau}(0)(z)$. The other non-recursive types are proven likewise.

Now we focus on the recursive types:

(Recursive type) $\mu X. \sigma$:

From Equations (5.1 – 5.2), it follows immediately that

\[
\text{d}^{\mu X. \sigma}(0) = \bot_{\mu X. \sigma \rightarrow \mu X. \sigma} = e^{\mu X. \sigma}(0) \subseteq \text{id}_{\mu X. \sigma}.
\]

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:

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

We want to show that

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

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 $\text{d}^{\sigma}(k + 1) = (\text{d}^{\sigma_1}(k + 1), \text{d}^{\sigma_2}(k + 1))$. By induction hypotheses, since $\text{d}^{\sigma_1}(k + 1) \sqsubseteq e^{\sigma_1}(k + 1) \sqsubseteq \text{id}_{\sigma_1}$ and $\text{d}^{\sigma_2}(k + 1) \sqsubseteq e^{\sigma_2}(k + 1) \sqsubseteq \text{id}_{\sigma_2}$, it follows that

\[
\text{d}^{\sigma}(k + 1) = (\text{d}^{\sigma_1}(k + 1), \text{d}^{\sigma_2}(k + 1))
\]

\[
\subseteq (e^{\sigma_1}(k + 1), e^{\sigma_2}(k + 1))
\]

\[
\subseteq e^{\sigma}(k + 1)
\]

\[
= (e^{\sigma_1}(k + 1), e^{\sigma_2}(k + 1))
\]

\[
\subseteq (\text{id}_{\sigma_1}, \text{id}_{\sigma_2})
\]

\[
\subseteq \text{id}_{\sigma}.
\]

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 $\tau$.

Case I(1): Type variable.

This case concerns $\tau = X$.

The proof is straightforward as follows:

\[
\text{d}^{\mu X. X}(k + 1) = \text{fold} \circ \text{d}^{\mu X. X}(k) \circ \text{unfold} \quad \text{(defn. of d}^{\mu X. X})
\]

\[
\subseteq \text{fold} \circ e^{\mu X. X}(k) \circ \text{unfold} \quad \text{(Ind. hyp.)}
\]

\[
= e^{\mu X. X}(k + 1) \quad \text{(defn. of e}^{\mu X. X})
\]
Case I(2): Product type.

This case concerns \( \tau = \tau_1 \times \tau_2 \).

The proof proceeds as follows:

\[
\begin{align*}
d^{\mu X} \cdot \tau (k + 1) &= d^{\mu X} \cdot \tau_1 \times \tau_2 (k + 1) \\
&= \text{fold} \circ d^{(\tau_1 \times \tau_2)/X}(k) \circ \text{unfold} \\
&= \text{fold} \circ (d^{\tau_1/X}(k), d^{\tau_2/X}(k)) \circ \text{unfold} \\
&\subseteq \text{fold} \circ (\rho_1^{\tau_1/X}(k), \rho_2^{\tau_2/X}(k)) \circ \text{unfold} \\
&= \text{fold} \circ e^{(\tau_1 \times \tau_2)/X}(k) \circ \text{unfold} \\
&= e^{\mu X} \cdot \tau (k + 1)
\end{align*}
\]

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 \( \rho \) 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 \( \rho' \) is not recursive.)

To show that

\[
d^{\mu X} \cdot \mu Y. \rho (k + 1) \subseteq e^{\mu X} \cdot \mu Y. \rho (k + 1)
\]

it will be sufficient if we can establish

\[
d^{\mu Y. \rho[\tau_0/X]}(k) \subseteq F_{X'}^{\mu Y. \rho}(e^{\tau_0/k})
\]

where \( \tau_0 := \mu X. \mu Y. \rho \).

To this end, we first prove that

\[
d^{\mu Y. \rho[\tau_0/X]}(k) \subseteq F_{X'}^{\mu Y. \rho}(d^{\tau_0/k})
\]

Then either \( X, Y \vdash \rho \) is \( \mu \)-free or it is not.

Assume that \( X, Y \vdash \rho \) is \( \mu \)-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 \( \mu \)-free, it follows that

\[
d^{\rho[\tau_0/X, \tau_1/y]}(k - 1) = F_{X,Y}^{\tau_1/y} \circ (\text{fold} \circ d^{\tau_0}(k - 1), d^{\tau_1}(k - 1)).
\]

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

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

where \( R := F_{X,Y}^{\tau_1/y} \). By monotonicity of realisable actions and extensionality properties, it follows from \( d^{\tau_0}(j) \subseteq d^{\tau_0}(j) \) that the above term is below

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

with respect to the contextual pre-order. Thus,

\[
\begin{align*}
\text{fold} \circ d^{\rho[\tau_0/X, \tau_1/y]}(k - 1) \circ \text{unfold} \subseteq \text{fold} \circ s \circ \text{unfold} \subseteq \bigcup_{j \geq 0} \Phi^j (d^{\tau_1}(0))
\end{align*}
\]

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

\[
d^{\mu Y. \rho[\tau_0/X]}(k) \subseteq F_{X'}^{\mu Y. \rho}(d^{\tau_0/k}).
\]
It now remains to settle the case when $X, Y \vdash \rho$ is not $\mu$-free. Since we have assumed that $\rho$ is not recursive, by Lemma 5.13 there exist a unique $\mu$-free type-in-context $\vec{X}, X, Y \vdash \delta$ and maximal recursive types $\alpha_i := \mu X_i, \beta_i$ such that

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

By induction hypothesis, we assume that $d^{\alpha_j}(k-1) \sqsubseteq id_{\alpha_j}$ for all $j = 1, \ldots, n$. Since $F_{\vec{X}, X, Y, \delta}(\id_{\tau_0}, d^{\tau_0}(k-1), d^{\tau_1}(k-1)) \sqsubseteq F_{X, Y, \rho}(d^{\tau_0}(k-1), d^{\tau_1}(k-1))$ and $F_{\vec{X}, X, Y, \delta}$ is monotone, it holds that

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

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

$$d^{\mu Y \cdot \rho^{(\tau_0/X)}}(k) \sqsubseteq F_{X, Y, \rho}(d^{\tau_0}(k)),
$$

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

To complete the proof, we apply the monotonicity of the realisable action $F_{X, Y, \rho}$ and the induction hypothesis that $d^{\tau_0}(k) \sqsubseteq e^{\tau_0}(k) \sqsubseteq id_{\tau_0}$ to obtain the desired result:

$$d^{\mu X \cdot \mu Y \cdot \rho}(k) \sqsubseteq \text{fold} \circ F_{X, Y, \rho}(d^{\tau_0}(k)) \circ \text{unfold}
\sqsubseteq \text{fold} \circ F_{X, Y, \rho}(e^{\tau_0}(k)) \circ \text{unfold}
\sqsubseteq \text{fold} \circ F_{X, Y, \rho}(id_{\tau_0}) \circ \text{unfold}
\sqsubseteq \text{fold} \circ \id_{\mu Y \cdot \rho^{(\tau_0/X)}} \circ \text{unfold}
\sqsubseteq \id_{\mu X \cdot \mu Y \cdot \rho},$$

where the fourth inequality holds by virtue of Lemma 5.5.

**Corollary 5.17.** For any closed type $\sigma$, it holds that

$$d^{\sigma}(\infty) \sqsubseteq e^{\sigma}(\infty) \sqsubseteq id_{\sigma}.$$

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

### 5.4. Compilation and canonical deflationary structure

The focus of this subsection is to prove that the family $d^{\sigma}$ (defined in Subsection 5.3) induces a canonical deflationary structure on each closed FPC type $\sigma$. This means that in addition to showing that for each $n \in \sigma$ and $n < \infty$,

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

we must prove that

$$d^{\sigma}(\infty) = id_\sigma.$$

Note that property (2) has already been shown in Theorem 5.16. Property (1) will be essential in the proof of minimal invariance, and property (3) will not be used.
An Operational Domain-theoretic Treatment of Recursive Types

As a consequence of \( d^\sigma(\infty) = \text{id}_\sigma \) and Theorem 5.16, we have
\[ d^\sigma(\infty) = e^\sigma(\infty) = \text{id}_\sigma \]
and this is crucial in establishing the operational minimal invariance theorem, as we shall explain later.

The following proposition is easy to establish:

**Lemma 5.18.** For any closed FPC type \( \sigma \) and for each \( n \in \omega \) and \( n < \infty \), \( d^\sigma(n) \)

1. is idempotent,
2. \( \sqsubseteq \text{id}_\sigma \) and
3. has finite image modulo contextual equivalence.

In addition, \( d^\sigma(\infty) \) satisfies (1) and (2).

**Proof.** Note that (2) is immediate from Theorem 5.16 (1) and (3) can be established by induction on the structure of closed type \( \sigma \). In particular, (1) is straightforward. For (3), we only give a proof sketch since this property is not used later. The only interesting bit lies in the recursive type. In order to show that \( d^{\mu X. \sigma}(n) \) has finite image modulo contextual equivalence, one invokes Lemma 5.13 to define \( d^{\mu X. \sigma}(n) \) 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 \( d^\sigma \) does indeed define a deflationary structure on each closed FPC type \( \sigma \), we make essential use of the compilation of terms and contexts. This 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 \( \text{Exp}_\sigma(\Gamma) \) is defined by induction on the derivation of \( \Gamma \vdash t : \sigma \), given by the axioms and rules in Figure 6. It is intended that the compilation relation applies canonical deflations \( d(\infty) \) a given FPC term \( t \) (only when an introduction rule is invoked) in a uniform way, and compiles \( t \) to such a resulting term. Motivated by this idea, it comes as no surprise that the compilation relation \( \Rightarrow \) is a function.

