The University of Birmingham
School of Computer Science

Operational Domain Theory and Topology of Sequential Functional Languages

Weng Kin Ho

Dr. Martín Escardó
Supervisor

Dr. Alex Simpson
External Examiner

Dr. Paul Blain Levy
Internal Examiner

A dissertation submitted for the degree of
DOCTOR OF PHILOSOPHY
in
Computer Science
The University of Birmingham
Submitted August 18, 2006
Defended October 4, 2006

Declaration

The results reported in Part III consist of joint work with Martín Escardó [14]. All the other results reported in this thesis are due to the author, except for background results, which are clearly stated as such. Some of the results in Part IV have already appeared as [28].

Note This version of the thesis, produced on October 31, 2006, is the result of completing all the minor modifications as suggested by both the examiners in the viva report (Ref: CLM/AC/497773).

Abstract

We develop an operational domain theory to reason about programs in sequential functional languages. The central idea is to export domain-theoretic techniques of the Scott denotational semantics directly to the study of contextual pre-order and equivalence. We investigate to what extent this can be done for two deterministic functional programming languages: PCF (Programming-language for Computable Functionals) and FPC (Fixed Point Calculus).

Traditionally, domain theory and topology in programming languages have been applied to manufacture and study denotational models, for instance, the Scott model of PCF. For a sequential language like this, it is well-known that the match of the model with the operational semantics is imprecise: computational adequacy holds but full abstraction fails.

One of the main achievements is a reconciliation of a good deal of domain theory and topology with sequential computation. This is accomplished by side-stepping denotational semantics and reformulating domain-theoretic and topological notions directly in terms of programming concepts, interpreted in an operational way. Regarding operational domain theory, we introduce operational finiteness. The upshot is the SFP theorem: Every PCF type has an SFP structure. In particular, the set of finite elements of each type forms a basis. Regarding operational topology, we work with an operational notion of compactness. The elegance of the theory lies not only in the interplay of these two notions but also in the reasoning principles that emerge. For instance, we show that total programs with values on certain types are uniformly continuous on compact sets of total elements. We apply this and other conclusions to prove the correctness of non-trivial PCF programs that manipulate infinite data.

For FPC, an operational domain theory is developed for treating recursive types. The principal approach taken here deviates from classical domain theory in that we do not produce recursive types via inverse limit constructions - we have it for free by working directly with the operational semantics of FPC. The important step taken in this work is to extend type expressions to legitimate n-ary functors on suitable 'syntactic' categories. To achieve this, we rely on operational versions of the Plotkin's uniformity principle and the minimal

invariance property. This provides a basis for us to introduce the operational notion of algebraic compactness. We then establish algebraic compactness results in this operational setting. In addition, a "pre-deflationary" structure is derived on closed FPC types and this is used to generalise the "Generic Approximation Lemma" recently developed by Hutton and Gibbons. This lemma provided a powerful tool for proving program equivalence by simple inductions, where previously various other more complex methods had been employed.

For my beloved family
"As for me and my house we will serve the Lord." (Joshua 24:15)

Acknowledgements

I thank God for giving me this once-in-a-lifetime opportunity to pursue a PhD. While in Singapore, a joint work with my M.Sc. supervisor, Dongsheng Zhao, in the Nanyang Technological University regarding Scott-closed sets saw a small breakthrough. Things just somehow started to unfold in my favour, beginning with a rather fortuitous acquaintance with my PhD supervisor, Martín Escardó. In our communication, we were amazed that my work [27] on characterising the Scott-closed set lattices was closely related to Martín's work [12] on injective locales over perfect sublocale embeddings. Subsequently, I received the kind invitation from Martín Escardó and Achim Jung to present my findings in the workshop Domains VI in Birmingham, September 16-19, 2002. Besides meeting the domain theory and programming language semantics community, I received an unexpected opportunity to be interviewed by Peter Hancox, the Admissions Tutor. In 2003, I was awarded an International Research Studentship from the School of Computer (The University of Birmingham), which made this work possible. I am grateful to the School of Computer Science, The University of Birmingham, and especially to Martín Escardó, Achim Jung and Peter Hancox.

My three years of study in Birmingham proved to be a rich experience. I benefitted from a rich resource of research material and expert advice. I am extremely grateful to my supervisor, Martín Escardó, for all his help and encouragement. This was especially the case during my first year of study in which I must put in more effort to understand the "computer science" component of my research. I benefitted from the weekly meetings with him, in which I was inspired and intrigued by his vast knowledge and ingenious ideas. His deep insights (especially in his work [13]) so very much motivated and shaped the entire course of my research that he deserves most of the credit for my work. I am particularly thankful for the opportunity of a joint work (Escardó & Ho [14]) with him, in which we had many fruitful discussions. I shall always remain in intellectual debt with Martín because he has taught me how to do research.

I want to extend my special gratitude to the School's research group: Mathematical Foundations of Computer Science. They provided a ready audience (together with constructive criticisms) to whom I shared my research

findings. In particular, I thank Paul Blain Levy for giving me opportunities in the informal lunch talks where I was allowed to rehearse for more formal ones. Regarding external seminars, I am also thankful to Dongsheng Zhao (Nanyang Technological University, Singapore) and Alexander Kurz (University of Leicester, UK) for inviting me to speak in several occasions about my work. In particular, I am very encouraged by Alexander Kurz and Neil Ghani (University of Leicester, UK) when they expressed enthusiasm in my work.

My study was enriched by the various research conferences which I attended. I would like to acknowledge the Research Committee in funding my trips to academic events, such as MGS 2003 and 2004, LICS 2005 and MFPS 2006. In these events, not only did I learn about the works of others but also meet several excellent researchers from all over the world. In particular, I got acquainted with Andrew Pitts and Thomas Streicher who later offered me very timely and useful advice. Andrew Pitts patiently entertained my series of emails seeking clarification on the operational machineries which he developed in Pitts [41]. Following his suggestion, I was able to develop the operational toolkit for arguing about program equivalence for FPC (cf. Part II). Thomas Streicher went the extra mile in offering expert advice as I fixed a gap in Alexander Rohr's reasoning regarding the minimal invariance of syntactic functors (cf. Rohr [47]). This resulted in the establishment of operational algebraic compactness with respect to the class of syntactic functors. I am grateful to both of them for patiently and carefully reading my (manuscript) preprint of Ho [28]. Regarding the Midland Graduate School, I was most inspired by a series of lectures given by Achim Jung on denotational semantics in MGS 2004.

During these three years, I have had many invaluable discussions with many researchers such as Steve Vickers (topology via logic), Paul Levy (denotational semantics) and Achim Jung (domain theory, especially Chapter 5 of [3]). My fellow colleagues, including José Raymundo Marcial-Romero, Thomas Anberrée and Mohamed El-Zawawy, often lent me their ears. We had a wonderful time together in our reading group on the domain-theory bible [21]. I am grateful to Thomas Anberrée for proof-reading some parts of this thesis. I want to especially thank Martín for carefully having proof-read the entire thesis. All the remaining mistakes are, of course, due to myself.

All the commutative diagrams in this thesis were produced with Paul Taylor's commutative diagrams package.

Special thanks goes to my wife, Hwee Hoong, and my (now four-year old) son, Samuel, for supporting me in every possible way throughout my study in the UK. Each time I get back from work, their warm welcome and hugs meant an entire world to me. Also I would like to thank the unflagging

support of my father and my parents-in-law for my overseas studies despite their old age.

Finally, I thank my family-in-Christ from the Birmingham Chinese Methodist Church (UK) and the Aldersgate Methodist Church (Singapore) for giving me the spiritual support and guidance during these three years in the UK.

Contents

List of Figures

Chapter 1

Introduction

We develop an operational domain theory to reason about programs in sequential functional languages. The central idea is to export domain-theoretic techniques of the Scott denotational semantics directly to the study of contextual preorder and equivalence. We investigate to what extent this can be done for two call-by-name deterministic functional programming languages: PCF (Programming language for Computable Functions) and FPC (Fixed Point Calculus).

Traditionally, domain theory and topology in programming languages have been applied to manufacture and study denotational models. The Scott model, for instance, uses mathematical theory of domains as the foundation for developing methods for reasoning about program equivalence for languages such as PCF and FPC (cf. Scott [51], Plotkin [43]) . But for sequential languages like these, it is well-known that the match of the model with the operational semantics is imprecise: computational adequacy holds but full abstraction fails (Plotkin [42]).

One solution to this problem is to bypass denotational semantics and reformulate domain-theoretic notions directly in terms of programming concepts, interpreted in an operational way. The idea that order-theoretic techniques from domain theory can be directly understood in terms of operational semantics goes back to Mason, Smith, Talcott [36] and Sands [48]. In fact, operational methods, such as co-inductive techniques, have been imported into functional programming earlier by several people: Dybjer and Sander [11], Abramsky [2], Howe [29, 30] and Gordon [22]. Notably, these operationally-based theories of program equivalence have been systematically reworked for the functional language, PCFL (PCF with pairs and lazy Lists) in Pitts [41].

However, works on operational domain theory have, more often than not, focused on the reasoning principles based on operational methods (such as the co-inductive principle and the 'compactness' of evaluation, cf. Pitts [41])

and neglected the topological side of the story. One clear exception is Escardó [13] in which it is demonstrated that topological techniques can be directly understood in terms of the operational semantics, and moreover, are applicable to sequential languages.

One main objective of our study is to achieve a reconciliation of a good deal of domain theory and topology with sequential computation. We accomplish this by side-stepping denotational semantics and reformulating both domain-theoretic and topological notions directly in terms of programming concepts, interpreted in an operational way. Exploiting the strong interplay between order theory and topology, understood purely in terms of computational notions, we produce more powerful reasoning principles.

Another objective of our study is to understand the operational interpretation of recursive types. Recently there has been a steady stream of literature which deals with this aspect, e.g., Gordon [23], Pitts [41], Abadi & Fiore [1] and Birkedal & Harper [9]. The first two works developed operational techniques, such as the co-induction principle, for various versions of PCF and only give a slight indication (no details) of how these can also be done in FPC. Moreover, important and well-developed notions in classical domain theory, such as minimal invariance of endofunctors and algebraic compactness have yet to find their place in the operational setting. Regarding minimal invariance, there are two exceptions: (1) Birkedal & Harper [9], and (2) Lassen [33, 34][1]. In both these works, a 'syntactic' minimal invariance theorem had been established in a purely operational way. Unfortunately, the languages they considered has only one top-level recursive type. This means that the machineries developed therein are not readily applicable to languages like FPC which do have facilities for handling user-declared recursive data types and nested recursion.

Our aim, in this present work, is to fill in the gap by giving a comprehensive operational treatment of recursive type. This involves establishing operational principles of minimal invariance and operational algebraic compactness results. Additionally, we show how these lead to a powerful, yet simple, proof techniques for reasoning about program equivalence and correctness in FPC.

[1] I was pointed to Lassen's works near the completion of writing of this thesis. Special thanks to Paul B. Levy who drew my attention to the 'syntactic minimal invariance' in Lassen [33].

1.1 Brief summary of contributions

We now proceed to a slightly more detailed and technical exposition of our main results and underlying ideas.

1.1.1 Operational domain theory and topology for PCF

The operational domain theory and topology developed for the language PCF reported in Part III of this thesis consists of joint work with Martín Escardó [14].

Rational-chain completeness. One major highlight of Pitts' work [41] is that the collection of PCFL terms preordered by the contextual preorder enjoys a restricted amount of chain-completeness, known as rational-chain completeness. We identify this completeness condition as a salient feature in the study of the contextual preorder. So the most crucial step in developing an operational domain theory is to replace the directed sets by rational chains. These rational chains, we observe, are equivalent to programs defined on a "vertical natural numbers" type $\overline{\omega}$. Many of the classical definitions and theorems go through smoothly with this modification. For example, (1) rational chains have suprema in the contextual order, and (2) programs of functional type preserve suprema of rational chains.

Operational topology. Regarding topology, we define open sets of elements via programs with values on a "Sierpinski" type, and compact sets of elements via Sierpinski-valued universal-quantification programs. Then (1) the open sets of any type are closed under the formation of finite intersections and rational unions, (2) open sets are "rationally Scott open", (3) compact sets satisfy the "rational Heine–Borel property", (4) total programs with values on certain types are uniformly continuous on compact sets of total elements.

The idea that topological techniques can be directly understood in terms of operational semantics, and, moreover, are applicable to sequential languages, is due to Escardó [13]. In particular, we have taken our operational notion of compactness and some material about it from that reference. The main novelty here is a uniform-continuity principle, which plays a crucial role in the sample applications given in Chapter 12. We also have a Kleene-Kreisel density theorem for total elements, and a number of continuity principles based on finite elements.

Operational finiteness. Various ways have been proposed to formulate notions of finiteness in operational settings. Our approach is to take the classical domain-theoretic formulation, with directed sets replaced by rational chains. Again well known classical results regarding finiteness continue to hold. For instance, (1) every element (closed term) of any type is the supremum of a rational chain of finite elements, and (2) two programs of functional type are contextually equivalent if and only if they produce a contextually equivalent result for every finite input. Crucially, we have an SFP-style characterisation of finiteness using rational chains of deflations. Already in Mason *et al* [36], one can find, in addition to rational-chain principles, two equivalent formulations of an operational notion of finiteness. One is similar to ours except that directed sets of closed terms are used instead of rational chains, and the other is analogous to SFP-characterisation of finiteness. In addition to redeveloping their formulations in terms of rational chains, we add a topological characterisation.

Data language. In order to be able to formulate certain specifications of higher-type programs without invoking a denotational semantics, we work with a "data language" for our programming language PCF, which consists of the latter extended with first-order "oracles" (Escardó [13]). The idea is to have a more powerful environment in order to get stronger program specifications. In this work, we establish some folkloric results, namely that program equivalence defined by ground data contexts coincides with program equivalence defined by ground program contexts, but the notion of totality changes.

Program correctness. We illustrate the scope and flexibility of the theory by applying our conclusions to prove the correctness of non-trivial programs that manipulate infinite data. We take one such example from Simpson [52]. In order to avoid having exact real-number computation as a prerequisite, as in that reference, we consider modified versions of the program and its specification that retain their essential aspects. We show that the given specification and proof in the Scott model can be directly understood in our operational setting. This is relevant because, although this program is sequential, its original specification and proof are developed in the Scott model, which, as discussed above, doesn't faithfully model sequential computation.

Although our development is operational, we never invoke evaluation mechanisms directly. We instead rely on known extensionality, monotonicity, and rational-chain principles for contextual equivalence and order. Moreover,

with the exception of the proof of the density theorem, we don't perform syntactic manipulations with terms.

1.1.2 Operational domain theory for FPC

We continue a similar program of exporting operational domain-theoretic techniques to treat recursive types. This is done for the language FPC which has facilities for defining user-declared recursive types. The principal approach taken here deviates from classical domain theory in that we do not produce recursive types via inverse limits constructions - we have it for free by working directly with the operational semantics of FPC.

Part of the operational domain theory developed for the language FPC reported in Part IV consists of work that appeared in Ho [28][2].

Type expressions as functors. The important step taken in this part of the work is to view type expressions (more accurately, types-in-context) as legitimate n-ary functors on certain 'syntactic' categories of closed types. In this operational setting, such functors arising from type expressions exhibit familiar properties such as monotonicity and local continuity with respect to the contextual preorder. In the process of establishing the functoriality of type expressions, we prove operational analogues of useful domain-theoretic results such as the Plotkin's uniformity principle and the minimal invariance property. The functoriality of type expressions was first developed by M. Abadi and M. Fiore [1] using equational theories, and we closely follow their approach although there are some differences (to be explained in the technical development).

Operational algebraic compactness. In classical domain theory, it is already well established that every locally continuous endofunctor has an initial algebra and a final coalgebra and most crucially they coincide. In a sequence of influential works of P.J. Freyd [17, 18, 19] during the 1990s, the notion of algebraic completeness and algebraic compactness have been axiomatised in his categorical treatment. One important consequence is the famous Freyd's Product Theorem which asserts that a finite product of algebraically compact categories is again algebraically compact. The reader should note that the works of Freyd can be understood in Kleisli categorical settings (cf. Simpson [53]). However, these notions have not found their

[2]Since its publication, materials contained therein have been improved on and included in various chapters of Part IV. In addition, mistakes in the [28] have also been rectified. The interested reader may find these improvements listed in the Appendix A.

places in a concrete operational setting. The functorial status of types-in-context now provides a sound basis for us to introduce an operational notion of algebraic compactness. It turns out that the syntactic categories we are working with are algebraically compact with respect to definable functors.

Generic approximation lemma. In Hutton & Gibbons [31], a "Generic Approximation Lemma" was established, via denotational semantics, for polynomial types (i.e., types built only from unit, sums and products). In the same reference, they suggested it would be possible to generalise the lemma "to mutually recursive, parametrised, exponential and nested datatypes" (cf. p.4 of Hutton & Gibbons [31]). In this present work, we confirm this by deriving a pre-deflationary structure on closed FPC types. We also demonstrate that the "Generic Approximation Lemma" is a powerful tool for proving program equivalence by simple inductions, where previously various other more complex methods had been employed.

1.2 Additional contributions

In order to make use of several important domain-theoretic facts concerning the contextual preorder and equivalence in both the languages, it is necessary for us to rework the results of Pitts [41] to suit our languages. Since PCFL (which A. Pitts considered) and our version of PCF are similar, we have chosen to outline in Chapter 6 the necessary modifications without detailed proofs. However, because the existing literature[3] does not provide explicit details about developing operationally based methods of reasoning about recursively typed programs, we choose to rework all the details for FPC following Pitts' work [41] closely. There are two main results proven here: (1) Contextual equivalence is characterised as the largest FPC bisimulation. (2) Rational chains have suprema in the contextual order (rational-chain completeness) and programs of functional type preserve suprema of rational chains (rational continuity). In short, the entire set of operational machinery necessary to develop our theory is collected at one place in Part II (Operational Toolkit).

1.3 Background

The prerequisites of this work are basic category theory [32, 46], domain theory [3, 21, 43], operational and denotational semantics of PCF [24, 40,

[3]There are two exceptions here: Birkedal & Harper [9] and Lassen [33, 34].

42, 58] and FPC [24, 37]. To appreciate the development of operational topology in this thesis, it is ideal to have a nodding acquaintance with basic topology [10, 55, 60, 61] though not necessary.

Since PCF and FPC are heavily used in this thesis, we include background chapters on these subjects. The background chapters also contain some materials on domain theory and category theory which are essential for later development.

1.4 Organisation

This thesis is organised in four parts:

I Background

II Operational Toolkit

III Operational Domain Theory for PCF

IV Operational Domain theory for FPC

An index of definitions is included - it contains the *emphasised* defined terms and some mathematical symbols.

Part I

Background

This part serves as a reference. In our organisation of the background material, we introduce essential concepts and highlight important underlying ideas of well-known results without spelling out the details. In Chapter 2, we introduce domain theory and category theory necessary for the development of our theory. In Chapters 3 and 4, we introduce the syntax and the operational semantics of the languages PCF and FPC. In Chapter 5, we introduce important computational analogues of various topological notions, such as open set, continuous map, Hausdorff space and compact set. The material presented in this chapter is taken from Escardó [13] where these notions were first introduced. Note that in this chapter, we have taken proofs directly from [13] and also included some proofs which are meant to be exercises in [13].

Chapter 2

Prerequisites

In this chapter, we cover essential notions in domain theory and category theory. Regarding domain theory, we have included material on the solution of recursive domain equations. The reader can find more comprehensive treatments of these subjects in [3, 21, 43] for domain theory (including recursive domain equations), and [32, 46] for category theory.

2.1 Domain theory

Domain theory may be considered a branch of topology that has a convenient presentation via order-theoretic notions. This perspective, essentially due to Dana Scott, was first introduced in his seminal papers [49, 50]. The idea here is to employ order-theoretic and topological techniques in understanding the meaning of data types.

2.1.1 Directed complete posets

A *preordered set* is a set equipped with a reflexive and transitive binary relation $\sqsubseteq$ (called a *preorder*). Preordered sets are prevalent in topology as any topological space X can be endowed with the following preorder:

$$x \sqsubseteq y \iff \forall \text{ open subset } U \subseteq X.(x \in U \implies y \in U)$$

which is called the *specialisation order* of X. A *poset* (*partially ordered set*) is a pre-ordered set $(P, \sqsubseteq)$ with $\sqsubseteq$ being antisymmetric. Any T_0 space, i.e., a topological space in which no two distinct points share exactly the same family of open neighbourhoods, is a poset with respect to its specialisation order.

Given a poset $(P, \sqsubseteq)$, $p \in P$ and $X \subseteq P$, we adopt the following notations:

1. $\uparrow p := \{x \in P | p \sqsubseteq x\}$ and $\downarrow p := \{x \in P | x \sqsubseteq p\}$,

2. $\uparrow X := \bigcup_{x \in X} \uparrow x$, and $\downarrow X := \bigcup_{x \in X} \downarrow x$.

$X \subseteq P$ is *lower* if $X = \downarrow X$. Dually, we define the notion of an *upper* subset. The element $p \in P$ is an *upper bound* of X if for all $x \in X$, it holds that $x \sqsubseteq p$. Dually, we define the notion of a *lower bound*. The *least upper bound* (or the *supremum*) of X, if it exists, is denoted by $\bigsqcup X$. Dually, $\bigsqcap X$ denotes the *greatest lower bound* (or the *infimum*) of X.

A subset X of a poset D is *directed* if every finite subset of X has an upper bound in X. Note that a directed subset, by its definition, cannot be empty since the empty set is finite. A lower directed subset is called an *ideal*. The set of all the ideals of a poset D is denoted by $\mathrm{Id}(D)$. We adopt the notation $\bigsqcup^{\uparrow} X$ to mean the supremum of a directed set if it exists. A poset $(D, \sqsubseteq)$ is a *dcpo* (*directed complete poset*) if for every directed subset X of D, $\bigsqcup^{\uparrow} X$ exists. Note that if D is a dcpo, then so is $(\mathrm{Id}(D), \subseteq)$.

A *monotone* function between posets is one which preserves order. A (*order-*)*continuous* function between dcpos is one which preserves directed suprema. Such a function is necessarily monotone.

2.1.2 Scott topology

The *Scott topology* on a dcpo D is one in which the open sets U are

1. upper, i.e., $\uparrow U = U$, and

2. inaccessible by directed suprema, i.e.,
   $\forall$ directed subset $X \subseteq D.(\bigsqcup X \in U \implies \exists x \in X.x \in U)$.

By taking complements, a set is Scott-closed if and only if it is lower and contains the suprema of its directed subsets. One pleasant aspect of the Scott topology on dcpos is that the order-continuous functions are exactly the topologically continuous ones with respect to the Scott topologies. In addition, because sets of the form $D \setminus \downarrow p$ are Scott-open the specialisation order of a dcpo D with respect to the Scott topology coincides with the underlying order.

2.1.3 Dcpos and least fixed-points

A dcpo which has a least element is called a *pointed* dcpo. The least element, also called the *bottom*, of a pointed dcpo is denoted by $\bot$. The category of pointed dcpos and continuous functions is denoted by $\mathbf{DCPO}_{\bot}$. A function $f : D \to E$ between dcpos is *strict* if it preserves the bottom. The category

of pointed dcpos and strict continuous functions is denoted by $\mathbf{DCPO}_{\perp!}$. A *fixed-point* of an endofunction $f : X \to X$ is an element $x \in X$ such that $f(x) = x$. It turns out that every continuous endofunction $f : D \to D$ on a pointed dcpo D always has a *least fixed-point* denoted by $\mu(f)$ and given by $\bigsqcup_{n\in\mathbb{N}} f^{(n)}(\perp)$. With regards to strict functions and fixed-points, one handy lemma commonly known as the Plotkin's "axiom" (also known as Plotkin's uniformity principle) stands out amongst others.

Lemma 2.1.1. *Let D and E be pointed dcpo's and let*

$$\begin{array}{ccc} D & \xrightarrow{h} & E \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ D & \xrightarrow[h]{} & E \end{array}$$

be a commutative diagram of continuous functions where h is strict. Then

$$\mu(g) = h(\mu(f)).$$

2.1.4 Complete lattices and the Tarski-Knaster fixed-point theorem

A *complete lattice* is a poset $(L, \sqsubseteq)$ for which every subset $S \subseteq L$ has a least upper bound. This is equivalent to requiring that every subset has a greatest lower bound. Let f be an endofunction on a complete lattice L. A *post-fixed point* of f is an element x of L such that $x \sqsubseteq f(x)$.

Theorem 2.1.2. *(Tarski-Knaster fixed point theorem)*
Every monotone endofunction f on a complete lattice $(L, \sqsubseteq)$ possesses a greatest post-fixed point, $\nu(f)$. This element is in fact the greatest element of the set $\{x \in L | x = f(x)\}$ of fixed points of f.

2.1.5 Domains and algebraic domains

In order to define domains, we must first define the *way-below* relation $\ll$ on a given dcpo D:

$$x \ll y \iff \forall \text{ directed subset } A \subseteq D.(y \sqsubseteq \bigsqcup\!\!\uparrow A \implies \exists a \in A.x \sqsubseteq a).$$

Using the notion of ideals, the defining condition is equivalent to:

$$\forall X \in \mathrm{Id}(D).(y \sqsubseteq \bigsqcup^{\uparrow} X \implies x \in X).$$

The following standard properties regarding $\ll$ can be readily verified:

(1) $x \ll y \implies x \sqsubseteq y$.

(2) $\bot \ll x$ for any $x \in D$.

(3) $u \sqsubseteq x \ll y \sqsubseteq v \implies u \ll v$.

(4) If $u \ll x, v \ll x$ and $u \sqcup v$ exists, then $u \sqcup v \ll x$.

A dcpo D is *continuous* if for every $x \in D$,

1. the set $\downarrow x := \{d \in D | d \ll x\}$ is a directed subset of D, and
2. $\bigsqcup^{\uparrow} \downarrow x = x$.

In the literature, condition (2) is called the axiom of approximation. Moreover, this axiom is equivalent to:

$$x \not\sqsubseteq y \implies \exists u \ll x.u \not\sqsubseteq y.$$

The term *domain* is used throughout this thesis to mean a continuous dcpo. One characteristic feature of a domain is that the relation $\ll$ satisfies the following interpolation property:

$$x \ll y \implies \exists u \in D.x \ll u \ll y.$$

A *basis* of a domain D is a subset B such that for every $x \in D$, the set $\downarrow x \cap B$ is directed and it holds that

$$x = \bigsqcup^{\uparrow} \downarrow x \cap B.$$

Thus a domain as a subset of itself is a basis. For any basis B of a domain D, the sets $\Uparrow b$ for $b \in B$ form a base of the Scott topology on D. Thus, if D and E are domains with bases B and C, then a function $f : D \to E$ is continuous at x if and only if for every $c \in C$,

$$c \ll f(x) \iff \exists b \in B.(b \ll x) \wedge (c \ll f(b)).$$

This is also refered to as the ϵ-δ characterisation of continuity[1]. Furthermore

$$f \text{ is continuous } \iff f(x) = \bigsqcup_{b \ll x}{}^{\uparrow} f(b).$$

Given a domain D, an element $x \in D$ is *finite* (or compact) if $x \ll x$. In other words, the defining condition is equivalent to:

$$\forall \text{ directed subset } A \subseteq D. x \sqsubseteq \bigsqcup^{\uparrow} A \implies \exists a \in A. x \sqsubseteq a.$$

Let B be a basis of a domain D. Then by definition of $\ll$, whenever $x \ll y$, there exists $b \in B$ such that $x \sqsubseteq b \ll y$. This implies that every finite element belongs to B. In other words, any basis of a domain contains the set of compact elements. A dcpo D is *algebraic* if the finite elements form a basis, which we denote by $\mathrm{K}(D)$. An example of an algebraic dcpo is $\mathrm{Id}(D)$ where D is a dcpo.

The following facts will provide motivation for our definition of operational finiteness in Chapter 10.

Proposition 2.1.3. (e.g. Abramsky and Jung [3], Proposition 2.2.13)
If a dcpo D has a countable basis (in the sense of p. 13), then every directed subset of D contains an ω-chain with the same supremum.

Proposition 2.1.4. *Let D be a dcpo with a countable base (in the sense of p. 13) and $\overline{\omega} := \omega \cup \{\infty\}$ the ordinal domain. Then the following statements are equivalent:*

(i) $x \in D$ is finite.

(ii) For every continuous function $f : \overline{\omega} \to D$, $x \sqsubseteq f(\infty)$ implies that there is $i \in \mathbb{N}$ such that $x \sqsubseteq f(i)$.

Proof. (i) $\Rightarrow$ (ii): Let $x \in D$ be finite and $f : \overline{\omega} \to D$ a continuous function with $x \sqsubseteq f(\infty)$. Since $\infty = \bigsqcup^{\uparrow}_{i \in \mathbb{N}} i$ and f preserves directed suprema, it follows that $f(\infty) = \bigsqcup^{\uparrow}_{i \in \mathbb{N}} f(i)$. Because x is finite, there exists $i \in \mathbb{N}$ such that $x \sqsubseteq f(i)$.
(ii) $\Rightarrow$ (i): Assume $x \in D$ satisfies the condition of (ii) and suppose further that $A \subseteq D$ is directed with $x \sqsubseteq \bigsqcup^{\uparrow} A$. Then Proposition 2.1.3 ensures the existence of an ω-chain C in A with $\bigsqcup^{\uparrow} C = \bigsqcup^{\uparrow} A$. The chain C defines an obvious continuous function $c : \overline{\omega} \to D$ and $c(\infty) = \bigsqcup^{\uparrow} A$. Thus by assumption we have $i \in \mathbb{N}$ such that $x \sqsubseteq c(i)$, i.e., there is $a \in A$ such that $x \sqsubseteq a$. So x is finite. □

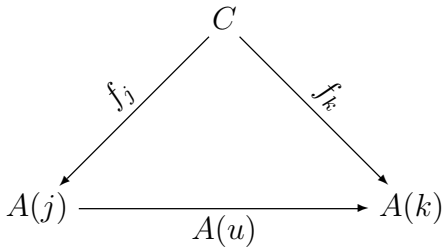
[1] One can compare this with the formulation of continuity in real analysis.

2.2 Essential categorical notions

In this section, we recall some categorical notions used in our operational treatment of recursive types.

2.2.1 Limits and colimits

Let $\mathcal{C}$ be a category and $\mathcal{J}$ a small category. A *diagram* F in $\mathcal{C}$ of type $\mathcal{J}$ is a functor $F : \mathcal{J} \to \mathcal{C}$. For each $\mathcal{C}$-object C, we can define a constant diagram $\Delta_{\mathcal{J}}(C) : \mathcal{J} \to \mathcal{C}, j \mapsto C$. The functor $\Delta_{\mathcal{J}} : \mathcal{C} \to \mathcal{C}^{\mathcal{J}}$ is called the *diagonal functor*. A natural transformation π from $\Delta_{\mathcal{J}}(C)$ to some other diagram A consists of morphisms $\pi_j : C \to A(j)$ such that for each $\mathcal{J}$-morphism $u : j \to k$, the following triangle commutes:



Such a natural transformation is called a cone $\pi : C \to A$ with vertex C. A cone $\pi : L \to A$ with vertex L is *universal* if for every cone $f : C \to A$, there is a unique mediating morphism $g : C \to L$ such that $\pi_j \circ g = f_j$ for all $j \in \mathcal{J}$. The universal cone $\pi : L \to A$ (or less accurately, its vertex L) is called the *limit* of the diagram A, denoted by

$$L = \lim_{\leftarrow \mathcal{J}} A.$$

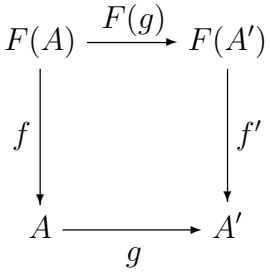
The dual notion is known as *colimit*.

Many categorical notions can be defined in terms of limits or colimits. However, we only invoke the use of limits and colimits in the construction of canonical solutions for recursive domain equations in Section 2.3. One important aspect of this canonicity is the coincidence of the initial algebras and the final coalgebras. We define these two categorical notions in the next section.

2.2.2 Algebras and coalgebras

Let F be an endofunctor on a category $\mathcal{C}$. An *F-algebra* is given by an object A together with a morphism $f : F(A) \to A$, denoted by (A, f). An *F-algebra*

homomorphism from (A, f) to (A', f') is a $\mathcal{C}$-morphism $g : A \to A'$ such that the following diagram commutes:



We denote by $\mathcal{C}^F$ the category of F-algebras and F-algebra homomorphisms. (A, f) is an *initial F-algebra* if it is an initial object in $\mathcal{C}^T$, i.e., for every F-algebra (A', f'), there is a unique algebra homomorphism $g : A \to A'$. The dual notion is known as *coalgebra* (respectively, *final* coalgebra). The following lemma regarding initial algebras is useful.

Lemma 2.2.1. (Lambek's Lemma)
If $i : F(A) \to A$ is an initial F-algebra, then i is an isomorphism.

2.2.3 Adjunctions

An *adjunction* (F, G) between two categories $\mathcal{C}$ and $\mathcal{D}$ is a pair of functors

$$F : \mathcal{C} \to \mathcal{D} \text{ and } G : \mathcal{D} \to \mathcal{C}$$

such that for all $C \in \mathcal{C}$ and $\mathcal{D} \in \mathcal{D}$, there is a bijection between the hom-sets

$$\theta : \mathcal{C}(C, GD) \cong \mathcal{D}(FC, D)$$

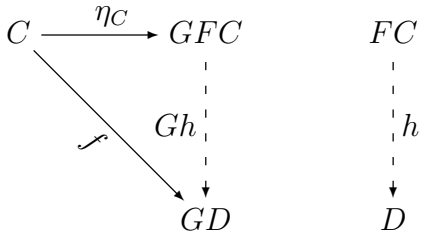
natural in C and D.

It is well-known that the following are equivalent:

(i) $F : \mathcal{C} \leftrightarrows \mathcal{D} : G$ is an adjunction.

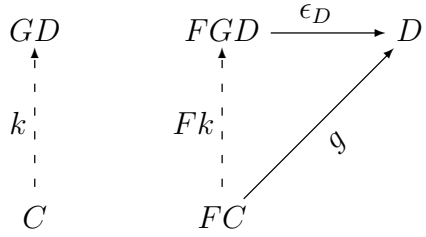
(ii) There exists a natural transformation $\eta : \mathrm{id}_{\mathcal{C}} \to GF$ (called the *unit*) such that for each $\mathcal{C}$-morphism $f : C \to GD$ there is a unique $\mathcal{D}$-

morphism $h : FC \to D$ such that the left triangle



commutes.

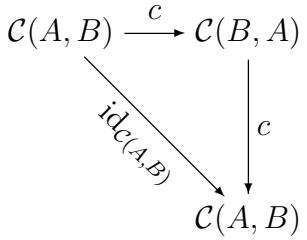
(iii) There exists a natural transformation $\epsilon : FG \to \mathrm{id}_{\mathcal{D}}$ (called the *counit*) such that for each $\mathcal{D}$-morphism $g : FC \to D$ there is a unique $\mathcal{C}$-morphism $k : C \to GD$ such that the right triangle



commutes.

2.2.4 Involutory and locally involutory categories

A *locally involutory category*[2] is a category $\mathcal{C}$ with a local involution $c : \mathcal{C} \to \mathcal{C}^{\mathrm{op}}$, i.e., for all $C \in \mathcal{C}$ it holds that $c(C) = C$ and for any objects $A, B \in \mathcal{C}$ the following diagram



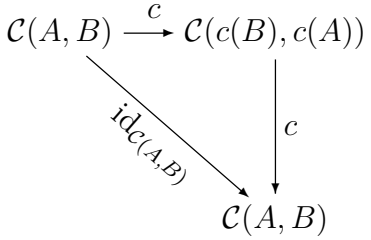
commutes in **Set**.

The category of locally involutory categories, **LocInvCat**, has as objects the locally involutory categories $(\mathcal{C}, c)$ and as morphisms those functors

[2]The term "locally involutory category" was introduced to me in a personal communication with Paul B. Levy.

$F : (\mathcal{C}, c) \to (\mathcal{D}, d)$ between locally involutory categories such that $F \circ c = d \circ F$.

An *involutory category* is a category $\mathcal{C}$ with an involution $c : \mathcal{C} \to \mathcal{C}^{\mathrm{op}}$, i.e., for all $C \in \mathcal{C}$, it holds that $c^2(C) = C$ and for any objects $A, B \in \mathcal{C}$, the following diagram



commutes in **Set** (cf. Fiore & Plotkin [16]).

The category of involutory categories, **InvCat**, has as objects the involutory categories (C, c) and as morphisms those functors $F : (\mathcal{C}, c) \to (\mathcal{D}, d)$ between involutory categories such that $F^{\mathrm{op}} \circ c = d \circ F$.

The following adjunctions are well-known:

$$\mathbf{LocInvCat} \underset{G_2}{\overset{U}{\rightleftarrows}} \mathbf{InvCat} \underset{G_1}{\overset{U}{\rightleftarrows}} \mathbf{Cat}$$

where U's are the forgetful functors, $G_1 : \mathcal{C} \mapsto (\breve{\mathcal{C}}, (\_)^{\S})$ where

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

and $G_2 : (\mathcal{C}, c) \mapsto (\hat{\mathcal{C}}, c)$ where

$$\hat{\mathcal{C}} := \{C \in \mathcal{C} | c(C) = C\}$$

is the full subcategory of $\mathcal{C}$ consisting of all the *symmetric objects*, i.e., the fixed points of c.

Since the composition of adjunctions gives an adjunction, there is an adjunction between the following categories:

$$\mathbf{LocInvCat} \underset{G}{\overset{U}{\rightleftarrows}} \mathbf{Cat}$$

where U is the forgetful functor and $G : \mathcal{C} \mapsto (\mathcal{C}^{\delta}, (\_)^{\S})$ where $\mathcal{C}^{\delta}$ is the

diagonal category, i.e., the full subcategory of $\mathcal{C}$ consisting of all the objects on the diagonal.

Note that for every (locally small) category $\mathcal{C}$, we have that $(\breve{\mathcal{C}}, (\_)^{\S})$ is an object of **InvCat**. Morphisms $F : (\breve{\mathcal{C}}, (\_)^{\S}) \to (\breve{\mathcal{D}}, (\_)^{\S})$ are functors $F : \breve{C} \to \breve{D}$ such that for every $f' \in \mathcal{C}^{\mathrm{op}}$ and for every $f \in \mathcal{C}$, it holds that $F_1(f', f) = F_2(f, f')$. Motivated by this example, we call all morphisms in **InvCat** *symmetric functors*.

Via the **InvCat-Cat** adjunction, the involutory categories $(\breve{\mathcal{B}}, (\_)^{\S})$ are universal in that they are characterised by a natural bijective correspondence

$$\frac{F : \mathcal{C} \to \mathcal{D}}{\breve{F} : (\mathcal{C}, (\_)^c) \xrightarrow[\text{symmetric}]{} (\breve{D}, (\_)^{\S})}$$

given by the mapping $F \mapsto \breve{F} = (F^{\mathrm{op}} \circ (\_)^c, F)$.

We employ the following technique frequently in Chapter 4 and Chapter 14 to turn mixed variant functors into covariant ones.

Example 2.2.2. Let $F : (\breve{\mathcal{C}})^n \to \mathcal{C}$ be a functor. Then by the **InvCat-Cat** adjunction, the bijective correspondence gives rise to a symmetric functor $\breve{F} : ((\breve{C})^n, (\_)^{\S}) \to (\breve{C}, (\_)^{\S})$ defined by

$$\begin{aligned} \breve{F}(P_1^-, P_1^+, \ldots, P_n^-, P_n^+) &= (F^{\mathrm{op}}(P_1^+, P_1^-, \ldots, P_n^-, P_n^+), F(P_1^-, P_1^+, \ldots, P_n^-, P_n^+)) \\ \breve{F}(f_1^-, f_1^+, \ldots, f_n^-, f_n^+) &= (F^{\mathrm{op}}(f_1^+, f_1^-, \ldots, f_n^+, f_n^-), F(f_1^-, f_1^+, \ldots, f_n^-, f_n^+)) \end{aligned}$$

2.3 Recursive domain equations

In this section, we recall some well known results regarding the construction of canonical solutions of recursive domain equations. We refer the reader to Smyth & Plotkin [56], Streicher [58] and Abramsky & Jung [3] for details of these constructions.

We begin by describing the problem in the environment of $\mathbf{DCPO}_\perp$ (the category of pointed dcpo's and continuous functions, not necessarily strict). Given an endofunctor F on $\mathbf{DCPO}_\perp$, the problem is to find a solution to the recursive domain equation

$$F(D) \cong D$$

i.e., to construct a pointed dcpo D and an isomorphism $\mathrm{fold} : F(D) \to D$. The word "equation" is used to mean "equality up to isomorphisms". First we must restrict our attention to a particular class of endofunctors. In this case, we consider *locally continuous endofunctors* on $\mathbf{DCPO}_\perp$, i.e., those

whose morphism part

$$(D_1 \to D_2) \longrightarrow (F(D_1) \to F(D_2))$$

is a Scott-continuous function for all pointed dcpos D_1 and D_2. Here $(D \to E)$ denotes the function space from D to E, which is defined to be the set of continuous functions $f : D \to E$ with respect to the pointwise order. Notice that functors built from $1 = \{\bot\}$ by using functors of the form $(-)_\bot$ (lifting), $\times$ (cartesian product) and $+$ (separated sum) are all locally continuous.

2.3.1 Construction of solutions

Let F be a locally continuous endofunctor on $\mathbf{DCPO}_\bot$. There are only two major steps in solving the recursive domain equation $F(D) \cong D$, namely:

(1) Form the dcpo D via a particular limit construction.

(2) Exploit the universal property of limits to obtain the desired isomorphism fold.

Step 1
Denote by $1 = \{\bot\}$ the one-point domain and define the sequence of projections $(p_n : F^{n+1}(1) \to F^n(1))_{n\in\mathbb{N}}$ by

$$p_0 := \lambda x : F(1).\bot_1 \text{ and } p_n := F^n(p_0).$$

This gives rise to the following diagram in $\mathbf{DCPO}_\bot$

$$1 \xleftarrow{p_0} F(1) \xleftarrow{p_1} F^2(1) \xleftarrow{p_2} \ldots.$$

whose limit is given by the vertex

$$\mathrm{Fix}(F) := \{d \in \prod_{n\in\mathbb{N}} F^n(1) | \forall n \in \mathbb{N}.d_n = p_n(d_{n+1})\}$$

together with the cone of morphisms $q_n : \mathrm{Fix}(F) \to F^n(1), d \mapsto d_n$. The embeddings $(e_n : F^n(1) \to F^{n+1}(1))_{n\in\mathbb{N}}$ defined by

$$e_0 := \lambda x : 1.\bot_{F(1)} \text{ and } e_n := F^n(e_0)$$

are such that each (e_n, p_n) is an embedding-projection pair (*e-p pair* for short), i.e., $p_n \circ e_n = \mathrm{id}_{F^{(n)}(1)}$ and $e_n \circ p_n \sqsubseteq \mathrm{id}_{F^{(n+1)}(1)}$. Precisely because these are e-p pairs, we can further characterise the limit of the sequence $(p_n)_{n\in\mathbb{N}}$ in a purely local manner as follows:

Lemma 2.3.1. *Associated to the projections q_n's are the embeddings $i_n : F^n(1) \to \mathrm{Fix}(F)$ defined explicitly by*

$$i_n(x)_m = \begin{cases} (e_{m-1} \circ \ldots \circ e_n)(x) & \text{if } n \leq m \\ (p_m \circ \ldots \circ p_{n-1})(x) & \text{if } n > m \end{cases}$$

Moreover, we have

$$\bigsqcup_{n \in \mathbb{N}} i_n \circ q_n = \mathrm{id}_{\mathrm{Fix}(F)}$$

and this property together with the requirement that $q_n = p_n \circ q_{n+1}$ characterises the limit up to isomorphism.

Step 2
Because both $(\mathrm{Fix}(F), (q_n)_{n\in\mathbb{N}})$ and $(F(\mathrm{Fix}(F)), F(q_n)_{n\in\mathbb{N}})$ are limiting cones for the sequence $(p_n)_{n\in\mathbb{N}}$, it follows from the universal properties of these limiting cones that there is a unique morphism $\mathrm{fold} : F(\mathrm{Fix}(F)) \to \mathrm{Fix}(F)$ with $q_{n+1} \circ \mathrm{fold} = F(q_n)$ for all $n \in \mathbb{N}$. Invoking Lemma 2.3.1, we have:

Lemma 2.3.2. *$F(\mathrm{Fix}(F))$ is isomorphic to $\mathrm{Fix}(F)$ via*

$$\begin{array}{lcll} \mathrm{fold} & = & \bigsqcup^{\uparrow}_{n\in\mathbb{N}} i_{n+1} \circ F(q_n) & : \quad F(\mathrm{fix}(F)) \to \mathrm{Fix}(F) \\ \mathrm{unfold} & = & \bigsqcup^{\uparrow}_{n\in\mathbb{N}} F(i_n) \circ q_{n+1} & : \quad \mathrm{Fix}(F) \to F(\mathrm{Fix}(F)) \end{array}$$

Moreover, for each $n \in \mathbb{N}$, they satisfy the equations:

$$\begin{array}{lcl} F(q_n) & = & q_{n+1} \circ \mathrm{fold} \\ F(i_n) & = & \mathrm{unfold} \circ i_{n+1}. \end{array}$$

2.3.2 Canonicity

The canonicity of the solution $(\mathrm{Fix}(F), \mathrm{fold})$ of the recursive domain equation $F(D) \cong D$ can be succinctly captured in the following theorems (due to Smyth & Plotkin [56] but formulated as in Abramsky & Jung [3]) whose proofs rely crucially on Lemmas 2.1.1, 2.3.1 and 2.3.2.

Theorem 2.3.3. *Let F be a locally continuous endofunctor on the category of pointed dcpos $\mathcal{D}$ and $i : F(D) \to D$ be an isomorphism. Then the following are equivalent:*

(i) $D \cong \mathrm{Fix}(F)$ *as F-algebras.*

(ii) id_D *is the least F-algebra endomorphism of D.*

(iii) $\mathrm{id}_D = \mathrm{fix}(\phi)$ *where* $\mathrm{fix} : (D \to D) \to (D \to D)$ *is defined by* $\phi(g) = i \circ F(g) \circ i^{-1}$.

(iv) id_D *is the only strict* F*-algebra endomorphism of* D.

Theorem 2.3.4. *Let* $F : \mathbf{DCPO}_{\perp!} \to \mathbf{DCPO}_{\perp!}$ *be a locally continuous functor. Then* $\mathrm{fold} : F(D) \to D$ *is an initial* F*-algebra where* $D = \mathrm{Fix}(F)$.

Theorem 2.3.5. *Let* $F : \mathbf{DCPO}_{\perp} \to \mathbf{DCPO}_{\perp}$ *be a locally continuous functor with canonical fixpoint* $D = \mathrm{Fix}(F)$. *Then* $\mathrm{unfold} : D \to F(D)$ *is a final co-algebra.*

Following Freyd's [17], for a given locally continuous functor on the category of pointed domains $\mathcal{D}$, by a *minimal* F*-invariant* we mean an F-algebra (D, i) such that (1) i is an isomorphism, and (2) the only endomorphism $e : D \to D$ for which the following diagram

$$\begin{array}{ccc} D & \xrightarrow{\;e\;} & D \\ {\scriptstyle i}\uparrow & & \uparrow{\scriptstyle i} \\ F(D) & \xrightarrow[F(e)]{} & F(D) \end{array}$$

commutes is the identity morphism id_D.

Theorem 2.3.6. (cf. Freyd [18])
Let $\mathcal{D}$ *be the category of pointed domains and strict maps, and* $F : \mathcal{D} \to \mathcal{D}$ *a locally continuous functor. The following statements are equivalent:*

(1) (D, i) *is a minimal* F*-invariant.*

(2) (D, i) *is an initial* F*-algebra.*

(3) (D, i^{-1}) *is a final* F*-coalgebra.*

2.3.3 Mixed variance

We first extend the notion of local continuity for mixed variant functors. A functor $F : \mathcal{D}^{\mathrm{op}} \times \mathcal{D}' \to \mathcal{E}$, contravariant in its first, covariant in its second variable, is called *locally continuous* if for directed sets $A \subseteq \mathcal{D}(D_2, D_1)$ and

$A' \subseteq \mathcal{D}'(D'_1, D'_2)$ (where D_1, D_2 are objects in $\mathcal{D}$ and D'_1, D'_2 are objects in $\mathcal{D}'$) we have

$$F(\bigsqcup\nolimits^{\uparrow} A, \bigsqcup\nolimits^{\uparrow} A') = \bigsqcup_{f \in A, f' \in A'} F(f, f')$$

in $\mathcal{E}(F(D_1, D'_1), F(D_2, D'_2))$.

The following theorems (also taken from Abramsky & Jung [3]) will prove handy later.

Theorem 2.3.7. *Let $\mathcal{D}$ be the category of pointed dcpos and $F : \mathcal{D}^{\mathrm{op}} \times \mathcal{D} \to \mathcal{D}$ be a mixed variant and locally continuous functor. Let $i : F(D, D) \to D$ be an isomorphism. Then the following are equivalent:*

(i) $D \cong \mathrm{Fix}(F)$ *where* $\mathrm{Fix}(F)$ *is the limit*[3] *of the diagram:*

$$1 \xleftarrow{p_0} F(1,1) \xleftarrow{F(e_0, p_0)} F(F(1,1), F(1,1)) \longleftarrow \dots$$

(ii) id_D *is the least mixed F-endomorphism of D.*

(iii) $\mathrm{id}_D = \mathrm{fix}(\phi)$ *where $\phi : (D \to D) \to (D \to D)$ is defined by $\phi(g) = i \circ F(g, g) \circ i^{-1}$.*

(iv) id_D *is the only strict mixed F-endomorphism of D.*

Theorem 2.3.8. *Let $\mathcal{D}_{\perp!}$ be the category of pointed dcpos and strict maps, and $F : \mathcal{D}_{\perp!}^{\mathrm{op}} \times \mathcal{D}_{\perp!} \to \mathcal{D}_{\perp!}$ be a mixed variant and locally continuous functor and $D = \mathrm{Fix}(F)$. Then for every pair of strict continuous functions $f : A \to F(B, A)$ and $g : F(A, B) \to A$ there are unique strict functions $h : A \to D$ and $k : D \to B$ such that the following diagrams commute:*

$$\begin{array}{ccc} F(B,A) & \xrightarrow{F(k,h)} & F(D,D) \\ {\scriptstyle f}\uparrow & & \uparrow{\scriptstyle \text{unfold}} \\ A & \xrightarrow[h]{} & D \end{array} \qquad \begin{array}{ccc} F(D,D) & \xrightarrow{F(h,k)} & F(A,B) \\ {\scriptstyle \text{fold}}\downarrow & & \downarrow{\scriptstyle g} \\ D & \xrightarrow[k]{} & B \end{array}$$

Given a locally continuous mixed-variant functor $F : \mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!} \to \mathbf{DCPO}_{\perp!}$, we say that a domain D together with an isomorphism $i : F(D, D) \to$

[3]Interested readers may refer to p. 78 of Abramsky & Jung [3] for the details of the construction of $\mathrm{Fix}(F)$.

D is a bifree solution of $X = F(X,X)$ if every strict $e : D \to D$ with $e = i \circ F(e,e) \circ i^{-1}$ is equal to id_D (cf. Streicher [58]).

So with this terminology, $(\mathrm{Fix}(F), i)$ is a bifree solution of $X = F(X,X)$. In view of Theorem 2.3.8, we also say that the canonical solution $(\mathrm{Fix}(F), i)$ a *bifree F-algebra*.

2.4 Algebraic completeness and compactness

In order to facilitate the discussion of operational algebraic compactness in Chapter 14, it is necessary to supply some background information on the concept of algebraic compactness. The material presented here comes from five sources: Freyd [17, 18, 19], Fiore & Plotkin [16] and Fiore [15]. Here we understand these notions in the setting of **DCPO**-categories. For an axiomatic treatment regarding algebraic completeness and compactness, the reader should refer to Fiore & Plotkin [16] and Fiore [15].

By a **DCPO**-*category*, we mean a locally small category whose hom-sets come equipped with a directed complete partial order with respect to which composition of morphisms is a continuous operation. Examples of **DCPO**-categories are **DCPO**, $\mathbf{DCPO}_{\perp!}$ and $\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}$.

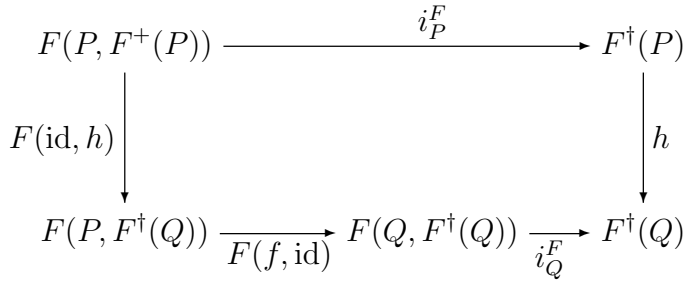
A **DCPO**-functor $F : \mathcal{C} \to \mathcal{D}$ between **DCPO**-categories $\mathcal{C}$ and $\mathcal{D}$, consists of a mapping associating every $C \in \mathcal{C}$ with some $F(C) \in \mathcal{D}$ and a functorial mapping associating every $C, C' \in \mathcal{C}$ with some Scott-continuous function $F_{C,C'} : \mathcal{C}(C,C') \to \mathcal{D}(FC, FC')$. An ordinary functor $F : \mathcal{C} \to \mathcal{D}$ is said to **DCPO**-*enrich* if for every $C, C' \in \mathcal{C}$, the function $F_{C,C'}$ is Scott-continuous. As an example, any locally continuous mixed variant functor $F : \mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}$ is a **DCPO**-functor.