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

**Proof.** By induction on the derivation of \( \Gamma \vdash t : \sigma \).

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

**Proof.** By induction on the derivation of \( \Gamma \vdash t : \sigma \Rightarrow \|t\| \). The cases for \( (\Rightarrow \text{var}) \), \( (\Rightarrow \text{pair}) \), \( (\Rightarrow \text{inl}) \), \( (\Rightarrow \text{inr}) \), \( (\Rightarrow \text{abs}) \), \( (\Rightarrow \text{up}) \) and \( (\Rightarrow \text{fold}) \) rely on the idempotence of \( d(\infty) \) (cf. Lemma 5.18) without having to invoke the induction hypothesis. We show the case for \( (\Rightarrow \text{var}) \) here.

Given that \( \Gamma \vdash x : \sigma \Rightarrow \|x\| \). By definition, \( \|x\| = d^\sigma(\infty)(x) \). We are to show that \( \Gamma \vdash d^\sigma(\infty)|x| =_\sigma \|x\| \). But this follows from the idempotence of \( d^\sigma(\infty) \), i.e., \( \Gamma \vdash d^\sigma(\infty)|x| =_\sigma \|x\| \).
\[ \Gamma \vdash x : \sigma \Rightarrow d^\sigma(\infty)(x) \text{ (if } x \in \text{dom}(\Gamma)) \quad (\Rightarrow \text{ var}) \]

\[ \Gamma \vdash s : \sigma \Rightarrow \sigma \quad \Gamma \vdash t : \tau \Rightarrow \tau \quad (\Rightarrow \text{ pair}) \]

\[ \Gamma \vdash (s, t) : \sigma \times \tau \Rightarrow d^{\sigma \times \tau}(\infty)(\sigma, \tau) \]

\[ \Gamma \vdash p : \sigma \times \tau \Rightarrow \tau \quad (\Rightarrow \text{ fst}) \]

\[ \Gamma \vdash s : \sigma \Rightarrow \sigma \quad \Gamma \vdash t : \tau \Rightarrow \tau \quad (\Rightarrow \text{ snd}) \]

\[ \Gamma \vdash \text{inl}(s) : \sigma + \tau \Rightarrow \sigma + \tau \quad \text{(inl)} \]

\[ \Gamma \vdash \text{inr}(s) : \sigma + \tau \Rightarrow \sigma + \tau \quad \text{(inr)} \]

\[ \Gamma \vdash \text{case}(s) \text{ of inl}(x). t_1 \text{ or inr}(y). t_2 \Rightarrow \text{case}(\sigma + \tau) \text{ of inl}(x). t_1 \text{ or inr}(y). t_2 \quad (\Rightarrow \text{ case}) \]

\[ \Gamma \vdash s : \sigma \Rightarrow \tau \Rightarrow \sigma \quad \Gamma \vdash t : \sigma \Rightarrow \tau \quad (\Rightarrow \text{ app}) \]

\[ \Gamma, x : \sigma \vdash t : \sigma \Rightarrow t \quad (\Rightarrow \text{ abs}) \]

\[ \Gamma \vdash \text{up}(t) : \sigma_\bot \Rightarrow d^{\sigma_\bot}(\infty)(\text{up}(t)) \quad (\Rightarrow \text{ up}) \]

\[ \Gamma \vdash \text{case}(s) \text{ of up}(x). t : \tau \Rightarrow \text{case}(\sigma_\bot) \text{ of up}(x). t \quad (\Rightarrow \text{ case up}) \]

\[ \Gamma \vdash t : \mu X. \sigma \Rightarrow t \quad (\Rightarrow \text{ unfold}) \]

\[ \Gamma \vdash \text{fold}(t) : \mu X. \sigma \Rightarrow d^{\text{fold}(t)}(\infty)(\text{fold}(t)) \quad (\Rightarrow \text{ fold}) \]

Fig. 6. Definition of \( \Gamma \vdash t : \sigma \Rightarrow \tau \)
\[d^\tau(\infty)(d^\tau(\infty)(x)) =_{\sigma} d^\tau(\infty)(x) = |x|,\]
The rest of the cases are fairly routine. We now show the case \((\Rightarrow \text{unfold})\) which requires
the expansion of \(d^{\mu X. \sigma}\).

Let \(\Gamma \vdash \text{unfold}(t) : \sigma[\mu X. \sigma/X] \Rightarrow \text{unfold}(t)|\) be given. We must show that
\[
\Gamma \vdash d^{\sigma[\mu X. \sigma/X](\infty)\text{unfold}(t)} =_{\sigma[\mu X. \sigma/X]} \text{unfold}(t).
\]

The inference rule \((\Rightarrow \text{unfold})\)
\[
\frac{\Gamma \vdash t : \mu X. \sigma \Rightarrow \text{unfold}(t)}{\Gamma \vdash \text{unfold}(t) : \sigma[\mu X. \sigma/X] \Rightarrow \text{unfold}(t)}
\]
guarantees that \(|\text{unfold}(t)| \equiv \text{unfold}(|t|)\). The induction hypothesis asserts that \(\Gamma \vdash d^{\mu X. \sigma(|t|)} =_{\mu X. \sigma} |t|\). It then follows that
\[
\begin{align*}
\Gamma \vdash d^{\sigma[\mu X. \sigma/X](\infty)\text{unfold}(t)} &\equiv d^{\sigma[\mu X. \sigma/X](\infty)\text{unfold}(|t|)} \\
&= \text{unfold} \circ d^{\mu X. \sigma}(\infty)(|t|) \quad \text{(def. of } d^{\mu X. \sigma}(\infty)) \\
&= \text{unfold}(|t|) \quad \text{(Ind. hyp.)} \\
&= |\text{unfold}(t)|.
\end{align*}
\]

\textbf{Lemma 5.21.} If \(\Gamma \vdash t : \sigma \Rightarrow |t|\), then \(\Gamma \vdash |t| \subseteq_{\sigma} t\).

\textit{Proof.} By induction on \(\Gamma \vdash t : \sigma \Rightarrow |t|\). The proof for each case typically involves
unwinding of the definition of \(d^{\sigma}(\infty)\) according to the type structure of \(\sigma\), followed by
an application of the previous lemma to the induction hypothesis/hypotheses, and then
the use of inequational logic \([5.28]\). \hfill \Box

\textbf{5.5. Compilation of a context}

One last technical gadget is to compile a context \(C[\neg \sigma] \in \text{Ctx}_{\tau}(\Gamma)\). For a given context
\(C[\neg \sigma] \in \text{Ctx}_{\tau}(\Gamma)\), we define a compiled context \(|C|[\neg \sigma] \in \text{Ctx}_{\tau}(\Gamma)\) using the axioms and
rules similar to those for defining \(\Gamma \vdash t : \sigma \Rightarrow |t|\). The axioms and rules for defining \(|C|\)
are an extension of those in Figure 6 by

— appending a parameter axiom (par):
\[
\Gamma \vdash \neg \sigma \Rightarrow d^{\tau}(\infty)(\neg \sigma) \quad (\Rightarrow \text{par})
\]
— retaining the rule (var), and
— for each of the remaining rule in Figure 6, replace every occurrence of term by context.
For example, the rules \((\Rightarrow \text{inr})\) and \((\Rightarrow \text{unfold})\) are given by

\[
\Gamma \vdash S : \tau \Rightarrow |S|
\]

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

\[
\Gamma \vdash T : \mu X. \sigma \Rightarrow |T|
\]

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

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

\[
\Gamma \vdash |C[t]| = |\sigma| = |t|.
\]

**Proof.** By induction on the structure of \(C[-\sigma]\) and Lemma 5.20.

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

\[
|C[t]| \sqsubseteq \tau C[t].
\]

**Proof.** By induction on the structure of \(C[-\sigma]\) and Lemma 5.21.

5.6. A crucial lemma

**Lemma 5.24.**

\[
(\emptyset \vdash t : \sigma \Rightarrow |t| \land t \Downarrow v) \implies \emptyset \vdash |t| =_\tau |v|.
\]

**Proof.** By induction on the derivation of \(t \Downarrow v\).

(1) \((\Downarrow \text{can})\): Trivial.

(2) \((\Downarrow \text{fst,snd})\):

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

\[
p \Downarrow (s, t) \quad s \Downarrow v.
\]

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

\[
\begin{align*}
\emptyset \vdash |\text{fst}(p)| &= \text{fst}(|p|) =_\sigma |\text{fst}((s, t))| =_{\sigma \times \tau} |(s, t)| =_\sigma d^{\sigma}(\infty)(|s|) =_\sigma |s| \quad (\text{def. of } |\text{fst}(p)|) \\
&= |v| \quad (\text{Lemma } 5.20) \quad (\text{Ind. hyp.})
\end{align*}
\]

The case for \((\Downarrow \text{snd})\) is similar.

(3) \((\Downarrow \text{app})\):

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 \((\Downarrow \text{app})\) whose
An Operational Domain-theoretic Treatment of Recursive Types

premise is given by

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

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:

\[ \emptyset \vdash |s(t)| \equiv |s([t])| \quad \text{(def. of } |s(t)|) \]

\[ =_\tau |\lambda x. r|([t]) \quad \text{(Ind. hyp.)} \]

\[ \equiv (d^{\sigma \to \tau}(\infty)(\lambda x. |r|))[|t|] \quad \text{(def. of } \lambda x. r) \]

\[ \equiv (\lambda x. d^7(\infty) \circ |r| \circ d^{\sigma}(\infty))[|t|] \quad \text{(def. of } d^{\sigma \to \tau}) \]

\[ =_\tau (\lambda x. d^7(\infty) \circ |r|)(d^{\sigma}(\infty)(|t|)) \quad \text{(Lemma 5.20)} \]

\[ =_\tau d^7(\infty)(|r|[[t]/x]) \quad \text{(\beta-rule (3.30))} \]

\[ =_\tau d^7(\infty)(|[r[t/x]|]) \quad \text{(Lemma 5.22)} \]

\[ =_\tau |v| \quad \text{(Ind. hyp.)} \]

\[ (4) (\Downarrow \text{ case}): \]

Given that

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

and case(s) of \( \text{inl}(x). t_1 \text{ or } \text{inr}(y). t_2 \Downarrow v \). We want to prove that

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

W.l.o.g., let us assume that the following evaluation rule (\( \Downarrow \text{ case inl} \)) derives the given evaluation:

\[
\begin{array}{c}
s \Downarrow \text{inl}(t) \quad t_1[t/x] \Downarrow v \\
\hline
\text{case(s) of } \text{inl}(x). t_1 \text{ or } \text{inr}(y). t_2 \Downarrow v
\end{array}
\]

The induction hypothesis asserts that

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

It then follows that

\[
\begin{array}{c}
\emptyset \vdash |\text{case(s) of } \text{inl}(x). t_1 \text{ or } \text{inr}(y). t_2| \\
\equiv |\text{case}(s)\text{ of } \text{inl}(x). |t_1| \text{ or } \text{inr}(y). |t_2| \quad \text{(by def.)} \\
=_{\rho} |\text{case}(\text{inl}(|t|))\text{ of } \text{inl}(x). |t_1| \text{ or } \text{inr}(y). |t_2| \quad \text{(Ind. hyp.)} \\
=_{\rho} |t_1||t_1[/x] | \quad \text{(\beta-rule (3.33))} \\
=_{\rho} |t_1[t/x]| \quad \text{(Lemma 5.22)} \\
=_{\rho} |v| \quad \text{(Ind. hyp.)}
\end{array}
\]

\[ (5) (\Downarrow \text{ case up}): \]

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

\( \emptyset \vdash |\text{case(s) of } \text{up}(x). t| =_\rho |v| \).
The premise of the evaluation rule which derives case(s) of up(x). t \downarrow v consists of
\[ s \downarrow \text{up}(t') \quad t'[t'/x] \downarrow v. \]
The induction hypothesis asserts that
\[ |s| =_{\sigma_{\perp}} |\text{up}(t')| \quad \text{and} \quad |t'[t'/x]| =_{\rho} |v|. \]
The desired result then follows from
\[ \emptyset \vdash |\text{case}(s)\ |	ext{ of up(x). t}| \equiv |\text{case}(|s|)\ |	ext{ of up(x). t}| \quad \text{(def. of |case(s)\ | of up(x). t|)} \]
\[ =_{\rho} \text{case}(|\text{up}(t')|)\ |	ext{ of up(x). t}| \quad \text{(Ind. hyp.)} \]
\[ =_{\rho} |t'[t'/x]| \quad \text{(\beta-rule (3.32))} \]
\[ =_{\rho} |v|. \quad \text{(Lemma 5.22)} \]
\[ =_{\rho} |v|. \quad \text{(Ind. hyp.)} \]

(6) (\downarrow \ unfolding): Given that \( \emptyset \vdash \text{unfold}(t) \Rightarrow |\text{unfold}(t)|\) and unfold(t) \( \downarrow v\). We must show that
\[ \emptyset \vdash |\text{unfold}(t)| =_{\sigma_{\mu X. \sigma/X}} |v|. \]
The premise of the evaluation rule which derives unfold(t) \( \downarrow v\) consists of
\[ t \downarrow \text{fold}(s) \quad s \downarrow v. \]
The induction hypothesis asserts that
\[ \emptyset \vdash |t| =_{\mu X. \sigma} |\text{fold}(s)| \quad \text{and} \quad \emptyset \vdash |s| =_{\sigma_{\mu X. \sigma/X}} |v|. \]
The desired result follows from
\[ \emptyset \vdash |\text{unfold}(t)| \equiv |\text{unfold}(\text{fold}(s))| \quad \text{(def. of |\text{unfold}(t)|)} \]
\[ =_{\sigma_{\mu X. \sigma/X}} \text{unfold}(\text{fold}(s)) \quad \text{(Ind. hyp.)} \]
\[ =_{\sigma_{\mu X. \sigma/X}} \text{unfold}(d^{\sigma_{\mu X. \sigma/X}}(\infty)(\text{fold}(s))) \quad \text{(def. of |\text{fold}(s)|)} \]
\[ =_{\sigma_{\mu X. \sigma/X}} \text{unfold}(\text{fold}(s)) \quad \text{(Proposition 5.15)} \]
\[ =_{\sigma_{\mu X. \sigma/X}} |s| \quad \text{(\beta-rule (3.35))} \]
\[ =_{\sigma_{\mu X. \sigma/X}} |v|. \quad \text{(Lemma 5.20)} \]
\[ =_{\sigma_{\mu X. \sigma/X}} |v|. \quad \text{(Ind. hyp.)} \]

5.7. Proof of functoriality

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

**Lemma 5.25.** Let \( f, g \in \text{Exp}_{\sigma_{\mu X. \sigma/X}} \) be given. Suppose that for all \( t \in \text{Exp}_{\sigma_{\mu X. \sigma/X}} \) and for all contexts of the form \[ C[\text{fold}(t)] \in \text{Ctx}_\Sigma \] it holds that
\[ C[f(\text{fold}(t))] \subseteq_{\Sigma} C[g(\text{fold}(t))]. \]
Then \( f \subseteq_{\mu X. \sigma_{\mu X. \sigma/X}} g. \)
Proof. By the extensionality property (3.42), in order to prove that \( f \sqsubseteq g \), it suffices to prove that for all \( s \in \text{Exp}_{\mu X. \sigma} \), \( f(s) \sqsubseteq \mu X. \sigma \) holds. Let \( s \in \text{Exp}_{\mu X. \sigma} \) be given and suppose \( C[-\mu X. \sigma] \in \text{Ctx}_\Sigma \) is such that \( C[f(s)] \downarrow \top \). Because of \( \eta \)-rule (3.47), it follows from the definition of \( \sqsubseteq \) that

\[
C[f(s)] \downarrow \top \iff C[f(\text{fold}(\text{unfold}(s)))].
\]

Thus by assumption that \( C[f(\text{fold}(t))] \subseteq \Sigma C[g(\text{fold}(t))] \) for all \( t \in \text{Exp}_{\sigma/\mu X. \sigma} \), it follows (by defining \( t := \text{unfold}(s) \)) that \( C[g(\text{fold}(\text{unfold}(s)))] \downarrow \top \). Again invoking \( \eta \)-rule (3.47) and the definition of \( \subseteq \), we have that \( C[g(s)] \downarrow \top \), as required.

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

\[
\emptyset \vdash \text{id}_{\mu X. \sigma} \sqsubseteq d_{\mu X. \sigma}(\infty).
\]

Proof. By Lemma 5.25, it suffices to show that for any \( t \in \text{Exp}_{\sigma/\mu X. \sigma} \) and for any context \( C[-\mu X. \sigma \rightarrow \mu X. \sigma(\text{fold}(t))] \in \text{Ctx}_\Sigma \), it holds that

\[
C[\text{id}_{\mu X. \sigma}(\text{fold}(t))] \subseteq \Sigma C[d_{\mu X. \sigma}(\infty)(\text{fold}(t))].
\]

Let \( C[-\mu X. \sigma \rightarrow \mu X. \sigma][\text{fold}(t)] \in \text{Ctx}_\Sigma \) be arbitrary. Since \( \text{id}_{\mu X. \sigma}(\text{fold}(t)) =_{\mu X. \sigma} \text{fold}(t) \) holds (an instance of Kleene equivalence), it suffices to prove that

\[
C[\text{fold}(t)] \subseteq \Sigma C[d_{\mu X. \sigma}(\infty)(\text{fold}(t))].
\]

By Lemma 5.21 it suffices to show that

\[
C[\text{fold}(t)] \subseteq \Sigma C[d_{\mu X. \sigma}(\infty)(\text{fold}(t))].
\]

But by Lemma 5.20 it suffices to show that

\[
C[\text{fold}(t)] \subseteq \Sigma C[\text{fold}(t)].
\]

By Lemma 5.24 \( C[\text{fold}(t)] \downarrow \top \) implies that \( |C[\text{fold}(t)]| = |\top| = \top. \) It then follows that

\[
C[\text{fold}(t)] \downarrow \top \implies |C[\text{fold}(t)]| \downarrow \top \quad \text{(Lemma 5.24)}
\]

\[
\implies |C[[\text{fold}(t)]]| \downarrow \top \quad \text{(Lemma 5.22)}
\]

\[
\implies C[[\text{fold}(t)]] \downarrow \top. \quad \text{(Lemma 5.23)}
\]

which is what we aim to show.