2.4.1 Parametrised algebraic completeness

A **DCPO**-category is *algebraically complete* if every **DCPO**-functor on it has an initial algebra.

Let χ and $\mathcal{C}$ be **DCPO**-categories, and $F : \chi \times \mathcal{C} \to \mathcal{C}$ a **DCPO**-functor. Assume that $\mathcal{C}$ is algebraically complete. For each $P \in \chi$, we have that $F(P, \_) : \mathcal{C} \to \mathcal{C}$ is a locally continuous functor so that we can set $(F^\dagger(P), i_P^F)$ to be an initial $F(P,\_)$-algebra. To extend $F^\dagger$ to a functor, we define its morphism part as follows. For every χ-morphism $f : P \to Q$, let $F^\dagger(f) : F^\dagger(P) \to F^\dagger(Q)$ be the unique $F(P,\_)$-algebra homomorphism h from $(F^\dagger(P), i_P^F)$ to $(F^\dagger(P), i_Q^F \circ F(f, F^\dagger(Q)))$, i.e., which makes the following

diagram



commutes. By the universal property of initial algebras, $F^\dagger$ is a functor $\chi \to \mathcal{C}$ and, by construction, i^F is a natural transformation $F(\mathrm{Id}, F^\dagger) \to F^\dagger$. The pair $(F^\dagger, i^F)$ called an *initial parametrised F-algebra*.

A **DCPO**-category $\mathcal{C}$ is *parametrised algebraically complete* if it is algebraically complete and for every **DCPO**-functor $F : \chi \times \mathcal{C} \to \mathcal{C}$ and every family $\{i_P^F : F(P, F^\dagger(P)) \to F^\dagger(P)\}_{P \in \chi}$ of initial $F(P, \_)$-algebras, the induced functor $F^\dagger : \chi \to \mathcal{C}$ **DCPO**-enriches.

2.4.2 Parametrised algebraic compactness

A **DCPO**-category is *algebraically compact* if it is algebraically complete and the initial algebra of every **DCPO**-endofunctor on it is bifree, in the sense that its inverse is a final coalgebra.

A **DCPO**-category is *parametrised algebraically compact* if it is algebraically compact and parametrised algebraically complete.

Here are some well-known results specialised to the category $\mathbf{DCPO}_{\perp!}$ of pointed dcpos with strict maps.

Proposition 2.4.1. *(1)* $\mathbf{DCPO}_{\perp!}$ *is algebraically complete and hence is parametrised algebraically complete.*

(2) $\mathbf{DCPO}_{\perp!}$ *is algebraically compact and hence so is* $\mathbf{DCPO}_{\perp!}^{\mathrm{op}}$.

2.4.3 The Product Theorem

Theorem 2.4.2. (Product Theorem, Freyd [19], Fiore & Plotkin [16])
If $\mathcal{C}$ and $\mathcal{D}$ are (parametrised) algebraically compact then so is $\mathcal{C} \times \mathcal{D}$.

Corollary 2.4.3.

(1) If $\mathcal{C}$ is (parametrised) algebraically compact then so is $\mathcal{C}^{\mathrm{op}}$.

(2) If $\mathcal{C}$ is (parametrised) algebraically compact, so is $\breve{C}$.

For us, it is important to know, as an example, that $\mathbf{DCPO}_{\perp!}^{\mathrm{op}} \times \mathbf{DCPO}_{\perp!}$ is parametrised algebraically compact.

The last property of algebraic compactness (also known as the Fundamental Property of Algebraically Compact Categories) is stated below:

Theorem 2.4.4. (Fiore [15][4], Fiore & Plotkin [16])
Let χ and $\mathcal{C}$ be **DCPO***-categories. Assume that $\mathcal{C}$ is parametrised algebraically compact. For a symmetric* **DCPO***-functor $F : \breve{\chi} \times \breve{C} \to \breve{C}$, every initial parametrised F-algebra $(F^\dagger, i^F)$ canonically induces an initial parametrised F-algebra $(F^\ddagger, \varphi^F)$ such that $F^\ddagger$ is a symmetric* **DCPO***-functor and $\varphi^{\S}_P = \varphi_P^{-1}$ for every symmetric P.*

[4]The interested reader is refered to this reference for further explanation concerning this theorem.

Chapter 3

The programming language PCF

We work with the language[1] PCF which is a simply-typed λ-calculus with function and finite product types, base types `Nat` for natural numbers and `Bool` for booleans, as well as fixed-point recursion. For clarity of exposition, we also include a *Sierpinski* base type Σ and an *ordinal* base type $\overline{\omega}$, although such types can be easily encoded in other existing types if one so desires (for instance, via retractions - for this, see Scott [50]). We regard this as a programming language under the call-by-name[2] evaluation strategy.

This chapter introduces the syntax and operational semantics of the language PCF. In addition, we also bring to the attention of the reader some extensions of PCF which will be useful to us later. Based on Streicher [58] (with minor adaptations and simplifications), the material presented here is sufficient for us, in Part II, to develop an operational domain and topology for the language. However, it is not intended to give a comprehensive introduction to the language. For a good reference to PCF, the reader is asked to consult Streicher [58] and Gunter [24].

3.1 The language PCF

The language PCF is a *typed language* whose set `Type` of types is defined inductively as follows:

[1] PCF (an acronym for *Programming language for Computable Functions*) was introduced by Gordon Plotkin in his paper [42].

[2] There seems no difficulty developing the results of this thesis in a call-by-value setting as indicated in p. 436 of Escardó & Ho [14] (see also p. 282 of Pitts [41]), and this should definitely be done.

(i) The base types are

(1) `Nat`: (flat) natural number type,

(2) `Bool`: Boolean type,

(3) Σ: Sierpinski's data type, and

(4) $\overline{\omega}$: ordinal type (or the vertical natural number type).

(ii) Whenever σ and τ are types, then so are $\sigma \to \tau$ and $\sigma \times \tau$.

As discussed above, we have included the Sierpinski type Σ and the ordinal type $\overline{\omega}$ for clarity of exposition. The reader should note that the original version of PCF in Plotkin [42] does not include these two data types.

The constructor $\to$ is a right associative binary operation on `Type` meaning that, for instance, $\sigma_1 \to \sigma_2 \to \sigma_3$ is understood as $\sigma_1 \to (\sigma_2 \to \sigma_2)$. The constructor $\times$ is a left associative binary operation on `Type`, i.e., $\sigma_1 \times \sigma_2 \times \sigma_3$ is taken to mean $(\sigma_1 \times \sigma_2) \times \sigma_3$.

The first three base types are collectively known as the *ground types* and are intended to be types of printable values. We often use the symbol γ to range over ground types.

The PCF raw terms are given by the syntax trees generated by the grammar, modulo α-equivalence, in Figure 3.1. Terms of the form $s(t)$ are called *applications*. Terms of the form $\lambda x.t$ are called *abstractions*. Parentheses around applications and abstractions are sometimes omitted with the convention that juxtaposition is left-associative, i.e., $t_1 \dots t_n$ stands for $t_1(t_2) \dots (t_n)$.

For variables bound by λ's, we employ the usual convention of α-conversion according to which terms are considered as equal if they can be obtained from each other by an appropriate renaming of bound variables. Also, when substituting term t for variable x in term s we first rename the bound variables of t in such a way that free variables of s do not get bound by the λ-abstractions, i.e., employing the so-called capture-free substitution.

A *type assignment* (or typing context) consists of finitely many variables declared together with their types, i.e., it is of the form:

$$\Gamma \equiv x_1 : \sigma_1, \dots, x_n : \sigma_n$$

where the σ_i's are the types and the x_i's are pairwise distinct *term variables*. Formally, a type assignment Γ is a finite partial function from term variables to types, i.e., $\mathrm{dom}(\Gamma) = \{x_1, \dots, x_n\}$ and $\Gamma : x_i \mapsto \sigma_i$ $(i = 1, \dots, n)$. We inductively define terms in valid type assignments of the form $\Gamma \vdash t : \sigma$ (where t is a *term* of type σ in context Γ) by the typing rules given in Figure 3.2. The letters m, p, s, t range over terms while Δ, Γ range over

$t :=$	x	term variables
	$\mid \underline{0}$	zero
	$\mid \mathrm{succ}(t)$	successor
	$\mid \mathrm{pred}(t)$	predecessor
	$\mid (t \stackrel{?}{=} 0)$	test for zero
	$\mid \mathrm{T}$	true
	$\mid \mathrm{F}$	false
	$\mid$ if t then t else t	boolean conditional
	$\mid \top$	top
	$\mid$ if t then t	Sierpinski conditional
	$\mid t + 1$	ordinal successor
	$\mid t - 1$	ordinal predecessor
	$\mid (t > 0)$	test of convergence for ordinals
	$\mid (s, t)$	pairs
	$\mid \mathrm{fst}(p)$	first projection
	$\mid \mathrm{snd}(p)$	second projection
	$\mid \lambda x.t$	function abstraction
	$\mid s(t)$	function application
	$\mid \mathrm{fix}(f)$	fixed-point recursion

Figure 3.1: PCF syntax

type assignments. The notation $\Gamma, x : \sigma$ denotes the partial function which properly extends Γ by mapping x to σ. So by $\Gamma, x : \sigma$ we implicitly mean that $x \notin \mathrm{dom}(\Gamma)$. Similarly $\Gamma, \Gamma' \vdash t : \sigma$ is intended to imply that Γ and Γ' have disjoint domains.

Notice that every typable term has a unique type. Moreover we have:

Proposition 3.1.1. *(1) If* $\Gamma \vdash t : \sigma$*, then* $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$.

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

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

(4) If $\Gamma \vdash t_i : \sigma_i$ *for* $i = 1, \ldots, n$ *and* $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash s : \sigma$*, then* $\Gamma \vdash s[t_1/x_1, \ldots, t_n/x_n] : \sigma$.

Proof. (1)-(3) are proven by induction on the derivation of $\Gamma \vdash t : \sigma$ whereas (4) is proven by induction on the derivation of $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash s : \sigma$, using (2). □

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ (\text{var}) \qquad \frac{}{\Gamma \vdash \underline{0} : \texttt{Nat}}\ (\text{zero})$$

$$\frac{\Gamma \vdash m : \texttt{Nat}}{\Gamma \vdash \text{succ}(m) : \texttt{Nat}}\ (\text{succ}) \qquad \frac{\Gamma \vdash m : \texttt{Nat}}{\Gamma \vdash \text{pred}(m) : \texttt{Nat}}\ (\text{pred})$$

$$\frac{}{\Gamma \vdash \text{T} : \texttt{Bool}}\ (\text{true}) \qquad \frac{}{\Gamma \vdash \text{F} : \texttt{Bool}}\ (\text{false})$$

$$\frac{}{\Gamma \vdash \top : \Sigma}\ (\text{top}) \qquad \frac{\Gamma \vdash s : \overline{\omega}}{\Gamma \vdash (s > 0) : \Sigma}\ (> 0)$$

$$\frac{\Gamma \vdash s : \overline{\omega}}{\Gamma \vdash s + 1 : \overline{\omega}}\ (+1) \qquad \frac{\Gamma \vdash s : \overline{\omega}}{\Gamma \vdash s - 1 : \overline{\omega}}\ (-1)$$

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

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

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

$$\frac{\Gamma \vdash m : \texttt{Nat}}{\Gamma \vdash m \stackrel{?}{=} 0 : \texttt{Bool}}\ (\stackrel{?}{=} 0) \qquad \frac{\Gamma \vdash t : \texttt{Bool} \quad \Gamma \vdash s_1, s_2 : \sigma}{\Gamma \vdash \text{if } t \text{ then } s_1 \text{ else } s_2 : \sigma}\ (\text{cond})$$

$$\frac{\Gamma \vdash s : \Sigma \quad \Gamma \vdash t : \sigma}{\Gamma \vdash \text{if } s \text{ then } t : \sigma}\ (\text{if})$$

Figure 3.2: Rules for type assignment in PCF

We use the symbol $\text{Exp}_{\sigma}(\Gamma)$ to denote the set of PCF terms that can be assigned the type σ, given Γ:

$$\text{Exp}_{\sigma}(\Gamma) := \{t | \Gamma \vdash t : \sigma\}.$$

Note that Proposition 3.1.1 implies that all the free variables of $t \in \text{Exp}_{\sigma}(\Gamma)$ are contained in $\text{dom}(\Gamma)$. In the special case when t is valid under the empty typing assignment, we say that it is a *closed term*, i.e. one with no free variables. A PCF term with free variables is called an *open term*. We write Exp_{σ} for $\text{Exp}_{\sigma}(\emptyset)$. The elements of Exp_{σ} are also called *programs* of type σ. In particular, programs of ground type are simply known as programs.

3.2 Operational semantics

The big-step operational semantics of PCF is given by the *evaluation* relation which takes the form:

$$t \Downarrow v$$

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

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

$$\frac{p \Downarrow (s,t) \quad t \Downarrow v}{\text{snd}(p) \Downarrow v}\ (\Downarrow \text{snd}) \qquad \frac{m \Downarrow \underline{n}}{\text{succ}(m) \Downarrow \underline{n+1}}\ (\Downarrow \text{succ})$$

$$\frac{m \Downarrow \underline{0}}{\text{pred}(m) \Downarrow \underline{0}}\ (\Downarrow \text{pred1}) \qquad \frac{m \Downarrow \underline{n+1}}{\text{pred}(m) \Downarrow \underline{n}}\ (\Downarrow \text{pred2})$$

$$\frac{m \Downarrow \underline{0}}{(m \stackrel{?}{=} 0) \Downarrow \text{T}}\ (\Downarrow (\stackrel{?}{=} 0)1) \qquad \frac{m \Downarrow \underline{n+1}}{(m \stackrel{?}{=} 0) \Downarrow \text{F}}\ (\Downarrow (\stackrel{?}{=} 0)2)$$

$$\frac{t \Downarrow \text{T} \quad s_1 \Downarrow v}{\text{if } t \text{ then } s_1 \text{ else } s_2 \Downarrow v}\ (\Downarrow \text{cond1}) \qquad \frac{t \Downarrow \text{F} \quad s_2 \Downarrow v}{\text{if } t \text{ then } s_1 \text{ else } s_2 \Downarrow v}\ (\Downarrow \text{cond2})$$

$$\frac{s \Downarrow \top \quad t \Downarrow v}{\text{if } s \text{ then } t \Downarrow v}\ (\Downarrow \text{if}) \qquad \frac{s \Downarrow t+1 \quad t \Downarrow v}{s-1 \Downarrow v}\ (\Downarrow (-1))$$

$$\frac{s \Downarrow t+1}{(s>0) \Downarrow \top}\ (\Downarrow (>0))$$

Figure 3.3: Rules for evaluating PCF terms

where t and v are closed terms and v, the *canonical value* to which t evaluates, is given by the grammar:

$$v ::= \underline{n} \mid \text{T} \mid \text{F} \mid \top \mid \lambda x.t \mid (t,t) \mid t+1$$

The set of canonical values is denoted by Val_{σ}. The axioms and rules for inductively defining $\Downarrow$ is given in Figure 3.3.

For convenience, we use the following notations.

(1) For each type σ, the symbol $\bot_{\sigma}$ (read as *bottom*) is used to denote the term $\text{fix}(\lambda x^{\sigma}.x)$.

(2) In $\overline{\omega}$, define $0 := \bot_{\overline{\omega}}$ and for each $n \in \mathbb{N}$, the element $n : \overline{\omega}$ is defined to be:

$$(\ldots(0\underbrace{+1)+1\ldots)+1}_{n \text{ copies}}$$

and the element $\infty : \overline{\omega}$ is defined as $\text{fix}_{\overline{\omega}}(+1)$.

The evaluation relation $\Downarrow$ is deterministic and preserves typing.

Proposition 3.2.1.

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

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

Proof. Both (1) and (2) can be proven in a straightforward manner by induction on the structure of derivation of $t \Downarrow v$. □

3.3 Extensions of PCF

In our ensuing development, we shall encounter various extensions of PCF which we now describe.

3.3.1 Oracles

PCF_Ω is the extension of PCF with the following term-formation rule: For any function $\Omega : \mathbb{N} \to \mathbb{N}$, computable or not, we have:

$$\frac{\Gamma \vdash t : \mathtt{Nat}}{\Gamma \vdash \Omega t : \mathtt{Nat}} \quad \text{(oracle)}.$$

Then the operational semantics is extended by the rule:

$$\frac{t \Downarrow \underline{n} \quad \Omega(n) = m}{\Omega t \Downarrow \underline{m}} \quad (\Downarrow \text{oracle}).$$

We think of Ω as an external input or *oracle*, and of the equation $\Omega(n) = m$ as a query with question n and answer m. Of course, the extension of the language with oracle is no longer a *programming language*. We shall regard it as a *data language*[3]. In summary, PCF_Ω admits the computational environment in which data supplied may not be programmable in the programming language. To emphasise that the syntax tree of a closed term is free of any oracles, we refer to it as a program.

3.3.2 Parallel features

In our study, we also consider PCF extended with certain parallel features. One such extension, PCF^+, includes a *parallel-or* construct (por) with the following term-formation rule:

$$\frac{\Gamma \vdash s, t : \mathtt{Bool}}{\Gamma \vdash \mathrm{por}(s, t) : \mathtt{Bool}} \quad \text{(por)}.$$

[3]The term "data language" originates from Escardó [13]

The operational semantics is accordingly extended by the following rules:

$$\frac{s \Downarrow \mathrm{T}}{\mathrm{por}(s,t) \Downarrow \mathrm{T}} \quad \frac{t \Downarrow \mathrm{T}}{\mathrm{por}(s,t) \Downarrow \mathrm{T}} \quad \frac{s \Downarrow \mathrm{F} \quad t \Downarrow \mathrm{F}}{\mathrm{por}(s,t) \Downarrow \mathrm{F}} \ (\Downarrow \mathrm{por}).$$

G.D. Plotkin, in his seminal paper [42], proved that the Scott model is *fully abstract* for PCF plus a parallel condition $\supset$, i.e., two programs have the same denotation if and only if they are contextually equivalent. It is later established in Stoughton [57] that the parallel conditional is definable from parallel-or, and hence the Scott model is fully abstract for PCF^+.

Sometimes, a weaker form of the parallel-or (also called *weak parallel-or*) is considered. The term formation rule for the weak parallel-or construct ($\vee$) is given by

$$\frac{\Gamma \vdash s, t : \Sigma}{\Gamma \vdash s \vee t : \Sigma} \quad (\vee).$$

The operational semantics includes the following rule for evaluation:

$$\frac{s \Downarrow \top}{s \vee t \Downarrow \top} \quad \frac{t \Downarrow \top}{s \vee t \Downarrow \top} \ (\Downarrow \vee).$$

3.3.3 Existential quantifier

A further extension of PCF^+, denoted by PCF^{++}, includes the Plotkin's *existential quantifier* ($\exists$) with following additional term-formation rule:

$$\frac{\Gamma \vdash f : \mathtt{Nat} \to \mathtt{Bool}}{\Gamma \vdash (\exists f) : \mathtt{Bool}}$$

together with the corresponding additional rules in its operational semantics:

$$\frac{f\underline{n} \Downarrow \mathrm{T}}{(\exists f) \Downarrow \mathrm{T}} \ (\text{for some } n \in \mathbb{N}) \quad \frac{f(\bot_{\mathtt{Nat}}) \Downarrow \mathrm{F}}{(\exists f) \Downarrow \mathrm{F}} \quad (\Downarrow \exists).$$

The reader may be interested to know that in [42] Plotkin introduced the extension PCF^{++} to remedy the situation that not all "computable" elements of the Scott model can be denoted by terms of PCF^+. Here an element is "computable" iff the set of codes of approximating compact elements is recursively enumerable. In other words, by adding the existential quantifier, the model becomes Turing-universal.

3.3.4 $\mathbf{PCF}^{++}_{\Omega}$

$\mathrm{PCF}^{++}_{\Omega}$ is the extension of PCF which includes oracles, parallel-or and the Plotkin's existential quantifier. It is folkloric[4] that the Scott model is absolutely universal for $\mathrm{PCF}^{++}_{\Omega}$, i.e., every element of the Scott model becomes definable in the language.

3.4 PCF context

One fundamental question in computer science is to determine whether two given programs P_1 and P_2 are the same. It is immediate that one is not concerned whether they have the same syntax, for what one really cares is that they exhibit the same behaviour (for instance, whether they complete the same task). The common practice is to put these programs through tests: P_1 and P_2 are regarded as (observationally) equivalent if and only if they pass the same tests.

In order to formalise the notion of interchanging occurrences of terms in programs, we use 'contexts' - syntax trees containing parameters (or placeholders, or holes) which yield a term when the parameters are replaced by terms.

The *PCF contexts*, C, are the syntax trees generated by the grammar of PCF augmented by the clause:

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

where p ranges over some fixed set of *parameters*. Note that the syntax trees of PCF terms are particular contexts, namely the ones with no occurrences of parameters.

Most of the time we will use contexts involving a single parameter, we write as $-$. We write $C[-]$ to indicate that C is a context containing only one parameter. If t is a PCF term, then $C[t]$ will denote the term resulting from choosing a representative syntax tree for t, substituting it for the parameter in C, and forming the α-equivalence class of the resulting PCF syntax tree (which is independent of the choice of representative for t).

[4]Only recently did this fact appear in print. For this, see Section 12.15 of [13] and Theorem 13.10 of [59]

3.5 Typed contexts

We will assume given a function that assigns types to parameters. We write $-_\sigma$ to indicate that a parameter $-$ has type σ. Just as we only consider a PCF term to be well-formed if it can be assigned a type, we restrict attention to contexts that can be typed. The relation

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

assigning a type σ to a context C given a finite partial function Γ assigning types to variables, is inductively generated by axioms and rules just like in Figure 3.2, together with the following axiom for parameters:

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

One should take note that when the axioms and rules applied to syntax trees rather than α-equivalence classes of syntax trees (as in the case when typing contexts), it should be borne in mind that they enforce a separation between free and bound variables and hence are not closed under α-equivalence. For example, if $x \neq y$, then $x : \mathtt{Nat} \vdash \lambda y.-_{\mathtt{Nat}} : \mathtt{Nat} \to \mathtt{Nat}$ is a valid typing assertion, whereas $x : \mathtt{Nat} \vdash \lambda x.-_{\mathtt{Nat}} : \mathtt{Nat} \to \mathtt{Nat}$ is not.

Let $\mathrm{Ctx}_\sigma(\Gamma)$ denote the set of PCF contexts that can be assigned type σ, given Γ:

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

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

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

but not in the context

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

The operation $t \mapsto C[t]$ of substituting a PCF term for a parameter in a context to obtain a new PCF term respects typing in the following sense:

Lemma 3.5.1. *Suppose $t \in \mathrm{Exp}_\sigma(\Gamma, \Gamma')$, $C[-_\sigma] \in \mathrm{Ctx}_{\sigma'}(\Gamma)$ and that Γ' is trapped within $C[-_\sigma]$. Then $C[t] \in \mathrm{Exp}_{\sigma'}(\Gamma)$.*

Proof. By induction on the derivation of $\Gamma \vdash C[-_\sigma] : \sigma'$. □

3.6 Contextual equivalence and preorder

Let Γ be a finite partial function from variables to PCF types. Given $s, t \in \mathrm{Exp}_\sigma(\Gamma)$, we write

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

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

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

The relation $\sqsubseteq$ is called the *contextual preorder* between PCF terms (of the same type, given a typing of free variables). *Contextual equivalence* is the symmetrisation of this relation:

$$\Gamma \vdash s =_\sigma t \iff (\Gamma \vdash s \sqsubseteq_\sigma t) \wedge (\Gamma \vdash t \sqsubseteq_\sigma s).$$

For closed terms $s, t \in \mathrm{Exp}_\sigma$, we just write $s \sqsubseteq_\sigma t$ for $\emptyset \vdash s \sqsubseteq_\sigma t$. We also define *contextual order* to mean the contextual preorder modulo contextual equivalence and denote it by the same symbol $\sqsubseteq$ as there will be no confusion.

Remark 3.6.1. The need to apply proofs by induction on the derivation of terms-in-context forces us to define contextual order and equivalence for open terms, and not just the closed terms.

It does not matter which ground type (i.e., $\mathtt{Nat}$, $\mathtt{Bool}$, Σ) we choose to make the observation for the testing of programs.

Proposition 3.6.2. *The following are equivalent for any terms $s, t : \sigma$.*

(i) $\forall C \in \mathrm{Ctx}_\Sigma. C[s] \Downarrow \top \implies C[t] \Downarrow \top.$

(ii) $\forall C' \in \mathrm{Ctx}_{\mathtt{Nat}}. \forall n \in \mathbb{N}. C'[s] \Downarrow \underline{n} \implies C'[t] \Downarrow \underline{n}.$

(iii) $\forall C'' \in \mathrm{Ctx}_{\mathtt{Bool}}. \forall \mathtt{b} \in \mathbb{B}. C''[s] \Downarrow \mathtt{b} \Rightarrow C''[t] \Downarrow \mathtt{b}.$

Proof. We prove only the equivalence of (i) and (ii).
(i) $\Rightarrow$ (ii): Let $C'[-_\sigma] \in \mathrm{Ctx}_{\mathtt{Nat}}$ and $n \in \mathbb{N}$ be given. Suppose that $C'[s] \Downarrow \underline{n}$. We want to prove that $C'[t] \Downarrow \underline{n}$. To do this, consider the following context $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ defined by:

$$C[-_\sigma] := (C'[-_\sigma] == \underline{n})$$

where $x == y$ is the Sierpinski-valued equality-test on $\mathtt{Nat}$. Note that this equality test is PCF-definable: $(x == y) \equiv \mathrm{fix}_{\mathtt{Nat}\to\mathtt{Nat}\to\mathtt{Nat}}(G)$ where

$$\begin{aligned} G \equiv \quad & \lambda g.\lambda x.\lambda y.\mathrm{if}\ x \stackrel{?}{=} 0 \ \ \mathrm{then}\ (\mathrm{if}\ \ y \stackrel{?}{=} 0\ \ \mathrm{then}\ \ \top\ \ \mathrm{else}\ \ \bot_\Sigma) \\ & \qquad\qquad\qquad\qquad \mathrm{else}\ (\mathrm{if}\ \ y \stackrel{?}{=} 0\ \ \mathrm{then}\ \ \bot_\Sigma\ \ \mathrm{else}\ \ g\ \mathrm{pred}(x)\ \mathrm{pred}(y)) \end{aligned}$$

Note that for any $s \in \mathrm{Exp}_\sigma$, $C[s] \Downarrow \top$ iff $C'[s] \Downarrow \underline{n}$. Now invoking (i), we deduce that $C[t] \Downarrow \top$. This then implies that $C'[t] \Downarrow \underline{n}$ as required.
(ii) $\Rightarrow$ (i): Let $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ be given. Consider the context $C' \in \mathrm{Ctx}_{\mathtt{Nat}}$ defined by:

$$C' := \mathrm{if}\ \ C[-_\sigma]\ \ \mathrm{then}\ \ \underline{1}.$$

Note that for any $s \in \mathrm{Exp}_\sigma$, $C'[s] \Downarrow \underline{1}$ iff $C[s] \Downarrow \top$. By applying (ii), we deduce that $C'[t] \Downarrow \underline{1}$ and consequently we have $C[t] \Downarrow \top$ as required. □

3.7 Extensionality and monotonicity

In this section, we record some well-known properties of the contextual preorder and equivalence which are collectively termed as "extensionality properties" in Pitts [41] (cf. p.255).

Convention and notation. Since our primary focus is the study of the PCF closed terms, we avoid the hassle of writing out all the explicit typing assignments. By an element t of a type σ, we mean the contextual equivalence class containing the closed term t of that type. We adopt the set-theoretic notation for the elements of a type in the sense just defined. For example, write $x \in \sigma$ and $f \in (\sigma \to \tau)$ to mean that x is an element of type σ and f is an element of type $\sigma \to \tau$. We are going to apply the above convention in Sections 3.7 and 6.6, Chapter 5 and in Part III.

Proposition 3.7.1.

(1) For any $x, y \in \gamma$ (where γ is a ground type, i.e., Σ, $\mathtt{Nat}$, $\mathtt{Bool}$),

$$x \sqsubseteq_\gamma y \iff \forall v \in \mathrm{Val}_\gamma.(x \Downarrow v \Rightarrow y \Downarrow v).$$

(2) For any $x, y \in \overline{\omega}$,

$$x \sqsubseteq_{\overline{\omega}} y \iff \forall s \in \overline{\omega}.(x \Downarrow s+1 \implies \exists t \in \overline{\omega}.y \Downarrow t+1 \wedge s \sqsubseteq_{\overline{\omega}} t).$$

(3) For any $f, g \in (\sigma \to \tau)$,

$$f \sqsubseteq_{\sigma\to\tau} g \iff \forall t \in \sigma.(f(t) \sqsubseteq_\tau g(t)).$$

(4) For any $p, q \in (\sigma \times \tau)$,

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

The following statements are immediate from the definitions of contextual preorder and equivalence, except for the converse of (2) which is a consequence of Proposition 3.7.1:

Corollary 3.7.2. *(1) Contextual equivalence is a congruence: If* $f = g$ *and* $x = y$*, then* $f(x) = g(y)$ *for any* $f, g \in (\sigma \to \tau)$ *and* $x, y \in \sigma$*.*

(2) Application is extensional: $f = g$ *iff* $f(x) = g(x)$ *for all* $x \in \sigma$*.*

(3) Application is monotone: If $f \sqsubseteq g$ *and* $x \sqsubseteq y$*, then* $f(x) \sqsubseteq g(y)$ *for any* $f, g \in (\sigma \to \tau)$ *and* $x, y \in \sigma$*.*

Chapter 4

The programming language FPC

We consider a call-by-name version of the language FPC (Fixed Point Calculus) whose call-by-value version was first introduced by G.D. Plotkin in his 1985 CSLI lecture notes [45]. In a nutshell, FPC does for recursive definitions of types what PCF does for recursive definitions of functions.

In this chapter, we introduce the syntax and operational semantics of the language FPC. In Part IV, we shall give an operational domain-theoretic treatment of recursive types for this language. The interested reader may also find information on call-by-name FPC in McCusker [37] and call-by-value FPC in Gunter [24].

4.1 The language FPC

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

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

For type expressions, we have type variables, product types, sum types, lifted types, recursive types and function types. A *closed type* is a type expression containing no free type variables, i.e., if any occurring type variable X is bound under the scope of a recursive type constructor μX. A *type context* is a list of distinct type variables (which may be empty). We write $\Theta \vdash \sigma$ for the type σ in context Θ, indicating that the set of free type variables occurring in σ is a subset of the type context Θ.

The raw FPC *terms* are given by the syntax trees generated by the following grammar, modulo α-equivalence:

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

Figure 4.1: FPC syntax

Terms containing no free variables are called *closed terms*. Otherwise, they are known as *open terms*. A *term context* is a list of distinct term variables with types. We write $\Theta; \Gamma \vdash t : \sigma$ for a term t in (term) context $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$ where $\Theta \vdash \sigma_i$ $(i = 1, \ldots, n)$ are well-formed types-in-context. When there is no confusion, we omit the type context Θ. The typing rules in FPC are given in Figure 4.2.

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

Lemma 4.1.1. *(1) If* $\Gamma \vdash t : \sigma$, *then* $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$.

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

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

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

Proof. (1) - (3) are proven by induction on the derivation of $\Gamma \vdash t : \sigma$ and (4) is proven by induction on the derivation of $\Gamma, \vec{x} : \vec{\sigma} : s : \sigma$, using (2). □

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

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

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

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

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

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

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

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

Figure 4.2: Rules for type assignments in FPC

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

4.2 Operational semantics

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

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

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

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

The relation $\Downarrow$ is inductively defined in Figure 4.3 below.

Proposition 4.2.1. *Evaluation is deterministic and preserves typing, i.e.,*

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

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

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

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

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

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

Figure 4.3: Rules for evaluating FPC terms

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

Proof. Both (1) and (2) are proven by induction on the derivation of $t \Downarrow v$ (invoking Lemma 4.1.1(iv)). □

4.3 Fixed point operator

Like existing works such as Gunter [24] and Rohr [47], we can define a fixed point operator using the recursive types. This is done as follows:

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

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

We shall prove in Chapter 7 that $\text{fix}(f)$ and $f(\text{fix}(f))$ are contextually equivalent. The reader should note that for this fixed point operator the following evaluation **does not** hold:

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

4.4 Some notations

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

To begin with, there are three special closed types worth mentioning:

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

The type 1 is called the *void type* and contains no canonical values. Lifting the type 1 produces the *Sierpinski type*, $1_\perp$, which we denote by Σ. The non-divergent element of Σ, $\mathrm{up}(\perp)$, is denoted by $\top$. We shall be exploiting Σ to make program observations[1]. Given $a : \Sigma$ and $b : \sigma$, we define

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

Notice that "if a then b" is an "if-then" construct without the usual "else".

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

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

We define $n - 1 := \mathrm{case}(\mathrm{unfold}(n)) \text{ of } \mathrm{up}(x).x$ and $\infty := \mathrm{fix}(+1)$ where $(+1) := \lambda x.x + 1$. The Σ-valued convergence test

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

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

Some of our programs in Chapter 15 makes use of the *lazy natural numbers type*, which we now introduce. We define the lazy natural number data type to be the recursive type

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

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

$$\begin{array}{lcllcl} 0 & := & \mathrm{fold}(\mathrm{inl}(\perp_1)) & \overline{0} & := & \mathrm{fold}(\mathrm{inr}(\perp_{\mathtt{Nat}})) \\ n+1 & := & \mathrm{succ}(n) & \overline{n+1} & := & \mathrm{succ}(\overline{n}) \\ & & & \infty & := & \mathrm{fix}(\mathrm{succ}) \end{array}$$

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

Remark 4.4.1. Because we want to work with a single evaluation strategy (i.e., call-by-name for both PCF and FPC) throughout this thesis, our version of FPC does not have flat natural numbers type and hence does not subsume PCF (as defined in Chapter 3). If one wishes to have the flat natural numbers type in the language, one can always introduce an infinitary sum constructor or a distingished flat natural natural numbers type.

[1]The two types Σ play the same role of program observation in PCF and FPC.

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

Figure 4.4: FPC contexts

4.5 FPC contexts

The *FPC contexts*, C, are syntax trees generated by the grammar for FPC terms in Figure 4.1 augmented by the clause:

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

where p ranges over a fixed set of *parameters* (or holes). The details of the defining grammar is spelt out in Figure 4.4.

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

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

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

assigning a closed type σ to a context C given the term Γ, is inductively generated by axioms and rules in Figure 4.5. We define

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

to be the set of FPC contexts that can be assigned to the closed type σ,

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

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

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

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

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

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

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

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

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

Figure 4.5: Typing rules for FPC contexts

given Γ. We write Ctx_{σ} for $\mathrm{Ctx}_{\sigma}(\emptyset)$.

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

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

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

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

The relation $\sqsubseteq$ is called the *contextual preorder* and its symmetrisation is called the *contextual equivalence*, denoted by $=$. For a given term σ, the order induced by the preorder $\sqsubseteq$ on the set of equivalence classes of closed terms of type σ is called the *contextual order*. Notice that we have chosen the ground type Σ to be the type on which program observations are based. Such a choice is motivated by aiming for compatibility with PCF (see Section 3.6).

Remark 4.5.1. Let $s, t : \sigma$ be closed terms. Then $s \sqsubseteq_{\sigma} t$ iff

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

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

4.6 Denotational semantics

In this section, we give the standard Scott (domain-theoretic) denotational semantics for FPC. The reader may consult Fiore & Plotkin [16] and McCusker [37] for details.

4.6.1 Interpretation of types

Types-in-context $\Theta \vdash \tau$ are interpreted as an n-ary symmetric locally continuous functors $[\![\Theta \vdash \tau]\!] : (\breve{D})^n \to \breve{D}$ where $\mathcal{D} = \mathbf{DCPO}_{\perp!}$ and this interpretation is given in Figure 4.6.

$$\begin{array}{rcl}
[\![\Theta \vdash X_i]\!] & = & \Pi_i \qquad\qquad (1 \leq i \leq |\Theta|) \\
[\![\Theta \vdash \tau_1 + \tau_2]\!] & = & (\Pi_2[\![\Theta \vdash \tau_1]\!] + \Pi_2[\![\Theta \vdash \tau_2]\!])^\smile \\
[\![\Theta \vdash \tau_\perp]\!] & = & ((\Pi_2[\![\Theta \vdash \tau]\!])_\perp)^\smile \\
[\![\Theta \vdash \tau_1 \times \tau_2]\!] & = & (\Pi_2[\![\Theta \vdash \tau_1]\!] \times \Pi_2[\![\Theta \vdash \tau_2]\!])^\smile \\
[\![\Theta \vdash \tau_1 \to \tau_2]\!] & = & (\Pi_1[\![\Theta \vdash \tau_1]\!] \to \Pi_2[\![\Theta \vdash \tau_2]\!])^\smile \\
[\![\Theta \vdash \mu X.\tau]\!] & = & [\![\Theta, X \vdash \tau]\!]^\ddagger
\end{array}$$

Figure 4.6: Definition of $[\![\Theta \vdash \Gamma]\!] : (\breve{\mathcal{D}})^{|\Theta|} \to \breve{\mathcal{D}}$

For the purpose of understanding the above figure, it may be helpful to recall that:

(1) Π_i is the projection functor in the ith component.

(2) The notation $(-)^\smile$ is as defined in Example 2.2.2.

(3) The notation $F^\ddagger$ is as defined in Theorem 2.4.4.

The reader may like to consult Section 8.4 of Fiore [15] for a detailed explanation of Figure 4.6 above.

The interpretation of types respects a substitution lemma:

Lemma 4.6.1. (Substitution lemma for types)
There exists a canonical natural isomorphism

$$\beta : [\![\Theta \vdash \sigma[\tau/X]]\!] \cong [\![\Theta, X \vdash \sigma]\!](\mathrm{Id}, [\![\Theta \vdash \tau]\!])$$

such that $\beta_P^{\S} = \beta_P^{-1}$ *for all symmetric* P.

4.6.2 Interpretation of terms

The interpretation of terms-in-context is standard: variables correspond to projections, inl/inr correspond to coproduct injections, case correspond to coproduct selection, $(-,-)$ to pairing, fst/snd to projections, $\lambda x.-$ to currying, $-(-)$ to evaluation and fold/unfold to folding/unfolding a recursive type. We give the interpretation of terms-in-context of the form $\Gamma \vdash t : \tau$ in Figure 4.7.

Lemma 4.6.2. (Substitution lemma for terms)
Suppose $\Gamma, x : \sigma \vdash s : \tau$ *and* $\Gamma \vdash t : \sigma$. *Then* $\Gamma \vdash s[t/x] : \tau$ *and*

$$[\![\Gamma \vdash s[t/x] : \tau]\!] = [\![\Gamma, x : \sigma \vdash s : \tau]\!] \circ (\mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \vdash t : \sigma]\!]).$$

4.6.3 Soundness and computational adequacy

G.D. Plotkin established in [45] that the Scott model for call-by-value FPC is *sound* and *computationally adequate*. With some modifications, it is possible to establish a similar result for call-by-name FPC.

Theorem 4.6.3. (Soundness and computational adequacy)

(1) The Scott model of call-by-name FPC is correct, i.e., for all $s \in \mathrm{Exp}_\sigma$ *and all* $v \in \mathrm{Val}_\sigma$, *we have*

$$s \Downarrow v \implies [\![s]\!] = [\![v]\!].$$

(2) The Scott model of call-by-name FPC is computationally adequate, i.e., for all $s \in \mathrm{Exp}_\Sigma$, *we have*

$$[\![s]\!] = \top \implies s \Downarrow \top.$$

Corollary 4.6.4. *Let* σ *be any closed type. Then for all* $s, t \in \mathrm{Exp}_\sigma$, *we have*

$$[\![s]\!] = [\![t]\!] \implies s =_\sigma t.$$

For every symmetric $P \in |\breve{\mathcal{D}}|^{|\Theta|}$ we define $[\![\Theta, \Gamma \vdash t : \tau]\!]_P$ as follows:

$$
\begin{array}{rcl}
[\![\Theta, \Gamma \vdash x_i]\!]_P & = & \pi_i \qquad\qquad (1 \leq i \leq |\Gamma|) \\
[\![\Theta, \Gamma \vdash \mathrm{inl}(t) : \tau_1 + \tau_2]\!]_P & = & \iota_1 \circ [\![\Gamma \vdash t : \tau_1]\!]_P \\
[\![\Theta, \Gamma \vdash \mathrm{inr}(t) : \tau_1 + \tau_2]\!]_P & = & \iota_2 \circ [\![\Gamma \vdash t : \tau_2]\!]_P \\
[\![\Theta, \Gamma \vdash \mathrm{case}(s) \text{ of } \begin{cases} \mathrm{inl}(x).t_1 \\ \mathrm{inr}(y).t_2 \end{cases} : \tau]\!]_P & = & [[\![\Theta, \Gamma, x : \tau_1 \vdash t_1 : \tau]\!]_P, \\
 & & [\![\Theta, \Gamma, y : \tau_2 \vdash t_2 : \tau]\!]_P \circ \delta \circ \\
 & & \langle \mathrm{id}, [\![\Theta, \Gamma \vdash s : \tau_1 + \tau_2]\!]_P \rangle \\
 & & \text{where } \delta \text{ is the canonical isomorphism} \\
 & & [\![\Theta, \Gamma]\!] \times ([\![\Theta \vdash \tau_1]\!] + [\![\Theta \vdash \tau_2]\!]) \cong \\
 & & ([\![\Theta, \Gamma]\!] \times [\![\Theta \vdash \tau_1]\!]) + ([\![\Theta, \Gamma]\!] \times [\![\Theta \vdash \tau_2]\!]) \\
[\![\Theta, \Gamma \vdash (s, t) : \tau_1 \times \tau_2]\!]_P & = & \langle [\![\Theta, \Gamma \vdash s : \tau_1]\!]_P, [\![\Theta, \Gamma \vdash t : \tau_2]\!] \rangle_P \\
[\![\Theta, \Gamma \vdash \mathrm{fst}(t) : \tau_1]\!]_P & = & \pi_1 \circ [\![\Theta, \Gamma \vdash t : \tau_1 \times \tau_2]\!]_P \\
[\![\Theta, \Gamma \vdash \mathrm{snd}(t) : \tau_2]\!]_P & = & \pi_2 \circ [\![\Theta, \Gamma \vdash t : \tau_1 \times \tau_2]\!]_P \\
[\![\Theta, \Gamma \vdash \lambda x : \tau_1 . t : \tau_1 \to \tau_2]\!]_P & = & \Lambda([\![\Theta, \Gamma, x : \tau_1 \vdash t : \tau_2]\!]_P) \\
[\![\Theta, \Gamma \vdash s(t) : \tau_2]\!]_P & = & \mathrm{eval} \circ \\
 & & \langle [\![\Theta, \Gamma \vdash s : \tau_1 \to \tau_2]\!]_P, [\![\Theta, \Gamma \vdash t : \tau_1]\!]_P \rangle \\
[\![\Theta, \Gamma \vdash \mathrm{fold}(t) : \mu X.\tau]\!]_P & = & I_P \circ [\![\Theta, \Gamma \vdash t : \tau[\mu X.\tau / X]]\!]_P \\
 & & \text{where } I_P := (\varphi_P^{[\![X \vdash \tau]\!]} \circ \beta_P)_2 \\
[\![\Theta, \Gamma \vdash \mathrm{unfold}(t) : \tau[\mu X.\tau / X]]\!]_P & = & E_P \circ [\![\Theta, \Gamma \vdash t : \mu X.\tau]\!]_P \\
 & & \text{where } E_P = I_P^{-1}
\end{array}
$$

Figure 4.7: Definition of $[\![\Theta, \Gamma \vdash t : \tau]\!]$

Chapter 5

Synthetic topology

The material presented in this chapter comes from Escardó [13] in which it is shown, via *synthetic topology*, how topological concepts can be directly understood in terms of the operational semantics, and, moreover, are applicable to sequential languages. In this section, we introduce operational topological notions that are essential to the development of our theory and these include space, continuous map, open set, closed set, discrete space, Hausdorff space and compact set. Whenever there is no confusion, we shall omit the word 'computational' as we talk about the various computational topological concepts. For instance, we speak of an open set instead of a computationally open set.

5.1 Continuous maps

We identify contextually equivalent programs and, by an abuse of notation, we denote by σ the set of closed terms of type σ, modulo contextual equivalence. A function from the set σ to the set τ may or may not be definable by a term of type $\sigma \to \tau$. If it is, we say that the function is *continuous*. Notice that, in a call-by-name framework, such as ours, every term of type $\sigma \to \tau$ is uniquely determined, modulo contextual equivalence, by its functional behaviour, by extensionality (cf. Corollary 3.7.2(2)). If $f : \sigma \to \tau$ is continuous, we use the notation $f \in (\sigma \to \tau)$.

Since continuity is dependent on the language, varying the language under consideration will vary the continuity of a given function. In this chapter, we take the internal view of data. So our base language is PCF. But what we develop applies just as well to the external view of data, i.e., by changing the base language to the data language PCF_Ω, which we are going to take up when we come to the notion of compactness.

Remark 5.1.1. Continuous maps are programs and are thus monotone with respect to the contextual order.

Example 5.1.2. There are four functions of type $\Sigma \to \Sigma$, namely

(1) $f_1 : \top \mapsto \bot, \bot \mapsto \bot$

(2) $f_2 : \top \mapsto \top, \bot \mapsto \bot$

(3) $f_3 : \top \mapsto \top, \bot \mapsto \top$

(4) $f_4 : \top \mapsto \bot, \bot \mapsto \top$

It is easy to see that f_i $(i = 1, 2, 3)$ are continuous as they can be respectively defined by the following programs:

(i) $\lambda x^{\Sigma}.\bot$

(ii) $\lambda x^{\Sigma}.x$

(iii) $\lambda x^{\Sigma}.\top$

Notice that f_4 cannot be continuous as it fails to be monotone.

5.2 Open and closed subsets

It is well known in classical topology that the notion of open sets can be defined in terms of continuity. We do the same here.

Definition 5.2.1. A set U of elements of a type σ is *open* if there is $\chi_U \in (\sigma \to \Sigma)$ such that for all $x \in \sigma$,

$$\chi_U(x) = \top \iff x \in U.$$

If such an element χ_U exists, then it is unique up to contextual equivalence, and we refer to it as the *characteristic function* of U. A set is called *closed* if its complement is open.

Example 5.2.2. The subsets $\emptyset, \{\top\}, \{\bot, \top\}$ of type Σ are open since their characteristic functions f_1, f_2 and f_3 are all continuous. But $\{\bot\}$ is not open as a characteristic function would have violated monotonicity.

As continuity is defined in terms of definability in the language, it is typical that proofs in computational topology are given by programs written to meet required specifications. One such example is given by the following proposition which states that programs of functional type are continuous in the topological sense:

Proposition 5.2.3. *If $f \in (\sigma \to \tau)$, then $f^{-1}(V) = \{x \in \sigma | f(x) \in V\}$ is open for every open set $V \subseteq \tau$.*

Proof. If $\chi_V \in (\tau \to \Sigma)$ is the characteristic function of the set V, then $\chi_V \circ f$ is that of $f^{-1}(V)$. □

5.3 Closure of open sets under set-union

While it is clear that for every type, the open sets are closed under the formation of finite intersections, it is not true in general that they are closed under the formation of arbitrary union. In what follows, we demonstrate how the addition of different parallel features (described in Section 3.3.2) gives rise to varying degrees of closure of open sets under set-theoretic union.

Proposition 5.3.1. *The following are equivalent:*

(i) For every type, the open sets are closed under the formation of finite unions.

(ii) There is $(\vee) \in (\Sigma \times \Sigma \to \Sigma)$ such that

$$p \vee q = \top \Leftrightarrow p = \top \text{ or } q = \top.$$

Proof. (i) $\Rightarrow$ (ii): $\chi_{\bigcup \emptyset}(x) = \bot$ and $\chi_{U \cup V}(x) = \chi_U(x) \vee \chi_V(x)$.
(i) $\Leftarrow$ (ii): The sets $U = \{(p,q) | p = \top\}$ and $V = \{(p,q) | q = \top\}$ are open in the type $\Sigma \times \Sigma$ because they have the first and second projections as their characteristic functions. Hence the set $U \cup V$ is also open, and so there is $\chi_{U \cup V}$ such that $\chi_{U \cup V}(p,q) = \top$ iff $(p,q) \in U \cup V$ iff $(p,q) \in U$ or $(p,q) \in V$ iff $p = \top$ or $q = \top$. Therefore $(\vee) = \chi_{U \cup V}$ gives the desired conclusion. □

The part (ii) of the above statement involves the same weak parallel-or $(\vee)$ which we already discussed on Section 3.3.2. This weak parallel-or is also known as the *disjunction operation*. Of course, we can do better:

Proposition 5.3.2. *In the language PCF extended with the disjunction operation, for every type, the open sets are closed under the formation of recursively enumerable unions.*

Proof. A recursively enumerable collection of open sets may be seen as a program $s \in (\mathtt{Nat} \to (\sigma \to \Sigma))$. To prove the proposition, it suffices to write

a program $p \in ((\mathtt{Nat} \to (\sigma \to \Sigma)) \to (\sigma \to \Sigma))$ such that $p(s)(x) = \top$ iff $s(i)(x) = \top$ for some $i \in \mathtt{Nat}$. To meet this requirement, define

$$p(s) = \lambda x.e(0)$$

where e is recursively given by $e(i) = s(i)(x) \vee e(i+1)$. Then $p(s)(x) = \top$ iff $e(0) = \top$ iff $s(i)(x) = \top$ for some $i \in \mathtt{Nat}$. □

Remark 5.3.3. For the language PCF extended with only the Plotkin's existential quantifier $\exists$, it is also true that the union of a recursively enumerable sequence of opens is open since the weak parallel-or can be defined from $\exists$.

Notice that even with the inclusion of parallel features, closure under arbitrary unions fails in general. However, the following holds:

Theorem 5.3.4. (Escardó [13], Theorem 4.1)
For the language PCF^{++}_{Ω}, the computational topology coincides with the Scott topology. In particular, computationally open sets are closed under the formation of arbitrary unions.

5.4 Subspace

Frequently we are working with only certain elements of a particular data type. In view of our topological development, it is natural to speak of a *subspace* as an arbitrary subset of a data type. For instance, σ, in itself, is a trivial subspace. If X is a subspace of σ, then we call σ an environment for space X.

The subspace N of $\mathtt{Nat}$ of non-divergent (i.e. non-bottom) elements is the *space of natural numbers*. The data type $\mathtt{Nat} \to \mathtt{Nat}$ is called the *Baire* data type and is denoted by $\mathtt{Baire}$. The *Baire space* is the subset B of strict total functions of type $\mathtt{Baire}$. The *Cantor space* is the subset C of B consisting of functions taking values 0 or 1 on all non-divergent arguments. So B (respectively, C) is an operational manifestation of the *Baire space* (respectively, *Cantor space*) in classical topology.

Since subspaces of data types are not necessarily data types, we are forced to work with *relative topology*. Let X and Y be subspaces of data types σ and τ. We say that a function $\phi : X \to Y$ is *relatively continuous* if there is at least one continuous function $f : \sigma \to \tau$ with $\phi(x) = f(x)$ for every $x \in X$. It does not concern us how f behaves outside on elements of σ outside X. We say that a subset of a space is *relatively open* if its Sierpinski-valued characteristic map is relatively continuous. The following is immediate from the definitions.

Proposition 5.4.1. *For a subspace X of a data type σ, a subset U of X is open in X iff there is an open subset U' of σ such that $X \cap U' = U$.*

Proof. ($\Rightarrow$) Suppose $U \subseteq X$ is relatively open in X. Then there is a continuous function $f : \sigma \to \Sigma$ such that $\chi_U(x) = f(x)$ for all $x : \sigma$. Now define $U' = f^{-1}(\top)$, which is certainly an open subset of σ and by definition $\chi_{U'} = f$. It is clear that for each $x \in U$, $f(x) = \top$ which implies that $\chi_{U'}(x) = \top$, i.e. $x \in U'$. Conversely if $x \in X \cap U'$, then $\chi_U(x) = f(x) = \chi_{U'}(x) = \top$, i.e. $x \in U$. This proves that $X \cap U' = U$.
($\Leftarrow$) Suppose that $X \cap U' = U$ for some open subset U' of σ. The characteristic function $\chi_{U'}$ of U' is continuous and thus that of U is since $\chi_U = \chi_{U'}$ when restricted to the subspace X. □

Example 5.4.2. (Exercise 3.6 of Escardó [13])
The subset of all sequences s which belong to the Baire space and satisfy $s(17) = 0$ is open in B but not open in `Baire`. For the first part, we must prove that T is open in B. To do so, observe that $T = C \cap U$ where $\chi_U(s) :=$ if $s(17) == 0$ then $\top$. The desired result then follows from Proposition 5.4.1. We prove the second part when we revisit this example in Chapter 10.

5.5 Separation axioms

In classical topology, it is traditional to study the various degrees of separation. Roughly speaking, we want to use the open neighbourhoods of the topology to distinguish between two points in space. Various degrees of separation arise when one considers different manners in which the distinction between points is to be made in terms of their open neighbourhoods. For instance, a space is T_0 if we require that there is at least one neighbourhood which contains exactly one of the two distinct points. A space is T_1 if we want to have two opens, each containing exactly one of the points. A space which satisfy yet a finer separation axiom that requires further that the abovementioned pair of opens be non-overlapping is called a Hausdorff space. An extreme case of Hausdorff separation arises when every singleton is open, i.e., equivalently, the space is discrete, in which every subset is open.

In our setting, the "equality" of programs really means contextual equivalence. So different separation conditions translate into the varying degrees of ability to tell two contextually inequivalent programs apart.

A subspace X of type σ is *Hausdorff* if there exists an apartness program $(\neq) \in (\sigma \times \sigma \to \Sigma)$ such that for every $x, y \in X$,

$$(\neq)(x, y) = \top \iff x \neq y.$$

Here, $\neq$ means contextually inequivalent.