The proof of Theorem 5.1 can now be given.

Proof. As remarked in the beginning of Section 5.4, we have by Theorem 5.16 that

\[
d_\sigma(n) \sqsubseteq e_\sigma(n) \sqsubseteq \text{id}_\sigma
\]

for all closed types \( \sigma \) and all \( n \in \mathbb{N} \). In particular, we must have

\[
d_\sigma(\infty) \sqsubseteq e_\sigma(\infty) \sqsubseteq \text{id}_\sigma.
\]

Now as a consequence of Lemmata 5.26 and 5.18 \( d_\sigma(\infty) = \text{id}_\sigma \). Thus it follows that

\[
e_\sigma(\infty) = \text{id}_\sigma.
\]
This means
\[
\text{id}_{\mu X. \tau} = e^{\mu X. \tau}(\infty)
\]
\[
= \text{fold} \circ F_{X \to \tau}(e^{\mu X. \tau}(\infty - 1)) \circ \text{unfold}
\]
\[
= \text{fold} \circ F_{X \to \tau}(\text{id}_{\mu X. \tau}) \circ \text{unfold}
\]
and thus \(\text{id}_{\mu X. \tau}\) is the least endomorphism \(e\) on \(\mu X. \tau\) which satisfies the equation
\[
e = \text{fold} \circ F_{X \to \tau}(e) \circ \text{unfold}.
\]
Consequently, for the more general case, the least endomorphism \(e\) on \(S(\bar{\sigma}) := \mu X. T(\bar{\sigma}, X)\) for which the following diagram
\[
\begin{array}{ccc}
S(\bar{\sigma}) & \xrightarrow{e} & S(\bar{\sigma}) \\
\downarrow i_{S(\bar{\sigma})} & & \downarrow \tilde{i}_{S(\bar{\sigma})} \\
T(\bar{\sigma}, S(\bar{\sigma})) & \xrightarrow{T(\text{id}_\bar{\sigma}, e)} & T(\bar{\sigma}, S(\bar{\sigma}))
\end{array}
\]
commutes is the identity morphism \(\langle \text{id}_{S(\bar{\sigma})}, \text{id}_{S(\bar{\sigma})} \rangle\).
Now, suppose an \(\text{FPC}\)-morphism \(h : S(\bar{\sigma}) \to S(\bar{\sigma})\) defines an endomorphism such that \(h \circ i_{S(\bar{\sigma})} = i_{S(\bar{\sigma})} \circ T(\text{id}_{S(\bar{\sigma})}, h)\). We apply the Plotkin’s uniformity principle, i.e., Lemma 3.13 to the diagram (where \(\phi := \lambda g. i_{S(\bar{\sigma})} \circ T(\bar{\sigma}, g) \circ i_{S(\bar{\sigma})}^{-1}\)):
\[
\begin{array}{ccc}
(S(\bar{\sigma}) \to S(\bar{\sigma})) & \xrightarrow{H} & (S(\bar{\sigma}) \to S(\bar{\sigma})) \\
\downarrow \phi & & \downarrow \phi \\
(S(\bar{\sigma}) \to S(\bar{\sigma})) & \xrightarrow{H} & (S(\bar{\sigma}) \to S(\bar{\sigma}))
\end{array}
\]
where \(H := \lambda g. h \circ g\). This is a strict program since \(h\) is assumed to be strict. The above diagram commutes since
\[
H \circ \phi(g) = H(i_{S(\bar{\sigma})} \circ T(\text{id}_\bar{\sigma}, g) \circ i_{S(\bar{\sigma})}^{-1})
\]
\[
= h \circ i_{S(\bar{\sigma})} \circ T(\text{id}_\bar{\sigma}, g) \circ i_{S(\bar{\sigma})}^{-1}
\]
\[
= i_{S(\bar{\sigma})} \circ T(\text{id}_\bar{\sigma}, h) \circ T(\text{id}_\bar{\sigma}, g) \circ i_{S(\bar{\sigma})}^{-1}
\]
\[
= i_{S(\bar{\sigma})} \circ T(\text{id}_\bar{\sigma}, h \circ g) \circ i_{S(\bar{\sigma})}^{-1}
\]
\[
= i_{S(\bar{\sigma})} \circ T(\text{id}_\bar{\sigma}, H(g)) \circ i_{S(\bar{\sigma})}^{-1}
\]
\[
= \phi \circ H(g).
\]
By Lemma 3.13 it follows that \(\text{id}_{S(\bar{\sigma})} = \text{fix}(\phi) = H(\text{fix}(\phi)) = h \circ \text{id}_{S(\bar{\sigma})} = h\). So, the identity is the least endomorphism such that \(h \circ i_{S(\bar{\sigma})} = i_{S(\bar{\sigma})} \circ T(\text{id}_{S(\bar{\sigma})}, h)\). \(\Box\)
Theorem 5.27. Let $\Theta \vdash \sigma$ be a type-in-context. Then, the realisable action $F_{\Theta \vdash \sigma}$ (defined on pp. 22 – 24) is in fact a realisable functor.

Proof. Since the construction given on pp. 22 – 24 justifies that realisable actions $F_{\Theta \vdash \sigma}$ preserve composition of morphisms, it remains to show that they preserve identity morphisms. The proof proceeds by induction on the structure of $\Theta \vdash \sigma$.

(1) Type variable.

If $\vec{X} \vdash X_i$, the morphism part of $F_{\vec{X} \vdash X_i}$ is realised by

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

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

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

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

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

For this purpose, we assume the following induction hypotheses:

(a) $F_{\vec{X} \vdash \tau_i}$ ($i = 1, 2$), $F_{\vec{X} \vdash \tau}$ and $F_{\vec{X}, X \vdash \sigma}$ are the realisable actions corresponding to $\vec{X} \vdash \tau_i$ ($i = 1, 2$), $\vec{X} \vdash \tau$ and $\vec{X}, X \vdash \sigma$ respectively, and they do preserve identity morphisms.

(b) The morphism parts of these realisable actions are respectively realised by

$$\vec{R}, \vec{S}, f : \vec{R} \rightarrow \vec{S} \vdash t_i : \tau_i[\vec{R}/\vec{X}] \rightarrow \tau_i[\vec{S}/\vec{X}] \quad (i = 1, 2),$$ 

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

and

$$\vec{R}, \vec{S}, f : \vec{R} \rightarrow \vec{S}, f : R \rightarrow S \vdash s : \sigma[\vec{R}/\vec{X}, R/X] \rightarrow \sigma[\vec{R}/\vec{X}, R/X].$$

For the non-recursive types, the proof is straightforward. Here we supply the proof for the case of product type.

The morphism part of $F_{\vec{X} \vdash \tau_1 \times \tau_2}$ is realised by

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

where

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

and

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

Note that $F_{\vec{X} \vdash \tau_1 \times \tau_2}$ preserves identity morphisms because $F_{\vec{X} \vdash \tau_i}$’s do by induction hypotheses.

For the recursive type, that $F_{\vec{X} \vdash \mu X. \sigma}$ preserves identity morphisms is just Theorem 5.1.

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, $\text{DCPO}_{\bot}$, the category of pointed cpos and strict maps) has an initial algebra and a final co-algebra and they are canonically isomorphic”. In the same reference, Freyd proved the Product Theorem which asserts that algebraic compactness is closed under finite products. Crucially, this implies that $\text{DCPO}_{\bot} \times \text{DCPO}^\text{op}_{\bot}$ 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 $\text{DCPO}_{\bot}$ is algebraically compact is no easy feat, assuming that one employs the usual proof technique of switching 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 section, we establish that the diagonal category $\text{FPC}^\delta$ is para-metrised algebraically compact. We also consider an alternative choice of categorical framework, namely the product category $\hat{\text{FPC}}_1 := \text{FPC}^{\text{op}}_1 \times \text{FPC}_1$, 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 3.13) in establishing the algebraic compactness results in Sections 6.1 - 6.2. Such a proof technique was probably first done in [Simpson, 1992] for Kleisli-categories.

A few definitions should be put in place.