A subspace X of type σ is *discrete* if there exists an equality test $(=) \in (\sigma \times \sigma \to \Sigma)$ such that for every $x, y \in X$,

$$(=)(x, y) = \top \iff x = y.$$

Here, $=$ refers to contextual equivalence.

Example 5.5.1. Any non-trivial data type is not Hausdorff since every open that contains the bottom element $\bot$ must contain every element of that type.

Example 5.5.2. The space of natural numbers, N, is Hausdorff since the apartness map $(\neq)$ is realised by the following recursion:

$$(\neq)(x, y) = \text{if } x \stackrel{?}{=} 0 \text{ then } a \text{ else } b$$

where the subprograms a and b are defined as follows:

$$\begin{array}{lcl} a & = & \text{if } y \stackrel{?}{=} 0 \text{ then } \bot \text{ else } \top \\ b & = & \text{if } y \stackrel{?}{=} 0 \text{ then } \top \text{ else } (\neq)(\text{pred}(x), \text{pred}(y)) \end{array}$$

Notice that also N is (relatively) discrete since we have a recursive recipe for the closed term which tests for equality on the natural numbers:

$$(=)(x, y) = \text{if } x \stackrel{?}{=} 0 \text{ then } a \text{ else } b$$

where the subprograms a and b are given by:

$$\begin{array}{lcl} a & = & \text{if } y \stackrel{?}{=} 0 \text{ then } \top \text{ else } \bot \\ b & = & \text{if } y \stackrel{?}{=} 0 \text{ then } \bot \text{ else } (=)(\text{pred}(x), \text{pred}(y)) \end{array}$$

The natural numbers type `Nat`, however, is not discrete since we always need to consider the divergent element $\bot$.

Example 5.5.3. The Baire space is not discrete since operationally we need to test for equality on each term of the sequence. We omit the proof of this since this is similar to Example 5.4.2 . One may be tempted to claim that it is not possible to have an equality (or even apartness) test whenever we deal with data types that seemingly require one to check an infinitude of data for equality. However, this is not true as already shown by Gandy and Berger (cf. Berger [6]). We shall recall these examples and study them in an operational setting.

Example 5.5.4. Given a pair of elements $(s, t) \in \mathtt{Baire} \times \mathtt{Baire}$, consider the program $\mathtt{apart} \in \mathtt{Nat} \to \Sigma$ which is recursively defined as follows:

$$\mathtt{apart}(i) = \text{if } s(i) \neq t(i) \text{ then } \top \text{ else } \mathtt{apart}(i+1)$$

where $(\neq)$ is the inequality test on $\mathtt{Nat}$. This program evaluates to $\top$ iff the sequences s and t disagree somewhere from the ith position onwards. So the program

$$\mathtt{apartB}(s, t) = \mathtt{apart}(0)$$

is the required inequality test, thus justifying that the Baire space is Hausdorff.

The following proposition[1] might be worth noting that

Proposition 5.5.5. *In a discrete space, singletons consisting of definable elements are open.*

Proof. Let $x \in \sigma$ be a member of a discrete space X. Then the characteristic function of the singleton $\{x\}$ is given by $\chi_x(y) = (=)(x, y)$ where $(=)$ is the existing equality test available from the discreteness of X. □

Remark 5.5.6. At the time of writing, it is not clear whether an example of a discrete non-Hausdorff subspace exists.

5.6 Specialisation order

Recall that the specialisation order $\sqsubseteq$ of a T_0 space X is defined by

$$x \sqsubseteq y \overset{\text{def}}{\Leftrightarrow} \forall \text{ open set } U.(x \in U \implies y \in U).$$

The following says that the contextual order is the "specialisation order" of the operational topology:

Proposition 5.6.1. *For $x, y \in \sigma$, the relation $x \sqsubseteq y$ holds iff $x \in U$ implies $y \in U$ for every open subset U of σ.*

Proof. Ground contexts of type Σ suffices to test the operational preorder - see Proposition 3.6.2. Because x and y are closed terms, applicative contexts, i.e., characteristic functions of open sets, suffice. □

Remark 5.6.2. For any $x \neq y \in \sigma$ (i.e., contextually inequivalent), there is an open that contains exactly one of them. This means our operational topology is always "T_0".

[1] In the process of making minor modifications in this thesis, this proposition has been strengthened: In a discrete space, singletons are *always* open.

5.7 Compact sets

The intuition behind the topological notion of compactness is that a compact set behaves, in many important aspects, as if it were a finite set. The official topological definition, which is more obscure, says that a subset Q is a topological space is compact iff it satisfies the Heine-Borel property: any collection of open sets that cover Q has a finite subcollection that already covers Q. In order to arrive at an operational notion of compactness, we reformulate this in two stages.

(1) Any collection of open sets of a topological space can be made directed by adding the unions of finite subcollections. Hence a set Q is compact iff every directed cover of Q by open sets includes an open set that already covers Q.

(2) Considering the Scott topology on the lattice of open sets of the topological space, this amounts to saying that the collection of open sets U with $Q \subseteq U$ is Scott open in this lattice.

Thus this last reformulation considers open sets of open sets. We take this as our definition, with "Scott open" replaced by "open" in the sense of Definition 5.2.1: we say that a collection $\mathcal{U}$ of open sets of type σ is *open* if the collection $\{\chi_U \mid U \in \mathcal{U}\}$ is open in the function type $(\sigma \to \Sigma)$.

Proposition 5.7.1. *For any set Q of elements of a type σ, the following two conditions are equivalent:*

(i) The collection $\{U$ is open $\mid Q \subseteq U\}$ is open.

(ii) There is $\forall_Q \in ((\sigma \to \Sigma) \to \Sigma)$ such that

$$\forall_Q(p) = \top \Leftrightarrow \forall x \in Q.p(x) = \top.$$

Proof. $\forall_Q = \chi_{\mathcal{U}}$ for $\mathcal{U} = \{\chi_U \mid Q \subseteq U\}$, because if $p = \chi_U$ then $Q \subseteq U \iff p(x) = \top$ for all $x \in Q$. □

Definition 5.7.2. We say that a set Q of elements of a type σ is *compact* if it satisfies the above equivalent conditions. In this case, for the sake of clarity, we write "$\forall x \in Q. \ldots$" instead of "$\forall_Q(\lambda x. \ldots)$".

Proposition 5.7.1(2) gives a sense in which a compact set behaves like a set of finite cardinality: it is possible to universally quantify over it in a mechanical fashion. So it is not surprising that

Proposition 5.7.3. *Every finite set of any type is compact.*

Proof. Let $Q = \{q_1, \ldots, q_n\}$ be finite set. Then the program $\forall_Q(p) = p(q_1) \wedge p(q_2) \wedge \cdots \wedge p(q_n)$ (where $x \wedge y :=$ if x then y as previously defined) satisfies the condition that $\forall_Q(p) = \top$ iff $\forall x \in Q.p(x) = \top$. □

We postpone the examples of infinite compact sets till we revisit compactness in Chapter 11.

5.8 Properties of compact sets

Properties of compact sets that are familiar from classical topology hold for our operational notion:

Proposition 5.8.1. *(1) The empty set is compact. If Q_1 and Q_2 are compact subsets of the same type then $Q_1 \cup Q_2$ is again compact.*

(2) For any $f \in (\sigma \to \tau)$ and any compact set Q in σ, the set $f(Q) = \{f(x) \mid x \in Q\}$ is compact in τ.

(3) If Q is compact in σ and R is compact in τ, then $Q \times R$ is compact in $\sigma \times \tau$.

(4) If Q is compact in σ and V is open in τ, then

$$N(Q, V) := \{f \in (\sigma \to \tau) \mid f(Q) \subseteq V\}$$

is open in $(\sigma \to \tau)$.

Proof. (1): $\forall z \in Q_1 \cup Q_2.p(z) = \forall z \in Q_1.p(z) \ \wedge \ \forall z \in Q_2.p(z)$.

(2): $\forall y \in f(Q).p(y) = \forall x \in Q.p(f(x))$.

(3): $\forall z \in Q \times R.p(z) = \forall x \in Q.\forall y \in R.p(x, y)$.

(4): $\chi_{N(Q,V)}(f) = \forall x \in Q.\chi_V(f(x))$.

□

Remark 5.8.2. For (3) of the above proposition, open sets of this form are known in classical topology: They form the subbase that defines the so-called compact-open topology on the set of continuous maps.

Example 5.8.3. The set of all elements of any type σ is compact, but for trivial reasons: $p(x) = \top$ holds for all $x \in \sigma$ iff it holds for $x = \bot$, by monotonicity, and hence the definition $\forall_\sigma(p) = p(\bot)$ gives a universal quantification program.

The following properties of compact sets are also familiar to us from classical topology:

Proposition 5.8.4. *If X is Hausdorff and $Q \subseteq X$ is compact, then Q is closed in X.*

Proof. It boils down to showing that $X \backslash Q$ is open, i.e., its characteristic map $\chi_{X\backslash Q} \in (\sigma \to \Sigma)$. But it is easy to see that the following program does the required job: $\chi_{X\backslash Q}(x) = \forall y \in Q.(\neq)(x, y)$. □

Proposition 5.8.5. *In the presence of the disjunction operator $(\vee)$, if X is compact and $F \subseteq X$ is closed then F is compact.*

Proof. The required program is $\forall x \in X.\chi_{X\backslash F}(x) \vee p(x)$. □

A collection $\mathcal{Q}$ of opens of a type σ is said to be *compact* if the corresponding set of characteristic maps

$$\{U \text{ is open} \mid U \in \mathcal{Q}\}$$

is compact in $(\sigma \to \Sigma)$.

Proposition 5.8.6. *If a set $\mathcal{Q}$ of opens is compact, then its intersection $\bigcap \mathcal{Q}$ is open.*

Proof. The required program $\forall U \in \mathcal{Q}.\chi_U(x)$ satisfies the property that $\forall U \in \mathcal{Q}.\chi_U(x) = \top$ iff $x \in U$ for all $U \in \mathcal{Q}$ iff $x \in \bigcap \mathcal{Q}$. □

Part II

Operational Toolkit

In Pitts [41], A. Pitts developed some mathematical methods for reasoning about program properties based upon the *operational semantics* of a language PCFL, in contrast to methods based upon domain-theoretic denotational semantics. In his paper, Pitts showed how the notion of bisimulation, together with a certain co-induction principle, can be used to establish program equivalence.

We show how Pitts' methods can be adapted to work for both PCF and FPC. For readers who want to understand operational domain theory and topology of PCF in Part III but do not wish to spend time on recursive types may refer to Chapter 6 and skip Chapters 7 and 8.

In Chapter 6, we only show the necessary modifications for PCF and omit the proofs. In Chapter 7, the reworking in FPC is shown in full detail. In Chapter 8, the proof of the operational extensionality theorem is provided. The reader should note reworking Pitts' work [41] for the languages PCF and FPC requires hard work but little insight.

Chapter 6

Contextual equivalence and PCF bisimilarity

In this chapter, we develop operational machinery to reason about contextual equivalence of PCF programs. We do this by using bisimulation techniques and the co-induction principle. As an example of how these principles may be applied, we study the contextual preorder of the ordinal data type $\overline{\omega}$ in Section 6.5.

6.1 Bisimulation and bisimilarity

Throughout this section, we will be concerned with one particular complete lattice, $(\mathrm{Rel}, \leq)$. The elements of Rel are typed-indexed families

$$\mathcal{R} = \{\mathcal{R}_\sigma \mid \sigma \in \mathtt{Type}\}$$

of binary relations R_σ between closed PCF terms of type σ. Thus each component of $\mathcal{R}$ is a subset $\mathcal{R}_\sigma \subseteq \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma$. The partial ordering on Rel is defined to be set-theoretic inclusion in each component:

$$\mathcal{R} \leq \mathcal{R}' \stackrel{\mathrm{def}}{\Leftrightarrow} \forall \sigma \in \mathtt{Type}.\mathcal{R}_\sigma \subseteq \mathcal{R}'_\sigma.$$

Clearly the least upper bound of a subset of Rel is given by set-theoretic union in each component.

Given $\mathcal{R} \in$ Rel, $\langle\mathcal{R}\rangle$ and $[\mathcal{R}]$ are defined in Figure 6.1.

Clearly, $\mathcal{R} \mapsto \langle\mathcal{R}\rangle$ and $\mathcal{R} \mapsto [\mathcal{R}]$ are both monotone operators on Rel. So we can apply Theorem 2.1.2 and form their greatest (post-)fixed points.

A family of relations $\mathcal{S} \in$ Rel satisfying $\mathcal{S} \leq \langle\mathcal{S}\rangle$ is called a *PCF simulation*; the greatest such is called *PCF similarity* and written $\preceq$. A family

(1) $\langle\mathcal{R}\rangle$:

$$
\begin{array}{rcll}
t\langle\mathcal{R}\rangle_{\Sigma}t' & \iff & (t \Downarrow \top \Rightarrow t' \Downarrow \top) & (6.1)\\
s\langle\mathcal{R}\rangle_{\mathtt{Nat}}s' & \iff & \forall n \in \mathbb{N}.(s \Downarrow \underline{n} \Rightarrow s' \Downarrow \underline{n}) & (6.2)\\
b\langle\mathcal{R}\rangle_{\mathtt{Bool}}b' & \iff & \forall \mathtt{b} \in \mathbb{B}.(b \Downarrow \mathtt{b} \Rightarrow b' \Downarrow \mathtt{b}) & (6.3)\\
s\langle\mathcal{R}\rangle_{\overline{\omega}}s' & \iff & \forall t : \overline{\omega}.(s \Downarrow t+1 \Rightarrow & (6.4)\\
 & & \exists t' : \overline{\omega}.(s' \Downarrow t'+1 \wedge t\ \mathcal{R}_{\overline{\omega}}\ t')) & \\
f\langle\mathcal{R}\rangle_{\sigma\to\tau}f' & \iff & \forall t \in \mathrm{Exp}_{\sigma}.(f(t)\ \mathcal{R}_{\tau}\ f'(t)) & (6.5)\\
p\langle\mathcal{R}\rangle_{\sigma\times\tau}p' & \iff & (\mathrm{fst}(p)\ \mathcal{R}_{\sigma}\ \mathrm{fst}(p') \wedge \mathrm{snd}(p)\ \mathcal{R}_{\tau}\ \mathrm{snd}(p')) & (6.6)
\end{array}
$$

(2) $[\mathcal{R}]$:

$$
\begin{array}{rcll}
t[\mathcal{R}]_{\Sigma}t' & \iff & (t \Downarrow \top \Leftrightarrow t' \Downarrow \top) & (6.7)\\
s[\mathcal{R}]_{\mathtt{Nat}}s' & \iff & \forall n \in \mathbb{N}.(s \Downarrow \underline{n} \Leftrightarrow s' \Downarrow \underline{n}) & (6.8)\\
b\langle\mathcal{R}\rangle_{\mathtt{Bool}}b' & \iff & \forall \mathtt{b} \in \mathbb{B}.(b \Downarrow \mathtt{b} \iff b' \Downarrow \mathtt{b}) & (6.9)\\
s[\mathcal{R}]_{\overline{\omega}}s' & \iff & \forall t : \overline{\omega}.(s \Downarrow t+1 \implies & (6.10)\\
 & & \exists t' : \overline{\omega}.(s' \Downarrow t'+1 \wedge t\ \mathcal{R}_{\overline{\omega}}\ t')) & \\
 & & \text{and} & \\
 & & \forall t' : \overline{\omega}.(s' \Downarrow t'+1 \implies & \\
 & & \exists t : \overline{\omega}.(s \Downarrow t+1 \wedge t\ \mathcal{R}_{\overline{\omega}}\ t')) & \\
f[\mathcal{R}]_{\sigma\to\tau}f' & \iff & \forall t \in \mathrm{Exp}_{\sigma}.(f(t)\ \mathcal{R}_{\tau}\ f'(t)) & (6.11)\\
p[\mathcal{R}]_{\sigma\times\tau}p' & \iff & (\mathrm{fst}(p)\ \mathcal{R}_{\sigma}\ \mathrm{fst}(p') \wedge \mathrm{snd}(p)\ \mathcal{R}_{\tau}\ \mathrm{snd}(p')) & (6.12)
\end{array}
$$

Figure 6.1: Definitions of $\langle\mathcal{R}\rangle$ and $[\mathcal{R}]$ in PCF

of relations $\mathcal{B} \in \mathrm{Rel}$ satisfying $\mathcal{B} \leq [\mathcal{B}]$ is called a *PCF bisimulation*; the greatest such is called *PCF bisimilarity* and written as $\simeq$.

In other words, a simulation is a post-fixed point of the operator $\langle\rangle$ and a similarity is the greatest (post-)fixed point of the operator $\langle\rangle$. Similarly, for a bisimulation and a bisimilarity.

We shall pause for a while to spell out what the conditions $\mathcal{S} \leq \langle\mathcal{S}\rangle$ and $\mathcal{B} \leq [\mathcal{B}]$ mean. A simulation $\mathcal{S}$ is specified by a type-indexed family of binary relations, $\mathcal{S}_{\sigma} \subseteq \mathrm{Exp}_{\sigma} \times \mathrm{Exp}_{\sigma}$, satisfying the conditions in Figure 6.2.

Similarly, a bisimulation is specified by a type-indexed family of binary relations, $\mathcal{B}_{\sigma} \subseteq \mathrm{Exp}_{\sigma} \times \mathrm{Exp}_{\sigma}$, satisfying the conditions in Figure 6.3.

Remark 6.1.1. Note that by Theorem 2.1.2, similarity and bisimilarity are

$$
\begin{array}{lcll}
(t\ \mathcal{S}_{\Sigma}\ t' \wedge t \Downarrow \top) & \Longrightarrow & t' \Downarrow \top & \text{(sim 1)} \\
(s\ \mathcal{S}_{\mathtt{Nat}}\ s' \wedge s \Downarrow \underline{n}) & \Longrightarrow & s' \Downarrow \underline{n} & \text{(sim 2)} \\
(b\ \mathcal{S}_{\mathtt{Bool}}\ b' \wedge b \Downarrow \mathtt{b}) & \Longrightarrow & b' \Downarrow \mathtt{b} & \text{(sim 3)} \\
(s\ \mathcal{S}_{\overline{\omega}}\ s' \wedge s \Downarrow t+1) & \Longrightarrow & \exists t'.(s' \Downarrow t'+1 \wedge t\ \mathcal{S}_{\overline{\omega}}\ t') & \text{(sim 4)} \\
f\ \mathcal{S}_{\sigma\to\tau}\ F' & \Longrightarrow & \forall t \in \mathrm{Exp}_{\sigma}.(f(t)\ \mathcal{S}_{\tau}\ f'(t)) & \text{(sim 5)} \\
p\ \mathcal{S}_{\sigma\times\tau}\ p' & \Longrightarrow & (\mathrm{fst}(p)\ \mathcal{S}_{\sigma}\ \mathrm{fst}(p') \wedge \mathrm{snd}(p)\ \mathcal{S}_{\tau}\ \mathrm{snd}(p')) & \text{(sim 6)}
\end{array}
$$

Figure 6.2: PCF simulation conditions

$$
\begin{array}{lcll}
(t\ \mathcal{B}_{\Sigma}\ t' \wedge t \Downarrow \top) & \Longrightarrow & t' \Downarrow \top & \text{(bis 1a)} \\
(t\ \mathcal{B}_{\Sigma}\ t' \wedge t' \Downarrow \top) & \Longrightarrow & t \Downarrow \top & \text{(bis 1b)} \\
(s\ \mathcal{B}_{\mathtt{Nat}}\ s' \wedge s \Downarrow \underline{n}) & \Longrightarrow & s' \Downarrow \underline{n} & \text{(bis 2a)} \\
(s\ \mathcal{B}_{\mathtt{Nat}}\ s' \wedge s' \Downarrow \underline{n}) & \Longrightarrow & s \Downarrow \underline{n} & \text{(bis 2b)} \\
(b\ \mathcal{B}_{\mathtt{Bool}}\ b' \wedge b \Downarrow \mathtt{b}) & \Longrightarrow & b' \Downarrow \mathtt{b} & \text{(bis 3a)} \\
(b\ \mathcal{B}_{\mathtt{Bool}}\ b' \wedge b' \Downarrow \mathtt{b}) & \Longrightarrow & b \Downarrow \mathtt{b} & \text{(bis 3b)} \\
(s\ \mathcal{B}_{\overline{\omega}}\ s' \wedge s \Downarrow t+1) & \Longrightarrow & \exists t'.(s' \Downarrow t'+1 \wedge t\ \mathcal{B}_{\overline{\omega}}\ t') & \text{(bis 4a)} \\
(s\ \mathcal{B}_{\overline{\omega}}\ s' \wedge s' \Downarrow t'+1) & \Longrightarrow & \exists t.(s \Downarrow t+1 \wedge t\ \mathcal{B}_{\overline{\omega}}\ t') & \text{(bis 4b)} \\
f\ \mathcal{B}_{\sigma\to\tau}\ f' & \Longrightarrow & \forall t \in \mathrm{Exp}_{\sigma}.(f(t)\ \mathcal{B}_{\tau}\ f'(t)) & \text{(bis 5)} \\
p\ \mathcal{B}_{\sigma\times\tau}\ p' & \Longrightarrow & (\mathrm{fst}(p)\ \mathcal{B}_{\sigma}\ \mathrm{fst}(p') \wedge \mathrm{snd}(p) B_{\sigma} \mathrm{snd}(p')) & \text{(sim 6)}
\end{array}
$$

Figure 6.3: PCF bisimulation conditions

fixed points (rather than just post-fixed points) of their associated monotone operators.

6.2 Co-induction principle

In this section, we present a powerful and important proof technique called the *co-induction principle*.

Proposition 6.2.1. (Co-induction principle for $\preceq$ and $\simeq$: PCF)
Given $s, t \in \mathrm{Exp}_{\sigma}$, to prove that $s \simeq_{\sigma} t$ holds, it suffices to find a bisimulation $\mathcal{B}$ such that $s\ \mathcal{B}_{\sigma}\ t$. Similarly, to prove $s \preceq_{\sigma} t$, it suffices to find a simulation $\mathcal{S}$ with $s\ \mathcal{S}_{\sigma}\ t$.

Proposition 6.2.2. *PCF similarity is a preorder and PCF bisimilarity is the equivalence relation induced by it. In other words, for all types σ and all closed terms $t, t', t'' \in \mathrm{Exp}_{\sigma}$, one has:*

(1) $t \preceq_{\sigma} t$.

(2) $(t \preceq_{\sigma} t' \wedge t' \preceq_{\sigma} t'') \Rightarrow t \preceq_{\sigma} t''$.

(3) $t \simeq_{\sigma} t' \Leftrightarrow (t \preceq_{\sigma} t' \wedge t' \preceq_{\sigma} t)$.

6.3 Operational extensionality theorem

We extend $\preceq$ and $\simeq$ from closed terms to all typable PCF terms by considering closed instantiations of open terms. For convenience, we introduce a notation for this process.

Suppose $\mathcal{R} \in$ Rel. For any finite partial function Γ assigning types to variables

$$\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$$

for any type σ, and for any terms $s, s' \in \mathrm{Exp}_\sigma(\Gamma)$, define

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

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

With these definitions, we can state the following characterisation of contextual equivalence.

Theorem 6.3.1. (Operational extensionality theorem for PCF)
Contextual preorder (respectively, equivalence) coincides with open similarity (respectively, open bisimilarity):

$$\Gamma \vdash s \sqsubseteq_\sigma t \iff \Gamma \vdash s \preceq^\circ_\sigma t$$

and

$$\Gamma \vdash s =_\sigma t \iff \Gamma \vdash s \simeq^\circ_\sigma t.$$

In particular, the following co-induction principle for contextual equivalence holds: To prove that two closed terms are contextually equivalent, it suffices to find a bisimulation which relates them.

The techniques involved in proving the above theorem are similar to those for establishing Theorem 7.4.4 (Operational extensionality theorem for FPC) whose proof is presented in Chapter 8.

6.4 Kleene preorder and equivalence

For each type σ, consider the following binary relations on Exp_σ:

$$s \sqsubseteq^{kl}_\sigma t \stackrel{\text{def}}{\Leftrightarrow} \forall v \in \mathrm{Val}_\sigma.(s \Downarrow v \Rightarrow t \Downarrow v)$$

and

$$s \cong^{kl}_\sigma t \stackrel{\text{def}}{\Leftrightarrow} (s \sqsubseteq^{kl}_\sigma t) \wedge (t \sqsubseteq^{kl}_\sigma s).$$

The relation $\sqsubseteq^{kl}$ is called the *Kleene preorder*. If $s \cong^{kl}_{\sigma} t$ holds we say that s and t are *Kleene equivalent*.

Proposition 6.4.1. *For each type σ, we have*

$$s \sqsubseteq^{kl}_{\sigma} t \implies s \preceq_{\sigma} t$$

and

$$s \cong^{kl}_{\sigma} t \implies s \simeq_{\sigma} t.$$

Hence in view of Theorem 6.3.1, Kleene equivalent closed terms are contextually equivalent.

The following contextual equivalences of open terms follow immediately from the fact that they are all Kleene equivalences:

$$\begin{aligned}
(\lambda x.s)t &= s[t/x] \\
\mathrm{fst}(s,t) &= s \\
\mathrm{snd}(s,t) &= t \\
\text{if T then } s \text{ else } s' &= s \\
\text{if F then } s \text{ else } s' &= s' \\
\text{if } \top \text{ then } t &= t \\
(n+1)-1 &= n \\
0-1 &= 0 \\
\infty+1 &= \infty \\
\mathrm{fix}(f) &= f(\mathrm{fix}(f))
\end{aligned}$$

Proposition 6.4.2. *For each type σ, $\bot_{\sigma} := \mathrm{fix}(\lambda x^{\sigma}.x)$ is the least element of type σ with respect to the contextual order.*

Proof. Notice that since $\bot_{\sigma}$ does not evaluate to anything, we have that

$$\bot_{\sigma} \sqsubseteq^{kl}_{\sigma} t$$

for any $t \in \mathrm{Exp}_{\sigma}$. By Proposition 6.4.1, we deduce that $\bot_{\sigma}$ acts as the least element with respect to the contextual preorder. □

Remark 6.4.3. In $\overline{\omega}$, the elements $(0-1)+1$ and 1 are distinct canonical values which are contextually equivalent. This just indicates that the Kleene equivalence is *strictly* contained in the contextual equivalence. Also it follows from Kleene equivalence and transitivity of contextual equivalence that if $\emptyset \vdash s =_{\sigma} t$ and $s \Downarrow u$ and $t \Downarrow v$, then $\emptyset \vdash u =_{\sigma} v$.

6.5 Elements of ordinal type

In this section, we show that the contextual order of the ordinal type $\overline{\omega}$ is indeed the ordinal domain:

$$0 \sqsubset_{\overline{\omega}} 1 \sqsubset_{\overline{\omega}} \cdots \sqsubset_{\overline{\omega}} n \sqsubset_{\overline{\omega}} \cdots \sqsubset_{\overline{\omega}} \infty.$$

In other words, we shall prove that the closed terms $0, 1, \ldots, \infty$ are contextually inequivalent terms of type $\overline{\omega}$. Diagrammatically, the contextual order of $\overline{\omega}$ is given in Figure 6.4.

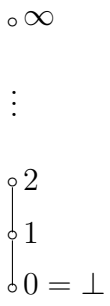


Figure 6.4: Vertical natural numbers: $\overline{\omega}$

At this juncture, the reader may refer to Section 3.2 to recall the definition of $0, \ldots, n, \ldots, \infty$.

To understand the contextual order of $\overline{\omega}$, we first consider a relation $\mathcal{S}$ between closed terms defined as follows:

$$\begin{array}{rcl}
t\ \mathcal{S}_{\Sigma}\ t' & \iff & (t \Downarrow \top \implies t' \Downarrow \top). \\
s\ \mathcal{S}_{\mathtt{Nat}}\ s' & \iff & (\forall n \in \mathbb{N}. s \Downarrow \underline{n} \implies s' \Downarrow \underline{n}). \\
b\ \mathcal{S}_{\mathtt{Bool}}\ b' & \iff & (\forall \mathtt{b}. b \Downarrow \mathtt{b} \implies b' \Downarrow \mathtt{b}). \\
s\ \mathcal{S}_{\overline{\omega}}\ s' & \iff & (\forall n \in \mathbb{N}. (s - n > 0) \Downarrow \top \implies (s' - n > 0) \Downarrow \top). \\
f\ \mathcal{S}_{\sigma\to\tau}\ F' & \iff & \forall f \in \mathrm{Exp}_{\sigma}. (f(t)\ \mathcal{S}_{\tau}\ f'(t)). \\
p\ \mathcal{S}_{\sigma\times\tau}\ p' & \iff & (\mathrm{fst}(p)\ \mathcal{S}_{\sigma}\ \mathrm{fst}(p')) \wedge (\mathrm{snd}(p)\ \mathcal{S}_{\tau}\ \mathrm{snd}(p')).
\end{array}$$

In the third clause, note that $(s - n)$ means

$$(\ldots((s \underbrace{-1) - 1) \cdots - 1}_{n \text{ copies}}).$$

To check that S defines a simulation, it is enough to verify that (sim 4) holds.

So suppose that $s \; \mathcal{S}_{\overline{\omega}} \; s'$ and $s \Downarrow t+1$. We must show that $s' \Downarrow t'+1$ for some $t' : \overline{\omega}$ and $t \; \mathcal{S}_{\overline{\omega}} \; t'$. Since $s \Downarrow t+1$, it follows that $(s > 0) \Downarrow \top$. It then follows from the definition of $\mathcal{S}_{\overline{\omega}}$ that $(s' > 0) \Downarrow \top$, i.e., $s' \Downarrow t'+1$ for some $t' : \overline{\omega}$. Notice that $s - 1 \cong^{kl}_{\overline{\omega}} t$ so that $s - 1 =_{\overline{\omega}} t$ by the co-induction principle. Similarly, $s' - 1 =_{\overline{\omega}} t'$. If $m \in \mathbb{N}$ is such that $(t - m > 0) \Downarrow \top$, then it follows that $(s - 1 - m > 0) \Downarrow \top$. Thus $(s - (m+1) > 0) \Downarrow \top$ by definition. From the definition of $\mathcal{S}_{\overline{\omega}}$, we have that $(s' - (m+1) > 0) \Downarrow \top$. Thus $((s'-1) - m > 0) \Downarrow \top$ and consequently $(t' - m > 0) \Downarrow \top$, as required. Thus we have proven that (sim 4) holds and $\mathcal{S}$ is a PCF simulation.

Notice that for each $n \in \mathbb{N}$, $n \; \mathcal{S}_{\overline{\omega}} \; n+1$. Now since $\mathcal{S}$ is a PCF simulation, the co-induction principle guarantees that $n \sqsubseteq_{\overline{\omega}} n+1$. Of course, the context $C[-_{\overline{\omega}}] := (-_{\overline{\omega}} - n > 0)$ distinguishes between n and $n+1$ so that they are obviously contextually inequivalent. Consequently for each $n \in \mathbb{N}$, it holds that

$$n \sqsubset_{\overline{\omega}} n+1.$$

We now argue that ∞ is the maximum element of type $\overline{\omega}$ with respect to the contextual order. First, an easy proof by induction on n shows that

$$\forall n \in \mathbb{N}. \infty - n \Downarrow \infty + 1.$$

This implies that $\forall n \in \mathbb{N}.(\infty - n > 0) \Downarrow \top$ and thus for every $t : \overline{\omega}$, it holds that $t \; \mathcal{S}_{\overline{\omega}} \; \infty$. Consequently, $t \sqsubseteq_{\overline{\omega}} \infty$ as we expected.

So far we have shown that for every closed term $t : \overline{\omega}$, it holds that

$$0 \sqsubseteq_{\overline{\omega}} t \sqsubseteq_{\overline{\omega}} \infty.$$

Let us prove that if $t \neq_{\overline{\omega}} \infty$, then $t =_{\overline{\omega}} m$ for some $m \in \mathbb{N}$. So suppose that $t \neq_{\overline{\omega}} \infty$. By the co-induction principle, (∞, t) does not belong to any simulation. In particular, for the simulation $\mathcal{S}$, there must exist an $n \in \mathbb{N}$ such that $(\infty - n > 0) \Downarrow \top$ and $(t - n > 0) \not\Downarrow \top$. Let m be the smallest such n. We now show that $t =_{\overline{\omega}} m$. Suppose $n \in \mathbb{N}$ is such that $(t - n > 0) \Downarrow \top$. Then by the minimality of m, we must have $n < m$. This then implies that $(m - n > 0) \Downarrow \top$ and thus $t \; \mathcal{S}_{\overline{\omega}} \; m$. It now remains to prove that $m \; \mathcal{S}_{\overline{\omega}} \; t$. For that purpose, let $n \in \mathbb{N}$ be such that $(m - n > 0) \Downarrow \top$. Then $m - n$ is of the form $x + 1$ for some $x : \overline{\omega}$ and it follows easily from the evaluation rule that $m > n$. Again by the minimality of m, we conclude that $(t - n > 0) \Downarrow \top$. Thus (t, m) and (m, t) are in $S_{\overline{\omega}}$. Finally, by the co-induction principle, we have $t =_{\overline{\omega}} m$, as required. Thus we have established that

Proposition 6.5.1. *The elements* $0, 1, 2 \ldots, n, \ldots, \infty$ *of* $\overline{\omega}$ *are contextually*

inequivalent. Moreover, the contextual order on $\overline{\omega}$ is given by:

$$0 \sqsubset_{\overline{\omega}} 1 \sqsubset_{\overline{\omega}} \cdots \sqsubset_{\overline{\omega}} n \sqsubset_{\overline{\omega}} \cdots \sqsubset_{\overline{\omega}} \infty.$$

More precisely, any element of type $\overline{\omega}$ is contextually equivalent to one of these elements.

6.6 Rational chains

An important property proven using operational methods in Pitts [41] is rational-chain completeness, which can be stated as follows:

Theorem 6.6.1. *For any $g \in (\tau \to \tau)$ and any $h \in (\tau \to \sigma)$, the sequence $h(g^{(n)}(\bot))$ is increasing and has $h(\mathrm{fix}(g))$ as a least upper bound in the contextual order:*

$$h(\mathrm{fix}(g)) = \bigsqcup_n h(g^{(n)}(\bot)).$$

Definition 6.6.2. A sequence x_n of elements of a type σ is called a *rational chain* if there exists $g \in (\tau \to \tau)$ and $h \in (\tau \to \sigma)$ with $x_n = h(g^{(n)}(\bot))$.

In our thesis, we identify rational-chain completeness to be the salient completeness condition in the development of an operational domain theory. The notion of rational chains will be revisited and built upon in Chapter 9 where the relation between rational-chain completeness and synthetically open sets is studied. The reader should note that rational-chain completeness also holds for the language FPC and is proven in Section 7.6. In this thesis, we use the terms "rational-chain completeness" and "rational completeness" interchangeably.

Chapter 7

Contextual equivalence and FPC bisimilarity

In this chapter, we rework the results of A.M. Pitts' work for the language FPC, following closely the structure of Pitts [41]. In particular, we develop the operational machineries necessary for reasoning about program equivalence without appeal to any denotational model. Readers who only want to understand operational domain theory and topology but do not wish to spend time on recursive types may safely skip this chapter and Chapter 8. Note that only hard work, but no insight, is required in the reworking of Pitts' work [41] for the language FPC.

7.1 Properties of FPC contextual equivalence

In this section, we gather at one place the following groups of properties concerning FPC contextual equivalence. Properties which are not a direct consequence of the definition of the contextual preorder $\sqsubseteq$ will be proven later. In such cases, the reader will be given the reference to where that particular property is established.

7.1.1 Inequational logic

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash t \sqsubseteq_\sigma t \tag{7.1}$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \vdash t' \sqsubseteq_\sigma t'') \implies \Gamma \vdash t \sqsubseteq_\sigma t'' \tag{7.2}$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \vdash t' \sqsubseteq_\sigma t) \iff \Gamma \vdash t =_\sigma t' \tag{7.3}$$

$$\Gamma, x:\sigma \vdash t \sqsubseteq_\tau t' \implies \Gamma \vdash \lambda x.t \sqsubseteq_{\sigma\to\tau} \lambda x.t' \quad (7.4)$$

$$(\Gamma \vdash s \sqsubseteq_{\sigma_\perp} s' \wedge \Gamma, x:\sigma \vdash t \sqsubseteq_\rho t') \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).t' \quad (7.5)$$

$$(\Gamma \vdash s \sqsubseteq_{\sigma+\tau} s' \wedge \Gamma, x:\sigma \vdash t_1 \sqsubseteq_\rho t_1' \wedge \Gamma, y:\tau \vdash t_2 \sqsubseteq_\rho t_2') \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \sqsubseteq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t_1'\ \mathrm{or}\ \mathrm{inr}(y).t_2' \quad (7.6)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma \subseteq \Gamma') \implies \Gamma' \vdash t \sqsubseteq_\sigma t' \quad (7.7)$$

$$(\Gamma \vdash t \sqsubseteq_\sigma t' \wedge \Gamma, x:\sigma \vdash s:\tau) \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s[t'/x] \quad (7.8)$$

$$(\Gamma \vdash t:\sigma \wedge \Gamma, x:\sigma \vdash s \sqsubseteq_\tau s') \implies \Gamma \vdash s[t/x] \sqsubseteq_\tau s'[t/x] \quad (7.9)$$

Properties (7.1) - (7.8) are direct consequences of the definitions of $\sqsubseteq_\sigma$ and $=_\sigma$. By contrast, (7.9) is not so straightforward to establish since the operation $s \mapsto s[t/x]$ is not necessarily of the form $s \mapsto C[s]$ for some context $C[-]$. We shall prove (7.9) in Lemma 8.4.2.

7.1.2 β-equalities

$$(\Gamma, x:\sigma \vdash s:\tau \wedge \Gamma \vdash t:\sigma) \implies \Gamma \vdash (\lambda x.s)t =_\tau s[t/x] \quad (7.10)$$

$$(\Gamma \vdash s:\sigma \wedge \Gamma \vdash t:\tau) \implies (\Gamma \vdash \mathrm{fst}(s,t) =_\sigma s \wedge \Gamma \vdash \mathrm{snd}(s,t) =_\tau t) \quad (7.11)$$

$$\Gamma \vdash t:\sigma \implies \Gamma \vdash \mathrm{case}(\mathrm{up}(t))\ \mathrm{of}\ \mathrm{up}(x).s =_\sigma s[t/x] \quad (7.12)$$

$$\Gamma \vdash t:\sigma \implies \forall \Gamma \vdash s:\tau, \Gamma \vdash s':\tau. (\Gamma \vdash \mathrm{case}(\mathrm{inl}(t))\ \mathrm{of}\ \mathrm{inl}(x).s\ \mathrm{or}\ \mathrm{inr}(y).s' =_\sigma s[t/x]) \quad (7.13)$$

$$\Gamma \vdash t:\tau \implies \forall \Gamma \vdash s:\tau, \Gamma \vdash s':\tau. (\Gamma \vdash \mathrm{case}(\mathrm{inr}(t))\ \mathrm{of}\ \mathrm{inl}(x).s\ \mathrm{or}\ \mathrm{inr}(y).s' =_\tau s'[t/y]) \quad (7.14)$$

$$\Gamma \vdash t:\sigma[\mu X.\sigma/X] \implies \Gamma \vdash \mathrm{unfold}(\mathrm{fold}(t)) =_{\sigma[\mu X.\sigma/X]} t \quad (7.15)$$

These *β-equalities* are valid because of the characterisation of contextual equivalence in terms of FPC bisimilarity to be given in the next section (i.e., Theorem 7.4.4). For in each case, (closed instantiations of) the term on the left hand side of $=_\sigma$ evaluates to a canonical value v if and only if (closed

instantiations of) the right hand term evaluates to the same canonical value. Thus each of (7.10) - (7.15) follows from The β-equalities then follow from the fact, shown in Proposition 7.5.1, that the Kleene equivalence is contained in the contextual equivalence, together with (7.16), which is the first of the following *extensionality properties*.

7.1.3 Extensionality properties

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

$$\vec{x} : \vec{\sigma} \vdash s \sqsubseteq_\sigma s' \iff \forall t_i \in \mathrm{Exp}_{\sigma_i}(i = 1, \ldots, n). \; (s[\vec{t}/\vec{x}] \sqsubseteq_\sigma s'[\vec{t}/\vec{x}]). \tag{7.16}$$

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

$$s \sqsubseteq_\Sigma s' \iff (s \Downarrow \top \implies s' \Downarrow \top). \tag{7.17}$$

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

$$p \sqsubseteq_{\sigma\times\tau} p' \iff (\mathrm{fst}(p) \sqsubseteq_\sigma \mathrm{fst}(p') \wedge \mathrm{snd}(p) \sqsubseteq_\tau \mathrm{snd}(p')). \tag{7.18}$$

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

$$\begin{aligned} s \sqsubseteq_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma.\forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.s' \Downarrow \mathrm{inl}(a') \wedge a \sqsubseteq_\sigma a') \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.s' \Downarrow \mathrm{inr}(b') \wedge b \sqsubseteq_\tau b'). \end{aligned} \tag{7.19}$$

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

$$\begin{aligned} t \sqsubseteq_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s'.t' \Downarrow \mathrm{up}(s') \wedge s \sqsubseteq_\sigma s'). \end{aligned} \tag{7.20}$$

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

$$t \sqsubseteq_{\mu X.\sigma} t' \iff \mathrm{unfold}(t) \sqsubseteq_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t'). \tag{7.21}$$

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

$$f \sqsubseteq_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.(f(t) \sqsubseteq_\tau f'(t)). \tag{7.22}$$

Extensionality properties analogous to the above hold by construction for the notion of FPC bisimilarity introduced in the next section. Thus (7.17) -

(7.22) will follow once it has been proven that FPC bisimilarity coincides with contextual equivalence (cf. Theorem 7.4.4). Note that by virtue of Theorem 7.4.4, property (7.16) follows once Lemma 7.4.2 has been established.

7.1.4 η-equalities

The following η-equalities follow by combining the extensionality properties with the corresponding β-equality:

$$(\Gamma \vdash f : \sigma \to \tau \wedge x \notin \mathrm{dom}(\Gamma)) \implies \Gamma \vdash f =_{\sigma\to\tau} \lambda x.f(x) \quad (7.23)$$

$$\Gamma \vdash p : \sigma \times \tau \implies \Gamma \vdash p =_{\sigma\times\tau} (\mathrm{fst}(p), \mathrm{snd}(p)) \quad (7.24)$$

$$\Gamma \vdash t : \sigma + \tau \implies \Gamma \vdash t =_{\sigma+\tau} \quad (7.25)$$
$$\mathrm{case}(t) \text{ of } \mathrm{inl}(x).\mathrm{inl}(x) \text{ or } \mathrm{inr}(y).\mathrm{inr}(y)$$

$$\Gamma \vdash t : \sigma_\perp \implies \Gamma \vdash t =_{\sigma_\perp} \mathrm{case}(t) \text{ of } \mathrm{up}(x).\mathrm{up}(x) \quad (7.26)$$

$$\Gamma \vdash t : \mu X.\sigma \implies \Gamma \vdash t =_{\mu X.\sigma} \mathrm{fold}(\mathrm{unfold}(t)) \quad (7.27)$$

For example, to prove (7.27), it is enough, by virtue of (7.21), to show that $\Gamma \vdash \mathrm{unfold}(t) =_{\mu X.\sigma} \mathrm{unfold}(\mathrm{fold}(\mathrm{unfold}(t)))$. But the β-equality (7.15) guarantees that $\Gamma \vdash \mathrm{unfold}(\mathrm{fold}(\mathrm{unfold}(t))) =_{\mu X.\sigma} \mathrm{unfold}(t)$. Hence (7.27) holds.

Notice that properties (7.15) and (7.27) together imply that:

Proposition 7.1.1. *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.

7.1.5 Unfolding recursive terms

Recall that $\mathrm{fix}_\sigma := \lambda f : (\sigma \to \sigma).k(\mathrm{fold}^\tau(k))$ where $\tau := \mu X.(X \to \sigma)$ and $k := \lambda x^\tau.f(\mathrm{unfold}^\tau(x)x)$. Using the β-equality (7.10), we have that

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

But writing out k explicitly, it follows from (7.10) and (7.15) that

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

Thus we have:

$$\Gamma \vdash f : \sigma \to \sigma \quad \Longrightarrow \quad \mathrm{fix}(f) =_\sigma f(\mathrm{fix}(f)) \tag{7.28}$$

7.1.6 Syntactic bottom

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

$$\Gamma \vdash t : \sigma \quad \Longrightarrow \quad \Gamma \vdash \bot_\sigma \sqsubseteq_\sigma t \tag{7.29}$$

The proof of this is found in Section 7.5.

7.1.7 Rational-chain completeness and continuity

In addition to the unfolding property (7.28), terms of the form $\mathrm{fix}(f)$ enjoy a least prefixed-point property: if $f \in \mathrm{Exp}_{\sigma\to\sigma}$ and $t \in \mathrm{Exp}_\sigma$, then

$$f(t) \sqsubseteq_\sigma t \quad \Longrightarrow \quad \mathrm{fix}(f) \sqsubseteq_\sigma t \tag{7.30}$$

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

Define the sequence of terms $(f^{(n)}(\bot_\sigma))_{n\in\mathbb{N}}$ as follows:

$$\begin{array}{rcl} f^{(0)}(\bot_\sigma) & := & \bot_\sigma \\ f^{(n+1)}(\bot_\sigma) & := & f(f^{(n+1)}(\bot_\sigma)) \end{array}$$

It follows from (7.29) and (7.8) that these terms form an ascending chain:

$$\bot_\sigma \sqsubseteq_\sigma f(\bot_\sigma) \sqsubseteq_\sigma f^{(2)}(\bot_\sigma) \sqsubseteq_\sigma \cdots. \tag{7.31}$$

We claim that

$$\mathrm{fix}(f) \;=_\sigma \bigsqcup_n f^{(n)}(\bot_\sigma) \tag{7.32}$$

In other words, for each $t \in \mathrm{Exp}_\sigma$,

$$\mathrm{fix}(f) \sqsubseteq_\sigma t \quad \Longleftrightarrow \quad \forall n \in \mathbb{N}.(f^{(n)}(\bot_\sigma)) \sqsubseteq_\sigma t). \tag{7.33}$$

Such a canonical chain as in (7.31) belongs to a class of chains called the

rational chains, which are of the form

$$g(\bot_\sigma) \sqsubseteq_\tau gh(\bot_\sigma) \sqsubseteq_\tau gh^{(2)}(\bot_\sigma) \sqsubseteq_\tau \ldots \tag{7.34}$$

where $g : \sigma \to \tau$ and $h : \sigma \to \sigma$ are function-type FPC closed terms. It will be shown that the collection of FPC terms preordered by $\sqsubseteq$ enjoys *rational-chain completeness*, i.e.

$$\bigsqcup_n g(h^{(n)}(\bot_\tau)) \quad = \quad g(\mathrm{fix}(h)) \tag{7.35}$$

The operations of FPC preserve these suprema in that for each context $C[-_\sigma] \in \mathrm{Ctx}_\rho$, it holds that

$$C[g(\mathrm{fix}(f))] \sqsubseteq_\rho t \quad \Longleftrightarrow \quad \forall n \in \mathbb{N}.(C[g(f^{(n)}(\bot_\sigma))] \sqsubseteq_\rho t). \tag{7.36}$$

Both properties (7.32) and (7.35) will be proven in Section 7.6 using operational methods (cf. Theorem 7.6.6).

7.2 FPC similarity and bisimilarity

Let $\mathcal{R}= \{\mathcal{R}_\sigma|\sigma\}$ be a type-indexed family of binary relations $\mathcal{R}_\sigma$ between closed FPC terms of type σ. Given $\mathcal{R}$, the definitions of $\langle\mathcal{R}\rangle$ and $[\mathcal{R}]$ are given in Figure 7.1. The reader is invited to compare the definition of $\langle\mathcal{R}\rangle$ with the extensionality properties (7.17) - (7.22) which we claim to hold. The idea is to first define $\langle\mathcal{R}\rangle$ (respectively, $[\mathcal{R}]$) in such a way as to 'model' the extensionality properties we have in mind and once we have established that the contextual preorder is a bisimulation, then it is immediate that it satisfies these extensionality properties.

Because the operators $\mathcal{R} \mapsto \langle\mathcal{R}\rangle$ and $\mathcal{R} \mapsto [\mathcal{R}]$ are monotone on the set of all typed-indexed families of binary relations between closed FPC terms, by Theorem 2.1.2 they have greatest (post-)fixed points.

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

An FPC simulation (respectively, bisimulation) $\mathcal{S}$ (respectively, $\mathcal{B}$) is specified by a type-indexed family of binary relations $\mathcal{S}_\sigma \subseteq \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma$ (respectively, $\mathcal{B}_\sigma \subseteq \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma$), satisfying the conditions in Figure 7.2

(1) $\langle\mathcal{R}\rangle$:

$$\forall s, s' : 1, \quad \text{define} \quad s\langle\mathcal{R}\rangle_1 s'. \tag{7.37}$$

$$p\langle\mathcal{R}\rangle_{\sigma\times\tau} p' \iff \mathrm{fst}(p)\ \mathcal{R}_\sigma\ \mathrm{fst}(p') \wedge \mathrm{snd}(p)\ \mathcal{R}_\tau\ \mathrm{snd}(p') \tag{7.38}$$

$$\begin{aligned} s\langle\mathcal{R}\rangle_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma.\forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.s' \Downarrow \mathrm{inl}(a') \wedge a\ \mathcal{R}_\sigma\ a') \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.s' \Downarrow \mathrm{inr}(b') \wedge b\ \mathcal{R}_\tau\ b') \end{aligned} \tag{7.39}$$

$$\begin{aligned} t\langle\mathcal{R}\rangle_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s' \in \mathrm{Exp}_\sigma.t' \Downarrow \mathrm{up}(s') \wedge s\ \mathcal{R}_\sigma\ s') \end{aligned} \tag{7.40}$$

$$t\langle\mathcal{R}\rangle_{\mu X.\sigma} t' \iff \mathrm{unfold}(t)\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') \tag{7.41}$$

$$f\langle\mathcal{R}\rangle_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.(f(t)\ \mathcal{R}_\tau\ f'(t)) \tag{7.42}$$

(2) $[\mathcal{R}]$:

$$\forall s, s' : 1, \quad \text{define} \quad s[\mathcal{R}]_1 s'. \tag{7.43}$$

$$p[\mathcal{R}]_{\sigma\times\tau} p' \iff \mathrm{fst}(p)\ \mathcal{R}_\sigma\ \mathrm{fst}(p') \wedge \mathrm{snd}(p)\ \mathcal{R}_\tau\ \mathrm{snd}(p') \tag{7.44}$$

$$\begin{aligned} s[\mathcal{R}]_{\sigma+\tau} s' \iff\ & \forall a \in \mathrm{Exp}_\sigma.\forall b \in \mathrm{Exp}_\tau. \\ & (s \Downarrow \mathrm{inl}(a) \implies \exists a' \in \mathrm{Exp}_\sigma.s' \Downarrow \mathrm{inl}(a') \wedge a\ \mathcal{R}_\sigma\ a') \wedge \\ & (s \Downarrow \mathrm{inr}(b) \implies \exists b' \in \mathrm{Exp}_\tau.s' \Downarrow \mathrm{inr}(b') \wedge b\ \mathcal{R}_\tau\ b') \wedge \\ & \text{and} \\ & \forall a' \in \mathrm{Exp}_\sigma, \forall b' \in \mathrm{Exp}_\tau. \\ & (s' \Downarrow \mathrm{inl}(a') \implies \exists a \in \mathrm{Exp}_\sigma.s \Downarrow \mathrm{inl}(a) \wedge a\ \mathcal{R}_\sigma\ a') \wedge \\ & (s' \Downarrow \mathrm{inr}(b') \implies \exists b \in \mathrm{Exp}_\tau.s \Downarrow \mathrm{inr}(b) \wedge b\ \mathcal{R}_\tau\ b') \end{aligned} \tag{7.45}$$

$$\begin{aligned} t[\mathcal{R}]_{\sigma_\perp} t' \iff\ & \forall s \in \mathrm{Exp}_\sigma. \\ & (t \Downarrow \mathrm{up}(s) \implies \exists s' \in \mathrm{Exp}_\sigma.t' \Downarrow \mathrm{up}(s') \wedge s\ \mathcal{R}_\sigma\ s') \\ & \text{and} \\ & \forall s' \in \mathrm{Exp}_\sigma. \\ & (t' \Downarrow \mathrm{up}(s') \implies \exists s \in \mathrm{Exp}_\sigma.t \Downarrow \mathrm{up}(s) \wedge s\ \mathcal{R}_\sigma\ s') \end{aligned} \tag{7.46}$$

$$t[\mathcal{R}]_{\mu X.\sigma} t' \iff \mathrm{unfold}(t)\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') \tag{7.47}$$

$$f[\mathcal{R}]_{\sigma\to\tau} f' \iff \forall t \in \mathrm{Exp}_\sigma.(f(t)\ \mathcal{R}_\tau\ f'(t)) \tag{7.48}$$

Figure 7.1: Definitions of $\langle\mathcal{R}\rangle$ and $[\mathcal{R}]$ in FPC

$$
\begin{array}{lcll}
\forall s, s' : 1,\ s\ \mathcal{S}_1\ s'. & & & \text{(sim 1)} \\
s\ \mathcal{S}_{\sigma\times\tau}\ s' & \Longrightarrow & \mathrm{fst}(s)\ \mathcal{S}_\sigma\ \mathrm{fst}(s') \wedge \mathrm{snd}(s)\ \mathcal{S}_\tau\ \mathrm{snd}(s') & \text{(sim 2)} \\
(s\ \mathcal{S}_{\sigma+\tau}\ s' \wedge s \Downarrow \mathrm{inl}(a)) & \Longrightarrow & \exists a' \in \mathrm{Exp}_\sigma.(s' \Downarrow \mathrm{inl}(a') \wedge a\ \mathcal{S}_\sigma\ a') & \text{(sim 3a)} \\
(s\ \mathcal{S}_{\sigma+\tau}\ s' \wedge s \Downarrow \mathrm{inr}(b)) & \Longrightarrow & \exists b' \in \mathrm{Exp}_\tau.(s' \Downarrow \mathrm{inl}(b') \wedge b\ \mathcal{S}_\tau\ b') & \text{(sim 3b)} \\
(t\ \mathcal{S}_{\sigma_\perp}\ t' \wedge t \Downarrow \mathrm{up}(s)) & \Longrightarrow & \exists s' \in \mathrm{Exp}_\sigma.(t' \Downarrow \mathrm{up}(s') \wedge s\ \mathcal{S}_\sigma\ s') & \text{(sim 4)} \\
t\ \mathcal{S}_{\mu X.\sigma}\ t' & \Longrightarrow & \mathrm{unfold}(t)\ \mathcal{S}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') & \text{(sim 5)} \\
f\ \mathcal{S}_{\sigma\to\tau}\ f' & \Longrightarrow & \forall t \in \mathrm{Exp}_\sigma.(f(t)\ \mathcal{S}_\tau\ f'(t)) & \text{(sim 6)}
\end{array}
$$

Figure 7.2: FPC simulation conditions

$$
\begin{array}{lcll}
\forall s, s' : 1,\ s\ \mathcal{B}_1\ s'. & & & \text{(bis 1)} \\
s\ \mathcal{B}_{\sigma\times\tau}\ s' & \Longrightarrow & \mathrm{fst}(s)\ \mathcal{B}_\sigma\ \mathrm{fst}(s') \wedge \mathrm{snd}(s)\ \mathcal{B}_\tau\ \mathrm{snd}(s') & \text{(bis 2)} \\
(s\ \mathcal{B}_{\sigma+\tau}\ s' \wedge s \Downarrow \mathrm{inl}(a)) & \Longrightarrow & \exists a' \in \mathrm{Exp}_\sigma.(s' \Downarrow \mathrm{inl}(a') \wedge a\ \mathcal{B}_\sigma\ a') & \text{(sim 3a)} \\
(s\ \mathcal{B}_{\sigma+\tau}\ s' \wedge s' \Downarrow \mathrm{inl}(a')) & \Longrightarrow & \exists a \in \mathrm{Exp}_\sigma.(s \Downarrow \mathrm{inl}(a) \wedge a\ \mathcal{B}_\sigma\ a') & \text{(sim 3b)} \\
(s\ \mathcal{B}_{\sigma+\tau}\ s' \wedge s \Downarrow \mathrm{inr}(b)) & \Longrightarrow & \exists b' \in \mathrm{Exp}_\tau.(s' \Downarrow \mathrm{inl}(b') \wedge b\ \mathcal{B}_\tau\ b') & \text{(sim 3c)} \\
(s\ \mathcal{B}_{\sigma+\tau}\ s' \wedge s' \Downarrow \mathrm{inr}(b')) & \Longrightarrow & \exists b \in \mathrm{Exp}_\tau.(s \Downarrow \mathrm{inl}(b) \wedge b\ \mathcal{B}_\tau\ b') & \text{(sim 3d)} \\
(t\ \mathcal{B}_{\sigma_\perp}\ t' \wedge t \Downarrow \mathrm{up}(s)) & \Longrightarrow & \exists s' \in \mathrm{Exp}_\sigma.(t' \Downarrow \mathrm{up}(s') \wedge s\ \mathcal{B}_\sigma\ s') & \text{(bis 4a)} \\
(t\ \mathcal{B}_{\sigma_\perp}\ t' \wedge t' \Downarrow \mathrm{up}(s')) & \Longrightarrow & \exists s \in \mathrm{Exp}_\sigma.(t \Downarrow \mathrm{up}(s) \wedge s\ \mathcal{B}_\sigma\ s') & \text{(bis 4b)} \\
t\ \mathcal{B}_{\mu X.\sigma}\ t' & \Longrightarrow & \mathrm{unfold}(t)\ \mathcal{B}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t') & \text{(bis 5)} \\
f\ \mathcal{B}_{\sigma\to\tau}\ f' & \Longrightarrow & \forall t \in \mathrm{Exp}_\sigma.(f(t)\ \mathcal{B}_\tau\ f'(t)) & \text{(bis 6)}
\end{array}
$$

Figure 7.3: FPC bisimulation conditions

(respectively, Figure 7.3).

7.3 Co-induction principle

Proposition 7.3.1. (Co-induction principle for $\preceq$ and $\simeq$: FPC)
Given $s, t \in \sigma$, to prove that $s \simeq_\sigma t$, it suffices to find an FPC bisimulation $\mathcal{B}$ such that $s\ \mathcal{B}_\sigma\ t$. Likewise, to show that $s \preceq_\sigma t$, it is enough to find an FPC simulation $\mathcal{S}$ such that $s\ \mathcal{S}_\sigma\ t$.

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

Once we have established that FPC bisimilarity and contextual equivalence coincide, the above proposition will provide a powerful tool for proving contextual equivalence. For the moment, we use this proposition to establish some basic facts about (bi)similarity.

Proposition 7.3.2. *FPC similarity is a preorder and FPC bisimilarity is the equivalence relation induced by it, i.e., for all closed types σ and all closed terms $t, t', t'' \in \mathrm{Exp}_\sigma$, it holds that:*

(1) $t \preceq_\sigma t$

(2) $(t \preceq_\sigma t' \wedge t' \preceq_\sigma t'') \implies t \preceq_\sigma t''$

(3) $t \simeq_\sigma t' \iff (t \preceq_\sigma t' \wedge t' \preceq_\sigma t)$

Proof. (1) The relation $\mathcal{R}$ defined by

$$\mathcal{R}_\sigma := \{(t,t) | t \in \mathrm{Exp}_\sigma\}$$

is trivially an FPC simulation. Thus, by Proposition 7.3.1, (1) holds.

(2) Consider the relation $\mathcal{R}$ defined by

$$\mathcal{R}_\sigma := \{(t,t'') \in \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma | \exists t' \in \mathrm{Exp}_\sigma . (t \preceq_\sigma t' \wedge t' \preceq_\sigma t'')\}.$$

Because $\preceq$ itself is a simulation, it follows that $\mathcal{R}$ is a simulation.

(3) Notice that since $\simeq$ satisfies the bisimulation conditions in Figure 7.3, both $\{(t,t') | t \simeq_\sigma t'\}$ and $\{(t,t') | t' \simeq_\sigma t\}$ trivially determine FPC simulations. Hence by Proposition 7.3.1, both of these are contained in $\preceq$, and thus we have proven that

$$t \simeq_\sigma t' \implies (t \preceq_\sigma t' \wedge t' \preceq_\sigma t).$$

It remains to show that the relation

$$\{(t,t') \in \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma | t \preceq_\sigma t' \wedge t' \preceq_\sigma t\}$$

is contained in $\simeq_\sigma$, i.e., it satisfies the conditions in Figure 7.3. But this is the case because of determinacy of evaluation and the fact that $\preceq$ is an FPC simulation. Thus, by Proposition 7.3.1, the other implication

$$(t \preceq_\sigma t' \wedge t' \preceq_\sigma t) \implies t \simeq_\sigma t'$$

holds and the proof of (3) is complete.

□

7.4 Operational extensionality theorem

Before establishing that bisimilarity and contextual equivalence coincide for FPC, we need to extend the definitions of $\preceq$ and $\simeq$ from closed terms to all typable FPC terms by considering closed instantiations of open terms.

Suppose $\mathcal{R}$ is a typed-indexed family of binary relations between closed FPC terms. For any term context $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$, for any closed type σ and for any terms $s, s' \in \mathrm{Exp}_\sigma(\Gamma)$, define

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

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

Proposition 7.4.1. *FPC open similarity is a preorder and FPC open bisimilarity is the equivalence relation induced by it. In other words, for all typing assignments $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and for all closed types σ and for all closed terms $t, t', t'' \in \mathrm{Exp}_\sigma(\Gamma)$, one has:*

(1) $\Gamma \vdash t \preceq^\circ_\sigma t$

(2) $(\Gamma \vdash t \preceq^\circ_\sigma t' \wedge \Gamma \vdash t' \preceq^\circ_\sigma t'') \implies \Gamma \vdash t \preceq^\circ_\sigma t''$.

(3) $(\Gamma \vdash t \preceq^\circ_\sigma t' \wedge \Gamma \vdash t' \preceq^\circ_\sigma t) \iff \Gamma \vdash t \simeq_\sigma t'$.

Proof. Because $\preceq^\circ$ is defined via an extension of $\preceq$ by considering closed instantiation of open terms, one just relies on the closed analogues i.e., those in Proposition 7.3.2. □

Lemma 7.4.2. *If $\Gamma \vdash t : \sigma$ and $\Gamma, x : \sigma \vdash s \preceq^\circ_\tau s'$, then $\Gamma \vdash s[t/x] \preceq^\circ_\tau s'[t/x]$.*

Proof. Let $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$ and suppose that $t_1 \in \mathrm{Exp}_{\sigma_1}, \ldots, t_n \in \mathrm{Exp}_{\sigma_n}$ are given. We wish to show that

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

To do this, recall from the definition of $\preceq^\circ$ that since $\Gamma, x : \sigma \vdash s \preceq^\circ_\tau s'$ it holds that

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

But $(s[t/x])[\vec{t}/\vec{x}] \equiv s[t/x, \vec{t}/\vec{x}]$ because $x \notin \mathrm{dom}(\Gamma)$. Thus we have:

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

□

Remark 7.4.3. Once we have established Theorem 7.4.4, the above Lemma then implies that (7.16) holds.

Theorem 7.4.4. (Operational extensionality theorem for FPC)
Contextual preorder (respectively, equivalence) coincides with similarity (respectively, bisimilarity):

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

and

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

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

Proof. The proof of this is presented in Chapter 8. □

7.5 Kleene preorder and equivalence

In this section, we look at a particular kind of program equivalence called *Kleene equivalence* which turns out to be a bisimulation. Consequently, the co-induction principle for contextual equivalence guarantees that the Kleene equivalence is (properly) contained in it.

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

$$t \sqsubseteq^{kl}_\sigma t' \stackrel{\text{def}}{\Leftrightarrow} \forall v.(t \Downarrow v \Rightarrow t' \Downarrow v)$$

and

$$t \cong^{kl}_\sigma t' \stackrel{\text{def}}{\Leftrightarrow} (t \sqsubseteq^{kl}_\sigma t') \wedge (t' \sqsubseteq^{kl}_\sigma t).$$

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

Proposition 7.5.1. *For any closed type σ and any $t, t' \in \mathrm{Exp}_\sigma$, it holds that*

(1) $t \sqsubseteq^{kl}_\sigma t' \implies t \preceq_\sigma t'$

(2) $t \cong^{kl}_\sigma t \implies t \simeq_\sigma t'$

Proof. Notice that because $\simeq$ is the symmetrisation of $\prec$, once (1) is established (2) will follow. Thus it remains to prove (1), i.e., we check that the relation

$$\{(t, t') \in \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma | t \sqsubseteq^{kl}_\sigma t'\}$$

satisfies the simulation conditions in Figure 7.2. Notice that (sim 1) holds vacuously. To prove (sim 2), suppose that $p \sqsubseteq^{kl}_{\sigma\times\tau} p'$. Assume that $\mathrm{fst}(p) \Downarrow v$. It then follows from ($\Downarrow$ fst) that the premise is $p \Downarrow (s,t)$ for some closed terms $s:\sigma, t:\tau$ and $s \Downarrow v$. But $p \sqsubseteq^{kl}_{\sigma\times\tau} p'$ so that $p' \Downarrow (s,t)$ and thus $\mathrm{fst}(p') \Downarrow v$, as desired. Similarly, one can deduce that $\mathrm{snd}(p) \sqsubseteq^{kl}_{\tau} \mathrm{snd}(p')$ and hence (sim 2) holds.

To prove that (sim 3a) holds, suppose that $s \sqsubseteq^{kl}_{\sigma+\tau} s'$ and $s \Downarrow \mathrm{inl}(a)$. We want to show that there exists $a' \in \mathrm{Exp}_\sigma$ such that $s' \Downarrow \mathrm{inl}(a')$ and $a \sqsubseteq^{kl}_{\sigma} a'$. Because $s \sqsubseteq^{kl}_{\sigma+\tau} s'$ and $s \Downarrow \mathrm{inl}(a)$, it follows that $s' \Downarrow \mathrm{inl}(a)$. Moreover we always have $a \sqsubseteq^{kl}_{\sigma} a$. Thus (sim 3a) holds trivially. Likewise, (sim 3b) holds.

To prove that (sim 4) holds, suppose that $t \sqsubseteq^{kl}_{\sigma_\perp} t'$ and $t \Downarrow \mathrm{up}(s)$ for some $s:\sigma$. Because $t \sqsubseteq^{kl}_{\sigma_\perp} t'$, it follows that $t' \Downarrow \mathrm{up}(s)$. Moreover, $s \sqsubseteq^{kl}_{\sigma} s$ holds trivially so that (sim 4) holds.

To verify that (sim 5) holds, suppose that $t \sqsubseteq^{kl}_{\mu X.\sigma} t'$. Assume that $\mathrm{unfold}(t) \Downarrow v$. Then from ($\Downarrow$ unfold) it must be that $t \Downarrow \mathrm{fold}(s) \wedge s \Downarrow v$. Since $t \sqsubseteq^{kl}_{\mu X.\sigma} t'$, it follows that $t' \Downarrow \mathrm{fold}(s)$. Consequently, it again follows from ($\Downarrow$ unfold) that $\mathrm{unfold}(t') \Downarrow v$. Thus, $\mathrm{unfold}(t) \sqsubseteq^{kl}_{\mu X.\sigma} \mathrm{unfold}(t')$ and (sim 5) is satisfied.

Finally we verify that (sim 6) holds. Suppose that $f \sqsubseteq^{kl}_{\sigma\to\tau} f'$ and let $t \in \mathrm{Exp}_\sigma$. Assume that $f(t) \Downarrow v$. Then by ($\Downarrow$ app) there exists s such that $f \Downarrow \lambda x.s$ and $s[t/x] \Downarrow v$. Since $f \sqsubseteq^{kl}_{\sigma\to\tau} f'$, it follows that $f' \Downarrow \lambda x.s$. So $f'(t) \Downarrow v$ by ($\Downarrow$ app) again and hence $f(t) \sqsubseteq^{kl}_{\tau} f'(t)$. Thus (sim 6) holds. $\square$

Now we apply Theorem 7.5.1 to establish the β-equalities (7.10) - (7.15) and property (7.29). The following Kleene equivalences all follow immediately from evaluation rules for FPC (suppressing type information).

$$\begin{aligned}
(\lambda x.s)(t) &\cong^{kl} s[t/x] \\
\mathrm{fst}(s,t) &\cong^{kl} s \\
\mathrm{snd}(s,t) &\cong^{kl} t \\
\mathrm{case}(\mathrm{inl}(t)) \text{ of } \mathrm{inl}(x).s \text{ or } \mathrm{inr}(y).s' &\cong^{kl} s[t/x] \\
\mathrm{case}(\mathrm{inr}(t)) \text{ of } \mathrm{inl}(x).s \text{ or } \mathrm{inr}(y).s' &\cong^{kl} s'[t/y] \\
\mathrm{case}(\mathrm{up}(t)) \text{ of } \mathrm{up}(x).y &\cong^{kl} y[t/x] \\
\mathrm{unfold}(\mathrm{fold}(t)) &\cong^{kl} t
\end{aligned}$$

Thus by the above proposition, these are also valid for FPC similarity. So by Theorem 7.4.4, these are also valid contextual equivalences which are precisely the β-equalities (7.10)-(7.15) stated in Section 7.1.2.

To prove that $\perp_\sigma := \mathrm{fix}_\sigma(\lambda x^\sigma.x)$ is the least element of Exp_σ with respect

to the contextual preorder (i.e., property (7.29)) we reason as follows. First of all, we observe that the term $\mathrm{unfold}^\tau(\mathrm{fold}^\tau(k))\mathrm{fold}^\tau(k)$ does not evaluate to any canonical value. We prove this by contradiction. Suppose not, i.e., there is a minimal derivation of

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

for some canonical value v. Because k is an abstraction, $k \Downarrow k$. Thus by the ($\Downarrow$ unfold) rule that

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

From the above evaluation, the derivation of $\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k) \Downarrow v$ must be:

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

which contradicts the minimality of the derivation $\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k) \Downarrow v$.

Thus by the definition of Kleene preorder, we conclude that the term $\mathrm{unfold}(\mathrm{fold}(k))\mathrm{fold}(k)$ is the least element of Exp_σ with respect to the contextual preorder. It then follows from the β-equality (7.10) that

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

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

7.6 Continuity of evaluation

Recall that in Section 7.1.7 we have made the following definition. For every $f : \sigma \to \sigma$ and $n \in \mathbb{N}$, we have defined $f^{(n)}(\bot_\sigma)$ as follows:

$$\begin{array}{rcl} f^{(0)}(\bot_\sigma) & = & \bot_\sigma \\ f^{(n+1)}(\bot_\sigma) & = & f(f^{(n)}(\bot_\sigma))) \end{array}$$

Since $\bot_\sigma$ is the least element of σ and application is monotone, we obtain an ascending chain in Exp_σ:

$$\bot_\sigma = f(\bot_\sigma) \sqsubseteq f^{(2)}(\bot_\sigma) \sqsubseteq \dots$$

In this section, we prove that $\mathrm{fix}(f)$ is the contextual supremum of this rational chain, i.e.,

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

Also we establish the rational continuity property, i.e., for every $g \in \sigma \to \tau$ and $f : \sigma \to \sigma$,

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

To further simplify the writing, we employ the notation:

$$f_n := f^{(n)}(\bot_\sigma) \text{ and } f_\infty := \mathrm{fix}(f).$$

Throughout this section, we consider only FPC contexts involving parameters of type σ. As usual, we write $C[\vec{\mathrm{p}}]$ to mean a context whose parameters are contained in the list $\vec{\mathrm{p}} = \mathrm{p}_1 \dots, \mathrm{p}_k$ of pairwise distinct parameters. Given a k-tuple of natural numbers $\vec{n} := (n_1, \dots, n_k)$ we use the abbreviations:

$$C[f_{\vec{n}}] := C[f_{n_1}, \dots, f_{n_k}] \text{ and } C[f_\infty] := C[f_\infty, \dots, f_\infty].$$

The *length* of a list $\vec{\mathrm{p}}$ of parameters is denoted by $|\vec{\mathrm{p}}|$.

For each $k \in \mathbb{N}$, we can order the set $\mathbb{N}^k$ componentwise from the usual ordering on $\mathbb{N}$:

$$\vec{m} \leq \vec{n} \iff \forall i = 1, \dots, k. m_i \leq n_i.$$

A subset $I \subseteq \mathbb{N}^k$ is *cofinal* if for every $\vec{m} \in \mathbb{N}^k$ there is always $\vec{n} \in I$ such that $\vec{m} \leq \vec{n}$. We write $\mathcal{P}_{\mathrm{cof}}(\mathbb{N}^k)$ for the set of all cofinal subsets of $\mathbb{N}^k$.

One can easily show by induction on n, using (7.29) and (7.8), that

$$f_n \sqsubseteq_\sigma f_{n+1} \text{ and } f_n \sqsubseteq_\sigma f_\infty.$$

Consequently, for any unary context $C[\mathrm{p}]$, the ascending chain

$$C[f_0] \sqsubseteq C[f_1] \sqsubseteq C[f_2] \sqsubseteq \dots$$

is bounded above by $C[f_\infty]$. We want to show that $C[f_\infty]$ is the supremum of this chain. More generally, if C involves several parameters $\vec{\mathrm{p}}$, then for any $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, $C[f_\infty]$ will be the least upper bound of the set $\{C[f_{\vec{n}}] | \vec{n} \in I\}$.

A context V is a *value context* if it is generated by the grammar:

$$(C_1, C_2) \mid \mathrm{inl}(C) \mid \mathrm{inr}(C) \mid \mathrm{up}(C) \mid \mathrm{fold}(C) \mid \lambda x.C$$

where C ranges over contexts.

We now define *evaluation of contexts* modulo f. Given an FPC context $C[\vec{\mathrm{p}}]$ and a value context $V[\vec{\mathrm{q}}]$, we write $C[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{q}}]$ to mean that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$,

$$\{\vec{m}\vec{n} | \vec{m} \in I \wedge C[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\} \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}).$$

The relation $C[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{p}'}]$ is preserved under renaming of the parameters $\vec{\mathrm{p}}$ and, independently, the parameters $\vec{\mathrm{p}'}$. As the following lemma shows, the relation is also preserved under addition or subtraction of extra parameters.

Lemma 7.6.1.

$$C[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{p}'}] \iff C[\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{p}'}\vec{\mathrm{q}'}].$$

Proof. This follows from elementary properties of cofinal sets which we explain below.
($\Rightarrow$): Let $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|})$. We want to show that the set

$$A_1 := \{\vec{m_1}\vec{m_2}\vec{n_1}\vec{n_2} | \vec{m_1}\vec{m_2} \in I \wedge C[f_{\vec{m_1}\vec{m_2}}] \Downarrow V[f_{\vec{n_1}\vec{n_2}}]\}$$

is cofinal in $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|+|\vec{\mathrm{p}'}|+|\vec{\mathrm{q}'}|}$. Because $\vec{\mathrm{q}}$ are extra parameters for the context C, we have that $C[f_{\vec{m_1}}] \equiv C[f_{\vec{m_1}\vec{m_2}}]$ for all $\vec{m_1}, \vec{m_2}$. Similarly, as $\vec{\mathrm{q}'}$ are extra parameters for the context V, $V[f_{\vec{n_1}}] \equiv V[f_{\vec{n_1}\vec{n_2}}]$ for all $\vec{n_1}, \vec{n_2}$. Thus, the above set can be rewritten as

$$A_1 := \{\vec{m_1}\vec{m_2}\vec{n_1}\vec{n_2} | \vec{m_1}\vec{m_2} \in I \wedge C[f_{\vec{m_1}}] \Downarrow V[f_{\vec{n_1}}]\}$$

Because $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{p}'}|})$, the set

$$I' := \{\vec{m_1} | \exists \vec{m_2}.\vec{m_1}\vec{m_2} \in I\}.$$

is a cofinal subset of $\mathbb{N}^{|\vec{p}|}$. With this set I', we form the set

$$A_2 := \{\vec{m_1}\vec{n_1} | \vec{m_1} \in I' \wedge C[f_{\vec{m_1}}] \Downarrow V[f_{\vec{n_1}}]\}.$$

Since $C[\vec{\mathrm{p}}] \Downarrow V[\vec{\mathrm{p}'}]$, it follows that A_2 is a cofinal subset of $\mathbb{N}^{|\vec{p}|+|\vec{\mathrm{p}'}|}$. Using the

cofinal set A_2, we now form the set

$$A_3 := \{\vec{m_1}\vec{m_2}\vec{n_1}\vec{n_2} | \vec{m_1}\vec{n_1} \in A_3 \wedge \vec{m_1}\vec{m_2} \in I \wedge \vec{n_2} \in \mathbb{N}^{|\vec{q'}|}\}.$$

Since A_3, I and $\mathbb{N}^{\vec{q'}}$ are cofinal, it is clear that A_3 is a cofinal subset of $\mathbb{N}^{|\vec{p}|+|\vec{p'}|+|\vec{q}|+|\vec{q'}|}$. Finally observe that by definition $A_1 = A_3$ and the cofinality of A_1 is established.

($\Leftarrow$): Let $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{p}|})$. We want to show that the set

$$B_1 := \{\vec{m_1}\vec{n_1} | \vec{m_1} \in I \wedge C[f_{\vec{m_1}}] \Downarrow V[f_{\vec{n_1}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{p}|+|\vec{p'}|}$. First, form the following cofinal subset of $\mathbb{N}^{|\vec{p}|+|\vec{q}|}$:

$$I' := \{\vec{m_1}\vec{m_2} | \vec{m_1} \in I \wedge \vec{m_2} \in \mathbb{N}^{|\vec{q}|}\}.$$

Then, since $C[\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow V[\vec{\mathrm{p}'}\vec{\mathrm{q}'}]$, it follows that the set

$$B_2 := \{\vec{m_1}\vec{m_2}\vec{n_1}\vec{n_2} | \vec{m_1}\vec{m_2} \in I' \wedge C[f_{\vec{m_1}\vec{m_2}}] \Downarrow V[f_{\vec{n_1}\vec{n_2}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{p}|+|\vec{q}|+|\vec{p'}|+|\,\mathrm{q}'\,|}$. As in the above argument, because $\vec{\mathrm{q}}$ and $\vec{\mathrm{q}'}$ are extra parameters, we have that $C[f_{\vec{m_1}\vec{m_2}}] \equiv C[f_{\vec{m_1}}]$ and $V[f_{\vec{n_1}\vec{n_2}}] \equiv V[f_{\vec{n_1}}]$ for all $\vec{m_1}, \vec{m_2}, \vec{n_1}$ and $\vec{n_2}$. Thus the set B_2 can be rewritten as

$$B_3 := \{\vec{m_1}\vec{m_2}\vec{n_1}\vec{n_2} | \vec{m_1} \in I \wedge \vec{m_2} \in \mathbb{N}^{|\vec{q}|} \wedge C[f_{\vec{m_1}}] \Downarrow V[f_{\vec{n_1}}] \wedge \vec{n_2} \in \mathbb{N}^{|\vec{q'}|}\}.$$

The cofinality of B_3 easily implies that of B_1, and the desired result follows. □

Lemma 7.6.2. *The relation $\Downarrow^f$ satisfies the following analogues of the axioms and rules for FPC evaluation given in Figure 4.3:*

(1) If $V[\vec{\mathrm{p}}]$ is a value context, then $V[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{p}}]$.

(2) If $S[\vec{\mathrm{p}}] \Downarrow^f (\lambda x.S')[\vec{\mathrm{q}}]$ and $S'[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, then $ST[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(3) If $P[\vec{\mathrm{p}}] \Downarrow^f (S,T)[\vec{\mathrm{q}}]$ and $S[\vec{q}] \Downarrow^f V[\vec{\mathrm{r}}]$, then $\mathrm{fst}(P)[\vec{p}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(4) If $P[\vec{\mathrm{p}}] \Downarrow^f (S,T)[\vec{\mathrm{q}}]$ and $T[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, then $\mathrm{snd}(P)[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(5) If $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{fold}(T)[\vec{\mathrm{q}}]$ and $T[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, then $\mathrm{unfold}(S)[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(6) If $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{up}(T')[\vec{\mathrm{q}}]$ and $T[T'/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, then $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(7) *If* $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{inl}(T)[\vec{\mathrm{q}}]$ *and* $T_1[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow V[\vec{\mathrm{r}}]$, *then* $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(8) *If* $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{inr}(T)[\vec{\mathrm{q}}]$ *and* $T_2[T/y][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow V[\vec{\mathrm{r}}]$, *then* $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

Proof. Each property follows from combining the corresponding evaluation rule in Figure 4.3 with the definition of $\Downarrow^f$.

(1) Since $V[\vec{\mathrm{p}}]$ is a value context, it follows that for all $\vec{m} \in \mathbb{N}^{|\vec{\mathrm{p}}|}$, we have $V[f_{\vec{m}}] \Downarrow V[f_{\vec{m}}]$. Therefore, for every $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, the set

$$\{\vec{m}\vec{n} | \vec{m} \in I \wedge V[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\}$$

which is equal to $\{\vec{m}\vec{m} | \vec{m} \in I\}$ must be cofinal.

(2) We must show that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, the set

$$K := \{\vec{m}\vec{n} | \vec{m} \in I \wedge ST[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. Given that $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$ and since $S[\vec{\mathrm{p}}] \Downarrow^f \lambda x.S'[\vec{\mathrm{q}}]$, it follows that the set

$$I' := \{\vec{m}\vec{o} | \vec{m} \in I \wedge S[f_{\vec{m}}] \Downarrow (\lambda x.S')[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}$. Also since $S'[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$ it follows that the following set

$$I'' := \{\vec{m}\vec{o}\vec{n} | \vec{m}\vec{o} \in I' \wedge S'[T/x][f_{\vec{m}\vec{o}}] \Downarrow V[f_{\vec{n}}]\}$$

is cofinal in $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|+|\vec{\mathrm{r}}|}$. Because of the cofinality of I'', the set

$$K' := \{\vec{m}\vec{n} | \exists \vec{o}.\vec{m}\vec{o}\vec{n} \in I''\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. If we can show that $K' \subseteq K$, then we are done. To this end, let $\vec{m}\vec{n} \in K'$. This means that $\vec{m}\vec{o} \in I'$ and $S'[T/x][f_{\vec{m}\vec{o}}] \Downarrow V[f_{\vec{n}}]$. Since $\vec{m}\vec{o} \in I'$, we have $\vec{m} \in I$. Moreover, it holds that $S[f_{\vec{m}}] \Downarrow (\lambda x.S')[f_{\vec{o}}]$. Finally, by the ($\Downarrow$ app) rule, it follows that $ST[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]$. Thus $\vec{m}\vec{n} \in K$, as required.

(3) We must prove that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, the set

$$K := \{\vec{m}\vec{n} | \vec{m} \in I \wedge \mathrm{fst}(P)[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. To do this, suppose we are given $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$. Since $P[\vec{\mathrm{p}}] \Downarrow^f (S,T)[\vec{\mathrm{q}}]$, it follows that

$$I' := \{\vec{m}\vec{o} | \vec{m} \in I \wedge P[f_{\vec{m}}] \Downarrow (S,T)[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}$. This yields another cofinal set

$$I'' := \{\vec{o} | \exists \vec{m}.\vec{m}\vec{o} \in I'\}.$$

Because $S[\vec{\mathrm{q}}] \Downarrow V[\vec{\mathrm{r}}]$, the set

$$I''' := \{\vec{o}\vec{n} | \vec{o} \in I'' \wedge S[f_{\vec{o}}] \Downarrow V[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{q}}|+|\vec{\mathrm{r}}|}$. Since I' and I''' are cofinal subsets, so is the set

$$K' := \{\vec{m}\vec{n} | \exists o.\vec{m}\vec{o} \in I' \wedge \vec{o}\vec{n} \in I'''\}.$$

To prove that K is cofinal, it suffices to show that $K' \subseteq K$. Let $\vec{m}\vec{n} \in K'$. By definitions of I', I'' and I''', there is $\vec{o}$ such that $\vec{m} \in I$, $P[f_{\vec{m}}] \Downarrow (S,T)[f_{\vec{o}}]$ and $S[f_{\vec{o}}] \Downarrow V[f_{\vec{n}}]$. One then invokes the evaluation rule ($\Downarrow$ fst) to conclude that $\mathrm{fst}(P)[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]$.

(4) Similar to (3).

(5) Suppose $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{fold}(T)[\vec{\mathrm{q}}]$ and $T[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. In order to verify that $\mathrm{unfold}(S)[\vec{\mathrm{p}}] \Downarrow V[\vec{\mathrm{r}}]$, we have to show that for any $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$,

$$K := \{\vec{m}\vec{o} | \vec{m} \in I \wedge \mathrm{unfold}(S)[f_{\vec{m}}] \Downarrow V[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. But given such an I, by the definition of $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{fold}(T)[\vec{\mathrm{q}}]$,

$$I' := \{\vec{m}\vec{n} | \vec{m} \in I \wedge S[f_{\vec{m}}] \Downarrow \mathrm{fold}(T)[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}$ and hence

$$I'' := \{\vec{n} | \exists \vec{m}.(\vec{m}\vec{n} \in I')\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{q}}|}$. Since $T[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, it follows that

$$I''' := \{\vec{n}\vec{o} | \vec{n} \in I'' \wedge T[f_{\vec{n}}] \Downarrow V[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{q}}|+|\vec{\mathrm{r}}|}$. Notice that the set

$$K' = \{\vec{m}\vec{o} | \exists \vec{n}.\vec{m}\vec{n} \in I' \wedge \vec{n}\vec{o} \in I'''\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. We now show that K' is a subset of K. If $\vec{m}\vec{o} \in K'$, then $\vec{m} \in I$ and there is $\vec{n}$ such that $S[f_{\vec{m}}] \Downarrow \mathrm{fold}(T)[f_{\vec{n}}]$ and $T[f_{\vec{n}}] \Downarrow V[f_{\vec{o}}]$. By the evaluation rule ($\Downarrow$ unfold), it holds that $\mathrm{unfold}(S)[f_{\vec{m}}] \Downarrow V[f_{\vec{o}}]$. Hence $K' \subseteq K$ and thus K is also a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$.

(6) We must show that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, the set

$$K := \{\vec{m}\vec{n} | \vec{m} \in I \wedge \mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. Given that I is cofinal and since $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{up}(T')[\vec{\mathrm{q}}]$ holds, it follows that

$$I' := \{\vec{m}\vec{n} | \vec{m} \in I \wedge S[f_{\vec{m}}] \Downarrow \mathrm{up}(T')[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}$. Because $T[T'/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, the set

$$I'' := \{\vec{m}\vec{n}\vec{o} | \vec{m}\vec{n} \in I' \wedge T[T'/x][f_{\vec{m}\vec{n}}] \Downarrow V[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|+|\vec{\mathrm{r}}|}$. Thus we can form the following cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$:

$$K' := \{\vec{m}\vec{n} | \exists \vec{o}.\vec{m}\vec{n}\vec{o} \in I''\}.$$

If we can show that $K' \subseteq K$, then the cofinality of K' will imply that of K. To this end, we take any $\vec{m}\vec{n} \in K'$. By definition, there is $\vec{o}$ such that $\vec{m}\vec{n}\vec{o} \in I''$. Thus $\vec{m}\vec{n} \in I'$ and $T[T'/x][f_{\vec{m}\vec{n}}] \Downarrow V[f_{\vec{o}}]$. Since $\vec{m}\vec{n} \in I'$, we have $\vec{m} \in I$ and $S[f_{\vec{m}}] \Downarrow \mathrm{up}(T')[\vec{n}]$. Using the ($\Downarrow$ case-up) rule, it follows that $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]$. Thus $\vec{m}\vec{n} \in K$.

(7) We show that for all $I \in \mathcal{P}_{\mathrm{cof}}(\mathbb{N}^{|\vec{\mathrm{p}}|})$, the set

$$K := \{\vec{m}\vec{n} | \vec{m} \in I \wedge \mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$. Given that I is cofinal and since $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{inl}(T_1)[\vec{\mathrm{q}}]$ holds, it follows that

$$I' := \{\vec{m}\vec{n} | \vec{m} \in I \wedge S[f_{\vec{m}}] \Downarrow \mathrm{inl}(T_1)[f_{\vec{n}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|}$. Because $T_1[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$, the set

$$I'' := \{\vec{m}\vec{n}\vec{o} | \vec{m}\vec{n} \in I' \wedge T_1[T/x][f_{\vec{m}\vec{n}}] \Downarrow V[f_{\vec{o}}]\}$$

is a cofinal subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{q}}|+|\vec{\mathrm{r}}|}$. Thus we can form the following cofinal

subset of $\mathbb{N}^{|\vec{\mathrm{p}}|+|\vec{\mathrm{r}}|}$:

$$K' := \{\vec{m}\vec{n} | \exists \vec{o}.\vec{m}\vec{n}\vec{o} \in I''\}.$$

To show that K is cofinal, it suffices to verify that $K' \subseteq K$. To do this, let $\vec{m}\vec{n} \in K'$ be arbitrary. By definition, there is $\vec{o}$ such that $\vec{m}\vec{n}\vec{o} \in I''$. Thus $\vec{m}\vec{n} \in I'$ and $T_1[T/x][f_{\vec{m}\vec{n}}] \Downarrow V[f_{\vec{o}}]$. Since $\vec{m}\vec{n} \in I'$, we have $\vec{m} \in I$ and $S[f_{\vec{m}}] \Downarrow \mathrm{inl}(T)[\vec{n}]$. Using the ($\Downarrow$ case inl) rule, it follows that $\mathrm{case}(S)$ of $\mathrm{inl}(x).T_1$ or $\mathrm{inr}(y).T_2[f_{\vec{m}}] \Downarrow V[f_{\vec{n}}]$. Thus $\vec{m}\vec{n} \in K$.

(8) Similar to (7).

□

Proposition 7.6.3. *For all FPC contexts $C[\vec{\mathrm{p}}]$, if $C[f_\infty] \Downarrow v$, then there is a value context $V[\vec{\mathrm{q}}]$ with $v = V[f_\infty]$ and $C[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{q}}]$.*

Proof. The proof proceeds by induction on the structure of derivation of $C[f_\infty] \Downarrow v$.

(1) ($\Downarrow$ can)
Suppose $C[f_\infty]$ is a canonical value. Since f_∞ is not a canonical value, the context C itself must be a value context, and hence by Lemma 7.6.2(1), $C[\vec{\mathrm{p}}] \Downarrow^f C[\vec{\mathrm{p}}]$.

(2) ($\Downarrow$ app)
Suppose $C[\vec{\mathrm{p}}] \equiv ST[\vec{\mathrm{p}}]$ and $ST[\vec{f_\infty}] \Downarrow v$. It follows from the evaluation rule ($\Downarrow$ app) that the premise consists of

$$S[f_\infty] \Downarrow \lambda x.s' \text{ and } s'[T[f_\infty]/x] \Downarrow v$$

for some term s'. The induction hypothesis then asserts that there is a value context $S'[\vec{\mathrm{q}}]$ such that $\lambda x.s' = \lambda x.S'[f_\infty]$ and $S[\vec{\mathrm{p}}] \Downarrow^f \lambda x.S'[\vec{\mathrm{q}}]$. Thus $S'[T/x][f_\infty] \Downarrow v$. By the induction hypothesis again, there is a value context $V[\vec{\mathrm{r}}]$ with $v = V[f_\infty]$ and $S'[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. Hence by Lemma 7.6.2(2) we have $ST[\vec{\mathrm{p}}] \Downarrow^f V[\mathrm{r}]$.

(3) ($\Downarrow$ fst, snd)
Suppose $C[\vec{\mathrm{p}}] \equiv \mathrm{fst}(P)[\vec{\mathrm{p}}]$ and $\mathrm{fst}(P)[f_\infty] \Downarrow v$. By the ($\Downarrow$ fst) rule, the premise consists of

$$P[f_\infty] \Downarrow (s,t) \text{ and } s \Downarrow v$$

The induction hypothesis asserts that there is a value context $(S,T)[\vec{\mathrm{q}}]$ with $(s,t) = (S,T)[f_\infty]$ and $P[\vec{\mathrm{p}}] \Downarrow^f (S,T)[\vec{\mathrm{q}}]$. Thus $S[f_\infty] \Downarrow v$ and by the induction hypothesis again, there is a value context $V[\vec{\mathrm{r}}]$ with $v =$

$V[f_\infty]$ and $S[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. Hence by Lemma 7.6.2(3) we have $\mathrm{fst}(P)[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{q}}]$.

(4) ($\Downarrow$ unfold)
Suppose $C[\vec{\mathrm{p}}] \equiv \mathrm{unfold}(S)[\vec{\mathrm{p}}]$ and $\mathrm{unfold}(S)[f_\infty] \Downarrow v$. The premise of this must be

$$S[f_\infty] \Downarrow \mathrm{fold}(t) \text{ and } t \Downarrow v.$$

Invoking the induction hypothesis, there exists a value context $\mathrm{fold}(T)[\vec{\mathrm{q}}]$ with $\mathrm{fold}(t) = \mathrm{fold}(T)[f_\infty]$ and $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{fold}(T)[\vec{\mathrm{q}}]$. Thus $T[f_\infty] \Downarrow v$. Again by the induction hypothesis there exists a value context $V[\vec{\mathrm{r}}]$ with $v = V[f_\infty]$ and $T[\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. Thus, by Lemma 7.6.2(5) it follows that $\mathrm{unfold}(S)[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{q}}]$.

(5) ($\Downarrow$ case-up)
Suppose $C[\vec{\mathrm{p}}] \equiv \mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[\vec{\mathrm{p}}]$ and $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[f_\infty] \Downarrow v$. It follows from the evaluation rule ($\Downarrow$ case-up) that the premise consists of

$$S[f_\infty] \Downarrow \mathrm{up}(t) \text{ and } \mathrm{up}(t) \Downarrow v.$$

The induction hypothesis asserts that there is a value context $\mathrm{up}(T)[\vec{\mathrm{q}}]$ with $\mathrm{up}(t) = \mathrm{up}(T)[f_\infty]$ and $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{up}(T)[\vec{\mathrm{q}}]$. Thus $T[T'/x][f_\infty] \Downarrow v$. The induction hypothesis then asserts that there is a value context $V[\vec{\mathrm{r}}]$ with $v = V[f_\infty]$ and $T[T'/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. It then follows from Lemma 7.6.2 that $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{up}(x).T[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

(6) ($\Downarrow$ case inl, inr)
Suppose $C[\vec{\mathrm{p}}] \equiv \mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[\vec{\mathrm{p}}]$ and $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[f_\infty] \Downarrow v$. One possibility is via the evaluation rule ($\Downarrow$ case inl) where the premise is

$$S[f_\infty] \Downarrow \mathrm{inl}(t) \text{ and } \mathrm{inl}(t) \Downarrow v.$$

The induction hypothesis asserts that there is a value context $\mathrm{inl}(T)[\vec{\mathrm{q}}]$ with $\mathrm{inl}(t) = \mathrm{inl}(T)[f_\infty]$ and $S[\vec{\mathrm{p}}] \Downarrow^f \mathrm{inl}(T)[\vec{\mathrm{q}}]$. Thus $T_1[T/x][f_\infty] \Downarrow v$. The induction hypothesis then asserts that there is a value context $V[\vec{\mathrm{r}}]$ with $v = V[f_\infty]$ and $T_1[T/x][\vec{\mathrm{p}}\vec{\mathrm{q}}] \Downarrow^f V[\vec{\mathrm{r}}]$. It then follows from Lemma 7.6.2 that $\mathrm{case}(S)\ \mathrm{of}\ \mathrm{inl}(x).T_1\ \mathrm{or}\ \mathrm{inr}(y).T_2[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{r}}]$.

The other possibility is via the evaluation rule ($\Downarrow$ case inr) where the premise is

$$S[f_\infty] \Downarrow \mathrm{inr}(t) \text{ and } \mathrm{inr}(t) \Downarrow v.$$

Here the argument is similar and we omit the proof.

The proof by induction is thus complete. □

Corollary 7.6.4. *For any FPC context $C[\mathrm{p}]$ of type Σ, if $C[\mathrm{fix}(f)] \Downarrow \top$ then $C[f^{(n)}(\bot_\sigma)] \Downarrow \top$ for some $n \in \mathbb{N}$.*

Proof. Suppose that $C[f_\infty] \Downarrow \top$. Because $\top$ is a canonical value, it follows from Proposition 7.6.3 that there is a context $V[\vec{\mathrm{q}}]$ such that $C[\vec{\mathrm{p}}] \Downarrow^f V[\vec{\mathrm{q}}]$ and $\top = V[f_\infty]$. Since $\top$ is a canonical value, it must be that $V[\vec{\mathrm{q}}] = \top[\,]$ and hence $C[\vec{\mathrm{p}}] \Downarrow^f \top[\,]$. Taking $I = \mathbb{N}$ in the definition of $\Downarrow^f$, we have that $\{n \in \mathbb{N} | C[f_n] \Downarrow \top\}$ is a cofinal subset of $\mathbb{N}$. In particular this set is inhabited and thus there exists some $n \in \mathbb{N}$ with $C[f_n] \Downarrow \top$, as required. □

Remark 7.6.5. The above phenomenon of 'continuity of evaluation' used to be labelled as 'compactness of evaluation' in Corollary 4.6 of Pitts [41]. However, we feel that continuity is a more appropriate topological notion to describe this property.

We can now complete the proof of the rational-chain completeness and continuity for FPC.

Theorem 7.6.6. *For any $f : \sigma \to \sigma$, it holds that*

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

In general, for any context $C[-_\sigma] \in \mathrm{Ctx}_\tau$, we have

$$C[\mathrm{fix}(f)] =_\tau \bigsqcup_n C[\mathrm{fix}^n(f)].$$

Proof. We prove the later statement and deduce the first one as a special case by taking $C[-_\sigma] = -_\sigma$. To achieve this, we prove by induction on n that

$$f^{(n)}(\bot_\sigma) \sqsubseteq_\sigma \mathrm{fix}(f).$$

Base case: Trivial since $\bot_\sigma$ is the least element of Exp_σ with respect to the contextual preorder, i.e., property (7.29).
Inductive step: Since $f^{(n)}(\bot_\sigma) \sqsubseteq_\sigma \mathrm{fix}(f)$, it follows from monotonicity, that $f(f^{(n)}(\bot_\sigma)) \sqsubseteq_\sigma f(\mathrm{fix}(f))$. Because $f(\mathrm{fix}(f)) =_\sigma \mathrm{fix}(f)$ by property (7.28), we conclude that $f^{(n+1)}(\bot_\sigma) \sqsubseteq_\sigma \mathrm{fix}(f)$, which completes the proof by induction.

It then follows from this that for any $C[-_\sigma] \in \mathrm{Ctx}_\tau$ and for all $n \in \mathbb{N}$,

$$C[f^{(n)}(\bot_\sigma)] \sqsubseteq_\tau C[\mathrm{fix}(f)].$$

Now suppose $x : \tau$ is such that $C[f^{(n)}(\bot_\sigma)] \sqsubseteq_\tau x$ for all $n \in \mathbb{N}$. If $N[-_\tau] \in \mathrm{Ctx}_\Sigma$ is any context with $N[C[\mathrm{fix}(f)]] \Downarrow \top$, then by applying Corollary 7.6.4 to the context $N[C[-]]$, there exists some $n \in \mathbb{N}$ so that already $N[C[f^{(n)}(\bot_\sigma)]] \Downarrow \top$. Since $C[f^{(n)}(\bot_\sigma)] \sqsubseteq_\tau x$, it holds that $N[x] \Downarrow \top$. Hence $C[\mathrm{fix}(f)] \sqsubseteq_\tau x$, as required. $\square$

Chapter 8

Operational extensionality theorem

In this chapter, we prove the operational extensionality Theorem 7.4.4. This again follows Pitts' work [41], but with some re-organisation. This proof comprises of two parts:

(a) We prove that the open extension $\preceq^\circ$ of similarity is an FPC precongruence. Using this fact, we show that the open similarity $\preceq^\circ$ is contained in the contextual preorder $\sqsubseteq$, i.e.,

$$\Gamma \vdash t \preceq^\circ_\sigma t' \implies \Gamma \vdash t \sqsubseteq_\sigma t'.$$

(b) We prove that the contextual preorder $\sqsubseteq$, when restricted to closed terms, is an FPC simulation. Using this fact, we then prove that the contextual preorder $\sqsubseteq$ is contained in the open similarity $\preceq^\circ$, i.e.,

$$\Gamma \vdash t \sqsubseteq_\sigma t' \implies \Gamma \vdash t \preceq^\circ_\sigma t'.$$

This chapter is organised in the following way:

1. In Section 8.1, we define the notions of FPC precongruence and congruence.
2. In Section 8.2, we define an auxiliary relation $\preceq^*$ in terms of $\preceq^\circ$ and establish that $\preceq^\circ$ and $\preceq^*$ are equivalent.
3. Exploiting this equivalence, we then prove in Section 8.3 that the open similarity $\preceq^\circ$ is an FPC precongruence and as a consequence, it is contained in the contextual preorder. This completes part (a).

4. Finally, we verify in Section 8.4 that the contextual preorder, when restricted to closed terms, is an FPC simulation and consequently, by the co-induction principle, is contained in the FPC similarity. Finally, we complete the proof of part (b) by showing how this implication can be extended from the closed to open terms.

Note that we make use of an adaptation of Howe's method (cf. [29, 30]) in items (2) and (3).

8.1 Precongruence and congruence

Suppose $\mathcal{R}$ is a family of binary relations $\mathcal{R}_{\Gamma,\sigma} \subseteq \mathrm{Exp}_\sigma(\Gamma) \times \mathrm{Exp}_\sigma(\Gamma)$, indexed by variable typings Γ and closed types σ. We write $\Gamma \vdash t\ \mathcal{R}_\sigma\ t'$ to mean a pair of terms (t, t') is in the relation $\mathcal{R}_{\Gamma,\sigma}$.

$\mathcal{R}$ is an *FPC precongruence relation* if it satisfies the following conditions.

$$(\Gamma \vdash t\ \mathcal{R}_\sigma\ t' \wedge \Gamma \subseteq \Gamma') \implies \Gamma' \vdash t\ \mathcal{R}_\sigma\ t' \tag{A.1}$$

$$(\Gamma \vdash t : \sigma \wedge \Gamma, x : \sigma \vdash s\ \mathcal{R}_\tau\ s') \implies \Gamma \vdash s[t/x]\ \mathcal{R}_\tau\ s'[t/x] \tag{A.2}$$

$$\Gamma \vdash t : \sigma \implies \Gamma \vdash t\ \mathcal{R}_\sigma\ t \tag{A.3}$$

$$(\Gamma \vdash t\ \mathcal{R}_\sigma\ t' \wedge \Gamma \vdash t'\ \mathcal{R}_\sigma\ t'') \implies \Gamma \vdash t\ \mathcal{R}_\sigma\ t'' \tag{A.4}$$

$$(\Gamma \vdash t\ \mathcal{R}_\sigma\ t' \wedge \Gamma, x : \sigma \vdash s : \tau) \implies \Gamma \vdash s[t/x]\ \mathcal{R}_\tau\ s[t'/x] \tag{A.5}$$

$$\Gamma, x : \sigma \vdash t\ \mathcal{R}_\tau\ t' \implies \Gamma \vdash \lambda x.t\ \mathcal{R}_{\sigma\to\tau}\ \lambda x.t' \tag{A.6}$$

$$(\Gamma \vdash s\ \mathcal{R}_{\sigma+\tau}\ s' \wedge \Gamma, x : \sigma \vdash t_1\ \mathcal{R}_\rho\ t_1' \wedge \Gamma, y : \tau \vdash t_2\ \mathcal{R}_\rho\ t_2' \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2\ \mathcal{R}_\rho\ \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t_1'\ \mathrm{or}\ \mathrm{inr}(y).t_2' \tag{A.7}$$

$$(\Gamma \vdash s\ \mathcal{R}_{\sigma_\perp}\ s' \wedge \Gamma, x : \sigma \vdash t\ \mathcal{R}_\tau\ t') \implies \Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t\ \mathcal{R}_\tau\ \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).t' \tag{A.8}$$

$\mathcal{R}$ is called an *FPC congruence relation* if, in addition, it is symmetric:

$$\Gamma \vdash t\ \mathcal{R}_\sigma\ t' \implies \Gamma \vdash t'\ \mathcal{R}_\sigma\ t.$$

Some properties are implied by these properties and hence need not be included in the above definition. We have proven three such properties below:

1. Preservation of functional application, i.e.,

$$(\Gamma \vdash f\ \mathcal{R}_{\sigma\to\tau}\ f' \wedge \Gamma \vdash t\ \mathcal{R}_\sigma\ t') \implies \Gamma \vdash f(t)\ \mathcal{R}_\tau\ f'(t').$$

   To prove this, we begin by observing that $x : \sigma, g : \sigma \to \tau \vdash g(x) : \tau$ can be weakened by to:

$$\Gamma, x : \sigma, g : \sigma \to \tau \vdash g(x) : \tau.$$

Given that $\Gamma \vdash f\ \mathcal{R}_{\sigma\to\tau}\ f'$, we weaken by condition (A.1) so as to obtain

$$\Gamma, x : \sigma \vdash f\ \mathcal{R}_{\sigma\to\tau}\ f'.$$

Together with $\Gamma, x : \sigma, g : \sigma \to \tau \vdash g(x) : \tau$ and by virtue of condition (A.5), it holds that

$$\Gamma, x : \sigma \vdash f(x)\ \mathcal{R}_{\tau}\ f'(x).$$

Again by invoking condition (A.5), together with $\Gamma \vdash t\ \mathcal{R}_{\sigma}\ t'$, we have that

$$\Gamma \vdash f(t)\ \mathcal{R}_{\tau}\ f'(t').$$

2. Preservation of liftings, i.e.,

$$\Gamma \vdash t\ \mathcal{R}_{\sigma}\ t' \implies \Gamma \vdash \mathrm{up}(t)\ \mathcal{R}_{\sigma_\perp}\ \mathrm{up}(t').$$

We first weaken the context $x : \sigma \vdash \mathrm{up}(x) : \sigma_\perp$ to $\Gamma, x : \sigma \vdash \mathrm{up}(x) : \sigma_\perp$. Then combining with $\Gamma \vdash t\ \mathcal{R}_{\sigma}\ t'$, we invoke condition (A.5) to obtain:

$$\Gamma \vdash \mathrm{up}(t)\ \mathcal{R}_{\sigma_\perp}\ \mathrm{up}(t').$$

3. Preservation of foldings and unfoldings, i.e.,

$$\Gamma \vdash t\ \mathcal{R}_{\mu X.\sigma}\ t' \implies \Gamma \vdash \mathrm{unfold}(t)\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ \mathrm{unfold}(t')$$

and

$$\Gamma \vdash t\ \mathcal{R}_{\sigma[\mu X.\sigma/X]}\ t' \implies \Gamma \vdash \mathrm{fold}(t)\ \mathcal{R}_{\mu X.\sigma}\ \mathrm{fold}(t').$$

These again can be established using condition (A.5), together with the corresponding typing contexts.

Note that (A.5) is equivalent to saying that the constructs that don't bind variables preserve the precongruence relation. The conditions (A.6)-(A.9) extend this preservation property to the variable binding constructs of the language. As the following lemma shows, these properties are all special cases of preservation of the precongruence relation by the operation of substituting for a parameter in a context.