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

\[
\begin{array}{ccc}
FA & \xrightarrow{Fg} & FA' \\
fA & \downarrow & fA' \\
A & \xrightarrow{g} & A'
\end{array}
\]

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

**6.1. Operational algebraic compactness**

**Theorem 6.2.** (Operational algebraic completeness I)

Every realisable endofunctor

$$F : \text{FPC}^\delta \to \text{FPC}^\delta$$

has an initial algebra.

\[\text{If non-strict maps are considered then the identity functor does not have an initial algebra.}\]
We say that the category \( FPC_\delta \) 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 \((\text{unfold}, \text{fold})\mu X. \tau \) 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 begin by defining \( k \) to be the least homomorphism for which the diagram

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

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:

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

where the programs \( \Phi, \Psi \) and \( G \) are defined as follows.

\[
\Phi := \lambda h : (D \to D) \times (D \to D). i \circ F(h) \circ i^{-1}
\]

\[
\Psi := \lambda k : (D' \to D) \times (D \to D'). i' \circ F(k) \circ i^{-1}
\]

\[
G := \lambda h : (D \to D) \times (D \to D). k' \circ h.
\]

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

\[
G(\Phi(h)) = k' \circ \Phi(h)
\]

\[
= k' \circ i \circ F(h) \circ i^{-1} \quad \text{(def of } \Phi) \]

\[
= i' \circ F(k') \circ i^{-1} \circ i \circ F(h) \circ i^{-1} \quad (k' = i' \circ F(k') \circ i^{-1}) \]

\[
= i' \circ F(k') \circ F(h) \circ i^{-1} \quad \text{(unfold } = \text{fold}^{-1})
\]

\[
= i' \circ F(k' \circ h) \circ i^{-1} \quad (F \text{ is a functor})
\]

\[
= \Psi(k' \circ h) \quad \text{(def of } \Psi)
\]

\[
= \Psi(G(h)).
\]

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

\[ k = \text{fix}(\Psi) = k' \circ \text{fix}(\Phi) = k' \circ (\text{id}_D, \text{id}_D) = k'. \]

Thus, the uniqueness of \( k \) is established.

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

**Theorem 6.4.** (Operational algebraic compactness I)

Let \( F : \text{FPC}_\delta \to \text{FPC}_\delta \) be a realisable endofunctor. Then, every initial \( F \)-algebra is bifree.

We say that the category \( \text{FPC}_\delta \) is *operationally algebraically compact* with respect to the class of realisable functors.

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

**Theorem 6.5.** (Operational parametrised algebraic compactness I)

Let \( F : (\text{FPC}_\delta)^{n+1} \to \text{FPC}_\delta \) be a realisable functor. Then there exists a realisable functor \( H : (\text{FPC}_\delta)^n \to \text{FPC}_\delta \) and a natural isomorphism \( i \) such that for all sequences of closed types \( P \) in \( (\text{FPC}_\delta)^n \), we have

\[ i_P : F(P, H(P)) \cong H(P). \]

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

\[ F(P, \_ : \text{FPC}_\delta) \to \text{FPC}_\delta. \]

We say that the category \( \text{FPC}_\delta \) is *operationally parametrised algebraically compact* with respect to the class of realisable functors.

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

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

Then, clearly, every \( P \in (\text{FPC}_\delta)^n \) induces a realisable endofunctor

\[ F(P, \_ : \text{FPC}_\delta) \to \text{FPC}_\delta \]

and by operational algebraic completeness of \( \text{FPC}_\delta \) 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
\[
\begin{array}{ccc}
F(P,H(P)) & \xrightarrow{i_P} & H(P) \\
\downarrow F(P,g) & & \downarrow g \\
F(P,H(Q)) & \xrightarrow{F(f,H(Q))} & F(Q,H(Q)) \\
& \xrightarrow{i_Q} & H(Q)
\end{array}
\]
commutes. By the universal property of initial algebras, \(H\) is a functor and by construction, \(i\) is a natural transformation. Moreover, Theorem \ref{thm:bifree-algebra} ensures that \((H(P), i_P)\) is a bifree \(F(P,_)\)-algebra. Finally, note that \(H\) is realized by the type-in-context \(\vec{X} \vdash \mu \vec{X}. \tau\) and the pair of terms-in-context \(\vec{R}, \vec{S} ; \vec{f} : \vec{R} \rightarrow \vec{S} \vdash \text{fix}(\lambda v. i^{-1}_{H(\vec{R})} \circ t[v/f] \circ i_{H(\vec{S})})\), where \(i := \langle \text{fold}, \text{unfold} \rangle\).

\section{6.2. Alternative choice of category}

The theory of recursive domain equations since the works of Freyd, Fiore and Plotkin in the early 1990’s centres around functors of the form
\[F : (\text{DCPO}^{op}_{\perp!} \times \text{DCPO}_{\perp!})^{n+1} \rightarrow (\text{DCPO}^{op}_{\perp!} \times \text{DCPO}_{\perp!}).\]

As noted before, \(\text{DCPO}^{op}_{\perp!} \times \text{DCPO}_{\perp!}\) is algebraically compact. But more generally \(\text{DCPO}^{op}_{\perp!} \times \text{DCPO}_{\perp!}\) is parameterised algebraically compact - a result implied by Corollary 5.6 of \cite{FiorePlotkin94}.

Let \(\mathbf{FPC}_{\dagger}\) denote the product category \(\mathbf{FPC}_{\dagger}^{op} \times \mathbf{FPC}_{\dagger}\) where \(\mathbf{FPC}_{\dagger}\) has been defined earlier in Definition \ref{def:FPC}. The natural question to ask is whether the category \(\mathbf{FPC}_{\dagger}\) is algebraically compact. In order that this question makes sense, one has to identify an appropriate class of functors, \(F\), with respect to which algebraic compactness is defined. In this section, we show that, with a suitable choice of \(F\), the category \(\mathbf{FPC}_{\dagger}\) is parameterised algebraically compact with respect to \(F\), i.e., for every \(F\)-functor \(T : (\mathbf{FPC}_{\dagger})^{n+1} \rightarrow \mathbf{FPC}_{\dagger}\), there exists an \(F\)-functor \(H : (\mathbf{FPC}_{\dagger})^{n} \rightarrow (\mathbf{FPC}_{\dagger})\) and a natural isomorphism \(i\) such that for every sequence of closed types \(\vec{\sigma} := \sigma_1^+ , \sigma_2^- , \ldots, \sigma_n^+ , \sigma_n^-\), the pair \((H(\vec{\sigma}), i_{\vec{\sigma}})\) is a bifree algebra of the endofunctor \(T(\vec{\sigma}, - , +) : \mathbf{FPC}_{\dagger} \rightarrow \mathbf{FPC}_{\dagger}\).

In the framework of the product category \(\mathbf{FPC}_{\dagger}\), 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 \(X + (X \rightarrow X)\), separation yields \(X^+ + (X^- \rightarrow X^+)\).
Notation. We use the following notations:

\[ \vec{X} := X_1^+, X_1^-, \ldots, X_n^-, X_n^+ \]
\[ \vec{X}^\pm := X_1^+, X_1^-, \ldots, X_n^+, X_n^- \]
\[ \vec{\sigma} := \sigma_1^+, \sigma_1^-, \ldots, \sigma_n^+, \sigma_n^- \]
\[ \vec{\sigma}^\pm := \sigma_1^+, \sigma_1^-, \ldots, \sigma_n^+, \sigma_n^- \]
\[ \vec{f} : \vec{R} \to \vec{S} := \vec{f}^+ : \vec{R}^+ \to \vec{S}^+, \vec{f}^- : \vec{S}^- \to \vec{R}^- \]

Sometimes, we also use \( P \) and \( Q \) to denote objects in \( \mathsf{FPC}_! \), and \( u \) for morphisms in \( \mathsf{FPC}_! \).

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

\( (\mathsf{FPC}_!)^n \to \mathsf{FPC}_! \).

A seemingly reasonable choice is the class of syntactic functors (originally used by A. Rohr in his Ph.D. thesis \cite{Rohr2002}) which is defined as follows.

A syntactic functor \( T : (\mathsf{FPC}_!)^n \to \mathsf{FPC}_! \) 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 \( \vec{\sigma} \in \mathsf{FPC}_!^n \), it holds that

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

and for any \( \vec{\rho}, \vec{\sigma} \in \mathsf{FP\bar{C}}_!^n \), and any \( \vec{u} \in \mathsf{FP\bar{C}}_!^n(\vec{\rho}, \vec{\sigma}) \), 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 \( \mathsf{FPC}_! \to \mathsf{FP\bar{C}}_! \) 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 \( F : (\mathsf{FP\bar{C}}_!)^n \to \mathsf{FP\bar{C}}_! \), there is a standard way of turning it to an endofunctor \( \check{F} : (\mathsf{FP\bar{C}}_!)^n \to \mathsf{FP\bar{C}}_! \). Define \( \check{F} \) by:

\[ \check{F}(\vec{\sigma}) = (F(\vec{\sigma}^+), F(\vec{\sigma}^-)) \]

So one might consider defining a functor \( G : \mathsf{FP\bar{C}}_!^n \to \mathsf{FP\bar{C}}_! \) to be syntactic if there exists a syntactic functor \( F : \mathsf{FP\bar{C}}_!^n \to \mathsf{FP\bar{C}}_! \) such that \( G = \check{F} \).

However, if we work with this definition, a serious problem arises. As we shall see in Theorem 6.9, the parametrised initial algebra of such functors are not of the form \( \check{H} \) for some functor \( H \).

This can be fixed by working with our official definition:

\[ * \] This problem was discovered by the author and T. Streicher during a private communication in January 2006.
Definition 6.6. An \( n \)-ary functor \( F : (\text{FPC}_n)^n \to \text{FPC}_1 \) 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 \( \vec{R}, \vec{S}; f : \vec{R} \to \vec{S} \)

\[
\begin{align*}
t^- : &\, \tau^-[\vec{S}/\vec{X}] \to \tau^-[\vec{R}/\vec{X}], \\
t^+ : &\, \tau^+[\vec{R}/\vec{X}] \to \tau^+[\vec{S}/\vec{X}].
\end{align*}
\]

such that for any \( \vec{\sigma} \in \text{FPC}_1^n \),

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

and for any \( \vec{\rho}, \vec{\sigma} \in \text{FPC}_1^n \) and any \( \vec{u} \in \text{FPC}_1^n \), we have

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

Before we establish operational algebraic completeness and compactness for the category \( \text{FPC}_1 \), we pause to look at some examples.

Example 6.7.

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

\[
\begin{align*}
\tau^- &\, \vdash \mu X_1.(X_1 \to X_2), (X_2 \to X_1) \\
\tau^+ &\, \vdash \mu X_1.(X_1 \to X_2).
\end{align*}
\]

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

\[
\begin{align*}
t^- &\, \coloneqq \lambda g : (S^+ \to S^-).f^- \circ g \circ f^+ \\
t^+ &\, \coloneqq \lambda h : (R^- \to R^+).f^+ \circ h \circ f^-.
\end{align*}
\]

(2) The type-in-context \( X_2 \vdash \mu X_1.(X_1 \to X_2) \) is not functorial in \( X_2 \) since one unfolding of \( \mu X_1.(X_1 \to X_2) \) yields \( \mu X_1.(X_1 \to X_2) \to X_2 \) and the latter expression does not respect the variance of \( X_2 \). It seems clear that there is no syntactic functor whose object part is realised by the type-in-context \( X_2 \vdash \mu X_1.(X_1 \to X_2) \).

Remark 6.8. Crucially, Example 6.7(2) indicates that if a bifree algebra for \( X_2, X_1 \vdash X_1 \to X_2 \) were to exist then it cannot be simply given by \( X_2, X_1 \vdash \mu X_1.(X_1 \to X_2) \). Theorems 6.9 and 6.11 below provide us with a way to obtain the required bifree algebra.

Theorem 6.9. (Operational algebraic completeness II)

Every syntactic functor

\[
F : \text{FPC}_1 \to \text{FPC}_1
\]

has an initial algebra.
We say that the category $\mathbf{FPc}_1$ is \textit{operationally algebraically complete} with respect to the class of syntactic functors.

\textit{Proof.} Recall that $F$ can be resolved into its coordinate functors

$$T^- : \mathbf{FPc}_1 \to \mathbf{FPc}_1^{op} \text{ and } T^+ : \mathbf{FPc}_1 \to \mathbf{FPc}_1,$$

which are explicitly defined as follows. Note that $T^-$ (respectively, $T^+$) is realised by a type-in-context $\vec{X} \vdash \tau^-$ (respectively $\tau^+$) and a term-in-context $\vec{R}, \vec{S}; f : \vec{R} \to \vec{S} \vdash t^- : \tau^-[\vec{S}/\vec{X}] \to \tau^-[\vec{R}/\vec{X}]$ (respectively, $t^+$).

We want to construct an initial $F$-algebra in stages.

(1) For each $\sigma^+$ in $\mathbf{FPc}_1$, consider the endofunctor

$$T^-(_-, \sigma^+) : \mathbf{FPc}_1^{op} \to \mathbf{FPc}_1^{op}.$$

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:

\[\operatorname{unfold}^{op} : T^-((\mu X^- T^- (X^-, \sigma^+), \sigma^+)) \to \mu X^- T^- (X^-, \sigma^+).\]

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

$$f^\sigma_- : T^- (F^- (\sigma^+), \sigma^+) \to F^- (\sigma^+).$$

One can then verify that this is an initial $T^-(_-, \sigma^+)$-algebra in $\mathbf{FPc}_1^{op}$, via a proof similar to that in Theorem 6.2.

(2) We now extend $F^-$ to be a functor $\mathbf{FPc}_1 \to \mathbf{FPc}_1^{op}$. Note that $F^-$ is realised by the type-in-context $X^+ \vdash \mu X^-. \tau^-$ and the term-in-context $R^+, S^+; f^+ : R^+ \to S^+ \vdash \text{fix}(\lambda v. \text{fold} \circ t^-[v/f^-] \circ \text{unfold})$, and is thus syntactic. We now define the morphism part of $F^-$. Let $w^+ : \rho^+ \to \sigma^+$ be a $\mathbf{FPc}_1$-morphism. Using the initiality of $F^- (\rho^+)$, define $F^- (w^+)$ to be the unique morphism which makes the following diagram commute in $\mathbf{FPc}_1^{op}$:

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

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

(3) In this stage, we define an endofunctor $G : \mathbf{FPc}_1 \to \mathbf{FPc}_1$ by

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

which is syntactic since it is realised by the type-in-context $X^+ \vdash \tau^+[(\mu X^- \cdot \tau^-)/X^-]$ and the term-in-context

\[R^+, S^+, f^+ : R^+ \to S^+ \vdash t^+[\text{fix}(\lambda v. \text{fold} \circ t^-[v/f^-] \circ \text{unfold})]/t^-].\]
We have the initial algebra for $G$ given by
\[
\text{fold}^{\mu X^+, G(X^+)} : T^+(F^-(\mu X^+, G(X^+)), \mu X^+, G(X^+)) \to \mu X^+, G(X^+).
\]
We use the notations $\delta^+$ for $\mu X^+, G(X^+)$ and $d^+$ for $\text{fold}^{\mu X^+, G(X^+)}$. Rewriting, we have the initial $G$-algebra given by
\[
d^+: T^+(F^-(\delta^+), \delta^+) \to \delta^+ \tag{6.1}
\]
We further define $\delta^- := F^-(\delta^+)$ and denote the initial $T^-(\_, \delta^+)$-algebra $f^-_{\delta^+} : T^-(F^-(\delta^+), \delta^+) \to F^-(\delta^+)$ by
\[
d^- : T^-(\delta^-, \delta^+) \to \delta^- \tag{6.2}
\]
bearing in mind that this is a morphism in $\mathbf{FPC}^{\text{op}}$.

(4) It can be shown, following closely the proof method outlined in [Freyd, 1990], that $(d^{-}, d^{+}) : (T^-(\delta^-, \delta^+), T^+(\delta^-, \delta^+)) \to (\delta^-, \delta^+)$ is an initial $(T^-, T^+)$-algebra, i.e., $F$-coalgebra, which we set out to find initially. The proof is now complete.

\[\square\]

**Theorem 6.10.** (Operational algebraic compactness II)

Let $F : \mathbf{FP\mathbf{C}}_n \to \mathbf{FP\mathbf{C}}$ 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 $\mathbf{FP\mathbf{C}}_n$ is \textit{operationally algebraically compact} with respect to the class of syntactic functors.

\[\begin{proof}
\text{Walking through the stages of the proof of Theorem 6.9 one can check at each stage that a final coalgebra results when each initial algebra structure map is inverted. Notice this works even for the definition of $F^-$ in stage (2).}
\end{proof}\]

**Theorem 6.11.** (Operational parametrised algebraic compactness II)

Let $F : (\mathbf{FP\mathbf{C}}_n)^{n+1} \to \mathbf{FP\mathbf{C}}$ be a syntactic functor. Then there exists a syntactic functor $H : \mathbf{FP\mathbf{C}}_n \to \mathbf{FP\mathbf{C}}$ and a natural isomorphism $i$ such that for all sequence of closed types $P$ in $(\mathbf{FP\mathbf{C}}_n)^n$ we have
\[
i_P : F(P, H(P)) \cong H(P).
\]
Moreover, $(H(P), i_P)$ is a bifree algebra for the endofunctor
\[
F(P, -) : \mathbf{FP\mathbf{C}}_n \to \mathbf{FP\mathbf{C}}.
\]
In other words, $\mathbf{FP\mathbf{C}}_n$ is \textit{parametrised operationally algebraically complete} with respect to the syntactic functors.

\[\begin{proof}
\text{Since $F : (\mathbf{FP\mathbf{C}}_n)^{n+1} \to \mathbf{FP\mathbf{C}}$ is syntactic, it is realised by a pair of types-in-contexts $\vec{X}; X^-, X^+ \vdash \tau^-, \tau^+$ and a pair of terms-in-context $\vec{R}, R^-, R^+, \vec{S}, S^-, S^+: f :}
\end{proof}\]
\[ \vec{R} \to \vec{S}, f^- : S^- \to R^-, f^+ : R^+ \to S^+ \]

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

For each \( P \in \left( \mathbf{FP} \mathbf{C}_1 \right)^n \), we have that \( F(P, \_ : \mathbf{FP} \mathbf{C}_1) \to \mathbf{FP} \mathbf{C}_1 \) is a syntactic endofunctor whose component functors are \( T^-(P, \_ ) \) and \( T^+(P, \_ ). \) Invoking Theorem 6.9, we form the initial \( F(P, \_ ) \)-algebra, \( (H(P), i_\_ P), \) where
\[ H^-(P) = \mu X^- \cdot T^-(P, X^-, H^+(P)), \]
\[ H^+(P) = \mu X^+ \cdot T^+(P, \mu X^-, T^- (P, X^-, X^+), X^+). \]

and \( i_\_ P = (d^-, d^+) = (\text{unfold} H^-(P), \text{fold} H^+(P)). \) Note that \( H \) is a syntactic functor realised by a pair of types-in-context:
\[ \vec{X}^- \vdash \mu X^- \cdot \tau^-[(\mu X^+, \tau^+[(\mu X^-, \tau^-)/X^-)]/X^+], \]
\[ \vec{X}^+ \vdash \mu X^+ \cdot \tau^+[(\mu X^-, \tau^-)/X^-] \]

and a pair of terms-in-context \( \vec{P}, \vec{Q} : \vec{f} : \vec{P} \to \vec{Q} \vdash \)
\[ s^- := \text{fix}(\lambda v. \text{fold} \circ t^-[v/f^-] \circ \text{unfold})[s^+/f^+], \]
\[ s^+ := \text{fix}(\lambda u. \text{fold} \circ t^+[\text{fix}(\lambda v. \text{fold} \circ t^-[v/f^-] \circ \text{unfold})]/f^-] \circ \text{unfold}) \]