Lemma 8.1.1. *Suppose that $\mathcal{R}$ is an FPC precongruence relation and suppose further that $\Gamma, \Gamma' \vdash t\ \mathcal{R}_{\sigma}\ t'$, that $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ and that Γ' is trapped within $C[-_\sigma]$. Then*

$$\Gamma \vdash C[t]\ \mathcal{R}_{\tau}\ C[t'].$$

Proof. The proof is by induction on the derivation of $\Gamma \vdash C[-_\sigma] : \tau$. We choose to only show the case for function abstraction, which is one of those cases which involve binding of variables. For that purpose, we suppose that $\Gamma, \Gamma' \vdash t \; \mathcal{R}_\sigma \; t'$ and that

$$C[-_\sigma] \equiv \lambda x.D[-_\sigma] \in \mathrm{Ctx}_{\rho\to\tau}(\Gamma).$$

Because Γ' is trapped within $C[-_\sigma]$, either (1) $x \in \mathrm{dom}(\Gamma')$ or (2) $x \notin \mathrm{dom}(\Gamma')$.

(1) $x \in \mathrm{dom}(\Gamma')$.
We write $\Gamma' \equiv x : \rho, \Gamma''$. Thus we have assume that

$$\Gamma, x : \rho, \Gamma'' \vdash t \; \mathcal{R}_\sigma \; t'.$$

Moreover, $D[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma, x : \rho)$ and Γ'' is trapped within $D[-_\sigma]$ so that by the induction hypothesis, we have that

$$\Gamma, x : \rho \vdash D[t] \; \mathcal{R}_\tau \; D[t'].$$

Finally, we invoke condition (A.6) to conclude that

$$\Gamma \vdash \lambda x.D[t] \; \mathcal{R}_{\rho\to\tau} \; \lambda x.D[t'].$$

(2) $x \notin \mathrm{dom}(\Gamma')$.
By weakening $(\Gamma, \Gamma' \vdash t \; \mathcal{R}_\sigma \; t')$ to

$$(\Gamma, \Gamma', x : \rho \vdash t \; \mathcal{R}_\sigma \; t')$$

by condition (A.1), and combining with the assumptions
$\Gamma, x : \rho \vdash D[-_\sigma] : \tau$ and Γ' is trapped within $D[-_\sigma]$, we obtain

$$\Gamma, x : \rho \vdash D[t] \; \mathcal{R}_\tau \; D[t']$$

and finally by the induction hypothesis and (A.5), we have that

$$\Gamma \vdash \lambda x.D[t] \; \mathcal{R}_{\rho\to\tau} \; \lambda x.D[t']$$

as desired.

□

8.2 An auxiliary relation

The auxiliary relation

$$\Gamma \vdash t \preceq^*_\sigma t' \qquad (t, t' \in \mathrm{Exp}_\sigma)$$

is inductively defined by the axioms and rules in Figure 8.1.

We collect at one place some useful properties of $\preceq^*$ in the form of Lemmas 8.2.1 and 8.2.2. In order not to interrupt the flow of the argument, the detailed proof of these two lemmas are placed in Section 8.5.

Lemma 8.2.1. *(1) If* $\Gamma \vdash t \preceq^*_\sigma t'$ *and* $\Gamma \vdash t' \preceq^\circ_\sigma t''$, *then* $\Gamma \vdash t \preceq^*_\sigma t''$.

(2) If $\Gamma \vdash t : \sigma$, *then* $\Gamma \vdash t \preceq^*_\sigma t$.

(3) If $\Gamma \vdash t \preceq^\circ_\sigma t'$, *then* $\Gamma \vdash t \preceq^*_\sigma t'$.

(4) If $\Gamma \vdash t \preceq^*_\sigma t'$ *and* $\Gamma, x : \sigma \vdash s \preceq^*_\tau s'$, *then* $\Gamma \vdash s[t/x] \preceq^*_\tau s'[t'/x]$.

Lemma 8.2.2. *If* $\emptyset \vdash s \preceq^*_\sigma t$ *and* $s \Downarrow v$, *then* $\emptyset \vdash v \preceq^*_\sigma t$.

Proposition 8.2.3. *For all* Γ, σ, s, t,

$$\Gamma \vdash s \preceq^\circ_\sigma t \iff \Gamma \vdash s \preceq^*_\sigma t.$$

Proof. ($\Rightarrow$): This is precisely Lemma 8.2.1(3).
($\Leftarrow$): To prove that $\Gamma \vdash s \preceq^*_\sigma t \implies \Gamma \vdash s \preceq^\circ_\sigma t$, it is enough to prove the implication just for closed terms, i.e.,

$$\emptyset \vdash s \preceq^*_\sigma t \implies s \preceq_\sigma t.$$

In order to see why this is sufficient, let us suppose that $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash s \preceq^*_\sigma t$ and assume that $\emptyset \vdash s \preceq^*_\sigma t \implies s \preceq_\sigma t$ holds. We wish to show that for all $t_1 \in \mathrm{Exp}_{\sigma_1}, \ldots, t_n \in \mathrm{Exp}_{\sigma_n}$,

$$s[\vec{t}/\vec{x}] \preceq_\sigma t[\vec{t}/\vec{x}].$$

But by Lemma 8.2.1(2), $\Gamma \vdash t_i : \sigma_i \implies \Gamma \vdash t_i \preceq^*_\sigma t_i$ $(i = 1, \ldots, n)$. It then follows from Lemma 8.2.1(4) that

$$\emptyset \vdash s[\vec{t}/\vec{x}] \preceq^*_\sigma t[\vec{t}/\vec{x}]$$

which is the same as $s[\vec{t}/\vec{x}] \preceq_\sigma t[\vec{t}/\vec{x}]$, as required.

Let us complete the proof that

$$\emptyset \vdash s \preceq^*_\sigma t \implies s \preceq_\sigma t.$$

$$\Gamma, x : \sigma \vdash x \preceq^*_\sigma t \text{ (if } \Gamma, x : \sigma \vdash x \preceq^\circ_\sigma t) \qquad (\preceq^* \text{ var})$$

$$\frac{\Gamma, x : \sigma \vdash t \preceq^*_\tau t'}{\Gamma \vdash \lambda x.t \preceq^*_{\sigma\to\tau} u} \text{ (if } \Gamma \vdash \lambda x.t' \preceq^\circ_{\sigma\to\tau} u) \qquad (\preceq^* \text{ abs})$$

$$\frac{\Gamma \vdash f \preceq^*_{\sigma\to\tau} f' \quad \Gamma \vdash t \preceq^*_\sigma t'}{\Gamma \vdash f(t) \preceq^*_\tau u} \text{ (if } \Gamma \vdash f'(t') \preceq^\circ u) \qquad (\preceq^* \text{ app})$$

$$\frac{\Gamma \vdash s \preceq^*_\sigma s' \quad \Gamma \vdash t \preceq^*_\tau t'}{\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} u} \text{ (if } \Gamma \vdash (s',t') \preceq^\circ_{\sigma\times\tau} u) \qquad (\preceq^* \text{ pair})$$

$$\frac{\Gamma \vdash p \preceq^*_{\sigma\times\tau} p'}{\Gamma \vdash \mathrm{fst}(p) \preceq^*_\sigma u} \text{ (if } \Gamma \vdash \mathrm{fst}(p') \preceq^\circ_\sigma u) \qquad (\preceq^* \text{ fst})$$

$$\frac{\Gamma \vdash p \preceq^*_{\sigma\times\tau} p'}{\Gamma \vdash \mathrm{snd}(p) \preceq^*_\tau u} \text{ (if } \Gamma \vdash \mathrm{snd}(p') \preceq^\circ_\tau u) \qquad (\preceq^* \text{ snd})$$

$$\frac{\Gamma \vdash t \preceq^*_\sigma t'}{\Gamma \vdash \mathrm{inl}(t) \preceq^*_{\sigma+\tau} u} \text{ (if } \Gamma \vdash \mathrm{inl}(t') \preceq^\circ_{\sigma+\tau} u) \qquad (\preceq^* \text{ inl})$$

$$\frac{\Gamma \vdash t \preceq^*_\tau t'}{\Gamma \vdash \mathrm{inr}(t) \preceq^*_{\sigma+\tau} u} \text{ (if } \Gamma \vdash \mathrm{inr}(t') \preceq^\circ_{\sigma+\tau} u) \qquad (\preceq^* \text{ inr})$$

$$\frac{\Gamma \vdash s \preceq^*_{\sigma+\tau} s' \quad \Gamma, x : \sigma \vdash t_1 \preceq^*_\rho t'_1 \quad \Gamma, y : \sigma \vdash t_2 \preceq^*_\rho t'_2}{\Gamma \vdash \mathrm{case}(s) \text{ of } \mathrm{inl}(x).t_1 \text{ or } \mathrm{inr}(y).t_2 \preceq^*_\rho u}$$
$$(\text{if } \Gamma \vdash \mathrm{case}(s') \text{ of } \mathrm{inl}(x).t'_1 \text{ or } \mathrm{inr}(y).t'_2 \preceq^\circ_\rho u) \qquad (\preceq^* \text{ case})$$

$$\frac{\Gamma \vdash t \preceq^*_\sigma t'}{\Gamma \vdash \mathrm{up}(t) \preceq^*_{\sigma_\perp} u} \text{ (if } \Gamma \vdash \mathrm{up}(t') \preceq^\circ_{\sigma_\perp} u) \qquad (\preceq^* \text{ up})$$

$$\frac{\Gamma \vdash s \preceq^*_{\sigma_\perp} s' \quad \Gamma, x : \sigma \vdash t \preceq^*_\rho t'}{\Gamma \vdash \mathrm{case}(s) \text{ of } \mathrm{up}(x).t \preceq^*_\rho u} \text{ (if } \Gamma \vdash \mathrm{case}(s') \text{ of } \mathrm{up}(x).t' \preceq^\circ_\rho u) \qquad (\preceq^* \text{ case up})$$

$$\frac{\Gamma \vdash t \preceq^*_{\mu X.\sigma} t'}{\Gamma \vdash \mathrm{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma/X]} u} \text{ (if } \Gamma \vdash \mathrm{unfold}(t') \preceq^\circ_{\sigma[\mu X.\sigma/X]} u) \qquad (\preceq^* \text{ unfold})$$

$$\frac{\Gamma \vdash t \preceq^*_{\sigma[\mu X.\sigma/X]} t'}{\Gamma \vdash \mathrm{fold}(t) \preceq^*_{\mu X.\sigma} u} \text{ (if } \Gamma \vdash \mathrm{fold}(t') \preceq^\circ_{\mu X.\sigma} u) \qquad (\preceq^* \text{ fold})$$

Figure 8.1: Definition of $\Gamma \vdash s \preceq^*_\sigma t$

By the co-induction principle, it is enough to show that

$$\mathcal{S} := \{(s,t) \in \mathrm{Exp}_\sigma \times \mathrm{Exp}_\sigma | \emptyset \vdash s \preceq^*_\sigma t\}$$

is an FPC simulation. So all we need to do is to verify that $\mathcal{S}$ satisfies all the conditions (sim 1) - (sim 6) in Figure 7.2.

(sim 1) Let $s, s' : 1$ be given. We must show that $\emptyset \vdash s \preceq^*_1 s'$. Now since $\preceq$ is an FPC simulation, it follows that (sim 1) that $s \preceq_1 s'$ holds. It then follows from Lemma 8.2.1(3) that $\emptyset \vdash s \preceq^*_1 s'$, as required.

(sim 2) Let $\emptyset \vdash p \preceq^*_{\sigma\times\tau} p'$ be given. We want to show that

$$(\emptyset \vdash \mathrm{fst}(p) \preceq^*_\sigma \mathrm{fst}(p') \wedge \emptyset \vdash \mathrm{snd}(p) \preceq^*_\tau \mathrm{snd}(p')).$$

Note that $\emptyset \vdash \mathrm{fst}(p') \preceq^\circ_\sigma \mathrm{fst}(p')$ holds by Proposition 7.3.2(1). Since $\emptyset \vdash p \preceq^*_{\sigma\times\tau} p'$ and $\emptyset \vdash \mathrm{fst}(p') \preceq^\circ \mathrm{fst}(p')$, it follows from $(\preceq^* \mathrm{fst})$ that

$$\emptyset \vdash \mathrm{fst}(p) \preceq^*_\sigma \mathrm{fst}(p').$$

That $\emptyset \vdash \mathrm{snd}(p) \preceq^*_\tau \mathrm{snd}(p')$ holds can be proven similarly.

(sim 3a) Given that $\emptyset \vdash s \preceq^*_{\sigma+\tau} s'$ and $s \Downarrow \mathrm{inl}(a)$, we want to show that there exists $a' \in \mathrm{Exp}_\sigma$ such that $s' \Downarrow \mathrm{inl}(a')$ and $\emptyset \vdash a \preceq^*_\sigma a'$. Applying Lemma 8.2.2 to the supposition, we have $\emptyset \vdash \mathrm{inl}(a) \preceq^*_{\sigma+\tau} s'$. But the only derivation of this is via an application of the rule $(\preceq^* \mathrm{inl})$ to $\emptyset \vdash a \preceq^*_\sigma a''$ for some $a'' \in \mathrm{Exp}_\sigma$ with $\mathrm{inl}(a'') \preceq_{\sigma+\tau} s'$. Since $\preceq$ is an FPC simulation, it follows from (sim 3a) that there exists $a' \in \mathrm{Exp}_\sigma$ such that $s' \Downarrow \mathrm{inl}(a')$ and $a'' \preceq_\sigma a'$. Finally, by Lemma 8.2.1(3) we conclude that $\emptyset \vdash a \preceq^*_\sigma a'$, as required.

(sim 3b) Proven similarly.

(sim 4) Given that $\emptyset \vdash t \preceq^*_{\sigma_\perp} t'$ and $t \Downarrow \mathrm{up}(s)$, we want to show that there exists $s' : \sigma$ such that $t' \Downarrow \mathrm{up}(s')$ and $\emptyset \vdash s \preceq^*_\sigma s'$. To do this, apply Lemma 8.2.2 to the supposition to yield $\emptyset \vdash \mathrm{up}(s) \preceq^*_{\sigma_\perp} t$. But the only derivation for this is via an application of $(\preceq^* \mathrm{up})$ to $\emptyset \vdash s \preceq^*_\sigma s''$ for some term $s'' : \sigma$ with $\mathrm{up}(s'') \preceq_{\sigma_\perp} t'$. Since $\preceq$ is an FPC simulation, it follows from (sim 4) that there exists $s' : \sigma$ such that $t' \Downarrow \mathrm{up}(s')$ and $s \preceq_\sigma s'$. Finally, we invoke Lemma 8.2.1(3) to conclude that $\emptyset \vdash s \preceq^*_\sigma s'$ as required.

(sim 5) Let $\emptyset \vdash t \preceq^*_{\mu X.\sigma} t'$ be given. We want to show that

$$\emptyset \vdash \mathrm{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t').$$

To do this, observe that we always have, by reflexivity, that:

$$\text{unfold}(t') \preceq_{\sigma[\mu X.\sigma/X]} \text{unfold}(t')$$

Then invoking ($\preceq^*$ unfold), we have:

$$\frac{\emptyset \vdash t \preceq^*_{\mu X.\sigma} t'}{\emptyset \vdash \text{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma]} \text{unfold}(t')} (\text{if } \emptyset \vdash \text{unfold}(t') \preceq^\circ_{\sigma[\mu X.\sigma/X]} \text{unfold}(t')).$$

Thus we have $\emptyset \vdash \text{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma/X]} \text{unfold}(t')$, as required.

(sim 6) Let $\emptyset \vdash f \preceq^*_{\sigma\to\tau} f'$ be given. We want to show that

$$\forall t \in \text{Exp}_\sigma. \emptyset \vdash f(t) \preceq^*_\tau f'(t).$$

Let $t \in \text{Exp}_\sigma$ be arbitrary. Note that we always have $\emptyset \vdash t \preceq^\circ_\sigma t$ and consequently by Lemma 8.2.1(3), $\emptyset \vdash t \preceq^*_\sigma t$ holds. Also by Lemma 8.2.1(2), $\emptyset \vdash f'(t) \preceq^\circ f'(t)$ always holds. Invoke the rule ($\preceq^*$ app) to get:

$$\frac{\emptyset \vdash f \preceq^*_{\sigma\to\tau} f' \quad \emptyset \vdash t \preceq^*_\sigma t'}{\emptyset \vdash f(t) \preceq^*_\tau f'(t)} (\text{if } \emptyset \vdash f'(t) \preceq^\circ f'(t)).$$

Thus $\emptyset \vdash f(t) \preceq^*_\tau f'(t)$ as required.

□

8.3 Open similarity is an FPC precongruence

In this section, we use the coincidence of $\preceq^\circ$ and $\preceq^*$ (i.e., Proposition 8.2.3) to prove that $\preceq^\circ$ is an FPC precongruence.

Theorem 8.3.1. *FPC similarity $\preceq^\circ$ is an FPC precongruence, and hence (by Proposition 7.3.2(3)) FPC bisimilarity is an FPC congruence.*

Proof. Recall from Proposition 7.4.1 that $\preceq^\circ$ is reflexive and transitive. So it just remains to show that $\preceq^\circ$ satisfies the properties (A.1), (A.2) and (A.5) - (A.9) of the definition of FPC precongruence relation.

The weakening property (A.1) is an immediate consequence of the construction of $\preceq^\circ$ from $\preceq$.

For the other properties, it suffices, by Proposition 8.2.3, to prove that they hold for the relation $\preceq^*$.

Note that (A.2) and (A.5) are both instances of Lemma 8.2.1(4) (using also the reflexivity of $\preceq^\circ$, established in (2) of that lemma). The details are as follows:

(A.2) Suppose that ($\Gamma \vdash t : \sigma \wedge \Gamma, x : \sigma \vdash s \preceq^*_\tau s'$). Now by the reflexivity of $\preceq^*$, i.e., Lemma 8.2.1(2), we have $\Gamma \vdash t \preceq^*_\sigma t$. Then it follows from Lemma 8.2.1(4) that

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

(A.5) Suppose that ($\Gamma \vdash t \preceq^*_\sigma t' \wedge \Gamma, x : \sigma \vdash s : \tau$. Now by the reflexivity of $\preceq^*$, i.e., Lemma 8.2.1(2), we have $\Gamma, x : \sigma \vdash s \preceq^*_\tau s$. Then it follows from Lemma 8.2.1(4) that

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

Properties (A.6) - (A.9) hold for $\preceq^*$ by construction. □

Corollary 8.3.2. *For all* Γ, σ, s, t,

$$\Gamma \vdash s \preceq^\circ_\sigma t \implies \Gamma \vdash s \sqsubseteq_\sigma t.$$

Proof. Suppose $\Gamma \vdash s \preceq^\circ_\sigma t$ and let $C[-_\sigma]$ be a context for which $C[s]$ and $C[t]$ belong to Exp_Σ. By Theorem 8.3.1, i.e., $\preceq^\circ$ is a precongruence relation, it follows from Lemma 8.1.1 that $\emptyset \vdash C[s] \preceq^\circ_\Sigma C[t]$. So if $C[s] \Downarrow \top$ then, by (sim 4), there exists $\star : 1$ such that $C[s] \Downarrow \mathrm{up}(\star)$ and $\star \preceq^\circ_1 \star$ (which always holds by definition of $\preceq^\circ_1$). But $\mathrm{up}(\star)$ is simply $\top$. Thus $C[t] \Downarrow \top$ as required. Since $C[-_\sigma]$ is arbitrary, we have that $\Gamma \vdash s \sqsubseteq_\sigma t$. □

8.4 Contextual preorder is an FPC simulation

Lemma 8.4.1. *The contextual preorder* $\sqsubseteq$*, restricted to the closed terms, is an FPC simulation, i.e., the relation*

$$\mathcal{S}_\sigma := \{(s, t) | \emptyset \vdash s \sqsubseteq_\sigma t\}$$

satisfies all the conditions sim(1) - sim(6).

Proof.

(sim 1) Let $s : s' : 1$ be given. Because $\emptyset \vdash s \preceq^\circ_1 s'$ always holds (since $\preceq^\circ$ is an FPC simulation), by Corollary 8.3.2 we have that $\emptyset \vdash s \sqsubseteq_1 s'$.

(sim 2) Let $p \sqsubseteq_{\sigma\times\tau} p'$ be given. We want to show that

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

Let $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ be given and suppose that $C[\mathrm{fst}(p)] \Downarrow \top$. Then define $C'[-_{\sigma\times\tau}] \in \mathrm{Ctx}_\Sigma$ by

$$C'[-_{\sigma\times\tau}] := C[\mathrm{fst}(-_{\sigma\times\tau})].$$

Since $p \sqsubseteq_{\sigma\times\tau} p'$, it follows that

$$C'[p] \Downarrow \top \implies C'[p'] \Downarrow \top$$

which is equivalent to saying that $C[\mathrm{fst}(p')] \Downarrow \top$, as required. The same kind of argument can be carried out for proving $\mathrm{snd}(p) \sqsubseteq_\tau \mathrm{snd}(p')$.

(sim 3a) Given that $s \sqsubseteq_{\sigma+\tau} s'$ and $s \Downarrow \mathrm{inl}(a)$, we want to show that there exists $a' \in \mathrm{Exp}_\sigma$ such that $s' \Downarrow \mathrm{inl}(a')$ and $a \sqsubseteq_\sigma a'$. To prove this, we consider the context

$$C[-_{\sigma+\tau}] := \mathrm{case}(-_{\sigma+\tau}) \text{ of } \mathrm{inl}(x).\top \text{ or } \mathrm{inr}(y).\bot.$$

Notice that $C[s] \Downarrow \top$ iff $s \Downarrow \mathrm{inl}(a)$ for some $a \in \mathrm{Exp}_\sigma$. Indeed we are given that $s \Downarrow \mathrm{inl}(a)$ and thus $C[s] \Downarrow \top$. But $s \sqsubseteq_{\sigma+\tau} s'$ implies that $C[s'] \Downarrow \top$, i.e., there exists $a' \in \mathrm{Exp}_\sigma$ such that $s' \Downarrow \mathrm{inl}(a')$. It now remains to show that $a \sqsubseteq_\sigma a'$. To achieve this, we consider the abstraction

$$f := \lambda z : \sigma + \tau.\mathrm{case}(z) \text{ of } \mathrm{inl}(x).x \text{ or } \mathrm{inr}(y).\bot.$$

We claim that $a =_\sigma f(s)$ and $a' =_\sigma f(s')$. By the co-induction principle, it suffices to show $a \cong^{kl}_\sigma f(s)$. This is done by observing that $(s \Downarrow \mathrm{inl}(a) \wedge a \Downarrow v) \iff f(s) \Downarrow v$.
Now we are ready to show that $a \sqsubseteq_\sigma a'$. Let $C[_\sigma] \in \mathrm{Ctx}_\Sigma$ be such that $C[a] \Downarrow \top$. We want to show that $C[a'] \Downarrow \top$. Then define the context $C'[-_{\sigma+\tau}] := C[f(-_{\sigma+\tau})]$. Since $a =_{\sigma+\tau} f(s)$ and $a' =_{\sigma+\tau} f(s')$, it follows that $C[f(s)] \Downarrow \top$. Consequently, $C'[s] \Downarrow \top$. Since $s \sqsubseteq_{\sigma+\tau} s'$, it follows that $C'[s'] \Downarrow \top$ which is the same as $C[f(s')] \Downarrow \top$. Thus $C[a'] \Downarrow \top$, required.

(sim 3b) Proven similarly.

(sim 4) Given that $t \sqsubseteq_{\sigma_\bot} t'$ and $t \Downarrow \mathrm{up}(s)$, we want to prove that there exists $s' \in \mathrm{Exp}_\sigma$ such that $t' \Downarrow \mathrm{up}(s')$ and $s \sqsubseteq_\sigma s'$. Let us consider the context

$$C[-_{\sigma_\bot}] := \mathrm{case}(-_{\sigma_\bot}) \text{ of } \mathrm{up}(x).\top.$$

Note that $C[t] \Downarrow \top \iff t \Downarrow \mathrm{up}(s)$ for some term $s \in \mathrm{Exp}_\sigma$. Indeed it

is given that $t \Downarrow \mathrm{up}(s)$ and thus $C[t] \Downarrow \top$. But since $t \sqsubseteq_{\sigma_\perp} t'$, it follows that $C[t'] \Downarrow \top$, i.e., $t' \Downarrow \mathrm{up}(s')$ for some term $s' \in \mathrm{Exp}_\sigma$. It remains to show that $s \sqsubseteq_\sigma s'$. The term to consider is:

$$f := \lambda z : \sigma_\perp.\mathrm{case}(z) \text{ of } \mathrm{up}(x).x.$$

It is not difficult to see that $s \cong^{kl}_\sigma f(t)$ and $s' \cong^{kl}_\sigma f(t')$. Hence by the co-induction principle, we have $s =_\sigma f(t)$ and $s' =_\sigma f(t')$.
We are now ready to show that $s \sqsubseteq_\sigma s'$. For that purpose, let $C[-_\sigma] \in \mathrm{Ctx}_\Sigma$ be such that $C[s] \Downarrow \top$. Then define $C'[-_{\sigma_\perp}] := C[f(-_{\sigma_\perp})]$. Clearly, $C[s] \Downarrow \top$ implies that $C[f(t)] \Downarrow \top$, i.e., $C'[t] \Downarrow \top$. Because $t \sqsubseteq_{\sigma_\perp} t'$ we have that $C'[t'] \Downarrow \top$, i.e., $C[s'] \Downarrow \top$ which is what we desire to prove.

(sim 5) Let $t \sqsubseteq_{\mu X.\sigma} t'$ be given. We want to show that

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

To do that, let $C[-_{\sigma[\mu X.\sigma/X]}] \in \mathrm{Ctx}_\Sigma$ be such that $C[\mathrm{unfold}(t)] \Downarrow \top$. We want to prove that $C[\mathrm{unfold}(t')] \Downarrow \top$. To do so, consider the context

$$C'[-_{\mu X.\sigma}] := C[\mathrm{unfold}(-_{\mu X.\sigma})].$$

Since $C[\mathrm{unfold}(t)] \Downarrow \top$, it follows from the definition of C' that $C'[t] \Downarrow \top$. Then our assumption that $t \sqsubseteq_{\mu X.\sigma} t'$ guarantees that $C'[t'] \Downarrow \top$. Consequently, $C[\mathrm{unfold}(t')] \Downarrow \top$ which is what we aim to show.

(sim 6) Let $f \sqsubseteq_{\sigma\to\tau} f'$ be given. We aim to show that

$$\forall t \in \mathrm{Exp}_\sigma.f(t) =_\tau f'(t).$$

So let $t \in \mathrm{Exp}_\sigma$ be arbitrary. Suppose the context $C[-_\tau] \in \mathrm{Ctx}_\Sigma$ is such that $C[f(t)] \Downarrow \top$. We must prove that $C[f'(t)] \Downarrow \top$. To do this, we first define the context

$$C'[-_{\sigma\to\tau}] := C[-_{\sigma\to\tau}(t)].$$

Since $C[f(t)] \Downarrow \top$, it follows that $C'[f] \Downarrow \top$. Because $f \sqsubseteq_{\sigma\to\tau} f'$, it follows that $C'[f'] \Downarrow \top$, i.e., $C[f'(t)] \Downarrow \top$.

The proof is thus complete. □

In order to complete the proof of Theorem 7.4, we must extend the implication in Lemma 8.4.1 from closed to open terms. To do this, we need to

verify that the substitutivity property holds for $\sqsubseteq$, i.e.,

Lemma 8.4.2. *If* $\Gamma, x : \sigma \vdash s \sqsubseteq_\tau s'$*, then it holds that*

$$\Gamma \vdash s[t/x] \sqsubseteq_\tau s'[t/x]$$

for every $t \in \mathrm{Exp}_\sigma(\Gamma)$*.*

Proof. We first claim that

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

To prove this claim, let $C[-_\tau] \in \mathrm{Ctx}_\Sigma(\Gamma)$ be such that $C[(\lambda x.s)t] \Downarrow \top$. Now define the context $C'[-_\tau] := C[(\lambda x.-_\tau)t]$. Clearly, $C'[s] \Downarrow \top \implies C'[t] \Downarrow \top$, i.e., $C[(\lambda x.s)t] \Downarrow \top$. Hence $\Gamma \vdash (\lambda x.s)t \sqsubseteq_\tau (\lambda x.s')t$. But since $\Gamma \vdash s[t/x] \sqsubseteq^{kl}_\tau (\lambda x.s')t$ and $\Gamma \vdash (\lambda x.s')t \sqsubseteq^{kl}_\tau s'[t/x]$, it then follows from the co-induction principle that $\Gamma \vdash s[t/x] \sqsubseteq_\tau (\lambda x.s)t$ and $\Gamma \vdash (\lambda x.s')t \sqsubseteq_\tau s'[t/x]$. By the transitivity of $\sqsubseteq_\tau$, we have that $\Gamma \vdash s[t/x] \sqsubseteq_\tau s'[t/x]$. □

We are now ready to extend Lemma 8.4.1 from closed terms to open terms.

Lemma 8.4.3. *For all* Γ, σ, s, t*,*

$$\Gamma \vdash s \sqsubseteq_\sigma t \implies \Gamma \vdash s \preceq^\circ_\sigma t.$$

Proof. Suppose that $\Gamma \equiv x_1 : \sigma_1, \ldots, x : \sigma_n \vdash s \sqsubseteq_\sigma t$ holds. For any $s_i \in \mathrm{Exp}_{\sigma_i}$ $(i = 1, \ldots, n)$, by applying Lemma 8.4.2 repeatedly, we get:

$$s[\vec{s}/\vec{x}] \sqsubseteq_\sigma t[\vec{s}/\vec{x}]$$

and since $\emptyset \vdash s \sqsubseteq_\sigma t$ implies $\emptyset \vdash s \preceq^\circ_\sigma t$, we have that:

$$s[\vec{s}/\vec{x}] \preceq_\sigma t[\vec{s}/\vec{x}].$$

Thus by definition of $\preceq^\circ$, it follows that

$$\Gamma \vdash s \preceq^\circ_\sigma t.$$

□

Thus the converse of Corollary 8.3.2 holds and we have completed the proof of Theorem 7.4.

8.5 Appendix

In this appendix, we record the proofs of Lemmas 8.2.1 and 8.2.2.

Proof of Lemma 8.2.1(1).
We aim to prove that:
If $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma \vdash t' \preceq^\circ_\sigma t''$, then $\Gamma \vdash t \preceq^*_\sigma t''$.
We prove this by induction on the derivation of $\Gamma \vdash t \preceq^*_\sigma t'$.

(1) ($\preceq^*$ var)
Given that $\Gamma', x : \sigma \vdash x \preceq^*_\sigma t'$ and $\Gamma', x : \sigma \vdash t' \preceq^\circ_\sigma t''$, we must prove that $\Gamma', x : \sigma \vdash x \preceq^*_\sigma t''$. From ($\preceq^*$ var), $\Gamma', x : \sigma \vdash x \preceq^*_\sigma t'$ holds only if $\Gamma', x : \sigma \vdash x \preceq^\circ_\sigma t'$. It follows from Proposition 7.4.1(2) that

$$(\Gamma', x : \sigma \vdash x \preceq^\circ_\sigma t' \wedge \Gamma', x : \sigma \vdash t' \preceq^\circ_\sigma t'') \implies \Gamma', x : \sigma \vdash x \preceq^\circ_\sigma t''.$$

Then the required result follows from the axiom ($\preceq^*$ var)

$$\frac{}{\Gamma', x : \sigma \vdash x \preceq^*_\sigma t''} (\text{if } \Gamma', x : \sigma \vdash x \preceq^\circ t'').$$

(2) ($\preceq^*$ abs)
Given that $\Gamma \vdash \lambda x.s \preceq^*_{\sigma\to\tau} t'$ and $\Gamma \vdash t' \preceq^\circ_{\sigma\to\tau} t''$, we must show that $\Gamma \vdash \lambda x.s \preceq^*_{\sigma\to\tau} t''$. The induction hypothesis asserts that $\Gamma, x : \sigma \vdash s \preceq^*_\tau s'$. From the inference rule ($\preceq^*$ abs), $\Gamma \vdash \lambda x.s \preceq^*_{\sigma\to\tau} t'$ provided $\Gamma \vdash \lambda x.s' \preceq^*_{\sigma\to\tau} t'$ holds. By Proposition 7.4.1(2), we have

$$(\Gamma \vdash \lambda x.s' \preceq^\circ_{\sigma\to\tau} t' \wedge \Gamma \vdash t' \preceq^\circ_{\sigma\to\tau} t'') \implies \Gamma \vdash \lambda x.s' \preceq^\circ_{\sigma\to\tau} t''.$$

The required result then follows from the inference rule ($\preceq^*$ abs)

$$\frac{\Gamma, x : \sigma \vdash s \preceq^*_\tau s'}{\Gamma \vdash \lambda x.s \preceq^*_{\sigma\to\tau} t''} (\text{if } \Gamma \vdash \lambda x.s' \preceq^\circ_{\sigma\to\tau} t'').$$

(3) ($\preceq^*$ app)
Given that $\Gamma \vdash f(s) \preceq^*_\tau t'$ and $\Gamma \vdash t' \preceq^\circ_\tau t''$, we must show that $\Gamma \vdash f(s) \preceq^*_\tau t''$. The induction hypothesis asserts that $\Gamma \vdash f \preceq^*_{\sigma\to\tau} f'$ and $\Gamma \vdash s \preceq^*_\sigma s'$ for some f' and s'. Note that $\Gamma \vdash f(s) \preceq^*_\sigma t'$ holds provided $\Gamma \vdash f'(s') \preceq^\circ_\tau t'$. By Proposition 7.4.1(2), it then follows that

$$(\Gamma \vdash f'(s') \preceq^\circ_\tau t' \wedge \Gamma \vdash t' \preceq^\circ_\tau t'') \implies \Gamma \vdash f'(s') \preceq^\circ_\tau t''.$$

The required result then follows from the inference rule ($\preceq^*$ app)

$$\frac{\Gamma \vdash f \preceq^*_{\sigma\to\tau} f' \quad \Gamma \vdash s \preceq^*_\sigma s'}{\Gamma \vdash f(s) \preceq^*_\tau t''}(\text{if } \Gamma \vdash f'(s') \preceq^\circ_\tau t'').$$

(4) ($\preceq^*$ pair)
Given that $\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} p'$ and $\Gamma \vdash t' \preceq^\circ_{\sigma\times\tau} p''$, we must show that $\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} p''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_\sigma s'$ and $\Gamma \vdash t \preceq^*_\tau t'$ for some terms s' and t'. Also $\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} p'$ holds provided $\Gamma \vdash (s',t') \preceq^\circ_{\sigma\times\tau} p'$. By Proposition 7.4.1(2), it follows that

$$(\Gamma \vdash (s',t') \preceq^\circ_{\sigma\times\tau} p' \ \wedge\ \Gamma \vdash p' \preceq^\circ_{\sigma\times\tau} p'') \implies \Gamma \vdash (s',t') \preceq^\circ_{\sigma\times\tau} p''.$$

The required result then follows from the rule ($\preceq^*$ pair)

$$\frac{\Gamma \vdash s \preceq^*_\sigma s' \quad \Gamma \vdash t \preceq^*_\tau t'}{\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} p''}(\text{if } \Gamma \vdash (s',t') \preceq^\circ_{\sigma\times\tau} p'').$$

(5) ($\preceq^*$ fst, snd)
Given that $\Gamma \vdash \text{fst}(p) \preceq^*_\sigma t'$ and $\Gamma \vdash t' \preceq^\circ_\sigma t''$, we must show that $\Gamma \vdash \text{fst}(p) \preceq^*_\sigma t''$. The induction hypothesis asserts that $\Gamma \vdash p \preceq^*_{\sigma\times\tau} p'$ for some term p'. Note that $\Gamma \vdash \text{fst}(p) \preceq^*_\sigma t'$ holds provided $\Gamma \vdash \text{fst}(p') \preceq^\circ_\sigma t'$. By Proposition 7.4.1(2), it follows that

$$(\Gamma \vdash \text{fst}(p') \preceq^\circ_\sigma t' \ \wedge\ \Gamma \vdash t' \preceq^\circ_\sigma t'') \implies \Gamma \vdash \text{fst}(p') \preceq^\circ_\sigma t''.$$

The required result then follows from the rule ($\preceq^*$ fst)

$$\frac{\Gamma \vdash p \preceq^*_{\sigma\times\tau} p'}{\Gamma \vdash \text{fst}(p) \preceq^*_\sigma t''}(\text{if } \Gamma \vdash \text{fst}(p') \preceq^\circ_\sigma t'').$$

Similarly, the case holds for ($\preceq^*$ snd).

(6) ($\preceq^*$ inl, inr)
Given that $\Gamma \vdash \text{inl}(s) \preceq^*_{\sigma+\tau} t'$ and $\Gamma \vdash t' \preceq^\circ_{\sigma+\tau} t''$, we must show that $\Gamma \vdash \text{inl}(s) \preceq^*_{\sigma+\tau} t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_\sigma s'$ for some s'. Note that $\Gamma \vdash \text{inl}(s) \preceq^*_{\sigma+\tau} t'$ holds provided $\Gamma \vdash \text{inl}(s') \preceq^\circ_{\sigma+\tau} t'$. By Proposition 7.4.1(2), it follows that

$$(\Gamma \vdash \text{inl}(s') \preceq^\circ_{\sigma+\tau} t' \ \wedge\ \Gamma \vdash t' \preceq^\circ_{\sigma+\tau}) \implies \Gamma \vdash \text{inl}(s') \preceq^\circ_{\sigma+\tau} t''.$$

The required result then follows from the inference rule ($\preceq^*$ inl)

$$\frac{\Gamma \vdash s \preceq^*_\sigma s'}{\Gamma \vdash \mathrm{inl}(s) \preceq^*_{\sigma+\tau} t''}(\text{if } \Gamma \vdash \mathrm{inl}(s') \preceq^\circ_{\sigma+\tau} t'').$$

Similarly, the case for ($\preceq^*$ inr) holds.

(7) ($\preceq^*$ case)
Given that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho t'$ and $\Gamma \vdash t' \preceq^\circ_\rho t''$, we must show that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\sigma+\tau} s'$, $\Gamma, x : \sigma \vdash t_1 \preceq^*_\rho t'_1$ and $\Gamma, y : \tau \vdash t_2 \preceq^*_\rho t'_2$ for some terms s', t'_1, t'_2. Note that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho t'$ holds provided

$$\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho t'.$$

By Proposition 7.4.1(2), it follows that
$(\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho t' \wedge \Gamma \vdash t' \preceq^\circ_\rho t'') \implies$
$\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho t''$.
Using the inference rule ($\preceq^*$ case), the required result:

$$\frac{\Gamma \vdash s \preceq^*_{\sigma+\tau} s' \quad \Gamma, x : \sigma \vdash t_1 : \rho \quad \Gamma, y : \tau \vdash t_2 : \rho}{\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho t''}$$

$$(\text{if } \Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho t'').$$

(8) ($\preceq^*$ up)
Given that $\Gamma \vdash \mathrm{up}(s) \preceq^*_{\sigma_\perp} t'$ and $\Gamma \vdash t' \preceq^\circ_{\sigma_\perp} t''$, we want to show that $\Gamma \vdash \mathrm{up}(s) \preceq^*_{\sigma_\perp} t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_\sigma s'$ for some s'. Note that $\Gamma \vdash \mathrm{up}(s) \preceq^*_{\sigma_\perp} t'$ holds provided $\Gamma \vdash \mathrm{up}(s') \preceq^\circ_{\sigma_\perp} t''$. By Proposition 7.4.1(2), it follows that

$$(\Gamma \vdash \mathrm{up}(s') \preceq^\circ_{\sigma_\perp} t' \ \wedge\ \Gamma \vdash t' \preceq^\circ_{\sigma_\perp} t'') \implies \Gamma \vdash \mathrm{up}(s') \preceq^\circ_{\sigma_\perp} t''.$$

The required result then follows from the inference rule ($\preceq^*$ up)

$$\frac{\Gamma \vdash s \preceq^*_\sigma s'}{\Gamma \vdash \mathrm{up}(s) \preceq^*_{\sigma_\perp} t''}(\text{if } \Gamma \vdash \mathrm{up}(s') \preceq^\circ_{\sigma_\perp} t'').$$

(9) ($\preceq^*$ case up)
Given that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \preceq^*_\rho t'$ and $\Gamma \vdash t' \preceq^*_\rho t''$, we must show that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \preceq^*_\rho t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\sigma_\perp} s'$ and $\Gamma, x : \sigma \vdash r \preceq^*_\rho r'$ for some terms s'

and r'. Note that $\Gamma \vdash \text{case}(s)\ \text{of}\ \text{up}(x).r \preceq^*_\rho t'$ holds provided that $\Gamma \vdash \text{case}(s')\ \text{of}\ \text{up}(x).r' \preceq^\circ_\rho t'$. By Proposition 7.4.1(2), it follows that

$(\Gamma \vdash \text{case}(s')\ \text{of}\ \text{up}(x).r' \preceq^\circ_\rho t' \ \wedge\ \Gamma \vdash t' \preceq^\circ_\rho t'') \implies$
$\Gamma \vdash \text{case}(s')\ \text{of}\ \text{up}(x).r' \preceq^\circ_\rho t''$.

The required result follows from the inference rule ($\preceq^*$ case up)

$$\frac{\Gamma \vdash s \preceq^*_{\sigma_\perp} s' \quad \Gamma, x:\sigma \vdash r \preceq^*_\rho r'}{\Gamma \vdash \text{case}(s)\ \text{of}\ \text{up}(x).r \preceq^*_\rho t''}(\text{if } \Gamma \vdash \text{case}(s')\ \text{of}\ \text{up}(x).r \preceq^\circ_\rho t'').$$

(10) ($\preceq^*$ unfold)
Given that $\Gamma \vdash \text{unfold}(s) \preceq^*_{\sigma[\mu X.\sigma/X]} t'$ and $\Gamma \vdash t' \preceq^*_{\sigma[\mu X.\sigma/X]} t''$, we must show that $\Gamma \vdash \text{unfold}(s) \preceq^*_{\sigma[\mu X.\sigma/X]} t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\mu X.\sigma} s'$ for some s'. Note that $\Gamma \vdash \text{unfold}(s) \preceq^*_{\sigma[\mu X.\sigma/X]} t'$ holds provided $\Gamma \vdash \text{unfold}(s') \preceq^\circ_{\sigma[\mu X.\sigma/X]} t'$. By Proposition 7.4.1(2), it follows that

$$(\Gamma \vdash \text{unfold}(s') \preceq^\circ_{\mu X.\sigma} t' \wedge \Gamma \vdash t' \preceq^\circ_{\mu X.\sigma} t'') \implies \Gamma \vdash \text{unfold}(s') \preceq^\circ_{\mu X.\sigma} t''.$$

The required result then follows from the inference rule ($\preceq^*$ unfold)

$$\frac{\Gamma \vdash s \preceq^*_{\mu X.\sigma} s'}{\Gamma \vdash \text{unfold}(s) \preceq^*_{\sigma[\mu X.\sigma/X]} t''}(\text{if } \Gamma \vdash \text{unfold}(s') \preceq^\circ_{\sigma[\mu X.\sigma/X]} t'').$$

(11) ($\preceq^*$ fold)
Given that $\Gamma \vdash \text{fold}(s) \preceq^*_{\mu X.\sigma} t'$ and $\Gamma \vdash t' \preceq^*_{\mu X.\sigma} t''$, we must show that $\Gamma \vdash r' \preceq^*_{\mu X.\sigma} t''$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\sigma[\mu X.\sigma/X]} s'$ for some term s'. Note that $\Gamma \vdash t' \preceq^*_{\mu X.\sigma} t'$ holds provided $\Gamma \vdash \text{fold}(s') \preceq^\circ_{\mu X.\sigma} t'$. By Proposition 7.4.1(2), it then follows that

$$(\Gamma \vdash \text{fold}(s') \preceq^\circ_{\mu X.\sigma} t' \ \wedge\ \Gamma \vdash t' \preceq^\circ_{\mu X.\sigma} t'') \implies \Gamma \vdash \text{fold}(s') \preceq^\circ_{\mu X.\sigma} t''.$$

The required result then follows from the inference rule ($\preceq^*$ fold)

$$\frac{\Gamma \vdash s \preceq^*_{\sigma[\mu X.\sigma/X]} s'}{\Gamma \vdash \text{fold}(s) \preceq^*_{\mu X.\sigma} t''}(\text{if } \Gamma \vdash \text{fold}(s') \preceq^\circ_{\sigma[\mu X.\sigma/X]} t'').$$

The proof by induction is now complete. □

Proof of Lemma 8.2.1(2).
We aim to prove that:
If $\Gamma \vdash t : \sigma$, then $\Gamma \vdash t \preceq^*_\sigma t$.
The proof proceeds by induction on the derivation of $\Gamma \vdash t : \sigma$.

(1) ($\vdash$ var)
Given that $\Gamma', x : \sigma \vdash x : \sigma$, we must show that $\Gamma', x : \sigma \vdash x \preceq^*_\sigma x$. By Proposition 7.4.1(1), we have $\Gamma', x : \sigma \vdash x \preceq^\circ_\sigma x$. Hence, by the axiom ($\preceq^*$ var), we have $\Gamma', x : \sigma \vdash x \preceq^*_\sigma x$.

(2) ($\vdash$ abs)
Given that $\Gamma \vdash \lambda x.t : \sigma \to \tau$, we must show that $\Gamma \vdash \lambda x.t \preceq^*_{\sigma\to\tau} \lambda x.t$. The induction hypothesis asserts that $\Gamma, x : \sigma \vdash t \preceq^*_\tau t$. Note that $\Gamma \vdash \lambda x.t \preceq^\circ \lambda x.t$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ abs), we have $\Gamma \vdash \lambda x.t \preceq^*_{\sigma\to\tau} \lambda x.t$.

(3) ($\vdash$ app)
Given that $\Gamma \vdash f(s) : \tau$, we must show that $\Gamma \vdash f(s) \preceq^*_\tau f(s)$. The induction hypothesis asserts that $\Gamma \vdash f \preceq^*_{\sigma\to\tau} f$ and $\Gamma \vdash s \preceq^*_\sigma s$. Note that $\Gamma \vdash f(s) \preceq^\circ_\tau f(s)$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ app), we have $\Gamma \vdash f(s) \preceq^*_\tau f(s)$.

(4) ($\vdash$ pair)
Given that $\Gamma \vdash (s,t) : \sigma \times \tau$, we must show that $\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} (s,t)$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_\sigma s$ and $\Gamma \vdash t \preceq^*_\tau t$. By Proposition 7.4.1(1), we have $\Gamma \vdash (s,t) \preceq^\circ_{\sigma\times\tau} (s,t)$. Hence, by the inference rule ($\preceq^*$ pair), we have $\Gamma \vdash (s,t) \preceq^*_{\sigma\times\tau} (s,t)$.

(5) ($\vdash$ fst, snd)
Given that $\Gamma \vdash \mathrm{fst}(p) : \sigma$, we must show that $\Gamma \vdash \mathrm{fst}(p) \preceq^*_\sigma \mathrm{fst}(p)$. The induction hypothesis asserts that $\Gamma \vdash p \preceq^*_{\sigma\times\tau} p$. By Proposition 7.4.1(1), it follows that $\Gamma \vdash \mathrm{fst}(p) \preceq^\circ_\sigma \mathrm{fst}(p)$. Hence, by the inference rule ($\preceq^*$ fst), we have $\Gamma \vdash \mathrm{fst}(p) \preceq^*_\sigma \mathrm{fst}(p)$.
Similarly, the case ($\vdash$ snd) holds.

(6) ($\vdash$ up)
Given that $\Gamma \vdash \mathrm{up}(t) : \sigma_\perp$, we must show that $\Gamma \vdash \mathrm{up}(t) \preceq^*_{\sigma_\perp} \mathrm{up}(t)$. The induction hypothesis asserts that $\Gamma \vdash t \preceq^*_\sigma t$. Note that $\Gamma \vdash \mathrm{up}(t) \preceq^\circ_{\sigma_\perp} \mathrm{up}(t)$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ up), we have $\Gamma \vdash \mathrm{up}(t) \preceq^*_{\sigma_\perp} \mathrm{up}(t)$.

(7) ($\vdash$ case up)
Given that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t : \rho$, we must show that
$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \preceq^*_\rho \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\sigma_\perp} s$ and $\Gamma, x : \sigma \vdash t \preceq^*_\rho t$. Note that $\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \preceq^\circ_\rho \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ case up), we have
$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t \preceq^*_\rho \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).t$.

(8) ($\vdash$ inl, inr)
Given that $\Gamma \vdash \mathrm{inl}(t) : \sigma + \tau$, we must show that $\Gamma \vdash \mathrm{inl}(t) \preceq^*_{\sigma+\tau} \mathrm{inl}(t)$. The induction hypothesis asserts that $\Gamma \vdash t \preceq^*_\sigma t$. Note that $\Gamma \vdash \mathrm{inl}(t) \preceq^\circ_{\sigma+\tau} \mathrm{inl}(t)$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ inl), we have $\Gamma \vdash \mathrm{inl}(t) \preceq^*_{\sigma+\tau} \mathrm{inl}(t)$.
Similarly, the case ($\vdash$ inr) holds.

(9) ($\vdash$ case)
Given that $\Gamma \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 : \rho$, we must show that $\Gamma \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \preceq^*_\rho \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2$. The induction hypothesis asserts that $\Gamma \vdash s \preceq^*_{\sigma+\tau} s$, $\Gamma, x : \sigma \vdash t_1 \preceq^*_\rho t_1$ and $\Gamma, y : \tau \vdash t_2 \preceq^*_\rho t_2$. Note that $\Gamma \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \preceq^\circ_\rho \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ case), we have $\Gamma \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \preceq^*_\rho \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2$.

(10) ($\vdash$ unfold)
Given that $\Gamma \vdash \mathrm{unfold}(t) : \sigma[\mu X.\sigma/X]$, we must show that $\Gamma \vdash \mathrm{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t)$. The induction hypothesis asserts that $\Gamma \vdash t \preceq^*_{\mu X.\sigma} t$. Note that $\Gamma \vdash \mathrm{unfold}(t) \preceq^\circ_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t)$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ unfold), we have $\Gamma \vdash \mathrm{unfold}(t) \preceq^*_{\sigma[\mu X.\sigma/X]} \mathrm{unfold}(t)$.

(10) ($\vdash$ fold)
Given that $\Gamma \vdash \mathrm{fold}(t) : \mu X.\sigma$, we must show that $\Gamma \vdash \mathrm{fold}(t) \preceq^*_{\mu X.\sigma} \mathrm{fold}(t)$. The induction hypothesis asserts that $\Gamma \vdash t \preceq^*_{\sigma[\mu X.\sigma/X]} t$. Note that $\Gamma \vdash \mathrm{fold}(t) \preceq^\circ_{\mu X.\sigma} \mathrm{fold}(t)$ by Proposition 7.4.1(1). Hence, by the inference rule ($\preceq^*$ fold), we have $\Gamma \vdash \mathrm{fold}(t) \preceq^*_{\mu X.\sigma} \mathrm{fold}(t)$.

□

Proof of Lemma 8.2.1. (3)
We aim to prove that:
If $\Gamma \vdash t \preceq^\circ_\sigma t'$, then $\Gamma \vdash t \preceq^*_\sigma t'$.
Note that by (2), $\Gamma \vdash t \preceq^*_\sigma t$ holds whenever $\Gamma \vdash t : \sigma$. Thus by (1), $(\Gamma \vdash t \preceq^*_\sigma t \ \wedge\ \Gamma \vdash t \preceq^\circ_\sigma t') \implies \Gamma \vdash t \preceq^*_\sigma t'$. □

Proof of Lemma 8.2.1. (4)
We aim to prove that:
If $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash s \preceq^*_\tau s'$, then $\Gamma \vdash s[t/x] \preceq^*_\tau s'[t'/x]$.
We prove this by induction on the derivation of $\Gamma, x : \sigma \vdash s \preceq^*_\sigma s'$.

(1) ($\preceq^*$ var)
Case 1: $\Gamma, x : \sigma \vdash x \preceq^*_\sigma s'$.
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash x \preceq^*_\sigma s'$, we must show that

$$\Gamma \vdash t \preceq^*_\sigma s'[t'/x].$$

Firstly, from the axiom ($\preceq^*$ var), one notes that $\Gamma, x : \sigma \vdash x \preceq^*_\sigma s'$ holds only if $\Gamma, x : \sigma \vdash x \preceq^\circ_\sigma s'$. Secondly, by Proposition 7.4.1(1), it holds that $\Gamma \vdash t' \preceq^\circ_\sigma t'$. It then follows from Lemma 7.4.2 that $\Gamma \vdash t' \preceq^\circ_\sigma s'[t'/x]$. By Lemma 8.2.1(1), we have

$$(\Gamma \vdash t \preceq^*_\sigma t' \ \wedge \ \Gamma \vdash t' \preceq^\circ_\sigma s'[t'/x]) \implies \Gamma \vdash t \preceq^*_\sigma s'[t'/x].$$

Case 2: $\Gamma, x : \sigma \vdash y \preceq^*_\tau s'$ where y is a term variable distinct from x.
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash y \preceq^*_\tau s'$, we must prove that

$$\Gamma \vdash y \preceq^*_\tau s'[t'/x].$$

From the axiom ($\preceq^*$ var), one notes that $\Gamma, x : \sigma \vdash y \preceq^*_\tau s'$ holds only if $\Gamma, x : \sigma \vdash y \preceq^\circ_\tau s'$. By Proposition 7.4.1(1), it holds that $\Gamma \vdash t' \preceq^\circ_\sigma t'$. Thus, by Lemma 7.4.2, it follows that $\Gamma \vdash y \preceq^\circ_\tau s'[t'/x]$. Lemma 8.2.1(2) ensures that $\Gamma \vdash y \preceq^*_\tau y$ always holds. By Lemma 8.2.1(1), we have

$$(\Gamma \vdash y \preceq^*_\tau y \ \wedge \ \Gamma \vdash y \preceq^\circ_\tau s'[t'/x]) \implies \Gamma \vdash y \preceq^*_\tau s'[t'/x].$$

(2) ($\preceq^*$ abs)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \lambda y.r \preceq^*_{\rho\to\tau} s'$, we must show that

$$\Gamma \vdash \lambda y.r[t/x] \preceq^*_{\rho\to\tau} s'[t'/x].$$

Since $\Gamma \vdash \lambda y.r \preceq^*_{\rho\to\tau} s'$, it follows from the inference rule ($\preceq^*$ abs) that $\Gamma, y : \rho \vdash r \preceq^*_\tau r'$ for some term r' with $\Gamma \vdash \lambda x.r' \preceq^\circ_{\rho\to\tau} s'$. The induction hypothesis then asserts that $\Gamma, y : \rho \vdash r[t/x] \preceq^*_\tau r'[t'/x]$. Note that one always has $\Gamma \vdash t' \preceq^\circ_\sigma t'$ by Proposition 7.4.1(1). Thus, by Lemma 7.4.2(2), we have

$$(\Gamma, x : \sigma \vdash \lambda y.r' \preceq^\circ_{\rho\to\tau} s' \wedge \Gamma \vdash t \preceq^\circ t') \implies \Gamma \vdash \lambda y.r'[t'/x] \preceq^\circ_{\rho\to\sigma} s'[t'/x].$$

Since $\Gamma, y : \rho \vdash r[t/x] \preceq^*_\tau r'[t'/x]$ and $\Gamma \vdash \lambda y.r'[t'/x] \preceq^\circ_{\rho\to\tau} s'[t'/x]$, it

follows from the inference rule ($\preceq^*$ abs) that

$$\Gamma \vdash \lambda y.r[t/x] \preceq^*_{\rho\to\tau} s'[t'/x].$$

(3) ($\preceq^*$ app)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash f(r) \preceq^*_\tau s'$, we must show that

$$\Gamma \vdash f(r)[t/x] \preceq^*_\tau s'[t'/x].$$

Since $\Gamma \vdash f(r) \preceq^*_\tau s'$, it follows from the inference rule ($\preceq^*$ app) that $\Gamma, x : \sigma \vdash f \preceq^*_{\rho\to\tau} f'$ and $\Gamma, x : \sigma \vdash r \preceq^*_\rho r'$ for some terms f' and r' with $\Gamma, x : \sigma \vdash f'(r') \preceq^\circ_\tau s'$. The induction hypothesis asserts that $\Gamma \vdash f[t/x] \preceq^*_{\rho\to\tau} f'[t'/x]$ and $\Gamma \vdash r[t/x] \preceq^*_\tau r'[t'/x]$. Since $\Gamma \vdash t' \preceq^\circ t'$ by Proposition 7.4.1(1) and $\Gamma, x : \sigma \vdash f'(r') \preceq^\circ_\tau s'$, it follows from Lemma 7.4.2 that $\Gamma \vdash f'(r')[t'/x] \preceq^\circ_\tau s'[t'/x]$. Thus it follows from the inference rule ($\preceq^*$ app) that

$$\Gamma \vdash f(r)[t/x] \preceq^*_\tau s'[t'/x].$$

(4) ($\preceq^*$ pair)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash (m_1, m_2) \preceq^*_{\tau_1\times\tau_2} s'$, we must show that

$$\Gamma \vdash (m_1, m_2)[t/x] \preceq^*_{\tau_1\times\tau_2} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash (m_1, m_2) \preceq^*_{\tau_1\times\tau_2} s'$, it follows from the inference rule ($\preceq^*$ pair) that $\Gamma, x : \sigma \vdash m_1 \preceq^*_{\tau_1} m'_1$ and $\Gamma, x : \sigma \vdash m_2 \preceq^*_{\tau_2} m'_2$ for some terms m'_1 and m'_2 with $\Gamma, x : \sigma \vdash (m'_1, m'_2) \preceq^\circ_{\tau_1\times\tau_2} s'$. On one hand, the induction hypothesis then asserts that $\Gamma \vdash m_1[t/x] \preceq^*_{\tau_1} m'_1[t'/x]$ and $\Gamma \vdash m_2[t/x] \preceq^*_{\tau_2} m'_2[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ always holds by Proposition 7.4.1, it follows that $\Gamma \vdash (m'_1, m'_2)[t'/x] \preceq^\circ_{\tau_1\times\tau_2} s'[t'/x]$. Finally, it follows from the inference rule ($\preceq^*$ pair) that

$$\Gamma \vdash (m_1, m_2)[t/x] \preceq^*_{\tau_1\times\tau_2} s'[t'/x].$$

(5) ($\preceq^*$ fst, snd)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \mathrm{fst}(p) \preceq^*_{\tau_1} s'$, we must show that

$$\Gamma \vdash \mathrm{fst}(p)[t/x] \preceq^*_{\tau_1} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash \mathrm{fst}(p) \preceq^*_{\tau_1} s'$, it follows from the inference rule that $\Gamma, x : \sigma \vdash p \preceq^*_{\tau_1\times\tau_2} p'$ for some term p' with $\Gamma, x : \sigma \vdash \mathrm{fst}(p') \preceq^\circ_{\tau_1} s'$. On one hand, the induction hypothesis then asserts that $\Gamma \vdash p[t/x] \preceq^*_{\tau_1}$

$p'[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ holds by Proposition 7.4.1(1) and that $\Gamma, x \vdash \mathrm{fst}(p') \preceq^\circ_{\tau_1} s'$, it follows by Lemma 7.4.2 that $\Gamma \vdash \mathrm{fst}(p')[t'/x] \preceq^\circ s'[t'/x]$. Finally, by the inference rule ($\preceq^*$ fst), one has

$$\Gamma \vdash \mathrm{fst}(p)[t/x] \preceq^*_{\tau_1} s'[t'/x].$$

Similarly, the case ($\preceq^*$ snd) holds.

(6) ($\preceq^*$ inl, inr)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \mathrm{inl}(r) \preceq^*_{\tau_1+\tau_2} s'$, we must show that

$$\Gamma \vdash \mathrm{inl}(r)[t/x] \preceq^*_{\tau_1+\tau_2} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash \mathrm{inl}(r) \preceq^*_{\tau_1+\tau_2} s'$ holds, it follows from the inference rule ($\preceq^*$ inl) that $\Gamma, x : \sigma \vdash r \preceq^*_{\tau_1} r'$ for some term r' with $\Gamma, x : \sigma \vdash \mathrm{inl}(r') \preceq^\circ_{\tau_1+\tau_2} s'$. On one hand, the induction hypothesis asserts that $\Gamma, x : \sigma \vdash r[t/x] \preceq^*_{\tau_1} r'[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ holds by Proposition 7.4.1(1) and that $\Gamma, x : \sigma \vdash \mathrm{inl}(r') \preceq^\circ_{\tau_1+\tau_2} s'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{inl}(r')[t'/x] \preceq^\circ_{\tau_1+\tau_2} s'[t'/x]$. Finally, by the inference rule ($\preceq^*$ inl), it follows that

$$\Gamma \vdash \mathrm{inl}(r)[t/x] \preceq^*_{\tau_1+\tau_2} s'[t'/x].$$

Similarly, the case ($\preceq^*$ inr) holds.

(7) ($\preceq^*$ case)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, z : \sigma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho r'$, we must show that

$$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2[t/z] \preceq^*_\rho r'[t'/z].$$

Since $\Gamma, z : \sigma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2 \preceq^*_\rho r'$, it follows from the inference rule ($\preceq^*$ case) that $\Gamma, z : \sigma \vdash s \preceq^*_{\tau_1+\tau_2} s'$, $\Gamma, z : \sigma, x : \tau_1 \vdash t_1 \preceq^*_\rho t'_1$ and $\Gamma, z : \sigma, y : \tau_2 \vdash t_2 \preceq^*_\rho t'_2$ for some terms s', t'_1, t'_2 with $\Gamma, z : \sigma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho r'$. On one hand, the induction hypothesis then asserts that $\Gamma \vdash s[t/z] \preceq^*_{\tau_1+\tau_2} s'[t'/z]$, $\Gamma, x : \tau_1 \vdash t_1[t/z] \preceq^*_\rho t'_1[t'/z]$ and $\Gamma, y : \tau_2 \vdash t_2[t/z] \preceq^*_\rho t'_2[t'/z]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ holds by Proposition 7.4.1(1) and that $\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2 \preceq^\circ_\rho r'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{inl}(x).t'_1\ \mathrm{or}\ \mathrm{inr}(y).t'_2[t'/z] \preceq^\circ_\rho r'[t'/z]$. Finally, applying the inference rule ($\preceq^*$ case), one has

$$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{inl}(x).t_1\ \mathrm{or}\ \mathrm{inr}(y).t_2[t/z] \preceq^*_\rho r'[t'/z].$$

(8) ($\preceq^*$ up)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \mathrm{up}(r) \preceq^*_{\tau_\perp} s'$, we must show that

$$\Gamma \vdash \mathrm{up}(r)[t/x] \preceq^*_{\tau_\perp} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash \mathrm{up}(r) \preceq^*_{\tau_\perp} s'$ holds, it follows from the inference rule that $\Gamma, x : \sigma \vdash r \preceq^*_\tau r'$ for some term r' with $\Gamma, x : \sigma \vdash \mathrm{up}(r') \preceq^\circ_\tau s'$. On one hand, the induction hypothesis asserts that $\Gamma \vdash r[t/x] \preceq^*_{\tau_\perp} r'[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ by Proposition 7.4.1(1) and that $\Gamma \vdash \mathrm{up}(r') \preceq^\circ s'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{up}(r')[t'/x] \preceq^\circ_{\tau_\perp} s'[t'/x]$. Finally, by the inference rule ($\preceq^*$ up), we have

$$\Gamma \vdash \mathrm{up}(r)[t/x] \preceq^*_{\tau_\perp} s'[t'/x].$$

(9) ($\preceq^*$ case up)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, z : \sigma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).u \preceq^*_\rho r'$, we must show that

$$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).u[t/x] \preceq^*_\rho r'[t'/z].$$

Since $\Gamma, z : \sigma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).u \preceq^*_\rho r'$, it follows from the inference rule ($\preceq^*$ case up) that $\Gamma, z : \sigma \vdash s \preceq^*_{\tau_\perp} s'$ and $\Gamma, z : \sigma, x : \tau \vdash u \preceq^*_\rho u'$ for some s' and u' such that $\Gamma, z : \sigma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).u' \preceq^\circ_\rho r'$. On one hand, the induction hypothesis asserts that $\Gamma \vdash s[t/z] \preceq^*_{\tau_\perp} s'[t'/z]$ and $\Gamma, x : \tau \vdash u[t/z] \preceq^*_\rho u'[t'/z]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ holds by Proposition 7.4.1(1) and that $\Gamma, z : \sigma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).u' \preceq^\circ_\rho r'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).u'[t'/x] \preceq^\circ_\rho r'[t'/x]$. Finally, by the inference rule ($\preceq^*$ case up), it follows that

$$\Gamma \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).u[t/x] \preceq^*_\rho r'[t'/x].$$

(10) ($\preceq^*$ unfold)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \mathrm{unfold}(r) \preceq^*_{\tau[\mu X.\tau/X]} s'$, we must show that

$$\Gamma \vdash \mathrm{unfold}(r)[t/x] \preceq^*_{\tau[\mu X.\tau/X]} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash \mathrm{unfold}(r) \preceq^*_{\tau[\mu X.\tau/X]} s'$ holds, it follows from the inference rule ($\preceq^*$ unfold) that $\Gamma, x : \sigma \vdash r \preceq^*_{\mu X.\tau} r'$ for some term r' with $\Gamma, x : \sigma \vdash \mathrm{unfold}(r') \preceq^\circ_{\tau\mu X.\tau/X]} s'$. On one hand, the induction hypothesis asserts that $\Gamma \vdash r[t/x] \preceq^*_{\mu X.\tau} r'[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ by Proposition 7.4.1(1) and that $\Gamma, x : \sigma \vdash \mathrm{unfold}(r') \preceq^\circ_{\tau[\mu X.\tau/X]} s'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{unfold}(r')[t'/x] \preceq^\circ_{\tau[\mu X.\tau/X]} s'[t'/x]$. Finally, by the inference rule ($\preceq^*$

unfold), we have

$$\Gamma \vdash \mathrm{unfold}(r)[t/x] \preceq^*_{\tau[\mu X.\tau/X]} s'[t'/x].$$

(11) ($\preceq^*$ fold)
Given that $\Gamma \vdash t \preceq^*_\sigma t'$ and $\Gamma, x : \sigma \vdash \mathrm{fold}(r) \preceq^*_{\mu X.\tau} s'$, we must show that

$$\Gamma \vdash \mathrm{fold}(r)[t/x] \preceq^*_{\mu X.\tau} s'[t'/x].$$

Since $\Gamma, x : \sigma \vdash \mathrm{fold}(r) \preceq^*_{\mu X.\tau]} s'$ holds, it follows from the inference rule ($\preceq^*$ unfold) that $\Gamma, x : \sigma \vdash r \preceq^*_{\tau[\mu X.\tau/X]} r'$ for some term r' with $\Gamma, x : \sigma \vdash \mathrm{fold}(r') \preceq^\circ_{\mu X.\tau} s'$. On one hand, the induction hypothesis asserts that $\Gamma \vdash r[t/x] \preceq^*_{\tau[\mu X.\tau/X]} r'[t'/x]$. On the other hand, since $\Gamma \vdash t' \preceq^\circ_\sigma t'$ by Proposition 7.4.1(1) and that $\Gamma, x : \sigma \vdash \mathrm{fold}(r') \preceq^\circ_{\mu X.\tau} s'$, it follows from Lemma 7.4.2 that $\Gamma \vdash \mathrm{fold}(r')[t'/x] \preceq^\circ_{\mu X.\tau} s'[t'/x]$. Finally, by the inference rule ($\preceq^*$ unfold), we have

$$\Gamma \vdash \mathrm{fold}(r)[t/x] \preceq^*_{\mu X.\tau} s'[t'/x].$$

The proof is now complete. □

Proof of Proposition 8.2.2.
We aim to show that:
If $\Gamma \vdash s \preceq^*_\sigma t$ and $s \Downarrow v$, then $\Gamma \vdash v \preceq^*_\sigma t$.
We proceed by induction on the derivation of $s \Downarrow v$.

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

(2) ($\Downarrow$ app)
Given that $\emptyset \vdash f(r) \preceq^*_\tau t$ and $f(r) \Downarrow v$, we must show that

$$\emptyset \vdash f(r) \preceq^*_\tau v.$$

Since $\emptyset \vdash f(r) \preceq^*_\tau t$, it follows from the inference rule ($\preceq^*$ app) that $\emptyset \vdash f \preceq^*_{\sigma\to\tau} f'$ and $\emptyset \vdash r \preceq^*_\tau r'$ with $f'(r') \preceq_\tau t$. Note that $f(r) \Downarrow v$ is derived from $f \Downarrow \lambda x.s$ and $s[r/x] \Downarrow v$. By the induction hypothesis, it follows that $\emptyset \vdash \lambda x.s \preceq^*_{\sigma\to\tau} f'$. From the inference rule ($\preceq^*$ abs), it must be that $x : \sigma \vdash s \preceq^*_\tau s'$ for some term s' with $\Gamma \vdash \lambda x.s' \preceq^\circ_{\sigma\to\tau} f'$. Now applying Lemma 8.2.1(4), we have $\emptyset \vdash s[r/x] \preceq^*_\tau s'[r'/x]$. Since $\emptyset \vdash s[r/x] \preceq^*_\tau s'[r'/x]$ and $s[r/x] \Downarrow v$, it follows from the induction hypothesis that $\emptyset \vdash v \preceq^*_\tau s'[r'/x]$. Because $\preceq$ is an FPC simulation and $\lambda x.s' \preceq_{\sigma\to\tau} f'$, it follows from (sim 3) that $\lambda x.s'(r') \preceq_\tau f'(r')$. Note that by definition of $\sqsubseteq^{kl}$, we always have $s'[r'/x] \sqsubseteq^{kl} (\lambda x.s')(r')$ and hence by

Proposition 7.5.1, $s'[r'/x] \preceq_\tau (\lambda x.s')(r')$. Thus, by transitivity, we have $\emptyset \vdash s'[r'/x] \preceq^\circ_\tau t$. Finally, since $\emptyset \vdash v \preceq^*_\tau s'[r'/x]$ and $\emptyset \vdash s'[r'/x] \preceq^\circ_\tau t$, it follows from Lemma 8.2.1(1) that $\emptyset \vdash v \preceq^*_\tau t$.

(3) ($\Downarrow$ fst, snd) Given that $\emptyset \vdash \mathrm{fst}(p) \preceq^*_\sigma t$ and $\mathrm{fst}(p) \Downarrow v$, we must show that

$$\emptyset \vdash v \preceq^*_\sigma t.$$

Since $\emptyset \vdash \mathrm{fst}(p) \preceq^*_\sigma t$, it follows from the inference rule ($\preceq^*$ fst) that $\emptyset \vdash p \preceq^*_{\sigma\times\tau} p'$ for some term p' with $\emptyset \vdash \mathrm{fst}(p') \preceq^\circ_{\sigma\times\tau} t$. Note that $\mathrm{fst}(p) \Downarrow v$ is derived from $p \Downarrow (m_1, m_2)$ and $s \Downarrow v$. The induction hypothesis then asserts that $\emptyset \vdash (m_1, m_2) \preceq^*_{\sigma\times\tau} p'$. But from the inference rule ($\preceq^*$ pair), it must be that $\emptyset \vdash m_1 \preceq^*_\sigma m'_1$ and $\emptyset \vdash m_2 \preceq^*_\tau m'_2$ with $\emptyset \vdash (m'_1, m'_2) \preceq^\circ_{\sigma\times\tau} p'$. By the induction hypothesis applied to $\emptyset \vdash m_1 \preceq^*_\sigma m'_1$ and $s \Downarrow v$, we have $\emptyset \vdash v \preceq^*_\sigma m'_1$. Now since $\preceq$ is an FPC simulation and $(m'_1, m'_2) \preceq_{\sigma\times\tau} p'$, it follows from (sim 4) that $\mathrm{fst}(m'_1, m'_2) \preceq_\sigma \mathrm{fst}(p')$. Note that by the definition of $\sqsubseteq^{kl}$, we always have $m'_1 \sqsubseteq^{kl}_\sigma \mathrm{fst}(m'_1, m'_2)$. Hence by Proposition 7.5.1, it holds that $m'_1 \preceq_\sigma \mathrm{fst}(m'_1, m'_2)$. Thus, by transitivity, we have $\emptyset \vdash m'_1 \preceq^\circ_\sigma t$. Finally, since $\emptyset \vdash v \preceq^*_\sigma m'_1$ and $\emptyset \vdash m_1 \preceq^\circ_\sigma t$, it then follows from Lemma 8.2.1(1) that $\emptyset \vdash v \preceq^*_\sigma t$.
Similarly, the case ($\preceq^*$ snd) holds.

(4) ($\Downarrow$ case inl)
Given that $\emptyset \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \preceq^*_\rho t$ and
$\mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \Downarrow v$, we must show that $\emptyset \vdash v \preceq^*_\rho t$. Since $\emptyset \vdash \mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \preceq^*_\rho t$ holds, it follows from the inference rule ($\preceq^*$ case inl) that $\emptyset \vdash s \preceq^*_{\sigma+\tau} s'$ and $x : \sigma \vdash t_1 \preceq^*_\rho t'_1$ and $y : \tau \vdash t_2 \preceq^*_\rho t'_2$ with $\emptyset \vdash \mathrm{case}(s')$ of $\mathrm{inl}(x).t'_1$ or $\mathrm{inr}(y).t'_2 \preceq^\circ_\rho t$. Note that for this case, the evaluation $\mathrm{case}(s)$ of $\mathrm{inl}(x).t_1$ or $\mathrm{inr}(y).t_2 \Downarrow v$ is derived from $s \Downarrow \mathrm{inl}(a)$ and $t_1[a/x] \Downarrow v$. The induction hypothesis applied to $\emptyset \vdash s \preceq^*_{\sigma+\tau} s'$ and $s \Downarrow \mathrm{inl}(a)$ then asserts that $\emptyset \vdash \mathrm{inl}(a) \preceq^*_{\sigma+\tau} s'$. But this holds provided that $\emptyset \vdash a \preceq^*_\sigma a'$ for some a' with $\emptyset \vdash \mathrm{inl}(a') \preceq^\circ_{\sigma+\tau} s'$. Since $\preceq$ is an FPC simulation and $\mathrm{inl}(a') \preceq_{\sigma+\tau} s'$, it follows from (sim 5) that there is $a'' : \sigma$ such that $s' \Downarrow \mathrm{inl}(a'')$ and $a' \preceq_\sigma a''$. Note that by the definition of $\sqsubseteq^{kl}$, we have $t'_1[a''/x] \sqsubseteq^{kl}_\rho \mathrm{case}(s')$ of $\mathrm{inl}(x).t'_1$ or $\mathrm{inr}(y).t'_2$ and hence by Lemma 7.5.1, it holds that $t'_1[a''/x] \preceq_\rho \mathrm{case}(s')$ of $\mathrm{inl}(x).t'_1$ or $\mathrm{inr}(y).t'_2$. Consequently, by transitivity, we have $t'_1[a''/x] \preceq_\rho t$. Because $\emptyset \vdash a \preceq^*_\sigma a'$ and $\emptyset \vdash a' \preceq^\circ_\sigma a''$, by Lemma 8.2.1(1) it holds that $\emptyset \vdash a \preceq^*_\sigma a''$. Now since $x : \sigma \vdash t_1 \preceq^*_\rho t'_1$, using Lemma 7.4.2 one deduces that $\emptyset \vdash t_1[a/x] \preceq^*_\rho t'_1[a''/x]$. Since $t_1[a/x] \Downarrow v$, the induction hypothesis then asserts that

$\emptyset \vdash v \preceq^*_\rho t'_1[a''/x]$. Finally, since $\emptyset \vdash t'_1[a''/x] \preceq^\circ_\rho t$, it follows from Lemma 8.2.1(1) that $\emptyset \vdash v \preceq^*_\rho t$.
Similarly, the case ($\preceq^*$ inr) holds.

(5) ($\Downarrow$ case up)
Given that $\emptyset \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \preceq^*_\rho t$ and $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \Downarrow v$, we must show that $\emptyset \vdash v \preceq^*_\rho t$. Since $\emptyset \vdash \mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \preceq^*_\rho t$ holds, it follows from the inference rule ($\preceq^*$ case up) that $\emptyset \vdash s \preceq^*_{\sigma_\perp} s'$ and $x : \sigma \vdash r \preceq^*_\rho r'$ with $\emptyset \vdash \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).r' \preceq^\circ_\rho t$. Note that for this case, the evaluation $\mathrm{case}(s)\ \mathrm{of}\ \mathrm{up}(x).r \Downarrow v$ is derived from $s \Downarrow \mathrm{up}(a)$ and $r[a/x] \Downarrow v$. The induction hypothesis applied to $\emptyset \vdash s \preceq^*_{\sigma_\perp} s'$ and $s \Downarrow \mathrm{up}(a)$ then asserts that $\emptyset \vdash \mathrm{up}(a) \preceq^*_{\sigma_\perp} s'$. But this holds provided that $\emptyset \vdash a \preceq^*_\sigma a'$ for some a' with $\emptyset \vdash \mathrm{up}(a') \preceq^\circ_{\sigma_\perp} s'$. Since $\preceq$ is an FPC simulation and $\mathrm{up}(a') \preceq_{\sigma_\perp} s'$, it follows from (sim 5) that there is $a'' : \sigma$ such that $s' \Downarrow \mathrm{up}(a'')$ and $a' \preceq_\sigma a''$. Note that by the definition of $\sqsubseteq^{kl}$, we have $r'[a''/x] \sqsubseteq^{kl}_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).r'$ and hence by Lemma 7.5.1, it holds that $r'[a''/x] \preceq_\rho \mathrm{case}(s')\ \mathrm{of}\ \mathrm{up}(x).r'$. Consequently, by transitivity, we have $r'[a''/x] \preceq_\rho t$. Because $\emptyset \vdash a \preceq^*_\sigma a'$ and $\emptyset \vdash a' \preceq^\circ_\sigma a''$, by Lemma 8.2.1(1) it holds that $\emptyset \vdash a \preceq^*_\sigma a''$. Now since $x : \sigma \vdash r \preceq^*_\rho r'$, using Lemma 7.4.2 one deduces that $\emptyset \vdash r[a/x] \preceq^*_\rho r'[a''/x]$. Since $r[a/x] \Downarrow v$, the induction hypothesis then asserts that $\emptyset \vdash v \preceq^*_\rho r'[a''/x]$. Finally, since $\emptyset \vdash r'[a''/x] \preceq^\circ_\rho t$, it follows from Lemma 8.2.1(1) that $\emptyset \vdash v \preceq^*_\rho t$.

(6) ($\Downarrow$ unfold)
Given that $\emptyset \vdash \mathrm{unfold}(s) \preceq^*_{\tau[\mu X.\tau/X]} t$ and $\mathrm{unfold}(s) \Downarrow v$, we must show that

$$\emptyset \vdash v \preceq^*_{\tau[\mu X.\tau/X]} t.$$

Since $\emptyset \vdash \mathrm{unfold}(s) \preceq^*_{\tau[\mu X.\tau/X]} t$ holds, it follows from the inference rule ($\preceq^*$ unfold) that $\emptyset \vdash s \preceq^*_{\mu X.\tau} s'$ for some s' with $\emptyset \vdash \mathrm{unfold}(s') \preceq^\circ_{\tau[\mu X.\tau/X]} t$. Note that $\mathrm{unfold}(s) \Downarrow v$ derives from $s \Downarrow \mathrm{fold}(r)$ and $r \Downarrow v$. Since $\emptyset \vdash s \preceq^*_{\mu X.\tau} s'$ and $s \Downarrow \mathrm{fold}(r)$, the induction hypothesis then asserts that $\emptyset \vdash \mathrm{fold}(r) \preceq^*_{\mu X.\tau} s'$. But this holds provided that $\emptyset \vdash r \preceq^*_{\tau[\mu X.\tau/X]} r'$ for some term r' with $\emptyset \vdash \mathrm{fold}(r') \preceq^\circ_{\mu X.\tau} s'$. Now since $\preceq$ is an FPC simulation and $\mathrm{fold}(r') \preceq_{\mu X.\tau} s'$, it follows from (sim 6) that $\emptyset \vdash \mathrm{unfold}(\mathrm{fold}(r)) \preceq^\circ_{\tau[\mu X.\tau/X]} \mathrm{unfold}(s')$. Note that by the definition of $\sqsubseteq^{kl}$, one always has $r' \sqsubseteq^{kl}_{\tau[\mu X.\tau/X]} \mathrm{unfold}(\mathrm{fold}(r'))$ and by Proposition 7.5.1, it holds that $\emptyset \vdash r' \preceq^\circ_{\tau[\mu X.\tau/X]} \mathrm{unfold}(s')$. Applying Lemma 8.2.1(1) to $\emptyset \vdash r \preceq^*_{\tau[\mu X.\tau/X]} r'$ and $\emptyset \vdash r' \preceq^\circ_{\tau[\mu X.\tau/X]} \mathrm{unfold}(s')$, we have $\emptyset \vdash r \preceq^*_{\tau[\mu X.\tau/X]} \mathrm{unfold}(s')$. Since $r \Downarrow v$, the induction hypothesis as-

serts that $\emptyset \vdash v \preceq^*_{\mu X.\tau}$ unfold(s'). Finally, since $\emptyset \vdash$ unfold$(s') \preceq^\circ_{\mu X.\tau} t$, it follows from Lemma 8.2.1(1) that $\emptyset \vdash v \preceq^*_{\tau[\mu X.\tau/X]} t$.

The proof is now complete. $\square$

Part III

Operational Domain Theory for PCF

In this part, we present an operational domain theory and topology for the language PCF. Rational chains and rational topology are dealt with in Chapter 9. In Chapter 10, an operational notion of finiteness is introduced and studied. An SFP-structure for PCF types is derived in that same chapter. In Chapter 11, compactness is revisited. We see how compactness interacts with finiteness. Note that the materials on saturated sets and well-filtered subspaces in Chapter 11 are new. In Chapter 12, reasoning principles developed in Chapters 9, 10 and 11 are applied to establish program correctness of some non-trivial PCF programs.

Chapter 9

Rational chains and rational topology

In this chapter, we show that rational chains are equivalent to programs defined on the "vertical natural number" type $\overline{\omega}$. A crucial step in the development of an operational domain theory for PCF is to replace the directed sets by rational chains. With this replacement, several classical results go through smoothly. The highlight in this chapter is that the open sets of any type (1) form a 'rational' topology and (2) are 'rationally' Scott-open.

9.1 Rationale for rational chains

In this section, we briefly discuss the need for using rational chains in the development of operational domain theory.

In pure PCF, there are types which fail to be chain complete because of computability reasons. This fact is well-known and appears, for instance, in the work of Mason, Smith & Talcott [36]. For clarity, we present the argument here.

Proposition 9.1.1. *In pure PCF, the contextual pre-order is not chain complete, i.e., there exist a type σ and a chain $D \subseteq \sigma$ with no least upper bound.*

Proof. Let $\phi : \mathbb{N} \to \mathbb{N}$ be a non-computable function. For each $k \in \mathbb{N}$, define the program $f_k : \mathtt{Nat} \to \mathtt{Nat}$ as follows:

$$f_k = \text{if } n \leq k \text{ then } \phi(n) \text{ else } \bot.$$

Note that the subset $D = \{f_k | k \in \mathbb{N}\} \subseteq (\mathtt{Nat} \to \mathtt{Nat})$ does not have a least upper bound because the least upper bound, if it existed, would have been contextually equivalent to ϕ. □

Note that this example 'disappears' if we move from PCF to PCF_Ω. However, Dag Normann recently constructed an explicit example of a chain in PCF which does not have a least upper bound in PCF_Ω but does have a bound in PCF^{++}. So, the least upper bound exists in some sense, and is computable, but not sequential. Therefore, PCF_Ω is also not chain complete. However, PCF_Ω^{++} is chain complete because it is equivalent to the Scott model of PCF (see, for example, Theorem 5.3.4). Because of the sheer complexity of its construction, Normann's example is beyond the scope of this thesis. But the interested reader may find it, together with detailed explanations, in Normann [39].

In view of the failure of chain completeness (with respect to contextual preorder) in PCF and PCF_Ω, we are forced to work with a restricted form of chain completeness, i.e., rational chain completeness, cf. Pitts [41].

9.2 Rational continuity

We begin by making a simple but crucial observation.

Lemma 9.2.1. *The sequence* $0, 1, \ldots, n, \ldots$ *in* $\overline{\omega}$ *is a rational chain with least upper bound* ∞*, and*

$$l(\infty) = \bigsqcup_n l(n) \text{ for every } l \in (\overline{\omega} \to \sigma).$$

Proof. $n = \mathrm{succ}^{(n)}(\bot)$ and $\infty = \mathrm{fix}(\mathrm{succ})$. □

Moreover, this is the "generic rational chain" with "generic lub" ∞ in the following sense:

Lemma 9.2.2. *A sequence* $x_n \in \sigma$ *is a rational chain if and only if there exists* $l \in (\overline{\omega} \to \sigma)$ *such that for all* $n \in \mathbb{N}$*,*

$$x_n = l(n)$$

and hence such that $\bigsqcup_n x_n = l(\infty)$*.*

Proof. ($\Rightarrow$): Given $g \in (\tau \to \tau)$ and $h \in (\tau \to \sigma)$ with $x_n = h(g^{(n)}(\bot))$, recursively define

$$f(y) = \text{if } y > 0 \text{ then } g(f(y-1)).$$

Then $f(n) = g^{(n)}(\bot)$ and hence we can take $l = h \circ f$.
($\Leftarrow$): Take $h = l$ and $g(y) = y + 1$. □

Elements of function type are *rationally continuous* in the following sense:

Proposition 9.2.3. *If $f \in (\sigma \to \tau)$ and x_n is a rational chain in σ, then*

(1) $f(x_n)$ is a rational chain in τ, and

(2) $f(\bigsqcup_n x_n) = \bigsqcup_n f(x_n)$.

Proof. By Lemma 9.2.2, there is $l \in (\overline{\omega} \to \sigma)$ such that $x_n = l(n)$. Then the definition $l'(y) = f(l(y))$ and the same lemma shows that $f(x_n)$ is a rational chain. By two applications of Lemma 9.2.1,

$$f(\bigsqcup_n x_n) = f(l(\infty)) = l'(\infty) = \bigsqcup_n l'(n) = \bigsqcup_n f(x_n).$$

□

Corollary 9.2.4. *For any rational chain $f_n \in (\sigma \to \tau)$ and any $x \in \sigma$,*

(1) $f_n(x)$ is a rational chain in τ, and

(2) $(\bigsqcup_n f_n(x)) = \bigsqcup_n f_n(x)$.

Proof. Apply Proposition 9.2.3 to $F \in ((\sigma \to \tau) \to \tau)$ defined by $F(f) = f(x)$. □

9.3 Rational topology

We say that a sequence of open sets in σ is a rational chain if the corresponding sequence of characteristic functions is rational in the function type $(\sigma \to \Sigma)$.

The following says that the open sets of any type form a *rational topology*:

Proposition 9.3.1. *For any type, the open sets are closed under the formation of finite intersections and rational unions.*

Proof. For the nullary intersection, define $\chi_{\bigcap \emptyset}(x) = \top$. For the binary intersection, define $\chi_{U \cap V}(x) = \chi_U(x) \wedge \chi_V(x)$ where $p \wedge q =$ if p then q.

For the rational unions, suppose $l \in (\omega \to (\sigma \to \Sigma))$ and $l(n) = \chi_{U_n}$. Then $l(\infty) = \bigsqcup_n \chi_{U_n}$. Because $U \subseteq V$ iff $\chi_U \sqsubseteq \chi_V$, we have that $\chi_{U_n} \sqsubseteq \chi_{\bigcup_n U_n}$. Again for the same reason, $U_n \subseteq V$ for all n iff $\chi_{\bigcup_n U_n} \sqsubseteq \chi_V$. Thus $\chi_{\bigcup_n U_n}$ is the least upper bound of the χ_{U_n}'s with respect to the contextual order, i.e., $\bigsqcup_n \chi_{U_n} = \chi_{\bigcup_n U_n}$ and hence $l(\infty) = \chi_{\bigcup_n U_n} \in (\sigma \to \Sigma)$ as desired. □

Open sets are *rationally Scott-open*:

Proposition 9.3.2. *For any open set U in a type σ,*

(1) if $x \in U$ and $x \sqsubseteq y$, then $y \in U$, and

(2) if x_n is a rational chain with $\bigsqcup_n x_n \in U$, then there is $n \in \mathbb{N}$ such that already $x_n \in U$.

Proof. (1) follows directly from Proposition 5.6.1. For (2), because x_n is a rational chain, there is $l \in (\overline{\omega} \to \sigma)$ such that $l(n) = x_n$ and $l(\infty) = \bigsqcup_n x_n$. Thus $\bigsqcup_n x_n \in U$ implies $\chi_U(l(\infty)) = \top$. By rational continuity of χ_U, we have $\bigsqcup_n \chi_U(l(n)) = \top$. So there is $n \in \mathbb{N}$ such that $\chi_U(l(n)) = \top$ and thus already $x_n = l(n) \in U$. □

Chapter 10

Finiteness and SFP-structure

Very often, in computer science and mathematics, an infinite entity is viewed as the limit of its finite components. For instance, every infinite set is the union of all its finite subsets. Identifying a common property satisfied by all its finite components can shed light on the behaviour of the infinite entity itself. This is usually carried out by verifying that the process of taking limits somehow preserves the property in question so that the "infinite" entity enjoys that common property. In many situations, it is possible to capture the notion of finiteness *without* mentioning cardinality at all. For example, Kuratowski's characterisation of finite subsets states that a set F is finite if and only if for every directed collection $\mathcal{G}$ of sets with $F \subseteq \bigcup \mathcal{G}$ there exists $G \in \mathcal{G}$ such that already $F \subseteq G$. This idea of capturing finiteness is picked up in classical domain theory, particularly in the study of algebraic dcpos (cf. Section 2.1.5).

In this chapter, we develop an appropriate notion of finiteness in our operational domain-theoretic setting. In addition to an SFP-style characterisation (Theorem 10.3.3), the novelty here is a topological characterisation of finiteness (Theorem 10.3.14). In this chapter, we also (1) develop a continuity principle for two special kinds of functions (Propositions 10.4.8 and 10.4.9), (2) define an ultrametric on PCF, and (3) prove an operational version of the Kleene-Kreisel density theorem for total elements (Theorem 10.6.3).

10.1 Finiteness

In view of Proposition 2.1.4 and Lemma 9.2.2, we are motivated to define operational finiteness as follows:

Definition 10.1.1. An element b is called (rationally) *finite* if every rational chain x_n with $b \sqsubseteq \bigsqcup x_n$, there is n such that already $b \sqsubseteq x_n$.

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

$$\mathrm{K}_\sigma := \{x : \sigma | x \text{ is finite}\}.$$

Examples 10.1.2. (1) For each type σ, $\bot_\sigma$ is trivially finite by definition.

(2) If x_n is a rational chain in Σ whose contextual supremum is $\top$, then there is $n \in \mathbb{N}$ such that $x_n = \top$. Thus $\top$ is finite. Similarly, all the numerals $\underline{n} \in \mathtt{Nat}$ are finite.

10.2 Rational algebraicity

The types of our language PCF are *rationally algebraic* in the following sense:

Theorem 10.2.1. *Every element of any type is the contextual supremum of a rational chain of finite elements.*

A proof of this will be given in Section 10.3. For the moment, we develop some consequences.

Corollary 10.2.2. *An element b is finite if and only if for every rational chain x_n with $b = \bigsqcup_n x_n$, there is n such that already $b = x_n$.*

Proof. ($\Longrightarrow$): If $b = \bigsqcup_n x_n$, then $b \sqsubseteq \bigsqcup_n x_n$ and hence $b \sqsubseteq x_n$ for some n. But by definition of upper bound, we also have $b \sqsupseteq x_n$. Hence $b = x_n$, as required.
($\Longleftarrow$): By Theorem 10.2.1, there is a rational chain of finite elements x_n with $b = \bigsqcup_n x_n$. By hypothesis, $b = x_n$ for some n, which shows that b is finite. □

Another easy consequence of Theorem 10.2.1 is:

Corollary 10.2.3. *Every element of any type has enough finite elements contextually below it, in the sense that for each $x \in \sigma$,*

$$x = \bigsqcup \{y \in \mathrm{K}_\sigma \,|\, y \sqsubseteq x\}.$$

Proof. By Theorem 10.2.1, there is a subset of $\mathrm{K}_\sigma \cap {\downarrow} x$ whose least upper bound is x. □

The following provides a proof method for contextual equivalence based on finite elements:

Proposition 10.2.4. *$f = g$ holds in $(\sigma \to \tau)$ iff $f(b) = g(b)$ for every finite $b \in \sigma$.*

Proof. ($\Longrightarrow$): Contextual equivalence is an applicative congruence by Corollary 3.7.2(1).
($\Longleftarrow$): By extensionality (cf. Corollary 3.7.2(2)), it suffices to show that $f(x) = g(x)$ for any $x \in \sigma$. By Theorem 10.2.1, there is a rational chain b_n of finite elements with $x = b_n$. Hence by two applications of rational continuity and the hypothesis that f and g agree on K_σ, we have

$$f(x) = f(\bigsqcup_n b_n) = \bigsqcup_n f(b_n) = \bigsqcup_n g(b_n) = g(\bigsqcup_n b_n) = g(x).$$

□

Remark 10.2.5. Of course, the above holds with contextual equivalence replaced by contextual order.

Another consequence of Theorem 10.2.1 is a continuity principle, which is reminiscent of the $\epsilon - \delta$ characterisation of continuity mentioned in Section 2.1.5.

Proposition 10.2.6. *For any $f \in (\sigma \to \tau)$, any $x \in \sigma$ and any finite $c \sqsubseteq f(x)$, there is a finite $b \sqsubseteq x$ with $c \sqsubseteq f(b)$.*

Proof. By Theorem 10.2.1, x is the least upper bound of a rational chain b_n of finite elements. By rational continuity, $c \sqsubseteq \bigsqcup_n f(b_n)$. By finiteness of c, there is n with $c \sqsubseteq f(b_n)$. □

As a result of the above, we have an operational version of a well-known result in domain theory:

Corollary 10.2.7. *If U is open and $x \in U$, then there is a finite $b \sqsubseteq x$ such that already $b \in U$.*

Proof. The hypothesis gives $\top \sqsubseteq \chi_U(x)$, and so there is some finite $b \sqsubseteq x$ with $\top \sqsubseteq \chi_U(b)$ because $\top$ is finite. To conclude, use the maximality of $\top$. □

10.3 Deflation and SFP structure

In order to prove Theorem 10.2.1, we need to invoke the following concepts:

Definition 10.3.1.

1. A *deflation* on a type σ is an element of type $(\sigma \to \sigma)$ that

   (i) is below id_σ the identity of σ, and

(ii) has finite image modulo contextual equivalence.

2. A (rational) *SFP structure* on a type σ is a rational chain id_n of idempotent deflations with $\bigsqcup_n \mathrm{id}_n^\sigma = \mathrm{id}_\sigma$, the identity of σ.

3. A type is (rationally) *SFP* if it has an SFP structure.

Let us pause here to construct a standard SFP[1] structure for each type. The construction of programs is defined by induction on types

$$\mathrm{d}^\sigma : \overline{\omega} \to (\sigma \to \sigma).$$

For the base case, we define:

$$\begin{array}{rcl}
\mathrm{d}^{\mathtt{Bool}}(n)(p) & = & p. \\
\mathrm{d}^{\Sigma}(n)(p) & = & p. \\
\mathrm{d}^{\mathtt{Nat}}(n)(k) & = & \text{if } n > 0 \text{ then } (\text{if } k \stackrel{?}{=} 0 \text{ then } 0 \text{ else succ } \mathrm{d}^{\mathtt{Nat}}(n-1)(\mathrm{pred}(k))). \\
\mathrm{d}^{\overline{\omega}}(n)(x) & = & \text{if } (n > 0 \wedge x > 0) \text{ then } 1 + \mathrm{d}^{\overline{\omega}}(n-1)(x-1).
\end{array}$$

For the induction step, we define

$$\begin{array}{rcl}
\mathrm{d}^{\sigma\to\tau}(n)(f)(x) & = & \mathrm{d}^\tau(n)(f(\mathrm{d}^\sigma(n)(x))). \\
\mathrm{d}^{\sigma\times\tau}(n)(x,y) & = & (\mathrm{d}^\sigma(n)(x), \mathrm{d}^\tau(n)(y)).
\end{array}$$

The above construction is instrumental in establishing the following fundamental lemma.

Lemma 10.3.2. *The rational chain* $\mathrm{id}_n^\sigma := \mathrm{d}^\sigma(n)$ *is an SFP structure on* σ *for every type* σ*.*

Proof. We prove by induction on σ.

(i) Base types.
($\sigma = \mathtt{Nat}$): We first show by induction on n that

$$\mathrm{id}_n^{\mathtt{Nat}}(x) = \begin{cases} x & \text{if } x < n \\ \bot & \text{otherwise.} \end{cases}$$

[1]The name "SFP" is an acronym for "Sequence of Finite Posets" which, in classical domain theory, arises from the fact that SFP domains are characterised as the bilimit of a sequence of finite posets, cf. Gunter & Scott [25].

The claim holds for $n = 0$ since $\mathrm{id}_0^{\mathtt{Nat}}(x) = \bot$ by definition. For the case $n = k + 1$, since $(k + 1 > 0) = \top$, we must have

$$\mathrm{id}_{k+1}^{\mathtt{Nat}}(x) = \text{if } x \stackrel{?}{=} 0 \text{ then } 0 \text{ else succ } \mathrm{id}_k^{\mathtt{Nat}}(\mathrm{pred}(x)).$$

Case (1): $x = 0$.
Then $\mathrm{id}_{k+1}^{\mathtt{Nat}}(x) = \mathrm{id}_{k+1}^{\mathtt{Nat}}(0) = 0 = x$.
Case (2): $x > 0$ and $x < k + 1$.
Then $\mathrm{pred}(x) < k$ and thus the induction hypothesis asserts that $\mathrm{id}_k^{\mathtt{Nat}}(\mathrm{pred}(x)) = \mathrm{pred}(x)$. Hence $\mathrm{id}_{k+1}^{\mathtt{Nat}} = \mathrm{succ}\ \mathrm{pred}(x) = x$.
Case (3): $x > 0$ and $x \geq k + 1$.
Then $\mathrm{pred}(x) \geq k$ and thus the induction hypothesis asserts that $\mathrm{id}_k^{\mathtt{Nat}}(\mathrm{pred}(x)) = \bot$. Hence $\mathrm{id}_{k+1}^{\mathtt{Nat}}(x) = \bot$.
In summary, we have shown that

$$\mathrm{id}_{k+1}^{\mathtt{Nat}}(x) = \begin{cases} x & \text{if } x < k + 1 \\ \bot & \text{otherwise.} \end{cases}$$

Hence $\mathrm{id}_n^{\mathtt{Nat}}$ is idempotent, below the identity and has finite image given by $\{0, 1, \ldots, n - 1\}$. Note that because $\infty = \infty - 1$, one can prove by a simple induction on x that $\mathrm{id}_\infty^{\mathtt{Nat}}(x) = x$. Since every non-divergent term of type $\mathtt{Nat}$ is contextually equivalent to n for some $n \in \mathbb{N}$, it follows from extensionality that $\mathrm{id}_\infty^{\mathtt{Nat}}$ is the identity.
($\sigma = \Sigma$): Trivial.
($\sigma = \overline{\omega}$): In fact, only this case is non-trivial. First we show by induction on n that, for every $n \in \mathbb{N}$,

$$\mathrm{d}^{\overline{\omega}}(n)(y) = \min(n, y).$$

The case holds when $n = 0$ since $\mathrm{d}^{\overline{\omega}}(0)(y) = 0$. Assuming that the case holds for $n = k + 1$, we proceed to show that it holds for $n = k + 1$. First suppose that $y = 0$, then $\mathrm{d}^{\overline{\omega}}(k + 1)(0) = 0 = \min(k + 1, 0)$. Now suppose that $y > 0$, then $\mathrm{d}^{\overline{\omega}}(k + 1)(y) = 1 + \mathrm{d}^{\overline{\omega}}(k)(y - 1)$. The induction hypothesis asserts that $\mathrm{d}^{\overline{\omega}}(k)(y - 1) = \min(k, y - 1)$. Thus $\mathrm{d}^{\overline{\omega}}(k+1)(y) = 1 + \min(k, y - 1)$. But $1 + \min(k, y - 1) = \min(k + 1, y)$ and thus it holds that

$$\mathrm{d}^{\overline{\omega}}(n, y) = \min(n, y)$$

for all $n, y \in \overline{\omega}$. Hence $\mathrm{d}^{\overline{\omega}}(n)$ is idempotent and below the identity, and has image $\{0, 1, \ldots, n\}$. Now calculate, for $k \in \mathbb{N}$, $\mathrm{d}^{\overline{\omega}}(\infty)(k) =$

$\bigsqcup_n \mathrm{d}^{\overline{\omega}}(n)(k) = \bigsqcup_n \min(n,k) = k$. Hence $\mathrm{d}^{\overline{\omega}}(\infty)(\infty) = \bigsqcup_k \mathrm{d}^{\overline{\omega}}(\infty)(k) = \bigsqcup_k k = \infty$. By extensionality, $\mathrm{d}^{\overline{\omega}}(\infty)$ is the identity.

(ii) Function type: $\sigma \to \tau$.
For any $f \in (\sigma \to \tau)$ and any $n \in \overline{\omega}$, it holds that

$$\begin{array}{ll} & \mathrm{id}_n^{\sigma\to\tau} \circ \mathrm{id}_n^{\sigma\to\tau}(f) \\ = & \mathrm{id}_n^{\tau} \circ (\mathrm{id}_n^{\tau} \circ f \circ \mathrm{id}_n^{\sigma}) \circ \mathrm{id}_n^{\sigma} \\ = & (\mathrm{id}_n^{\tau} \circ \mathrm{id}_n^{\tau}) \circ f \circ (\mathrm{id}_n^{\sigma} \circ \mathrm{id}_n^{\sigma}) \\ = & \mathrm{id}_n^{\tau} \circ f \circ \mathrm{id}_n^{\sigma} \qquad \text{(Ind. hyp.)} \\ = & \mathrm{id}_n^{\sigma\to\tau}(f) \end{array}$$

and for any $x \in \sigma$, we have

$$\begin{array}{ll} & (\mathrm{id}_n^{\sigma\to\tau}(f))(x) \\ = & (\mathrm{id}_n^{\tau} \circ f \circ \mathrm{id}_n^{\sigma})(x) \\ = & (\mathrm{id}_n^{\tau} \circ f)(\mathrm{id}_n^{\sigma}(x)) \\ \sqsubseteq & (\mathrm{id}_n^{\tau} \circ f)(x) \qquad \text{(by monotonicity and ind. hyp.)} \\ = & \mathrm{id}_n^{\tau}(f(x)) \\ \sqsubseteq & f(x) \qquad \text{(by ind. hyp.)} \\ = & (\mathrm{id}_{\sigma\to\tau}(f))(x) \end{array}$$

so that $\mathrm{id}_n^{\sigma\to\tau}$ is idempotent and below the identity. Also we have

$$\begin{array}{ll} & \mathrm{id}_\infty^{\sigma\to\tau}(f)(y) \\ = & \mathrm{d}^{\sigma\to\tau}(\infty)(f)(y) \\ = & \mathrm{d}^{\tau}(\infty)(f(\mathrm{d}^{\sigma}(\infty)(y))) \\ = & f(y) \qquad \text{(since } \mathrm{id}_\infty = \mathrm{id} \text{ by ind. hyp.)} \\ = & \mathrm{id}^{\sigma\to\tau}(f)(y). \end{array}$$

Let $f \in (\sigma \to \tau)$ be fixed. Consider the set map

$$\phi : \mathrm{id}_n^{\sigma\to\tau}[\sigma \to \tau] \to \mathrm{id}_n^{\tau}[\tau]^{\mathrm{id}_n^{\sigma}[\sigma]}, h \mapsto h.$$

Note that ϕ is well-defined. Moreover, ϕ is injective because $\phi(g) = \phi(h)$ iff their restrictions to $\mathrm{id}_n^{\sigma}[\sigma]$ are equal iff $g = h$. So it follows from

$$|\mathrm{id}_n^{\sigma\to\tau}[\sigma \to \tau]| \leq |\mathrm{id}_n^{\tau}[\tau]|^{|\mathrm{id}_n^{\sigma}[\sigma]|}$$

that $\mathrm{id}_n^{\sigma\to\tau}[\sigma \to \tau]$ is finite because the induction hypothesis asserts that $\mathrm{id}_n^{\sigma}[\sigma]$ and $\mathrm{id}_n^{\tau}[\tau]$ are finite sets.

(iii) Product type: $\sigma \times \tau$
For any $(x, y) \in (\sigma \times \tau)$ and any $n \in \overline{\omega}$, we have:

$$
\begin{array}{lll}
& \mathrm{id}_n^{\sigma\times\tau} \circ \mathrm{id}_n^{\sigma\times\tau}(x,y) & \\
= & \mathrm{id}_n^{\sigma\times\tau}(\mathrm{id}_n^{\sigma}(x), \mathrm{id}_n^{\tau}(y)) & \\
= & (\mathrm{id}_n^{\sigma} \circ \mathrm{id}_n^{\sigma}(x), \mathrm{id}_n^{\tau} \circ \mathrm{id}_n^{\tau}(y)) & \\
= & (\mathrm{id}_n^{\sigma}(x), \mathrm{id}_n^{\tau}(y)) & \text{(Ind. hyp.)} \\
= & \mathrm{id}_n^{\sigma\times\tau}(x,y) &
\end{array}
$$

and

$$
\begin{array}{lll}
& \mathrm{id}_n^{\sigma\times\tau}(x,y) & \\
= & (\mathrm{id}_n^{\sigma}(x), \mathrm{id}_n^{\tau}(y)) & \\
\sqsubseteq & (x,y) & \text{(ind. hyp.)} \\
= & \mathrm{id}_{\sigma\times\tau}(x,y) &
\end{array}
$$

so that $\mathrm{id}^{\sigma\times\tau}$ is idempotent and below the identity. In addition, we must have

$$
\begin{array}{lll}
& \mathrm{id}_\infty^{\sigma\times\tau}(x,y) & \\
= & (\mathrm{id}_\infty^{\sigma}(x), \mathrm{id}_\infty^{\tau}(y)) & \\
= & (x,y). & \text{(since } \mathrm{id}_\infty = \mathrm{id} \text{ by ind. hyp.)}
\end{array}
$$

The finiteness of the image set follows from

$$|\mathrm{id}_n^{\sigma\times\tau}[\sigma\times\tau]| = |\mathrm{id}^{\sigma}[\sigma]| \times |\mathrm{id}^{\tau}[\tau]|$$

and the induction hypothesis.

□

We obtain an SFP-style characterisation of rational finiteness.