Because of its repetitive nature (c.f. Theorem 6.5), we leave the action of \( H \) on morphisms \( f : P \to Q \) in \( \left( \mathbf{FP} \mathbf{C}_1 \right)^n \) for the diligent reader to work out.

\[ \square \]

**Definition 6.12.** In Theorem 6.11 the functor \( H : \left( \mathbf{FP} \mathbf{C}_1 \right)^n \to \mathbf{FP} \mathbf{C}_1 \) is a bifree \( F \)-algebra, where \( F : \left( \mathbf{FP} \mathbf{C}_1 \right)^{n+1} \to \mathbf{FP} \mathbf{C}_1 \) is a syntactic functor. To indicate this dependence, we write
\[ H := \mu F. \]

To each \( P \in \left( \mathbf{FP} \mathbf{C}_1 \right)^n, \) \( H \) assigns the following pair of closed types:
\[ H^-(P) \quad = \quad \mu X^- \cdot T^-(P, X^-, H^+(P)), \]
\[ H^+(P) \quad = \quad \mu X^+ \cdot T^+(P, \mu X^-, T^- (P, X^-, X^+), X^+). \]

To each morphism \( u \in \mathbf{FP} \mathbf{C}_1^0 (P, Q), \) the morphism \( H(u) \) is the least morphism \( h \) for which the diagram
\[
\begin{array}{ccc}
F(P, H(P)) & \xrightarrow{i_\_ P} & H(P) \\
\downarrow F(u, h) & & \downarrow h \\
F(Q, H(Q)) & \xrightarrow{i_\_ Q} & H(Q)
\end{array}
\]

commutes.

**Examples 6.13.** Responding to Remark 6.8, the correct bifree algebra of \( X^-, X^+ \vdash \)
An Operational Domain-theoretic Treatment of Recursive Types

\[(X^+ \to X^-), (X^- \to X^+)\] should be
\[
H^- = \mu X^- \cdot H^+ \to X^-, \\
H^+ = \mu X^+ \cdot ((\mu X^- \cdot (X^+ \to X^-)) \to X^+).
\]

**Remark 6.14.** The functor \(H := \mu F\) is never symmetric. But we expect that symmetry can be achieved in the form of an operational analogue of Fiore’s diagonalisation technique (cf. p.124 of [Fiore, 1996]). Although we do not adopt this approach in this paper, we aim to establish an important result, Proposition 6.15, which can be seen as a restricted form of diagonalisation. In the upcoming section, we develop this result which then establishes equivalence to a symmetric functor with respect to a restriction on the domain of the functor.

6.3. On the choice of categorical frameworks

In this section, we compare the two approaches via the diagonal category, \(\text{FPC}^!\delta\), and the product category, \(\text{FPC}_!\).

In the framework of the product category \(\text{FPC}_!\), it is appropriate to study the class of syntactic functors because all FPC types-in-context can be viewed as syntactic functors. Because of its repetitious nature (c.f. similar definition in Section 4), we show very briefly how this construction can be carried out by induction on the structure of \(\Theta \vdash \sigma\) for type variables, function types and recursive types. We denote the syntactic functor associated to \(\Theta \vdash \sigma\) by \(G_{\Theta \vdash \sigma}\), or simply \(G\).

— Type variable.
Let \(\Theta \vdash X_i\) be given. Define the functor \(G : \text{FPC}_! n \to \text{FPC}_!\) as follows.
For an object \(P \in \text{FPC}_! n\), define \(G(P) := P_i\).
For a morphism \(u \in \text{FPC}_! n(P,Q)\), define \(G(u) := u_i\).

— Function type.
Let \(\Theta \vdash \sigma_1, \sigma_2\) be given and \(G_1, G_2 : \text{FPC}_! n \to \text{FPC}_!\) be their associated realisable functors. For a given syntactic functor \(G : \text{FPC}_! n \to \text{FPC}_!\), we write \(G^- : \text{FPC}_! n \to \text{FPC}_!^{op}\) and \(G^+ : \text{FPC}_! n \to \text{FPC}_!\) for its two component functors.
For an object \(P \in \text{FPC}_! n\), define \(G(P) := (G_1^+(P) \to G_2^-(P), G_1^-(P) \to G_2^+(P))\)
and for a morphism \(u \in \text{FPC}_! n(P,Q)\), define
\(G(u) := (G_1^+(u) \to G_2^-(u), G_1^-(u) \to G_2^+(u))\)
where the component morphisms are defined as follows:
\[
G_1^+(u) \to G_2^-(u) = \lambda g : G_1^+(Q) \to G_2^-(Q), G_2^-(u) \circ g \circ G_1^+(u) \\
G_1^-(u) \to G_2^+(u) = \lambda h : G_1^-(Q) \to G_2^+(Q), G_2^+(u) \circ h \circ G_1^-(u).
\]

— Recursive type.
Let $\Theta, X \vdash \sigma$ be given and $F$ the syntactic functor realising it. Define $G_{\Theta \vdash X} \sigma$ to be $\mu F$ as in Definition 6.12.

**Notations.** From this point, we denote the syntactic functor (respectively, the realisable functor) associated to the type-in-context $\Theta \vdash \sigma$ by $G_{\Theta \vdash \sigma}$ (respectively, $F_{\Theta \vdash \sigma}$). We use $\text{Inj} : \text{FPC}_!^\delta \rightarrow \text{FPC}_!^\delta$ to denote the functor whose object part assigns to $\sigma \in \text{FPC}_!^\delta$ a pair of types $(\sigma^-, \sigma^+)$ with $\sigma^- = \sigma^+ = \sigma$, and whose morphism part assigns to $f^+ : \sigma \rightarrow \tau : f^- : \sigma^- \rightarrow \sigma = \sigma^-$ and $f^+ : \sigma^+ \rightarrow \tau = \tau^+$. The following proposition reveals how the classes of realisable functors and syntactic functors are related.

**Proposition 6.15.** For every type-in-context $\Theta \vdash \sigma$, the realisable functor $F_{\Theta \vdash \sigma}$ restricts and co-restricts to the syntactic functor $G_{\Theta \vdash \sigma}$, i.e., the diagram

$$
\begin{array}{ccc}
\text{Inj}^n & \xrightarrow{\mu F_{\Theta \vdash \sigma}} & \text{Inj}^n \\
\downarrow \text{Inj} & & \downarrow \text{Inj} \\
(G_{\Theta \vdash \sigma})^n & \xrightarrow{G_{\Theta \vdash \sigma}} & \text{FPC}_!^\delta
\end{array}
$$

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

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

(1) Type variable.

Let $\Theta \vdash X_i$ be given. Define $\eta : G_{\Theta \vdash X_i} \circ \text{Inj}^n \rightarrow \text{Inj} \circ F_{\Theta \vdash X_i}$ as follows. For every $\vec{\sigma} \in (\text{FPC}_!^\delta)^n$,

$$\eta_{\vec{\sigma}} := (\id_{\sigma_i}, \id_{\sigma_i}).$$

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

$$\eta_j : G_{\Theta \vdash \tau_j} \circ \text{Inj}^n \rightarrow \text{Inj} \circ F_{\Theta \vdash \tau_j}$$

for $j = 1, 2$. We write $\eta_j = (\eta_i^-, \eta_i^+)$. (2) Product type.

We define $\eta : G_{\Theta \vdash \tau_1 \times \tau_2} \circ \text{Inj}^n \rightarrow \text{Inj} \circ F_{\Theta \vdash \tau_1 \times \tau_2}$ as follows. For every $\vec{\sigma} \in (\text{FPC}_!^\delta)^n$,

$$\eta_{\vec{\sigma}} := ((\eta_i^-, \eta_i^-)_{\vec{\sigma}}, (\eta_i^+, \eta_i^+)_{\vec{\sigma}})$$

where

$$(\eta_i^- \times \eta_i^-)_{\vec{\sigma}} = \lambda p. (\eta_i^-(\text{fst}(p)), \eta_i^-(\text{snd}(p)))$$

$$(\eta_i^+ \times \eta_i^+)_{\vec{\sigma}} = \lambda q. (\eta_i^+(\text{fst}(q)), \eta_i^+(\text{snd}(q))).$$

(3) Sum type.