Theorem 10.3.3. *(1) Each type of the language is SFP.*

(2) For any SFP structure id_n *on a type* σ*, an element* $b \in \sigma$ *is finite iff* $b = \mathrm{id}_n(b)$ *for some* $n \in \mathbb{N}$*.*

Proof. (1): Same as Lemma 10.3.2.
(2)($\Longrightarrow$): The inequality $b \sqsupseteq \mathrm{id}_n(b)$ holds because id_n is a deflation. For the other inequality, we first calculate $b = (\bigsqcup_n \mathrm{id}_n)(b) = \bigsqcup_n \mathrm{id}_n(b)$ using Corollary 9.2.4 and Lemma 10.3.2. Then by finiteness of b, there is n with $b \sqsubseteq \mathrm{id}_n(b)$.
(2)($\Longleftarrow$): To show that b is finite, let x_i be a rational chain with $b \sqsubseteq \bigsqcup_i x_i$. Then $b = \mathrm{id}_n(b) \sqsubseteq \mathrm{id}_n(\bigsqcup_i x_i) = \bigsqcup_i \mathrm{id}_n(x_i)$ by rational continuity of id_n. Because id_n has a finite image, the set $\{\mathrm{id}_n(x_i) | i \in \mathbb{N}\}$ is finite and hence has a maximum element, which is its least upper bound. That is, there is $i \in \mathbb{N}$ with $b \sqsubseteq \mathrm{id}_n(x_i)$. But $\mathrm{id}_n(x_i) \sqsubseteq x_i$ and hence $b \sqsubseteq x_i$, as required. □

This concludes the proof of Theorem 10.2.1.

Corollary 10.3.4. *For any $x \in \sigma$ and any SFP structure id_n on σ, the element $\mathrm{id}_n^\sigma(x)$ is finite.*

Proof. The result follows from the idempotence of id_n^σ and Theorem 10.3.3(2). □

Corollary 10.3.5. *Every finite ordinal is rationally finite.*

Proof. For any $i < \infty$ in $\overline{\omega}$, it holds that $\mathrm{id}_{i+1}^{\overline{\omega}}(i) = \min(i+1, i) = i$. The desired result then follows from Theorem 10.3.3(2). □

Corollary 10.3.6. *An element s of* `Baire` *is finite iff there is $k \in \mathbb{N}$ such that $s(i) = \bot$ for all $i > k$.*

Proof. If $s \in$ `Baire` is finite, then since $s = \mathrm{id}_n(s)$ for some $n \in \mathbb{N}$ it follows that if $k > n$ then $s(k) = \bot$. Conversely, if there is $k \in \mathbb{N}$ for which $s(i) = \bot$ for every $i > k$ then amongst the (at most k) non-divergent elements of $\{s(i) | i = 0, \ldots, k-1\}$ define the maximum of these to be m. Finally set $n = \max(m+1, k)$. Then $s = \mathrm{id}_n(s)$ and hence is finite by Theorem 10.3.3(2). □

Example 10.3.7. (Example 5.4.2 revisited.)
We want to prove that the set

$$T = \{s \in C | s(17) = 0\}$$

is not open in the Baire data type. Suppose T is open in `Baire`. Take any sequence $s \in B$ such that $s(17) = 0$. By Corollary 10.2.7, there is a finite $s' \sqsubseteq s$ such that $s' \in T$. By Corollary 10.3.6, s cannot be total, thus contradicting its membership in T.

We additionally have the following proposition.

Definition 10.3.8. By a *finitary type* we mean a type that is obtained from Σ and `Bool` by finitely many applications of the product- and function-type constructions.

Proposition 10.3.9. *SFP structures $\mathrm{id}_n^\sigma \in (\sigma \to \sigma)$ can be chosen for each type σ in such a way that*

(1) id_n^σ *is the identity for every finitary type σ,*

(2) $\mathrm{id}_n^{\sigma\to\tau}(f)(x) = \mathrm{id}_n^\tau(f(\mathrm{id}_n^\sigma(x)))$, *and*

(3) $\mathrm{id}_n^{\sigma\times\tau}(x,y) = (\mathrm{id}_n^\sigma(x), \mathrm{id}_n^\tau(y))$.

Proof. Since conditions (2) and (3) are immediate from the construction of the standard deflations, it remains to establish (1) by induction on finitary types.

(i) ($\sigma = \Sigma, \mathtt{Bool}$) Follows by definition.

(ii) Function type: $\sigma \to \tau$
For any $f \in (\sigma \to \tau)$ where σ, τ are finitary types, it holds that $\mathrm{id}_n^{\sigma\to\tau}(f) = \mathrm{id}_n^\tau \circ f \circ \mathrm{id}_n^\sigma = \mathrm{id}_\tau \circ f \circ \mathrm{id}_\sigma = f$.

(iii) Product type: $\sigma \times \tau$
For any $(x,y) \in \sigma \times \tau$ where σ, τ are finitary types, it holds that $\mathrm{id}_n^{\sigma\times\tau}(x,y) = (\mathrm{id}_n^\sigma(x), \mathrm{id}_n^\tau(y)) = (\mathrm{id}_\sigma(x), \mathrm{id}_\tau(y)) = (x,y)$.

□

Combined with Theorem 10.3.3(2), Proposition 10.3.9(1) gives:

Corollary 10.3.10. *Every element of any finitary type is finite.*

Proposition 10.3.11. *For any SFP structure* id_n *on a type* σ*, if* $\mathrm{id}_n(x) = x$ *then* $\mathrm{id}_k(x) = x$ *for any* $k \geq n$.

Proof. This follows immediately from $\mathrm{id}_n(x) \sqsubseteq \mathrm{id}_k(x) \sqsubseteq x$. □

Combining the above proposition with parts (2) and (3) of Proposition 10.3.9, we have

Corollary 10.3.12. *(1) If* $f \in (\sigma \to \tau)$ *and* $x \in \sigma$ *are finite, then so is* $f(x) \in \tau$.

(2) If $x \in \sigma$ *and* $y \in \tau$ *are finite, then so is* $(x,y) \in (\sigma \times \tau)$.

We now develop a topological characterisation of the notion of finiteness.

We say that an open set in σ has *finite characteristic* if its characteristic function is a finite element of the function type $(\sigma \to \Sigma)$.

Lemma 10.3.13. *For any open set* U *in* σ *and any* $n \in \mathbb{N}$*, let*

$$U^{(n)} := \mathrm{id}_n^{-1}(U) = \{x \in \sigma | \mathrm{id}_n(x) \in U\}.$$

(1) The open set $U^{(n)} \subseteq U$ *has a finite characteristic.*

(2) The set $\{U^{(n)} | U$ *is open in* $\sigma\}$ *has finite cardinality.*

(3) U has finite characteristic iff $U^{(n)} = U$ for some n.

(4) The chain $U^{(n)}$ is rational and $U = \bigcup_n U^{(n)}$.

Proof. (1) and (3): $\mathrm{id}_n(\chi_U)(x) = \mathrm{id}_n(\chi_U(\mathrm{id}_n(x))) = \chi_U(\mathrm{id}_n(x))$, and hence $\mathrm{id}_n(\chi_U)$ is the characteristic function of $U^{(n)}$.
(2): Any two equivalent characteristic functions classify the same open set and $\mathrm{id}_n^{\sigma\to\Sigma}$ has finite image modulo contextual equivalence.
(4): $\mathrm{id}_n(\chi_U)$ is a rational chain with least upper bound χ_U, i.e., $\chi_U(x) = \top$ iff $\mathrm{id}_n(\chi_U)(x) = \top$ for some n. □

Theorem 10.3.14. *An element $b \in \sigma$ is finite if and only if the set $\uparrow b := \{x \in \sigma | b \sqsubseteq x\}$ is open.*

Proof. ($\Longrightarrow$): By Proposition 5.6.1, for any $x \in \sigma$ we have

$$\uparrow x = \bigcap \{U | U \text{ is open and } x \in U\}.$$

Because b is finite, there is $n \in \mathbb{N}$ such that $\mathrm{id}_n(b) = b$. Hence if b belongs to an open set U then $b \in U^{(n)} \subseteq U$ by Lemma 10.3.14(1). This shows that

$$\uparrow b = \bigcap \{U^{(n)} | U \text{ is open and } b \in U\}.$$

But this is the intersection of a set of finite cardinality by Lemma 10.3.14(2) and hence open by Proposition 9.3.1.
($\Longleftarrow$): If $b \sqsubseteq \bigsqcup_n x_n$ holds for a rational chain x_n, then $\bigsqcup_n x_n \in \uparrow b$ and hence $x_n \in \uparrow b$ for some $n \in \mathbb{N}$ by Proposition 9.3.2(2), i.e., $b \sqsubseteq x_n$. □

Hence, by Corollary 10.2.7, the open sets $\uparrow b$ with b finite form a base of the (rational) topology.

Corollary 10.3.15. *Every open set is a union of open sets of the form $\uparrow b$ with b finite.*

Remark 10.3.16. (1) Notice that the proof of Theorem 10.3.14 ($\Longrightarrow$) is not constructive. The reason is that we implicitly use the fact that a subset of a finite set is finite. In general, however, it is not possible to finitely enumerate the members of a subset of a finite set unless the defining property of the subset is decidable.

(2) Moreover, this non-constructivity in the theorem is unavoidable[2]. In fact, if we had a constructive procedure for finding $\chi_{\uparrow b}$ for every finite

[2]The impossibility of a constructive proof of Theorem 10.3.14 was found together with Vincent Danos during a visit to our institution.

element b, then we would be able to semidecide contextual equivalence for finite elements, because $b = c$ iff $\chi_{\uparrow b}(c) = \top = \chi_{\uparrow c}(b)$. As all elements of finitary PCF are finite, and contextual equivalence is co-semidecidable for finitary PCF, this would give a decision procedure for equivalence, contradicting Loader [35].

10.4 A continuity principle

In this section, we consider a continuity principle for two special kinds of functions.

Definition 10.4.1. For any elements x and y of the same type, define for each $n \in \mathbb{N}$,

$$x =_n y \text{ iff } \mathrm{id}_n(x) = \mathrm{id}_n(y).$$

Proposition 10.4.2. *Let σ and τ be types. Then for any $f \in (\sigma \to \tau)$, any $x \in \sigma$ and any $\epsilon \in \mathbb{N}$, there exists $\delta \in \mathbb{N}$ such that $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(\mathrm{id}_\delta(x)))$.*

Proof. Since $\mathrm{id}_\epsilon(f(x)) = \bigsqcup_\delta \mathrm{id}_\epsilon \circ f \circ \mathrm{id}_\delta(x)$, it follows from the finiteness of $\mathrm{id}_\epsilon(f(x))$ that there exists $\delta \in \mathbb{N}$ such that $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(\mathrm{id}_\delta(x)))$. □

For the purpose of the ensuing discussion and the presentation of a density theorem in Section 10.6, we need to define totality. The notion of totality is defined by induction on the structure of types.

Definition 10.4.3. An element of ground type is *total* iff it is maximal with respect to the contextual order. An element $f \in (\sigma \to \tau)$ is *total* iff $f(x) \in \tau$ is total whenever $x \in \sigma$ is total. An element of type $(\sigma \times \tau)$ is *total* iff its projections onto σ and τ are total. Define a term with free variables to be *total* if every instantiation of its free variables by total elements produces a total element.

Remark 10.4.4. In the above definition, it is assumed that the language does not have a void type (i.e., a base type with no values).

Lemma 10.4.5. *(1) Let $\gamma \in \{\mathtt{Nat}, \mathtt{Bool}, \Sigma\}$. Then $m \in \gamma$ is total iff $m = v$ for some canonical value $v \in \gamma$.*

(2) $t \in \overline{\omega}$ is total iff $t = \infty$.

Proof. (1) ($\Longrightarrow$): Assume that $m \in \gamma$ is total. Then by definition of totality, $m \neq \bot$. So there exists $v \in \mathrm{Val}_\gamma$ such that $m \Downarrow v$. So $m = v$ by Kleene equivalence.
($\Longleftarrow$): Assume that $m = v$ for some canonical value v and suppose $y \in \gamma$ is such that $m \sqsubseteq y$. We must show that $y = m$. Because $\sqsubseteq$ is a simulation, it follows from (sim 1) - (sim 3) that $m \Downarrow v$. Again by (sim 1) - (sim 3), we have $y \Downarrow v$. Thus $y = v$ by Kleene equivalence. By transitivity, $y = m$ and thus m is total.

(2) ($\Longrightarrow$): Assume that $t \in \overline{\omega}$ is total. Since for every $x \in \overline{\omega}$, $x \sqsubseteq x + 1$, it follows from the maximality of t that $t = t + 1$. ∞ is the least fixed-point of $\lambda x^{\overline{\omega}}.x+1$ and hence $\infty \sqsubseteq t$. But we already know from Section 6.5 that $x \sqsubseteq \infty$ for all $x \in \overline{\omega}$. So $t \sqsubseteq \infty$ and thus $t = \infty$.
($\Longleftarrow$): Assume that $t = \infty$ and suppose $y \in \overline{\omega}$ is such that $t \sqsubseteq y$. We must show that $y = t$. By transitivity, $\infty \sqsubseteq y$. Again because $x \sqsubseteq \infty$ for all $x \in \overline{\omega}$, we have $y \sqsubseteq \infty$ and thus $y = \infty$. It then follows from transitivity that $t = y$. Thus t is total.

□

Proposition 10.4.6. *Any type has total elements.*

Proof. We prove this by induction on the structure of types.

(1) Base types. This follows from Lemma 10.4.5.

(2) Function type: $\sigma \to \tau$.
By induction hypothesis, there is a total element $t \in \tau$. Let x be a fresh variable not appearing in t. Then $\lambda x^\sigma.t$ is total since for any total element $s \in \sigma$ we have $(\lambda x^\sigma.t)(s) = t$ is total.

(3) Product type: $\sigma \times \tau$.
By induction hypothesis, there are total elements $x \in \sigma$ and $y \in \tau$ so that from the definition of totality it follows that (x, y) is total in $\sigma \times \tau$.

□

Recall that the function type ($\mathtt{Nat} \to \mathtt{Nat}$) is called the Baire type, $\mathtt{Baire}$.

Lemma 10.4.7. *Define* $\overline{\mathrm{id}}_n : \mathtt{Baire} \to \mathtt{Baire}$ *by:*

$$\overline{\mathrm{id}}_n(s) = \lambda i.\text{if } i < n \text{ then } s(i) \text{ else } \bot.$$

Then $\overline{\mathrm{id}}_n(s)$ *is finite and above* $\mathrm{id}_n(s)$*, and if* $s, t \in \mathtt{Baire}$ *are total then*

$$\overline{\mathrm{id}}_n(s) \sqsubseteq t \implies s =_n t.$$

Proof. Because whenever $i > n$ we have $\overline{\mathrm{id}}_n(s)(i) = \bot$, by Corollary 10.3.6 $\overline{\mathrm{id}}_n(s)$ is finite. Moreover, by extensionality, it follows from the definition of $\overline{\mathrm{id}}_n$ that $\overline{\mathrm{id}}_n(s)$ is above $\mathrm{id}_n(s)$. If $s, t \in \mathtt{Baire}$ are total, then $\overline{\mathrm{id}}_n(s) \sqsubseteq t$ implies that $\overline{\mathrm{id}}_n(s) = \overline{\mathrm{id}}_n(t)$. Since s and t agree on the first n positions, we have that $\mathrm{id}_n(s) = \mathrm{id}_n(t)$, i.e., $s =_n t$. □

Proposition 10.4.8. *For total $f \in (\sigma \to \mathtt{Baire})$ and $x \in \sigma$,*

$$\forall \epsilon \in \mathbb{N}. \exists \delta \in \mathbb{N}. \forall \text{ total } y \in \sigma. x =_\delta y \implies f(x) =_\epsilon f(y).$$

Proof. Because $\overline{\mathrm{id}}_\epsilon(f(x))$ is finite and below $f(x)$, there is δ such that already $\overline{\mathrm{id}}_\epsilon(f(x)) \sqsubseteq f(\mathrm{id}_\delta(x))$ by Proposition 10.2.6. If $x =_\delta y$ then $f(\mathrm{id}_\delta(x)) = f(\mathrm{id}_\delta(y))$ and hence $\overline{\mathrm{id}}_\epsilon(f(x)) \sqsubseteq f(\mathrm{id}_\delta(y)) \sqsubseteq f(y)$. By Lemma 10.4.7, $f(x) =_\epsilon f(y)$, as required. □

Similarly, we have:

Proposition 10.4.9. *For total $f \in (\sigma \to \gamma)$ and total $x \in \sigma$, where $\gamma \in \{\mathtt{Nat}, \mathtt{Bool}, \Sigma\}$,*

$$\exists \delta. \forall \text{ total } y \in \sigma. x =_\delta y \implies f(x) = f(y).$$

Proof. Because every element of γ is finite, so is $f(x)$. Thus by Proposition 10.2.6, there is $\delta \in \mathbb{N}$ such that already $f(x) = f(\mathrm{id}_\delta(x))$. If $x =_\delta y$ then $f(x) = f(\mathrm{id}_n(x)) = f(\mathrm{id}_n(y)) \sqsubseteq f(y)$. Since f and x are total, $f(x)$ is maximal and so $f(x) = f(y)$. □

10.5 An ultrametric on PCF

In this section, we look at a metric induced on each PCF type by the relations $=_n$. This helps us appreciate the meaning of "continuity" as used in the previous section.

Lemma 10.5.1. *For any type σ, if $m \leq n$ then*

$$\mathrm{id}^\sigma_m \circ \mathrm{id}^\sigma_n = \mathrm{id}^\sigma_m.$$

Thus $x =_n y$ implies $x =_m y$.

Proof. We proceed by induction on σ.

(i) Base types.
The claim is trivially true for Σ and `Bool`. For `Nat`, we note that

$$\mathrm{id}_n^{\mathtt{Nat}}(x) = \begin{cases} x & \text{if } x < n \\ \bot & \text{otherwise.} \end{cases}$$

Thus we have

$$\begin{aligned} \mathrm{id}_m^{\mathtt{Nat}} \circ \mathrm{id}_n^{\mathtt{Nat}}(x) &= \begin{cases} \mathrm{id}_n(x) & \text{if } \mathrm{id}_n(x) < m \\ \bot & \text{otherwise} \end{cases} \\ &= \begin{cases} x & \text{if } x < n \wedge x < m \\ \bot & \text{otherwise} \end{cases} \\ &= \begin{cases} x & \text{if } x < m \\ \bot & \text{otherwise} \end{cases} \end{aligned}$$

from which we conclude that $\mathrm{id}_m^{\mathtt{Nat}} \circ \mathrm{id}_n^{\mathtt{Nat}}(x) = \mathrm{id}_m^{\mathtt{Nat}}$.

As for $\overline{\omega}$, we rely on the property that $\mathrm{id}_n^{\overline{\omega}}(y) = \min(n, y)$ and easily see that

$$\begin{aligned} \mathrm{id}_m^{\overline{\omega}} \circ \mathrm{id}_n^{\overline{\omega}}(x) &= \mathrm{id}_m^{\overline{\omega}}(\min(n, x)) \\ &= \min(m, \min(n, x)) \\ &= \min(m, x) \\ &= \mathrm{id}_m^{\overline{\omega}}(x). \end{aligned}$$

(ii) Function types.
The claim holds since

$$\begin{aligned} & \mathrm{id}_m^{\sigma\to\tau} \circ \mathrm{id}_n^{\sigma\to\tau}(f) \\ = \;& (\mathrm{id}_m^{\tau} \circ \mathrm{id}_n^{\tau}) \circ f \circ (\mathrm{id}_n^{\sigma} \circ \mathrm{id}_m^{\sigma}) \\ = \;& \mathrm{id}_m^{\tau} \circ f \circ \mathrm{id}_m^{\sigma} && (\mathrm{id}_n^{\sigma} \circ \mathrm{id}_m^{\sigma} = \mathrm{id}_m^{\sigma} \text{ by Prop. 10.3.11}) \\ = \;& \mathrm{id}_m^{\sigma\to\tau}(f) \end{aligned}$$

where we apply the induction hypothesis at the second equality.

(iii) Product types.
The claim holds since

$$\begin{aligned} \mathrm{id}_m^{\sigma\times\tau} \circ \mathrm{id}_n^{\sigma\times\tau}(x, y) &= (\mathrm{id}_m^{\sigma} \circ \mathrm{id}_n^{\sigma}(x), \mathrm{id}_m^{\tau} \circ \mathrm{id}_n^{\tau}(y)) \\ &= (\mathrm{id}_m^{\sigma}(x), \mathrm{id}_m^{\tau}(y)) \\ &= \mathrm{id}_m^{\sigma\times\tau}(x, y) \end{aligned}$$

where we apply the induction hypothesis at the second equality.

The second part holds because

$$\begin{aligned} \mathrm{id}_m(x) &= \mathrm{id}_m \circ \mathrm{id}_n(x) \\ &= \mathrm{id}_m \circ \mathrm{id}_n(y) \quad (\text{since } x =_n y) \\ &= \mathrm{id}_m(y). \end{aligned}$$

where the first and third equalities rely on the above result. □

Theorem 10.5.2. *Define $d : \sigma \times \sigma \to \mathbb{R}_0^+$ (where $\mathbb{R}_0^+$ is the set of non-negative numbers) by*

$$d(x, y) = \inf\{2^{-n} | x =_n y\}$$

where $\inf(\emptyset) := 2$. *Then (σ, d) is an ultrametric space.*

Proof. We check that d satisfies the three axioms of an ultrametric:

(i) Non-degeneracy.
By definition, $d(x, y) \geq 0$ for every $x, y \in \sigma$. Moreover $d(x, y) = 0$ iff $x =_\infty y$ iff $x = y$.

(ii) Symmetry.
Trivial by definition.

(iii) Ultrametric-inequality.

$$\begin{aligned} \max(d(x, y), d(y, z)) &= \max(\inf\{2^{-n} | x =_n y\}, \inf\{2^{-m} | y =_m z\}) \\ &\geq \inf\{2^{-p} | x =_p z\} \\ &= d(x, z) \end{aligned}$$

where the inequality holds by Lemma 10.5.1

□

Remark 10.5.3. The fact that d is an ultrametric metric is nothing magical. It is a consequence of a much more general result which states that a descending family of equivalence relations on a set always induces an ultrametric on it (cf. p.706 of Smyth [55]).

A subset U of type σ is *metrically open* if it is open with respect to the above metric space topology.

Proposition 10.5.4. *Every open set is metrically open.*

Proof. Suppose U is open and $x \in U$. By Corollary 10.2.7 there is a finite element $b \in U$ such that $b \sqsubseteq x$. Since b is finite, there is $n \in \mathbb{N}$ with $b = \mathrm{id}_n(b)$. We now show that $B_n(x) := \{y \in \sigma | x =_n y\}$ is a subset of U. Let $y \in B_n(x)$. Then $x =_n y$ implies that $\mathrm{id}_n(x) = \mathrm{id}_n(y)$. But $b \sqsubseteq x$ entails that $\mathrm{id}_n(b) \sqsubseteq \mathrm{id}_n(x)$ and hence $b \sqsubseteq \mathrm{id}_n(x)$. Since $b \sqsubseteq \mathrm{id}_n(x) = \mathrm{id}_n(y) \sqsubseteq y$, by transitivity we conclude that $b \sqsubseteq y$. Then $b \in U$ implies $y \in U$. □

Question 10.5.5. Is the metric space induced by the relations $=_n$ complete?

The answer is probably not for the same reason that not every ω-chain in σ has a supremum (cf. Normann [39]).

10.6 Dense sets

In this section, we develop an operational version of the *Kleene-Kreisel density theorem* for total elements (cf. Berger [6]).

Definition 10.6.1. A set D is *dense* if it intersects every non-empty open set.

We say that a term x is defined from terms $y_1, \ldots, y_n$ if it belongs to the smallest set that contains $y_1, \ldots, y_n$ and constants and is closed under application and λ-abstraction.

In order to cope with the fact that the only total element of $\overline{\omega}$, namely ∞, is defined by the fixed-point recursion, we need:

Lemma 10.6.2. *If x is an element defined from total elements $y_1, \ldots, y_n$ in such a way that the only occurrences of the fixed-point combinator in x are those of $y_1, \ldots, y_n$, if any, then x is total.*

Proof. We prove by induction on the formation of x from $y_1, \dot{}y_n$ that x is total.

(1) $x \in \{y_1, \ldots, y_n\}$.
By asumption, y_i's are total and hence so is x.

(2) x is a constant (not fix).

(i) Numerals, boolean values and $\top$ are total by Lemma 10.4.5(1).

(ii) Assume that $m \in \mathtt{Nat}$ is a total element. We must prove that $\mathrm{succ}(m)$ and $\mathrm{pred}(m)$ are total. We prove the first part. By Lemma 10.4.5(1), $m = \underline{n}$ for some $n \in \mathbb{N}$. Hence $\mathrm{succ}(m) = \underline{n+1}$

by Kleene equivalence. Thus by Lemma 10.4.5(1), $\mathrm{succ}(m)$ is total. As for the second part, either $m = \underline{0}$ or $m = \underline{n+1}$ for some $n \in \mathbb{N}$ since every non-divergent term of type `Nat` must converge to some numeral. If $m = \underline{0}$, then $\mathrm{pred}(m) = \underline{0}$ which is total by Lemma 10.4.5(1). If $m = \underline{n+1}$, then $\mathrm{pred}(m) = \underline{n}$ by Kleene equivalence. Then by Lemma 10.4.5(1), $\mathrm{pred}(m)$ is total.

(iii) Assume that $m \in$ `Nat` is a total element. We must prove that $(m \stackrel{?}{=} 0)$ is total. Since m is total, $m = \underline{n}$ for some $n \in \mathbb{N}$ by Lemma 10.4.5(1). By Kleene equivalence, $(m \stackrel{?}{=} 0) = \mathrm{T}$ or F, which is total by Lemma 10.4.5(1).

(iv) Assume that $b \in$ `Bool` and $t_1, t_2 \in \sigma$ are total. We must prove that if b then t_1 else t_2 is total. Since b is total, by Lemma 10.4.5(1), either $b = \mathrm{T}$ or $b = \mathrm{F}$. If $b = \mathrm{T}$, then $t_1 =$ if b then t_1 else t_2 by Kleene equivalence. By assumption, t_1 is total and hence if b then t_1 else t_2 is total. The other case is similar.

(v) Assume that $t \in \overline{\omega}$ is total. We must show that $(t > 0)$ is total. By Lemma 10.4.5(2), $t = \infty$. Since $\infty \Downarrow \infty + 1$, it follows that $(t > 0) = \top$. Hence by Lemma 10.4.5(1), $(t > 0)$ is total.

(vi) Assume that $t \in \overline{\omega}$ is total. We must show that $t + 1$ and $t - 1$ are total. By Lemma 10.4.5(2), $t = \infty$. Thus $t + 1$ and $t - 1$ are contextually equivalent to ∞, and hence are total by Lemma 10.4.5(2).

(vii) Assume that $s \in \Sigma$ and $t \in \sigma$ are total. We must prove that if s then t is total. Since s is total, by Lemma 10.4.5(1), we have that $s = \top$. Thus by Kleene equivalence, $t =$ if s then t. By assumption, t is total. Thus if s then t is total.

(3) Closure under application follows from the definition of totality.

(4) Assume that $\beta_1 : \sigma_1, \ldots, \beta_k : \sigma_k, \alpha : \tau \vdash t$ is total. We must prove that $\beta_1 : \sigma_1, \ldots, \beta_k : \sigma_k \vdash \lambda\alpha.t$ is total, i.e., for all total elements $z_1 : \sigma_1, \ldots, z_k : \sigma_k$, the element $m := (\lambda\alpha.t)[z_1/\beta_1, \ldots, z_k/\beta_k]$ is total. Let $u : \tau$ be any total element. We want to show that $m(u)$ is total. But by Kleene equivalence, $m(u) = t[z_1/\beta_1, \ldots, z_k/\beta_k][u/\alpha]$ which is total by assumption.

This completes the proof by induction. □

The set of total elements in a type σ is dense:

Theorem 10.6.3. (Operational Kleene-Kreisel density theorem)
Every finite element is below some total element. Hence any inhabited open set has a total element.

Proof. For each type τ and each $n \in \mathbb{N}$, define programs

$$F^\tau : \overline{\omega} \to ((\tau \to \tau) \to \tau) \text{ and } G_n^\tau : (\tau \to \tau) \to \tau$$

by

$$F(x)(f) = \text{if } \ x > 0 \ \text{ then } \ f(F(x-1))(f)$$

and

$$G_n(f) = f^n(t) \text{ for some chosen total element } t \in \tau.$$

Note that t can be chosen by virtue of Proposition 10.4.6.

We claim that $F(n)(f) = f^n(\bot_\sigma)$. This can be shown by induction on n.
Base case: $n = 0$.
This is immediate from the syntax of the program F.
Inductive step:
$F(n+1)(f) = f(F(n-1)(f)) = f(f^n(\bot_\sigma))$ where the last equality is obtained by inductive hypothesis. Thus $F(n+1)(f) = f^{n+1}(\bot_\sigma)$ and the claim is established.

As a consequence of this, Theorem 6.6.1 yields $F(\infty) = \text{fix}$. Moreover since $\bot_\sigma \sqsubseteq t$ for any total element $t \in \sigma$, it follows that $F(n) \sqsubseteq G_n$. Also it is obvious that G_n is total.

Now given a finite element b, choose a fresh syntactic variable x of type $\overline{\omega}$, and define a term $\tilde{b}$ from b by replacing all occurrences of fix_τ with $F^\tau(x)$. Then $b = (\lambda x^{\overline{\omega}}.\tilde{b})(\infty)$. Because b is finite, there is some $n \in \mathbb{N}$ such that already $b = (\lambda x^{\overline{\omega}}.\tilde{b})(n)$.

Now construct another term $\hat{b}$ from b by replacing all occurrences of fix_τ by G_n^τ. Then the fixed point combinator does not occur in $\hat{b}$ and since each G_n^τ is total, by Lemma 10.6.2, $\hat{b}$ is total. Moreover $(\lambda x^{\overline{\omega}}.\tilde{b})(n) \sqsubseteq \hat{b}$ (a consequence of $F(n) \sqsubseteq G_n$ and monotonicity) and hence $b \sqsubseteq \hat{b}$ by transitivity. □

Remark 10.6.4. Note that the above density theorem exploits the absence of a void type from the language (cf. Definition 10.4.3).

Chapter 11

Compactness revisited

In this chapter, we revisit the notion of compactness (see Definition 5.7.2) and study how compactness interacts with finiteness. Here we see a strong interplay between topology and order theory, understood in a computational setting. In particular, we prove that compact sets satisfy a rational Heine-Borel property (Proposition 11.1.1). We also introduce and study operational notions of saturated sets and well-filtered subspaces. In particular, we have the following results: (1) every compact saturated set is the intersection of upper parts of finite sets of finite elements (Theorem 11.4.4), and (2) every Hausdorff subspace is well-filtered. Additionally, we develop a number of uniform continuity principles (Lemma 11.7.1, Theorem 11.7.2 and Proposition 11.7.3).

11.1 Rational Heine-Borel property

At this point, the reader may like to recall the definition of a compact set (cf. Definition 5.7.2).

If the notion of rational Scott openness is the operational analogue of Scott openness, then one expects that compact sets satisfy the *rational Heine-Borel property* in the sense that:

Proposition 11.1.1. *If Q is compact and U_n is a rational chain of opens with $Q \subseteq \bigcup U_n$, then there is $n \in \mathbb{N}$ such that already $Q \subseteq U_n$.*

Proof. There is $l \in (\overline{\omega} \to (\sigma \to \Sigma))$ with $l(n) = \chi_{U_n}$ and $l(\infty) = \chi_{\bigcup_n U_n}$. Since Q is compact and $Q \subseteq \bigcup_n U_n$, it follows from rational continuity that $\forall x \in Q.l(\infty)(x)$ iff $\bigsqcup_n \forall x \in Q.l(n)(x)$ iff $\exists n \in \mathbb{N}.\forall x \in Q.l(n)(x)$ iff $\exists n \in \mathbb{N}.Q \subseteq U_n$. □

The following proposition provides us with some non-compact sets.

Proposition 11.1.2. *The total elements of* `Nat` *and* `Baire` *do not form compact sets.*

Proof. Consider the program $g \in (\overline{\omega} \times \texttt{Nat} \to \Sigma)$ defined recursively by

$$g(x,n) = \text{if } x > 0 \text{ then } (\text{if } n \stackrel{?}{=} 0 \text{ then } \top \text{ else } g(x-1, \text{pred}(n))).$$

Clearly $g(x,n) = \top$ iff $x > n$ for all $x \in \overline{\omega}$ and $n \in \mathbb{N}$. If the total elements of $\mathbb{N}$ did form a compact set, then we would have $u \in (\overline{\omega} \to \Sigma)$ defined by $u(x) = \forall n \in \mathbb{N}.g(x,n)$ that would satisfy $u(k) = \bot$ for all $k \in \mathbb{N}$ and $u(\infty) = \top$ and hence would violate rational continuity. Therefore $\mathbb{N}$ is not compact in `Nat`. If the total elements of `Baire` formed a compact set, then, considering $f \in (\texttt{Baire} \to \texttt{Nat})$ defined by $f(s) = s(0)$, the image of B under f is the set $\mathbb{N}$ which by Proposition 5.8.1(1) renders $\mathbb{N}$ compact, again producing a contradiction. □

Remark 11.1.3. The above proof relies on a continuity principle rather than on recursion theory. Thus, compactness of $\mathbb{N}$ in `Nat` fails even if the language includes an oracle for the Halting Problem.

11.2 Saturation

Definition 11.2.1. The *saturation* of a subset S of a type σ is defined to be the intersection of its open neighbourhoods and is denoted by $\text{sat}(S)$, i.e.,

$$\text{sat}(S) = \bigcap \{U \text{ open } | S \subseteq U\}.$$

A set S is said to be *saturated* if $S = \text{sat}(S)$.

Lemma 11.2.2. *Let S be a subset of a type.*

(1) For any open set U, $S \subseteq U$ iff $\text{sat}(S) \subseteq U$.

(2) $\uparrow S \subseteq \text{sat}(S)$.

(3) $\text{sat}(S)$ is saturated.

(4) $\text{sat}(S)$ is the largest set with the same neighbourhoods as S.

Proof. Clearly $S \subseteq \text{sat}(S)$. Hence $\text{sat}(S) \subseteq U$ implies $S \subseteq U$. Conversely, if $S \subseteq U$, then by construction of $\text{sat}(S)$, $\text{sat}(S) \subseteq U$. Hence (1) holds. If $t \in \uparrow S$, then $s \sqsubseteq t$ for some $s \in S$. Hence t belongs to every neighbourhood of S, and hence to $\text{sat}(S)$. Therefore $\uparrow S \subseteq \text{sat}(S)$, i.e. (2) holds. Since

$S \subseteq \mathrm{sat}(S)$ for all S, we have that $\mathrm{sat}(S) \subseteq \mathrm{sat}(\mathrm{sat}(S))$. Now suppose $x \in \mathrm{sat}(\mathrm{sat}(S))$. Then for each open U with $S \subseteq U$, it holds that $\mathrm{sat}(S) \subseteq U$. Thus $x \in \mathrm{sat}(S)$ by definition. Hence $\mathrm{sat}(S) = \mathrm{sat}(\mathrm{sat}(S))$, i.e., (3) holds. That (4) holds is clear. □

Remark 11.2.3. The inclusion in (2) is strict in general. Here we furnish with an example of S for which the set-inclusion is strict. Take $S = \{(\bot, \top), (\top, \bot)\}$ of type $\Sigma \times \Sigma$ in a sequential language. Because of sequentiality, $\mathrm{sat}(S) = \uparrow (\bot, \bot)$. Then $(\bot, \bot) \in \mathrm{sat}(S)$ but $(\bot, \bot) \notin \uparrow S$.

Corollary 11.2.4. *(1) Q is compact iff $\mathrm{sat}(Q)$ is compact, and in this case, $\forall_Q = \forall_{\mathrm{sat}(Q)}$.*

(2) For any compact sets Q and R of the same type, it holds that $\forall_Q \sqsubseteq \forall_R$ iff $R \subseteq \mathrm{sat}(Q)$.

Proof. (1) This is immediate in the light of Lemma 11.2.2(1).
(2)

$$\begin{array}{ll} & \forall_Q \sqsubseteq \forall_R \\ \iff & \forall U \in \mathcal{U}. \forall_Q(\chi_U) = \top \Rightarrow \forall_R(\chi_U) = \top \\ \iff & \forall U \in \mathcal{U}. Q \subseteq U \Rightarrow R \subseteq U \\ \iff & R \subseteq \bigcap\{U \in \mathcal{U} | Q \subseteq U\} \\ \iff & R \subseteq \mathrm{sat}(Q) \end{array}$$

□

Definition 11.2.5. Let X be a subspace of σ and $S \subseteq X$. The saturation of S with respect to the subspace X (X-saturation of S, for short) is defined to be the intersection of its relatively open neighbourhoods and is denoted by $\mathrm{sat}_X(S)$, i.e.,

$$\mathrm{sat}_X(S) = \bigcap\{V \text{ open in } X | S \subseteq V\}.$$

S is said to be X-saturated if $S = \mathrm{sat}_X(S)$.

Proposition 11.2.6. *Let $S \subseteq X \subseteq \sigma$. Then $\mathrm{sat}_X(S) = X \cap \mathrm{sat}_\sigma(S)$. Hence for every saturated subset S of σ, $S \cap X$ is X-saturated.*

Proof.

$$\begin{array}{rcl} \mathrm{sat}_X(S) & = & \bigcap\{V \text{ open in } X | S \subseteq V\} \\ & = & \bigcap\{U \cap X | S \subseteq U \cap X \ \wedge \ U \text{ open in } \sigma\} \\ & = & X \cap \bigcap\{U \text{ open in } X | S \subseteq U\} \\ & = & X \cap \mathrm{sat}_\sigma(S) \end{array}$$

□

The following is an extension of Corollary 11.2.4.

Proposition 11.2.7. *Let $Q, R \subseteq X \subseteq \sigma$ be compact sets. Then $\forall_Q \sqsubseteq \forall_R$ iff $\mathrm{sat}_X(R) \subseteq \mathrm{sat}_X(Q)$. So, if Q and R are X-saturated then*

$$\forall_Q \sqsubseteq \forall_R \iff R \subseteq Q.$$

Proof. Using Corollary 11.2.4 and Proposition 11.2.6, we have

$$\begin{array}{rcl} \forall_Q \sqsubseteq \forall_R & \iff & R \subseteq \mathrm{sat}(Q) \\ & \iff & R \subseteq X \cap \mathrm{sat}(Q) \\ & \iff & R \subseteq \mathrm{sat}_X(Q) \\ & \iff & \mathrm{sat}_X(R) \subseteq \mathrm{sat}_X(Q). \end{array}$$

□

11.3 Compact open sets

The following is a characterisation of compact open sets:

Proposition 11.3.1. *An open set is compact iff it has finite characteristic. Hence every open set is a rational union of compact open sets.*

Proof. By Proposition 5.7.1(1), an open set V is compact iff the collection $\{U \text{ open } | V \subseteq U\}$ is open iff $\{\chi_U | U \text{ open and } V \subseteq U\}$ is open iff $\uparrow \chi_V$ is open. It then follows from Theorem 10.3.14 that this is equivalent to χ_V being finite, i.e., V having finite characteristic. The last part of the proposition then follows from Lemma 10.3.13(4). □

11.4 Compact saturated sets

Armed with the results we have so far, it is easy to see that:

Proposition 11.4.1. *Let U be an open set. If $U = U^{(n)}$, then $U = \uparrow \mathrm{id}_n(U)$. Hence every open set of finite characteristic is the contextual upper set of a finite set of finite elements.*

Proof. For each $u \in U$, it holds that $\mathrm{id}_n(u) \sqsubseteq u$. Thus $U \subseteq \uparrow \mathrm{id}_n(U)$. For the reverse inclusion, pick $x \in \uparrow \mathrm{id}_n(U)$. Then there is $u \in U$ such that $\mathrm{id}_n(u) \sqsubseteq x$. Because $U = \mathrm{id}_n^{-1}(U)$ we have $u \in \mathrm{id}_n^{-1}(U)$. So $\mathrm{id}_n(u) \in U$ and hence $x \in U$. □

The following may be seen as a generalisation of Theorem 10.3.14.

Proposition 11.4.2. *If F is a finite set of finite elements, then* $\mathrm{sat}(F)$ *is an open of finite characteristic.*

Proof. For each $x \in F$, there exists n_x such that $\mathrm{id}_{n_x}(x) = x$. Let $n = \max\{n_x | x \in F\}$. Then $\mathrm{id}_n(x) = x$ for all $x \in F$. Hence if $F \subseteq U$ for some open U, then $F \subseteq \mathrm{id}_n^{-1}(U) \subseteq U$. So $\mathrm{sat}(F) = \bigcap\{\mathrm{id}_n^{-1}(U) | F \subseteq U\}$. Because this is the intersection of a finite set of open sets, it is open. Moreover, by the idempotence of id_n, we have:

$$\begin{aligned}(\mathrm{sat}(F))^{(n)} &= (\bigcap\{U^{(n)} | F \subseteq U, U \text{ open}\})^{(n)} \\ &= \bigcap\{(U^{(n)})^{(n)} | F \subseteq U, U \text{ open}\} \\ &= \bigcap\{U^{(n)} | F \subseteq U, U \text{ open}\} \\ &= \mathrm{sat}(F).\end{aligned}$$

□

Lemma 11.4.3. *If Q is compact, then* $\mathrm{id}_n(Q)$ *is compact and*

$$\mathrm{id}_n(\forall_Q) = \forall_{\mathrm{id}_n(Q)}.$$

Furthermore, if U is open with $Q \subseteq U$, then there is n such that $\mathrm{id}_n(Q) \subseteq U$.

Proof. That $\mathrm{id}_n(Q)$ is compact and $\mathrm{id}_n(Q) = \forall_{\mathrm{id}_n(Q)}$ follow immediately from Proposition 5.8(1). Now, if U is open with $Q \subseteq U$, then $\forall_Q(\chi_U) = \top$. Hence by rational continuity there is n such that already $\mathrm{id}_n(\forall_Q)(\chi_U) = \top$, i.e., $\forall_{\mathrm{id}_n(Q)}(\chi_U) = \top$. So there is n such that $\mathrm{id}_n(Q) \subseteq U$. □

Theorem 11.4.4. *If Q is compact then* $\mathrm{sat}(Q) = \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. *Hence every compact saturated set is the intersection of upper parts of finite sets of finite elements.*

Proof. Since for any n it holds that $\mathrm{id}_n(Q) \subseteq U$ implies $Q \subseteq U$, it follows that $Q \subseteq \mathrm{sat}(\mathrm{id}_n(Q))$. Thus $\mathrm{sat}(Q) \subseteq \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. For the reverse inclusion, take any U open with $Q \subseteq U$. Then there is n such that $\mathrm{id}_n(Q) \subseteq U$ and hence $\mathrm{sat}(\mathrm{id}_n(Q)) \subseteq U$. Hence $\mathrm{sat}(Q) = \bigcap_n \mathrm{sat}(\mathrm{id}_n(Q))$. □

11.5 Intersections of compact saturated sets

A family of compact sets $\{Q_i\}_i$ is said to be *rationally filtered* if $(\forall_{Q_i})_i$ is a rational chain in $(\sigma \to \Sigma) \to \Sigma$.

Proposition 11.5.1. *Let X be a subspace of σ and $\{Q_i\}_i$ a rationally filtered family of compact X-saturated subsets of X. Then the following statements are equivalent:*

(i) $\bigcap_i Q_i$ *is compact and* $\forall_{\bigcap_i Q_i} \sqsubseteq \bigsqcup_i \forall_{Q_i}$.

(ii) $\bigsqcup_i \forall_{Q_i}$ *universally quantifies over* $\bigcap_i Q_i$.

(iii) For every open set U in σ,

$$\bigcap_i Q_i \subseteq U \implies \exists i.Q_i \subseteq U.$$

Proof. Observe that (a) if $\bigcap_i Q_i$ is compact, then by Corollary 11.2.4 the inequality $\forall_{\bigcap_i Q_i} \sqsupseteq \bigsqcup_i \forall_{Q_i}$ always holds, and (b) the inequality in (i) is equivalent to the implication in (iii).
(i) $\iff$ (ii): This is immediate from observation (a).
(i) $\implies$ (iii): This follows from observation (b).
(iii) $\implies$ (ii): It suffices to show that for every open set U in σ, it holds that

$$\bigsqcup_i \forall_{Q_i}(\chi_U) = \top \iff \bigcap_i Q_i \subseteq U.$$

($\Longrightarrow$): $\bigsqcup_i \forall_{Q_i}(\chi_U) = \top$ implies the existence of i such that $\forall_{Q_i}(\chi_U) = \top$. Thus $Q_i \subseteq U$ and since $\bigcap_i Q_i \subseteq Q_i$, it follows that $\bigcap_i Q_i \subseteq U$.
($\Longleftarrow$): Suppose that $\bigcap_i Q_i \subseteq U$. By assumption, there is i such that $Q_i \subseteq U$. So $\forall_{Q_i}(\chi_U) = \top$ and hence $\bigsqcup_i \forall_{Q_i}(\chi_U) = \top$. □

11.6 A non-trivial example of a compact set

The simplest non-trivial example of a compact set, which is a manifestation of the "one-point compactification of the discrete space of natural numbers", is given in the following proposition.

We regard function types of the form $(\mathtt{Nat} \to \sigma)$ as sequence types and define "head", "tail" and "cons" constructs for sequences as follows:

(1) $\mathrm{hd}(s) = s(0)$ and $\mathrm{tl}(s) = \lambda i.s(i+1)$.

(2) $n :: s = \lambda i.\mathrm{if}\ i \stackrel{?}{=} 0\ \mathrm{then}\ n\ \mathrm{else}\ s(i-1)$.

We also use familiar notations such as $0^n 1^\omega$ as shorthands for evident terms such as $\lambda i.\mathrm{if}\ i < n\ \mathrm{then}\ 0\ \mathrm{else}\ 1$.

Proposition 11.6.1. *The set $\mathbb{N}_\infty$ of sequences of the forms 0^n1^ω and 0^ω is compact in* `Baire`.

Proof. Define, omitting the subscript $\mathbb{N}_\infty$ for $\forall$,

$$\forall(p) = p(\text{if } (p(1^\omega) \wedge \forall(\lambda s.p(0 :: s))) \text{ then } t)$$

where t is some element of $\mathbb{N}_\infty$. More formally, $\forall = \text{fix}(F)$ where

$$F(A)(p) = p(\text{if } (p(1^\omega) \wedge A(\lambda s.p(0 :: s))) \text{ then } t).$$

We must show that, for any given p, $\forall(p) = \top$ iff $p(s) = \top$ for all $s \in \mathbb{N}_\infty$.
($\Longleftarrow$): The hypothesis gives $p(0^\omega) = \top$. By Proposition 10.2.6, there is $n \in \mathbb{N}$ such that already $p(\text{id}_n(0^\omega)) = \top$. Recall that

$$\text{id}_n(0^\omega)(i) = \begin{cases} 0 & \text{if } i < n; \\ \bot & \text{otherwise.} \end{cases}$$

We now show by induction on k that:

$$\text{For all } p \in (\texttt{Baire} \to \Sigma) \text{ such that } p(s) = \top \text{ for every } s \in \mathbb{N}_\infty,$$

$$p(\text{id}_k(0^\omega)) = \top \implies F^k(\bot)(p) = \top.$$

Base case: $k = 0$
This means that p is the constant predicate $\lambda s.\top$ and the hypothesis holds trivially.
Inductive step:
Suppose that $p \in (\texttt{Baire} \to \Sigma)$ such that $p(s) = \top$ for all $s \in \mathbb{N}_\infty$ and that $p(\text{id}_{k+1}(0^\omega)) = \top$. Unwinding the definition of F reveals that

$$\begin{array}{lll} F^{k+1}(\bot)(p) & = F(F^k(\bot))(p) & \\ & = p(\text{if } (p(1^\omega) \wedge F^k(\bot)(\lambda s.p(0 :: s))) \text{ then } t) & \text{(by definition of } F) \\ & = p(\text{if } (F^k(\bot)(\lambda s.p(0 :: s))) \text{ then } t) & \text{(since } p(1^\omega) = \top) \end{array}$$

Denote the predicate $\lambda s.p(0 :: s)$ by q. Because $(0 :: s) \in \mathbb{N}_\infty$ whenever $s \in \mathbb{N}_\infty$, it holds that $q(s) = \top$ for every $s \in \mathbb{N}_\infty$. Moreover $p(\text{id}_{k+1}(0^\omega)) = \top$ implies that $q(\text{id}_k(0^\omega)) = \top$. Hence by the induction hypothesis we have that $F^k(\bot)(q) = \top$. Thus $F^{k+1}(\bot)(p) = p(t) = \top$ as desired.

Because $F^n(\bot) \sqsubseteq \forall$, it follows that $\forall(p) = \top$.

($\Longrightarrow$): By rational continuity, the hypothesis implies that $F^n(\bot)(p) = \top$ for some n. We now prove by induction on k that:

For all $q \in (\texttt{Baire} \to \Sigma), F^k(\bot)(q) = \bot \implies q(s) = \top$ for all $s \in \mathbb{N}_\infty$.

Base case: $k = 0$
This is vacuously true.
Inductive step:
Suppose $F^{k+1}(\bot)(q) = \top$. But unfolding $F^{k+1}(\bot)$ gives

$$F^{k+1}(\bot)(q) = q(\text{if } (q(1^\omega) \wedge F^k(\bot)(\lambda s.q(0 :: s))) \text{ then } t.$$

So it must be that $q(1^\omega) = \top$ and $F^k(\bot)(\lambda s.q(0 :: s)) = \top$. It then follows from the induction hypothesis that $q(0 :: s) = \top$ for all $s \in \mathbb{N}_\infty$. But since every $t \in \mathbb{N}_\infty$ is either of the form 1^ω or $0 :: s$ for some $s \in \mathbb{N}_\infty$, the result follows. □

11.7 Uniform-continuity principles

In this section, we consider some *uniform continuity* principles.

Lemma 11.7.1. *For total $f \in (\sigma \to \texttt{Baire})$ and Q a compact set of total elements of σ,*

$$\forall \epsilon \in \mathbb{N}.\exists \delta \in \mathbb{N}.\forall x \in Q.f(x) \equiv_\epsilon f(\mathrm{id}_\delta(x))$$

where $x \equiv_n x'$ denotes $\overline{\mathrm{id}}_n(x) = \overline{\mathrm{id}}_n(x')$.

Proof. For any given ϵ, we construct the following program $e \in (\texttt{Baire} \times \texttt{Baire} \to \Sigma)$.

$E : \texttt{Baire} \times \texttt{Baire} \times \texttt{Nat} \to \Sigma$
$E(s,t,n) = \text{if } n \stackrel{?}{=} 0 \text{ then } \top \text{ else } (\text{if } s(n) = t(n) \text{ then } E(s,t,\mathrm{pred}(n)))$
$e(s,t) = E(s,t,\epsilon)$.

Then this program e is such that

(1) if $s, t \in \texttt{Baire}$ are total then $s \equiv_\epsilon t \implies e(s,t) = \top$,

(2) for all $s, t \in \texttt{Baire}$, $e(s,t) = \top \implies s \equiv_\epsilon t$.

Note that both (1) and (2) can be easily proven by induction on ϵ.

If we define $p(x) := e(f(x), f(x))$, then, by hypothesis and (1), $\forall_Q(p) = \top$. By Proposition 10.2.6, $\forall_Q(\mathrm{id}_\delta(x)) = \top$ for some $\delta \in \mathbb{N}$, and, by Proposition 10.3.9(2), $\mathrm{id}_\delta(p)(x) = p(\mathrm{id}_\delta(x))$. It follows that $e(f(\mathrm{id}_\delta(x)), f(\mathrm{id}_\delta(x))) = \top$ for all $x \in Q$. By monotonicity, $e(f(x), f(\mathrm{id}_\delta(x))) = \top$, and, by (2), $f(x) \equiv_\epsilon f(\mathrm{id}_\delta(x))$, as required. □

Theorem 11.7.2. *For $f \in (\sigma \to \mathtt{Baire})$ total and Q a compact set of total elements of σ,*

$$\forall \epsilon \in \mathbb{N}.\exists \delta \in \mathbb{N}.\forall x, y \in Q.x =_\delta y \Longrightarrow f(x) =_\epsilon f(y).$$

Proof. Given $\epsilon \in \mathbb{N}$, first construct $\delta \in \mathbb{N}$ as in Lemma 11.7.1. For $x, y \in Q$, if $x =_\delta y$ then $\overline{\mathrm{id}}_\epsilon(f(x)) = \overline{\mathrm{id}}_\epsilon(f(\mathrm{id}_\delta(x))) = \overline{\mathrm{id}}_\epsilon(f(\mathrm{id}_\delta(y))) \sqsubseteq f(y)$. By Lemma 10.4.7, $\mathrm{id}_\epsilon(f(x)) = \mathrm{id}_\epsilon(f(y))$, as required. □

Similarly, we have:

Proposition 11.7.3. *For $\gamma \in \{\mathtt{Nat}, \mathtt{Bool}, \Sigma\}$, $f \in (\sigma \to \gamma)$ total and Q a compact set of total elements of σ,*

(1) $\exists \delta \in \mathbb{N}.\forall x \in Q.f(x) = f(\mathrm{id}_\delta(x))$,

(2) $\exists \delta \in \mathbb{N}.\forall x, y \in Q.x =_\delta y \Longrightarrow f(x) = f(y)$.

Proof. (1) Recall that there is an equality test $(==) \in (\gamma \times \gamma \to \Sigma)$ for the total elements of γ such that

(a) if $x, y \in \gamma$ are total then $x = y \implies (x == y) = \top$,

(b) for all $x, y \in \gamma$, $(x == y) = \top \implies x = y$.