We define $\eta : G_{\Theta \vdash \tau_1 + \tau_2} \circ \text{Inj}^n \rightarrow \text{Inj} \circ F_{\Theta \vdash \tau_1 + \tau_2}$ as follows. For every $\vec{\sigma} \in (\text{FPC}_!^\delta)^n$,

$$\eta_{\vec{\sigma}} := ((\eta_i^-, \eta_i^+)_{\vec{\sigma}}, (\eta_i^+ + \eta_i^+)_{\vec{\sigma}})$$
where
\[
(\eta_i^- + \eta_2^+) = \lambda z. \text{case}(z) \text{ of } \text{inl}(x). \text{inl}(\eta_i^-(x)) \text{ or } \text{inr}(y). \text{inr}(\eta_2^+(y)) \\
(\eta_1^+ + \eta_2^-) = \lambda z. \text{case}(z) \text{ of } \text{inl}(x). \text{inl}(\eta_1^+(x)) \text{ or } \text{inr}(y). \text{inr}(\eta_2^-(y)).
\]

(4) Function type.

We define \( \eta : G_{\Theta \vdash \tau_1 \rightarrow \tau_2} \circ \text{Inj}^n \rightarrow \text{Inj} \circ \text{F}_{\Theta \vdash \tau_1 \rightarrow \tau_2} \) as follows. For every \( \vec{\sigma} \in (\text{FPC}_\tau^\delta)^n \),
\[
\eta_{\vec{\sigma}} := (\eta_1^+ \rightarrow \eta_2^-, \eta_1^- \rightarrow \eta_2^+)
\]
where
\[
(\eta_1^+ \rightarrow \eta_2^-) = \lambda g. \eta_2^- \circ g \circ \eta_1^+
\]
\[
(\eta_1^- \rightarrow \eta_2^+) = \lambda h. \eta_2^+ \circ h \circ \eta_1^-.
\]

(5) Lifted type.

Let \( \Theta \vdash \tau \) be given. The induction hypothesis asserts that there is a natural isomorphism
\[
\eta : G_{\Theta \vdash \tau} \circ \text{Inj}^n \rightarrow \text{Inj} \circ \text{F}_{\Theta \vdash \tau}.
\]

We define a natural isomorphism
\[
\eta_\Lambda : G_{\Theta \vdash \tau \downarrow} \circ \text{Inj}^n \rightarrow \text{Inj} \circ \text{F}_{\Theta \vdash \tau \downarrow}
\]
as follows. For every \( \vec{\sigma} \in (\text{FPC}_\tau^\delta)^n \),
\[
(\eta_\Lambda)_{\vec{\sigma}} := \text{case}(x) \text{ of } \text{up}(x). \text{up}(\eta(x)).
\]

(6) Recursive type.

Let \( \Theta, X \vdash \tau \) be given. The induction hypothesis asserts that there is a natural isomorphism
\[
\zeta : G_{\Theta, X \vdash \tau} \circ \text{Inj}^n \rightarrow \text{Inj} \circ \text{F}_{\Theta, X \vdash \tau}.
\]

We define a natural isomorphism
\[
\eta : G_{\Theta \vdash \mu \tau} \circ \text{Inj}^n \rightarrow \text{Inj} \circ \text{F}_{\Theta \vdash \mu \tau}
\]
as follows. For every \( \vec{\sigma} \in (\text{FPC}_\tau^\delta)^n \), define \( \eta_{\vec{\sigma}} \) to be the unique map \( h \) which fits into the commutative diagram:
\[
\begin{array}{ccc}
G(\text{Inj}^n(\vec{\sigma}), H \circ \text{Inj}^n(\vec{\sigma})) & & \xrightarrow{i_{\text{Inj}^n(\vec{\sigma})}} \xrightarrow{i_{\text{Inj}^n(\vec{\sigma})}} H \circ \text{Inj}^n(\vec{\sigma}) \\
G(\text{Inj}^n(\vec{\sigma}), h) & & \\
G \circ \text{Inj}^{n+1}(\vec{\sigma}, F_{\Theta \vdash \mu \tau}(\vec{\sigma})) & \xrightarrow{\zeta_{\vec{\sigma}, F_{\Theta \vdash \mu \tau}(\vec{\sigma})}} \xrightarrow{\zeta_{\vec{\sigma}, F_{\Theta \vdash \mu \tau}(\vec{\sigma})}} \text{Inj} \circ F_{\Theta, X \vdash \tau}(\vec{\sigma}, F_{\Theta \vdash \mu \tau}(\vec{\sigma})) & \xrightarrow{\langle \text{unfold, fold} \rangle} \text{Inj} \circ F_{\Theta \vdash \mu \tau}(\vec{\sigma}) \\
& & \xrightarrow{h} \\
\end{array}
\]
where \( H := \mu G \). Note that the existence and uniqueness of \( h \) is guaranteed by the initiality of \( i_{\text{Inj}^n(\vec{\sigma})} : G(\text{Inj}^n(\vec{\sigma}), H \circ \text{Inj}^n(\vec{\sigma})) \rightarrow H \circ \text{Inj}^n(\vec{\sigma}) \). Since \( \zeta, i \) and \( \langle \text{unfold, fold} \rangle \) 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 \( i_{\text{Inj}^n(\vec{\sigma})}^{-1} : H \circ \text{Inj}^n(\vec{\sigma}) \rightarrow G(\text{Inj}^n(\vec{\sigma}), H \circ \text{Inj}^n(\vec{\sigma})) \),
Inj\(^n(\bar{\sigma})\) is a final coalgebra and \(\zeta\) is an isomorphism, there exists a unique \(g\) which fits into the following commutative diagram:

\[
\begin{array}{ccc}
G(\text{Inj}^n(\bar{\sigma}), H \circ \text{Inj}^n(\bar{\sigma})) & \overset{i_{\text{Inj}^n(\bar{\sigma})}^{-1}}{\longrightarrow} & H \circ \text{Inj}^n(\bar{\sigma}) \\
G(\text{Inj}^n(\bar{\sigma}), g) & \downarrow & \\
G \circ \text{Inj}^{n+1}(\bar{\sigma}, F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})) & \rightarrow & \text{Inj} \circ F_{\bar{\theta} X_1 \tau}(\bar{\sigma}, F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})) & \rightarrow & \text{Inj} \circ F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})
\end{array}
\]

We claim that \(g \circ h = \text{id}_{H \circ \text{Inj}^n(\bar{\sigma})}\) and \(h \circ g = \text{id}_{\text{Inj}^{n+1}(\bar{\sigma})}\). To prove the first equation, notice that \(g \circ h\) is an \(G(\text{Inj}^n(\bar{\sigma}), \_)-\)algebra endomorphism on \(H \circ \text{Inj}^n(\bar{\sigma})\).

Thus by initiality of \(i_{\text{Inj}^n(\bar{\sigma})}: G(\text{Inj}^n(\bar{\sigma}), H \circ \text{Inj}^n(\bar{\sigma})) \rightarrow H \circ \text{Inj}^n(\bar{\sigma})\), it must be that \(g \circ h = \text{id}_{H \circ \text{Inj}^n(\bar{\sigma})}\). 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:

\[
\begin{array}{ccc}
G(\text{Inj}^n(\bar{\sigma}), h \circ g) & \downarrow & \\
G \circ \text{Inj}^{n+1}(\bar{\sigma}, F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})) & \rightarrow & \text{Inj} \circ F_{\bar{\theta} X_1 \tau}(\bar{\sigma}, F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})) & \rightarrow & \text{Inj} \circ F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})
\end{array}
\]

where the dotted arrow is the morphism

\[
\langle F_{\bar{\theta} X_1 \tau}(\bar{\sigma}, (h \circ g)^-), F_{\bar{\theta} X_1 \tau}(\bar{\sigma}, (h \circ g)^+) \rangle.
\]

Go for the second quadrangle, \((h \circ g)^-\) and \((h \circ g)^+\) are both endomorphisms on \(F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})\). By the initiality of \(\text{fold} : F_{\bar{\theta} X_1 \tau}(\bar{\sigma}, F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})) \rightarrow F_{\bar{\theta} \mu X}. \tau(\bar{\sigma})\), we conclude that \(h \circ g = \text{id}_{\text{Inj}^{n+1}(\bar{\sigma})}\).

Within the framework of \(\mathbf{FP}\mathbf{C}_{\!1}\), 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, as described in the third paragraph of Section 6.2
2. Carry out treatment (i.e., the investigation in question), e.g. calculating the bifree algebra of some syntactic functors as in Theorem 6.11
3. Perform a diagonalisation in the sense of Proposition 6.15 to derive the relevant conclusion regarding the original type expression.

In view of Proposition 6.15, this three-fold process can be carried out directly in the
An Operational Domain-theoretic Treatment of Recursive Types

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 lighter categorical overhead.

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

7. 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;
2. the uses of our operational domain theory of recursive types in the field of exact real arithmetic (e.g., in reasoning with streams of signed-digits in the representation of real numbers such as in Simpson, 1998), and
3. the possibility of developing an operational domain-theory that caters for non-deterministic languages, such as (Hennessy and Ashcroft, 1980).

Acknowledgements. Many people have contributed to the birth and development of ideas in this paper. Special thanks go to M. H. Escardó, whose deep insight and gentle advice I have always benefited from, as his student. I am grateful to P. B. Levy for first suggesting product categories as an alternative categorical framework. Following this suggestion, I was further reassured by M. P. Fiore that there should be no problems if I proceed 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. Thanks to M. H. Escardó and A. K. Simpson for encouraging me (time and again) to write this journal article. My gratitude also extends to the anonymous referee for detailed proof-reading and many helpful comments which tremendously improved the style of this paper. Special thanks to my NIE colleagues for their support so that I can focus on the task of writing during term time.
References