If we define $p(x) := (f(x) == f(x))$ then $\forall_Q(x) = \top$ by hypothesis and (a). By Proposition 10.2.6, $\forall_Q(\mathrm{id}_\delta(p)) = \top$ for some $\delta \in \mathbb{N}$. Since $\mathrm{id}_\delta(p)(x) = p(\mathrm{id}_\delta(x))$ by Proposition 10.3.9(2), it follows that $(f(\mathrm{id}_\delta(x)) == f(\mathrm{id}_\delta(x))) = \top$ for all $x \in Q$ and, by monotonicity, $(f(x) == f(\mathrm{id}_\delta(x))) = \top$, and, by (b), $f(x) = f(\mathrm{id}_\delta(x))$, as required.

(2) From (1) there exists $\delta \in \mathbb{N}$ such that for all $x \in Q$, $f(x) = f(\mathrm{id}_\delta(x))$. Thus for all $x, y \in Q$, if $x =_\delta y$ then $f(x) = f(\mathrm{id}_\delta(x)) = f(\mathrm{id}_\delta(y)) = f(y)$.

□

Definition 11.7.4. For f and Q as in Proposition 11.7.3, we refer to the least $\delta \in \mathbb{N}$ such that (1) (respectively (2)) holds as the *big* (respectively *small*) *modulus of uniform continuity* of f at Q. From the above proof, it is clear that the small modulus is smaller than or equal to the big modulus.

Chapter 12

Sample applications

In this chapter, we illustrate the scope and flexibility of the operational domain theory developed so far by applying our conclusions in the previous chapters to prove the correctness of non-trivial programs that manipulate infinite data (see Section 12.3 - 12.5).

We use the data language $\mathcal{D}$ to formulate specifications of programs in the programming language $\mathcal{P}$. The notion of data language originates from Escardó [13]. For the purpose of this chapter, $\mathcal{P}$ and $\mathcal{D}$ do not include parallel features. As in Section 3.7, the notation $x \in \sigma$ means that x is a closed term of type σ in $\mathcal{D}$. This is compatible with the notation of Chapters 9 - 11 by taking $\mathcal{D}$ as the underlying language for them. Again maintaining compatibility, we take the notions of totality, open set and compact set with respect to $\mathcal{D}$. To indicate that openness or compactness of a set is witnessed by a program rather than just an element of the data language, we say *programmably open* or *compact*.

Like the Baire type, we think of the elements of the Cantor type as sequences, and, following topological tradition, in this context we identify the booleans `true` and `false` with numbers 0 and 1 (it does not matter in which order).

12.1 Data language: an extension with oracles

In an operational setting, one usually adopts the same language to construct programs of a type and to express data of the same type. But consider programs that can accept externally produced streams as inputs. Because such streams are not necessarily definable in the language, it makes sense to consider program equivalence define by quantification over more liberal

"data contexts" and ask whether the same notion of program equivalence is obtained.

Definition 12.1.1. Let $\mathcal{P}$ be the programming language PCF introduced in Chapter 3, perhaps extended with parallel features, but not with oracles, and let $\mathcal{D}$ be $\mathcal{P}$ extended with oracles. We think of $\mathcal{D}$ as a *data language* for the programming language $\mathcal{P}$. The idea is that the closed term of $\mathcal{P}$ are *programs* and those of $\mathcal{D}$ are (higher typed) *data*. Accordingly, in this context, the notation $x \in \sigma$ means that x is a closed term of type σ in the data language. Of course, this includes the possibility that x is a program.

12.2 Equivalence with respect to ground $\mathcal{D}$-contexts

The following is folklore and goes back to Milner [38].

Theorem 12.2.1. *For terms in $\mathcal{P}$, equivalence with respect to ground $\mathcal{P}$-contexts and equivalence with respect to ground $\mathcal{D}$-contexts coincide.*

Proof. This follows directly from Proposition 10.2.4 and Lemma 12.2.2 below. □

Lemma 12.2.2. *For any data element x of any type, $\mathrm{id}_n(x)$ is equivalent to some program with respect to ground $\mathcal{D}$-context.*

Proof. Given $x \in \sigma$ in $\mathcal{D}$, there exists a program $g \in \mathtt{Baire}^m \to \sigma$ and oracles $\Omega_1, \ldots, \Omega_m$ such that $x = g(\Omega_1, \ldots, \Omega_m)$. It follows from m applications of Proposition 10.4.2 that there exist $k_1, \ldots, k_m$ such that

$$\mathrm{id}_n(x) = \mathrm{id}_n(g(\mathrm{id}_{k_1}(\Omega_1), \ldots, \mathrm{id}_{k_m}(\Omega_m))).$$

But the right-hand term is equivalent to a program, because clearly for any oracle Ω and $n \in \mathbb{N}$, the data term $\mathrm{id}_n(\Omega)$ is equivalent to some program. □

On the other hand, the notion of totality changes:

Theorem 12.2.3. *There are programs that are total with respect to $\mathcal{P}$ but not with respect to $\mathcal{D}$.*

This kind of phenomenon is again folklore. There are programs of type, for instance $\mathtt{Cantor} \to \mathtt{Bool}$, where $\mathtt{Cantor} := (\mathtt{Nat} \to \mathtt{Bool})$, that, when seen from the point of view of the data language, map programmable total elements to total elements, but diverge at some non-programmable total

inputs. The construction uses Kleene trees (see Beeson [4]), which are recursive counter-examples to König's Lemma, and can be found in Section 3.11 of Escardó [13].

12.3 The Cantor space

In this section, we are ready to give a more sophisticated example of a compact space. The following also serves as our main tool in this chapter:

Theorem 12.3.1. (Escardó [13], Section 3.11) *The total elements of the Cantor type, C, form a programmably compact set.*

Proof. The universal quantification program $\forall : (\mathtt{Cantor} \to \Sigma) \to \Sigma$ can be defined recursively by:

$$\forall(p) = p(\text{if } \forall(\lambda s.p(0 :: s)) \wedge \forall(\lambda s.p(1 :: s)) \text{ then } t)$$

where t is some programmable total element of Cantor, such as 0^ω. Formally, $\forall = \text{fix}(F)$ where

$$F(A)(p) = p(\text{if } A(\lambda s.p(0 :: s)) \wedge A(\lambda s.p(1 :: s)) \text{ then } t)$$

It remains to prove the correctness of this program, i.e.,

$$\forall(p) = \top \text{ iff } \forall s \in C.p(s) = \top.$$

($\Longrightarrow$): By rational continuity, the hypothesis implies that already $F^n(\bot)(p) = \top$ for some $n \in \mathbb{N}$. We now prove by induction on k that:

$$\forall q.F^k(\bot)(q) = \top \Rightarrow q(s) \text{ for all } s \in C.$$

Base case: $k = 0$
Because q does not look at its argument and returns $\top$, it must be the constant predicate $\lambda s.\top$. Thus $q(s) = \top$ for all $s \in C$ in particular.
Inductive step:
Suppose $F^{k+1}(\bot)(q) = \top$. Unfolding F yields:

$$F(F^k(\bot))(q) \quad = q(\text{if } (F^k(\bot)(\lambda s.q(0 :: s)) \wedge F^k(\bot)(\lambda s.q(1 :: s))) \text{ then } t)$$

Thus it must be that $F^k(\bot)(\lambda s.q(0 :: s)) = \top = F^k(\bot)(\lambda s.q(1 :: s))$. Invoking the induction hypothesis, we deduce that both $(\lambda s.q(0 :: s))$ and $(\lambda s.q(1 :: s))$ hold for all $s \in C$. Because every element of C has to begin with either 0 or 1, it follows that q holds for all $s \in C$.

($\Longleftarrow$): The hypothesis gives $p(s) = \top$ for all $s \in C$. Notice that for each such given $s \in C$, there is $n \in \mathbb{N}$ such that already $p(\mathrm{id}_n(s)) = \top$. By the *Cantor tree* we mean the infinite binarily branching tree. We think of an element of C as an infinite path in the Cantor tree, starting from the root, where a sequence of digits 0 and 1 is interpreted as a sequence of instructions "turn left" and "turn right". Since p looks only at the first n positions of the path s, it induces a pruning of s at level n; after which p does not look at the rest of the path. Because $p(s) = \top$ for every path s in the Cantor tree, we prune *all* possible paths and this results in a finitely branching tree of which every path is finite. By König's Lemma, the resulting pruned tree is finite, i.e., the height of the tree is finite, which we denote by δ. Thus it is clear that for all $s \in C$, $p(s) = p(\mathrm{id}_\delta(s)) = \top$. This is precisely the big modulus of uniform continuity of p at C (cf. Definition 11.7.4).

Now we prove by induction on δ that:

If the big modulus of uniform continuity of p at C is δ, then $\forall(p) = \top$.

Base case: $\delta = 0$
p does not look at its argument and returns $\top$. From the way $\forall$ is defined, it holds that $\forall(p) = \top$.
Inductive step:
Suppose the big modulus of uniform continuity of p at C is $\delta + 1$. Unfolding $\forall(p)$ yields:

$$p(\text{if } (\forall(p_0) \wedge \forall(p_1)) \text{ then } t)$$

where $p_0 := \lambda s.p(0 :: s)$ and $p_1 := \lambda s.p(1 :: s)$. Notice that the big moduli of continuity of these predicates p_0 and p_1 at C are at most δ. Invoking the induction hypothesis on p_0 and p_1, it follows that $\forall(p_0) = \top = \forall(p_1)$. It then follows, from the hypothesis and $t \in C$, that $\forall(p) = \top$, as required. □

Remark 12.3.2. If the data language is taken to be $\mathcal{P}$ itself, Theorem 12.3.1 fails for the same reason that leads to Theorem 12.2.3. Of course, the above program can still be written down. But it no longer satisfies the required specification given in Proposition 5.7.1(2). In summary, it is easier to universally quantify over *all* total elements of the Cantor type than just over the *programmable* ones, to the extent that the former can be achieved by a program but the latter cannot.

Interestingly, the programmability conclusion of Theorem 12.3.1 is not invoked for the purposes of this chapter, because we only apply compactness to get uniform continuity.

12.4 Universal quantification for boolean-valued predicates

The following theorem is due to Berger [5], with domain-theoretic denotational specification and proof. As discussed in the introduction, our purpose is to illustrate that such specifications and proofs can be directly understood in our operational setting, and, moreover, apply to *sequential* programming languages.

Theorem 12.4.1. *There is a total program*

$$\varepsilon : (\mathtt{Cantor} \to \mathtt{Bool}) \to \mathtt{Cantor}$$

such that for any total $p \in (\mathtt{Cantor} \to \mathtt{Bool})$*, if* $p(s) = 0$ *for some total* $s \in \mathtt{Cantor}$*, then* $\varepsilon(p)$ *is such an* s*.*

Proof. Define

$$\begin{array}{lll} \varepsilon(p) & = \text{if } p(0 :: \varepsilon(\lambda s.p(0 :: s)) & \text{then} \quad 0 :: \varepsilon(\lambda s.p(0 :: s)) \\ & & \text{else} \quad 1 :: \varepsilon(\lambda s.p(1 :: s)) \end{array}$$

Since C is compact, every total (boolean-valued) predicate p has a big modulus of uniform continuity δ. We now prove by induction on δ that:
If the big modulus of uniform continuity of p at C is δ, then $p(s) = 0$ for some total $s \in \mathtt{Cantor}$ implies that $\varepsilon(p)$ is such an s.
Base case: $\delta = 0$
When p has a modulus zero, $p(\bot)$ is total and hence p is constant. So irrespective of what $\varepsilon(p)$ is, we always have $p(\epsilon(p)) = 0$.
Inductive step:
If p has modulus $\delta + 1$, then the predicates $p_0 := \lambda s.p(0 :: s)$ and $p_1 := \lambda s.p(1 :: s)$ have modulus at most δ. Given that $p(s) = 0$ for some $s \in C$. Without loss of generality, let us assume that s is of the form $0 :: t$ for some $t \in C$. Thus we have $p_0(t) = 0$ and it follows from the induction hypothesis that $\varepsilon(p_0)$ witnesses p_0, i.e., $p_0(\varepsilon(p_0))) = 0$. By the definition of ε, we have $\varepsilon(p) = 0 :: \varepsilon(p_0)$. Now $p(\varepsilon(p)) = p(0 :: \varepsilon(p_0)) = p_0(\varepsilon(p_0)) = 0$. Thus $\varepsilon(p)$ is a total element from $\mathtt{Cantor}$ that witnesses p. □

This gives rise to universal quantification for boolean-valued rather than Sierpinski-valued predicates:

Corollary 12.4.2. *There is a total program*

$$\forall' : (\mathtt{Cantor} \to \mathtt{Bool}) \to \mathtt{Bool}$$

such that for every total $p \in (\texttt{Cantor} \to \texttt{Bool})$,

$$\forall'(p) = 0 \text{ iff } p(s) = 0 \text{ for all total } s \in \texttt{Cantor}.$$

Proof. First define $\exists : (\texttt{Cantor} \to \texttt{Bool}) \to \texttt{Bool}$ by $\exists(p) = p(\varepsilon(p))$ and then define $\forall'(p) = \neg(\exists(\lambda s.\neg p(s)))$. If $p(s)$ for some total $s \in \texttt{Cantor}$, then by Theorem 12.4.1 it holds that $\varepsilon(s)$ is a witness to p, i.e., $\exists(p) = p(\varepsilon(p)) = 0$. Conversely if $\exists(p) = 0$ then $p(\varepsilon(p)) = 0$. This means $p(s) = 0$ for some total $s \in \texttt{Cantor}$ (in which case, $s = \varepsilon(p)$). Thus in summary we have:

$$\exists(p) = 0 \text{ iff } p(s) = 0 \text{ for some total } s \in \texttt{Cantor}.$$

Unwinding the definition of $\forall'$, we see that $\forall'(p) = 0$ iff $\exists(\lambda s.\neg p(s)) = 1$ iff $\neg p(s) = 1$ for all total $s \in \texttt{Cantor}$ iff $p(s) = 0$ for all total $s \in \texttt{Cantor}$. □

So quite surprising one has:

Corollary 12.4.3. *The function type* $(\texttt{Cantor} \to \texttt{Nat})$ *has decidable equality for total elements.*

Proof. Define a program

$$(==) : (\texttt{Cantor} \to \texttt{Nat}) \times (\texttt{Cantor} \to \texttt{Nat}) \to \texttt{Bool}$$

by $f == g := \forall'(\lambda s.f(s) == g(s))$ where the second $==$ is the equality test on the total elements of $\texttt{Nat}$. It is easy to see that $(f == g) = 0$ iff $f(s) == g(s)$ for all total $s \in \texttt{Cantor}$. □

12.5 The supremum of the values of a function

The *lexicographic order* on the total elements of the Baire type is defined by $s \leq t$ iff whenever $s \neq t$, there is $n \in \mathbb{N}$ with $s(n) < t(n)$ and $s(i) = t(i)$ for all $i < n$.

Lemma 12.5.1. *There is a total program*

$$\max : \texttt{Baire} \times \texttt{Baire} \to \texttt{Baire}$$

such that

1. $\max(s,t)$ *is the maximum of* s *and* t *in the lexicographic order of all total* $s,t \in$ `Baire`*, and*

2. $(s,t) \equiv_\epsilon (s',t') \Rightarrow \max(s,t) \equiv_\epsilon \max(s',t')$ *for all* $s,t,s',t' \in$ `Baire` *(total or not) and all* $\epsilon \in \mathbb{N}$*.*

Proof. Define the program

$$\begin{array}{lll} \max(s,t) = & \text{if } \mathrm{hd}(s) == \mathrm{hd}(t) & \\ & \text{then} \quad \mathrm{hd}(s) :: \max(\mathrm{tl}(s), \mathrm{tl}(t)) & \\ & \text{else} \quad (\text{if } (\mathrm{hd}(s) > \mathrm{hd}(t) \text{ then } s \text{ else } t). & \end{array}$$

1. We now show that for all total $s,t \in$ `Baire`,

$$\max(s,t) = t \iff s \leq t.$$

Case 1: $s \neq t$.
For a given pair $s \neq t$ we define the minimum $n \in \mathbb{N}$ for which $s(n) \neq t(n)$ as the disagreement index of s and t. We prove by induction on n that:
For every total $s,t \in$ `Baire` $(s \neq t)$ whose disagreement index is n,

$$\max(s,t) = t \iff s \leq t.$$

Base case: $n = 0$
$\max(s,t) = t$ iff $\mathrm{hd}(s) < \mathrm{hd}(t)$ iff $s(0) < t(0)$.
Inductive step:
Suppose $s \neq t \in$ `Baire` with disagreement index of $n+1$. Then $\max(s,t) = \mathrm{hd}(s) :: \max(\mathrm{tl}(s), \mathrm{tl}(t))$. Since $tl(s) \neq tl(t)$ has disagreement index n, it follows from the induction hypothesis that

$$\max(\mathrm{tl}(s), \mathrm{tl}(t)) = \mathrm{tl}(t) \iff \mathrm{tl}(s) \leq \mathrm{tl}(t).$$

It immediately follows that $\max(s,t) = t \iff \leq t$.

Case 2: $s = t$.
We prove by induction on k that for all $k \in \mathbb{N}$, for all total $s \in$ `Baire`, it holds that

$$\max(s,s)(k) = s(k)$$

which then implies that $\max(s,s) = s$ by extensionality.
Base case: $k = 0$
Since $\mathrm{hd}(s) == \mathrm{hd}(s)$, we have $\max(s,s) = \mathrm{hd}(s) :: \max(\mathrm{tl}(s), \mathrm{tl}(s))$.

Hence $\max(s,s)(0) = \mathrm{hd}(\max(s,s)) = \mathrm{hd}(s) = s(0)$.
Inductive step:

$$\begin{aligned}\max(s,s)(k+1) &= (\mathrm{hd}(s) :: \max(\mathrm{tl}(s), \mathrm{tl}(s)))(k+1)\\ &= \max(\mathrm{tl}(s), \mathrm{tl}(s))(k)\\ &= \mathrm{tl}(s)(k)\\ &= s(k+1).\end{aligned}$$

2. By induction on ϵ.

□

The following is an adaptation of an example taken from Simpson [52]. In order to avoid having exact real-number computation as a pre-requisite, as in this reference, we have made suitable modifications to the program and its specification but retaining their essential aspects.

Theorem 12.5.2. *There is a total program*

$$S : (\mathtt{Cantor} \to \mathtt{Baire}) \to \mathtt{Baire}$$

such that for every total $f \in (\mathtt{Cantor} \to \mathtt{Baire})$*,*

$$S(f) = \sup\{f(s) | s \in \mathtt{Cantor} \textit{ is total}\},$$

where the supremum is taken in the lexicographic order.

Proof. Let $t \in \mathtt{Cantor}$ be a programmable total element and define

$$\begin{aligned}S(f) \;=\; & \text{if } \forall'(\lambda s.\mathrm{hd}(f(s)) == \mathrm{hd}(f(t)))\\ & \text{then } \mathrm{hd}(f(t)) :: S(\lambda s.\mathrm{tl}(f(s)))\\ & \text{else } \max(S(\lambda s.f(0 :: s)), S(\lambda s.f(1 :: s)))\end{aligned}$$

where $\forall' : (\mathtt{Cantor} \to \mathtt{Bool}) \to \mathtt{Bool}$ is the total program in Corollary 12.4.2. For convenience, we use the following notation:

$$\sup\{f(s) | s \in \mathtt{Cantor} \text{ is total}\} := \sup f.$$

We show by induction on $n \in \mathbb{N}$ that, for every total $f \in (\mathtt{Cantor} \to \mathtt{Baire})$,

$$S(f) \equiv_n \sup f.$$

Recall that $x \equiv_n y \iff \overline{\mathrm{id}}_n(x) = \overline{\mathrm{id}}_n(y)$.
Base case: $n = 0$.

Trivial.
Inductive step:
Assume that the statement holds for n, we must show that for every total $f \in (\texttt{Cantor} \to \texttt{Baire})$, it holds that

$$S(f) \equiv_{n+1} \sup f.$$

Recall from Proposition 11.7.4(2) that for each total $f \in (\texttt{Cantor} \to \texttt{Baire})$, the total function $\mathrm{hd} \circ f \in (\texttt{Cantor} \to \texttt{Nat})$ has a small modulus of uniform continuity , denoted by δ. We now prove by a further induction on δ that, for every total $f \in (\texttt{Cantor} \to \texttt{Baire})$ such that $\mathrm{hd} \circ f$ has modulus δ,

$$S(f) \equiv_{n+1} \sup f.$$

Base case: $\delta = 0$.
If $\mathrm{hd} \circ f$ has modulus 0 then $\mathrm{hd} \circ f$ is a constant, i.e. $\mathrm{hd}(f(s)) = \mathrm{hd}(f(t))$ for all total s. So $S(f) = \mathrm{hd}(f(t)) :: S(\lambda s.\mathrm{tl}(f(s)))$. But by assumption, $S(\lambda s.\mathrm{tl}(f(s))) \equiv_n \sup(\lambda s.\mathrm{tl}(f(s)))$, and thus:

$$\begin{array}{rcl} S(f) & = & \mathrm{hd}(f(t)) :: S(\lambda s.\mathrm{tl}(f(s))) \\ & \equiv_{n+1} & \mathrm{hd}(f(t)) :: \sup\,(\lambda s.\mathrm{tl}(f(s))) \\ & = & \sup((\lambda s.\mathrm{hd}(f(t)) :: s) \circ (\lambda s.\mathrm{tl}(f(s)))) \\ & = & \sup f. \end{array}$$

Note that the second equality relies on the fact that for all h,

$$\sup((\lambda s.d :: s) \circ h) = d :: \sup h.$$

Inductive step:
If $\mathrm{hd} \circ f$ has modulus $\delta + 1$ then $\mathrm{hd} \circ (\lambda s.f(0 :: s))$ and $\mathrm{hd} \circ (\lambda s.f(1 :: s))$ have modulus δ. Then $\mathrm{hd} \circ f(s) \neq \mathrm{hd} \circ f(t)$ for some total $s \in \texttt{Cantor}$. Thus $S(f) = \max(S(\lambda s.f(0 :: s)), S(\lambda s.f(1 :: s)))$. By assumption, we have $S(\lambda s.f(0 :: s)) \equiv_n \sup(\lambda s.f(0 :: s))$ and $S(\lambda s.f(1 :: s)) \equiv_n \sup(\lambda s.f(1 :: s))$. And by the (second) induction hypothesis, we have $S(\lambda s.f(0 :: s)) \equiv_{n+1} \sup(\lambda s.f(0 :: s))$ and $S(\lambda s.f(1 :: s)) \equiv_{n+1} \sup(\lambda s.f(1 :: s))$. Now by the non-expansiveness property given by Lemma 12.5.1(2) we have that $S(f) \equiv_{n+1} \max(\sup(\lambda s.f(0 :: s)), \sup(\lambda s.f(1 :: s)))$. Finally, we use the fact that

$$\sup h = \max(\sup(\lambda s.h(0 :: s)), \sup(\lambda s.h(1 :: s)))$$

to deduce that $\max(\sup(\lambda s.f(0 :: s)), \sup(\lambda s.f(1 :: s))) = \sup f$. Hence $S(f) \equiv_{n+1} \sup f$ and this completes the second induction.

Thus by Lemma 10.4.7, $S(f) =_n \sup f$ for all $n \in \mathbb{N}$, and by Lemma 10.3.2 it follows that

$$S(f) = \sup f.$$

□

Part IV

Operational Domain Theory for FPC

In this part, an operational domain theory is developed for FPC to treat recursive types. In Chapter 13, we establish the functoriality of type expressions. In Chapter 14, we establish the operational algebraic compactness of both the diagonal category, $\mathbf{FPC}_!^{\delta}$, and the product category, $\mathbf{F\check{P}C}_!$. In Chapter 15, we derive a pre-deflationary structure on the closed types and develop, as a consequence of this, a proof technique called the *Generic Approximation Lemma*. We demonstrate the versatility of this lemma in establishing program equivalence, where previously many other more complex methods had been employed.

Chapter 13

FPC considered as a category

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

The first category we consider, called $\mathbf{FPC}_!$, allows us to interpret types-in-context $X_1, \ldots, X_n \vdash \sigma$ as functors $\mathbf{FPC}_!^n \to \mathbf{FPC}_!$, provided type recursion in σ does not occur in contravariant positions (Sections 13.1 and 13.2). The second category, which is constructed out of $\mathbf{FPC}_!$, called $\mathbf{FPC}_!^{\delta}$, allows us to remove this restriction (Section 13.3).

13.1 The category of FPC types

The main purpose of this chapter is to give an account of the categorical framework within which our theory is organised. Our approach, largely adapted from Abadi & Fiore [1], turns out to be a convenient option among others. We carefully explain this in two stages:

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

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

The objects of the category **FPC** are the closed FPC types (i.e., type expressions with no free variables) and the morphisms are closed terms of function-type (modulo contextual equivalence). Given closed type σ, the

identity morphism id_σ is just the closed term $\lambda x^\sigma.x$ and the composition of two morphisms f and g is defined as

$$g \circ f := \lambda x.g(f(x)).$$

The category $\mathbf{FPC}_!$ is the subcategory of $\mathbf{FPC}$ whose morphisms are the strict $\mathbf{FPC}$-morphisms.

We make use of the following notations:

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

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

13.2 Basic functors

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

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

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

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

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

(2) a term-in-context

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

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

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

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

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

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

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

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

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

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

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

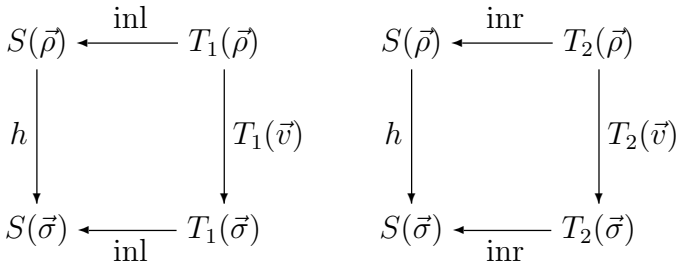
$$\begin{array}{ccc} S(\vec{\rho}) & \xrightarrow{\pi_j} & T_j(\vec{\rho}) \\ \Big\downarrow{\scriptstyle h} & & \Big\downarrow{\scriptstyle T_j(\vec{v})} \\ S(\vec{\sigma}) & \xrightarrow[\pi_j]{} & T_j(\vec{\sigma}) \end{array}$$

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

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

(4) Sum type.

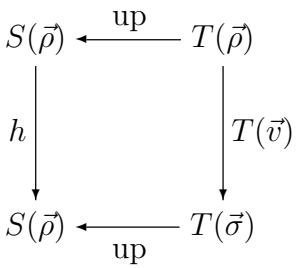
For object $\vec{\sigma} \in \mathbf{FPC}^n_!$, define $S(\vec{\sigma}) = T_1(\vec{\sigma}) + T_2(\vec{\sigma})$.
For morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the unique morphism h which makes the diagrams



commute. The morphism part of S is realised by

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

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



commute. The morphism part of S is realised by

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

where the morphism part of T is realised by

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

(6) Recursive type.
Let $\Theta, X \vdash \mathtt{B}$ ($X \notin \{X_1, \ldots, X_n\}$) and T the associated basic functor.

For object $\vec{\sigma} \in \mathbf{FPC}_!^n$, define $S(\vec{\sigma}) = \mu X.T(\vec{\sigma}, X)$. We write $T(\vec{\sigma}, S(\vec{\sigma}))$ for $\mathtt{B}[\vec{\sigma}/\vec{X}, S(\vec{\sigma})/X]$.
For the morphism $\vec{v} \in \mathbf{FPC}_!(\vec{\rho}, \vec{\sigma})$, define $S(\vec{v})$ to be the least morphism h that makes the diagram

$$\begin{array}{ccc} S(\vec{\rho}) & \xrightarrow{\text{unfold}^{S(\vec{\rho})}} & T(\vec{\rho}, S(\vec{\rho})) \\ \Big\downarrow h & & \Big\downarrow T(\vec{v}, h) \\ S(\vec{\sigma}) & \xrightarrow[\text{unfold}^{S(\vec{\sigma})}]{} & T(\vec{\sigma}, S(\vec{\sigma})) \end{array}$$

commute. The morphism part of S is realised by

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

where the morphism part of T is realised by

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

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

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

The morphism part of S is realised by

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

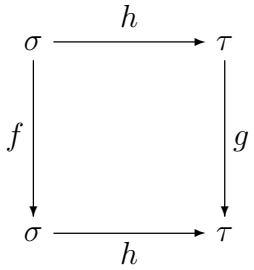
where the morphism part of T is realised by

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

Functoriality relies on the following two key lemmas.

Lemma 13.2.1. (Plotkin's uniformity principle)
Let $f : \sigma \to \sigma$, $g : \tau \to \tau$ be FPC programs and $h : \sigma \to \tau$ be a strict program,

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



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

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

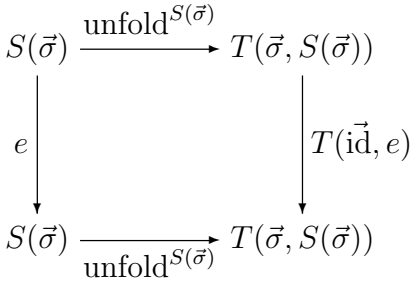
Proof. Using rational-chain completeness, rational continuity, $h \circ f = g \circ h$ in turn, it follows that

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

□

Remark 13.2.2. Notice that this uniformity of least fixed point relies on the rational-chain completeness enjoyed by FPC types and rational continuity of function-type programs. Both facts have been established in Part II of this thesis (cf. Theorem 7.6.6).

Lemma 13.2.3. (Operational minimal invariance for basic functors)
Let $T : \mathbf{FPC}_!^{n+1} \to \mathbf{FPC}_!$ *be a basic functor and* $\vec{\sigma}$ *any sequence of closed types. Write* $S(\vec{\sigma})$ *for* $\mu X.T(\vec{\sigma}, X)$*. Then the least endomorphism* $e : S(\vec{\sigma}) \to S(\vec{\sigma})$ *for which the diagram*



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

Proof. The interpretation of e in the Scott model, denoted by $[\![e]\!]$, makes the corresponding diagram commute. By Theorem 2.3.7, the only such endomorphism on $[\![S(\vec{\sigma})]\!]$ must be $\mathrm{id}_{[\![S(\vec{\sigma})]\!]}$. Now, by Lemma 13.2.4 below, it follows that $e = \mathrm{id}_{S(\vec{\sigma})}$. □

Lemma 13.2.4. *Let $e : \tau \to \tau$ be a closed term such that $[\![e]\!] = \mathrm{id}_{[\![\tau]\!]}$ in the Scott-model. Then $e = \mathrm{id}_\tau$.*

Proof. Notice that $[\![e]\!] = \mathrm{id}_{[\![\tau]\!]} = [\![\mathrm{id}_\tau]\!]$. By Corollary 4.6.4, it follows that $e = \mathrm{id}_\tau$. □

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

$$\vec{\sigma} \xrightarrow[\vec{u}]{} \vec{\rho} \xrightarrow[\vec{v}]{} \vec{\tau}$$

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

$$\begin{array}{ccccc}
\mu X.T(\vec{\sigma}, X) & \xrightarrow{\mu X.T(\vec{u}, X)} & \mu X.T(\vec{\rho}, X) & \xrightarrow{\mu X.T(\vec{v}, X)} & \mu X.T(\vec{\tau}, X) \\
= \downarrow & & & & \uparrow = \\
\mu X.T(\vec{\sigma}, X) & & \xrightarrow[\mu X.T(\vec{v} \circ \vec{u}, X)]{} & & \mu X.T(\vec{\tau}, X)
\end{array}$$

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

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

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

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

Because $- \circ S(\vec{u})$ is a strict program, by Lemma 13.2.1 it follows that

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

Next we prove the preservation of identity morphisms. By definition $S(\mathrm{id}_{\vec{\sigma}})$ is the least solution e of the equation $e = \mathrm{fold}^{S(\vec{\sigma})} \circ T(\mathrm{id}_{\vec{\sigma}}, e) \circ \mathrm{unfold}^{S(\vec{\sigma})}$. But Lemma 13.2.3 already asserts that $e = \mathrm{id}_{S(\vec{\sigma})}$.

13.3 Realisable functors

An unrestricted FPC type expression is more problematic.

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

(2) A particular type variable may be covariant and contravariant. For example, the type variable X in $X \to X$.

The usual solution to this problem of mixed variance, following Freyd [19], is to work with the category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$. In this chapter, we do not do so[1] but instead work with a full subcategory, $\mathbf{FPC}_!{}^{\delta}$, of this. Define $\mathbf{FPC}_!{}^{\delta}$, the *diagonal category*, to be the full subcategory of $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$ whose objects are those of $\mathbf{FPC}_!$ and morphisms being pairs of $\mathbf{FPC}_!$-morphisms, denoted by $u : \sigma \to \tau$ (or $\langle u^-, u^+ \rangle$), of the form:

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

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

[1] We shall consider the product category $\mathbf{FPC}_!^{\mathrm{op}} \times \mathbf{FPC}_!$ in Chapter 14.

The reader should note the following use of notations.

Notations. In order to avoid excessive use of $+$, $-$ and $\leftrightarrows$, we write

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

Definition 13.3.1. A *realisable functor* $T : (\mathbf{FPC}_!{}^\delta)^n \to \mathbf{FPC}_!{}^\delta$ is a functor which is realised by:

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

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

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

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

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

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

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

Remark 13.3.2. (1) Let $\vec{u}, \vec{v} \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$ be given and suppose that $\vec{u} \sqsubseteq \vec{v}$. Then by monotonicity, any realisable functor is *locally monotone* in the sense that $T(\vec{u}) \sqsubseteq T(\vec{v})$.

(2) Let $\vec{u}_k \in (\mathbf{FPC}_!{}^\delta)^n(\vec{\rho}, \vec{\sigma})$ be rational chains. Then by rational continuity, any realisable functor is *locally continuous* in the sense that $T(\bigsqcup_k \vec{u}_k) = \bigsqcup_k T(\vec{u}_k)$.

Definition 13.3.3. A type expression is *functional* if it is of the form $\tau_1 \to \tau_2$ for some types-in-context $\Theta \vdash \tau_1, \tau_2$.

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

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

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

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

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

The morphism part of T is given by:

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

where

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

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

(2) Non-functional type expressions.

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

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

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

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

(b) Product type.

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

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

The morphism part of T are realised by

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

where

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

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

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

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

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

The morphism part of T is realised by

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

where

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

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

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

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

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

If the morphism part of T is realised by

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

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

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

where

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

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

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

context

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

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

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

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

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

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

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

The morphism part of S is realised by:

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

By Lemmas 13.3.4 and 13.3.7 below, S is a functor.

Lemma 13.3.4. *S preserves composition of morphisms.*

Proof. The proof strategy[2] used here is the same as that used for establishing that basic type expressions (as functors) do preserve compositions. To show that S preserves, we must prove that for any morphism pairs $\vec{u} \in (\mathbf{FPC}_!^{\ \delta})^n(\vec{\rho}, \vec{\sigma})$ and $\vec{v} \in (\mathbf{FPC}_!^{\ \delta})^n(\vec{\sigma}, \vec{\tau})$, it holds that

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

[2]This proof strategy can be found in Lemma 5.3.1 of [3] where it is proven that least algebra homomorphisms compose.

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

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

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

Then the following diagram

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

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

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

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

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

is strict. Therefore, by Plotkin's uniformity Lemma 13.2.1, we have

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

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

Definition 13.3.5. An $\mathbf{FPC}_!^{\delta}$-morphism is said to be *twin* if it is of the form

$$u : \sigma \leftrightarrows \sigma : u.$$

Lemma 13.3.6. *Let $\vec{X} \vdash \tau$ be a type-in-context and $T_{\vec{X} \vdash \tau}$ as defined in the construction. Then for any $\vec{\sigma} \in (\mathbf{FPC}_!^{\delta})^n$ and for any sequence of twin morphism $\vec{u} \in (\mathbf{FPC}_!^{\delta})^n(\vec{\sigma},\vec{\sigma})$ (i.e., $u_i : \sigma_i \leftrightarrows \sigma_i : u_i$ $(i = 1,\ldots,n)$), the morphism $T(\vec{u})$ is again twin.*

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

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

commutes. Here we denote $\langle \mathrm{fold}, \mathrm{unfold} \rangle$ by i. Let $\phi := \lambda t. i^{-1} \circ T(\vec{u}, t) \circ i$. Then on one hand, by the definition of $S(\vec{u})$, we have $t = \mathrm{fix}(\phi)$. On the other hand, $\mathrm{fix}(\phi) = \bigsqcup_n \phi^{(n)}(\bot, \bot)$ by rational completeness. A further induction on n then shows that $\phi^{(n)}(\bot, \bot)$ is twin for every $n \in \mathbb{N}$. The proof is easy. For $n = 0$, we have the trivial twin $(\bot, \bot)$. For the inductive step, it follows from the two induction hypotheses that $\phi(n+1)(\bot, \bot) = i^{-1} \circ T(\vec{u}, \phi^{(n)}(\bot, \bot)) \circ i$ must be twin. Finally, invoking rational continuity, we have that $\mathrm{fix}(\phi)$ is twin and the proof is complete. □

Lemma 13.3.7. (Operational minimal invariance for realisable functors)
Let $T : (\mathbf{FPC}_!^{\delta})^{n+1} \to \mathbf{FPC}_!^{\delta}$ be a realisable functor and $\vec{\sigma} \in (\mathbf{FPC}_!^{\delta})^n$. As usual, we write $S(\vec{\sigma})$ for $\mu X.T(\vec{\sigma}, X)$. Then the least $\mathbf{FPC}_!^{\delta}$-endomorphism

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

for which the following commutes

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

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

i.e.,

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

Proof. First observe that e is twin by Lemma 13.3.6. So $e = (\varepsilon, \varepsilon)$ for some $\mathbf{FPC}_!$-morphism $\varepsilon : S(\vec{\sigma}) \to S(\vec{\sigma})$. Based on this observation, the proof will be complete once we have shown that $\varepsilon = \mathrm{id}_{S(\vec{\sigma})}$. Notice that $F := T(\vec{\sigma}, -) : \mathbf{FPC}_!{}^{\delta} \to \mathbf{FPC}_!{}^{\delta}$ realises the type-in-context $X \vdash \tau[\vec{\sigma}/\vec{X}]$. We want to prove that the least morphism $e = (\varepsilon, \varepsilon)$ such that

$$e = i^{-1} \circ F(e) \circ i$$

is contextually equivalent to $\langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle$. To achieve this, interpret this equation in the Scott model so that $[\![e]\!]$ is the least solution to the corresponding equation. Note that $[\![e]\!] = [\![(\varepsilon, \varepsilon)]\!] = ([\![\varepsilon]\!], [\![\varepsilon]\!])$. By Theorem 2.3.7, we have that $[\![\varepsilon]\!] = \mathrm{id}_{[\![S(\vec{\sigma})]\!]}$. It then follows from Lemma 13.2.4 that $\varepsilon = \mathrm{id}_{S(\vec{\sigma})}$. Hence $e = \langle \mathrm{id}_{S(\vec{\sigma})}, \mathrm{id}_{S(\vec{\sigma})} \rangle$. □

Remark 13.3.8. Note that the above proof cannot be easily replaced by a direct proof by induction on types. Such an attempt is deemed to fail because the least fixed point of a type expression is not in any way built from those of its constituents.

Remark 13.3.9. Notice that though denotational techniques have been employed in both the proofs of Lemmas 13.2.3 and 13.3.7, the results are purely operational. Some attempts have been made to produce a purely operational proof of these two lemmas and these are presented in Chapter 16.

Chapter 14

Operational algebraic compactness

In [18], P.J. Freyd introduced the notion of algebraic compactness to capture the bifree nature of the canonical solution to the domain equation

$$X = FX$$

in that "every endofunctor (on cpo-enriched categories, for example, $\mathbf{DCPO}_{\perp !}$, the category of pointed cpos and strict maps[1]) has an initial algebra and a final co-algebra and they are canonically isomorphic". In the same reference, Freyd proved the *Product Theorem* which asserts that algebraic compactness is closed under finite products. Crucially, this implies that $\mathbf{DCPO}_{\perp !} \times \mathbf{DCPO}_{\perp !}^{\mathrm{op}}$ is algebraically compact (since its components are) and thus allows one to cope well with the mixed-variant functors - making the study of recursive domain equations complete. Now proving that $\mathbf{DCPO}_{\perp !}$ is algebraically compact is no easy feat as one inevitably has to switch to the category of embeddings and projections, together with a bilimit construction (cf. Section 2.3.1). Using the operational machinery developed so far, we shall establish operational algebraic compactness with respect to the class of realisable functors.

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

The reader should note that we rely on uniformity (cf. Lemma 13.2.1) in

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

establishing the algebraic compactness results in Sections 14.1 - 14.2. Such a proof technique was probably first done in Simpson [53] for a more general setting of Kleisli-categories.

14.1 Operational algebraic compactness

Theorem 14.1.1. (Operational algebraic completeness I)
Every realisable endofunctor

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

has an initial algebra.

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

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

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

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

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

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

makes the above diagram commute. Then we consider the following diagram:

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

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

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

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

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

Note that $\mathrm{fix}(\Phi) = (\mathrm{id}_D, \mathrm{id}_D)$ by Lemma 13.3.7. Since G is strict, it follows from Lemma 13.2.1 that

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

Thus, the uniqueness of k is established. □

Theorem 14.1.2. (Operational algebraic compactness I)
Let $F : \mathbf{FPC}_!{}^{\delta} \to \mathbf{FPC}_!{}^{\delta}$ be a realisable endofunctor. Then every initial F-algebra is bifree, i.e., its inverse is also a final coalgebra.

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

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

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

where (D, i) is as defined in the proof of Theorem 14.1.1. Note that $i^{-1} = (\text{fold}, \text{unfold})^D$ so that

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

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

Theorem 14.1.3. (Operational parametrised algebraic compactness I)
Let $F : (\mathbf{FPC}_!{}^{\delta})^{n+1} \to \mathbf{FPC}_!{}^{\delta}$ be a realisable functor. Then there exists a realisable functor $H : (\mathbf{FPC}_!{}^{\delta})^n \to \mathbf{FPC}_!{}^{\delta}$ and a natural isomorphism i such that for all sequences of closed types P in $(\mathbf{FPC}_!{}^{\delta})^n$, we have

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

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

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

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

Proof. Every $P \in (\mathbf{FPC}_!{}^{\delta})^n$ induces a realisable endofunctor

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

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

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

commutes. By the universal property of initial algebras, H is a functor and by construction, i is a natural transformation. Moreover, Theorem 14.1.2 ensures that $(H(P), i_P)$ is a bifree $F(P, \_)$-algebra. □

14.2 Alternative choice of category

The classical theory of recursive domain equations centres around functors of the form $F : (\mathbf{DCPO}^{\mathrm{op}}_{\perp!} \times \mathbf{DCPO}_{\perp!})^{n+1} \to (\mathbf{DCPO}^{\mathrm{op}}_{\perp!} \times \mathbf{DCPO}_{\perp!})$. As noted before, $\mathbf{DCPO}^{\mathrm{op}}_{\perp!} \times \mathbf{DCPO}_{\perp!}$ is algebraically compact. But more generally $\mathbf{DCPO}^{\mathrm{op}}_{\perp!} \times \mathbf{DCPO}_{\perp!}$ is parameterised algebraically compact - a result implied by Corollary 5.6 of Fiore & Plotkin [16].

Let $\breve{\mathbf{FPC}}_!$ denote the product category $\mathbf{FPC}^{\mathrm{op}}_! \times \mathbf{FPC}_!$ where $\mathbf{FPC}_!$ is defined in Section 13.1. The natural question to ask is whether the category $\breve{\mathbf{FPC}}_!$ is algebraically compact. In order that this question makes sense, one has to identify an appropriate class of functors, $\mathcal{F}$, with respect to which algebraic compactness is defined. In this section, we show that, with a suitable choice of $\mathcal{F}$, the category $\breve{\mathbf{FPC}}_!$ is parametrised algebraically compact with respect to $\mathcal{F}$, i.e., for every $\mathcal{F}$-functor $T : (\breve{\mathbf{FPC}}_!)^{n+1} \to \breve{\mathbf{FPC}}_!$, there exists an $\mathcal{F}$-functor $H : (\breve{\mathbf{FPC}}_!)^n \to (\breve{\mathbf{FPC}}_!)$ and a natural isomorphism i such that for every sequence of closed types $\vec{\sigma} := \sigma_1^-, \sigma_1^+, \ldots, \sigma_n^-, \sigma_n^+$, the pair $(H(\vec{\sigma}), i_{\vec{\sigma}})$ is a bifree algebra of the endofunctor $T(\vec{\sigma}, -, +) : \breve{\mathbf{FPC}}_! \to \breve{\mathbf{FPC}}_!$.

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

Notation. We use the following notations:

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

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

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

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

A seemingly reasonable choice is the class of *syntactic functors* (originally used by A. Rohr in his Ph.D. thesis [47]) which is defined as follows.

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

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

(2) a term-in-context of the form:

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

such that for any $\vec{\sigma} \in \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!^{\ n}$, it holds that

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

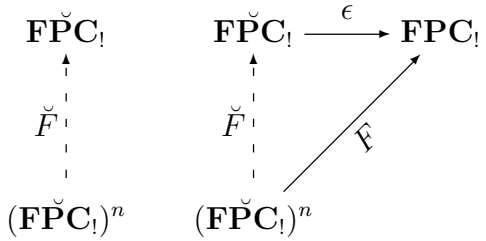
and for any $\vec{\rho}, \vec{\sigma} \in \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!^{\ n}$, and any $\vec{u} \in \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!^{\ n}(\vec{\rho}, \vec{\sigma})$, we have

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

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

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

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



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

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

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

However, if we work with this definition, a serious problem[2] arises. As we shall see in Theorem 14.2.4, the parametrised initial algebra of such functors are not of the form $\breve{H}$ for some functor H.

This can be fixed by working with our official definition:

Definition 14.2.1. An n-ary functor $F : (\mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!)^n \to \mathbf{F}\breve{\mathbf{P}}\mathbf{C}_!$ is said to be *syntactic* if it is given by:

(i) a pair of types-in-context $\vec{X} \vdash \tau^-, \tau^+$, and

(ii) a pair of terms-in-context $\vec{R}, \vec{S}; \vec{f} : \vec{R} \to \vec{S} \vdash$

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

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

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

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

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

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

Example 14.2.2. (1) Consider the type-in-context $X \vdash X \to X$. The object part part of the syntactic functor $T_{\vec{X} \vdash X \to X}$ is realised by the types-in-context

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

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

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

where

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

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

(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 14.2.3. Crucially, Example 14.2.2(2) indicates that if a minimal invariance for $X_2, X_1 \vdash X_1 \to X_2$ were to exist then it cannot be simply given by $X_2 \vdash \mu X_1.(X_1 \to X_2)$. Theorems 14.2.4 and 14.2.6 below provide us with a way to calculate the minimal invariance.

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

Theorem 14.2.4. (Operational algebraic completeness II)
Every syntactic functor

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

has an initial algebra.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We now show that $\mathrm{fix}(\Phi) = \mathrm{id}_{F^-(\sigma^+)}$. Note that because T^- is syntactic, so is $T^-(\_, \sigma^+)$. Denote $\mathrm{fix}(\Phi)$ by k. Again appealing to minimal invariance (cf. Lemma 13.3.7), it follows that $k = \mathrm{id}_{F^-(\sigma^+)}$.

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

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

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

that

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

Note that $- \circ h'$ is always strict. Invoking Lemma 13.2.1, we conclude that

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

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

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

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

Rediagramming a little gives:

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

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

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

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

In a similar way, we have the initial algebra for G given by

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

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

$$d^+ : T^+(F^-(\delta^+), \delta^+) \to \delta^+ \tag{14.1}$$

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

$$d^- : T^-(\delta^-, \delta^+) \to \delta^- \tag{14.2}$$

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

(4) We aim to show that

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

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

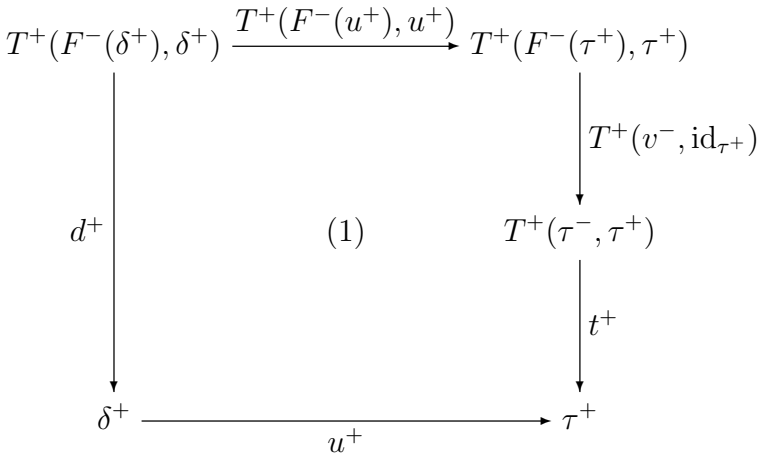
$$(t^-, t^+) : (T^-(\tau^-, \tau^+), T^+(\tau^-, \tau^+)) \to (\tau^-, \tau^+).$$

We want to show that there is a unique algebra homomorphism from $(d^-, d^+) : (T^-(\delta^-, \delta^+), T^+(\delta^-, \delta^+)) \to (\delta^-, \delta^+)$ to this. In this stage, we show the existence. By the initiality of $F^-(\tau^+)$, there is a unique morphism $v^- : F^-(\tau^+) \to \tau^-$ in $\mathbf{FPC}^{\mathrm{op}}_!$ so that the following diagram in $\mathbf{FPC}^{\mathrm{op}}_!$:

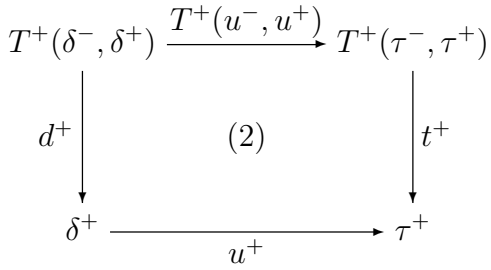
$$\begin{array}{ccc} T^-(F^-(\tau^+), \tau^+) & \xrightarrow{T^-(v^-, \mathrm{id}_{\tau^+})} & T^-(\tau^-, \tau^+) \\ \Big\downarrow{\scriptstyle f^-_{\tau^+}} & & \Big\downarrow{\scriptstyle t^-} \\ F^-(\tau^+) & \xrightarrow[v^-]{} & \tau^- \end{array}$$

Next we use the initiality of δ^+ to define $u^+ : \delta^+ \to \tau^+$ to be the unique

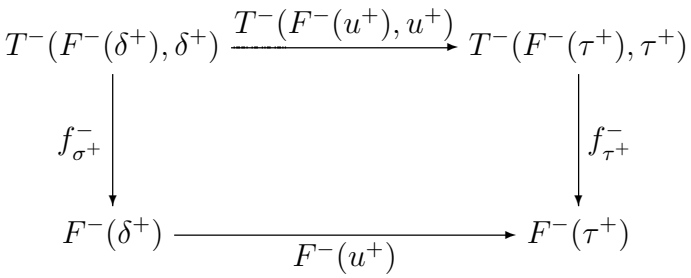
$\mathbf{FPC}_!$-morphism so that the Diagram (1) commutes in $\mathbf{FPC}_!$.



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



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



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

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

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

$$\begin{array}{ccc}
T^-(\delta^-,\delta^+) & \xrightarrow{T^-(u^-,u^+)} & T^-(\tau^-,\tau^+) \\
\downarrow{\scriptstyle f^-_{\delta^+}} & (3) & \downarrow{\scriptstyle t^-} \\
\delta^- & \xrightarrow[u^-]{} & \tau^-
\end{array}$$

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

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

On the other hand, we know from an earlier part of the proof that for any u^+, we may take $u^- = v^- \circ F^-(u^+)$ in $\mathbf{FPC}_!^{\mathrm{op}}$ to obtain Diagram (3). Hence we can conclude that $u^- = v^- \circ F^-(u^+)$ in $\mathbf{FPC}_!^{\mathrm{op}}$. Con-

sequently, putting these into Diagram (2) yields the commutativity of Diagram (1). Now by the initiality of δ^+, we can conclude that u^+ is unique.

□

Theorem 14.2.5. (Operational algebraic compactness II)
Let $F : \mathbf{F\breve{P}C}_! \to \mathbf{F\breve{P}C}_!$ be a syntactic functor. Then the initial algebra of F is bifree in the sense that the inverse

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

is a final F-coalgebra.

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

Proof. Walking through the stages of the proof of Theorem 14.2.4, one can check at each stage that a final coalgebra results when each initial algebra structure map is inverted. Notice this works even for the definition of F^- in stage (2). □

Theorem 14.2.6. (Operational parametrised algebraic compactness II)
Let $F : (\mathbf{F\breve{P}C}_!)^{n+1} \to \mathbf{F\breve{P}C}_!$ be a syntactic functor. Then there exists a syntactic functor $H : \mathbf{F\breve{P}C}_!^{\ n} \to \mathbf{F\breve{P}C}_!$ and a natural isomorphism i such that for all sequence of closed types P in $(\mathbf{F\breve{P}C}_!)^n$ we have

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

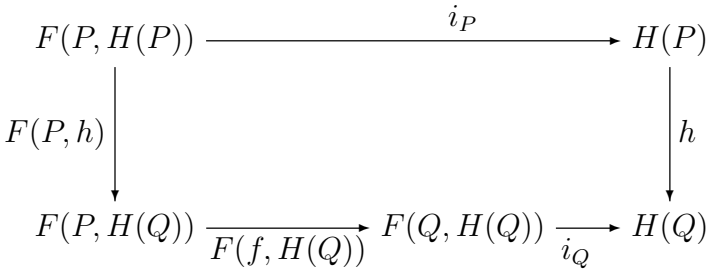
Moreover, $(H(P), i_P)$ is a bifree algebra for the endofunctor

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

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

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

commutes:



Notice that this unique h is also the least map for which the diagram commutes because initiality is derived from least fixed point construction (cf. Stage (1) of the proof of Theorem 14.2.4). By the universal property of initial algebras, H is a functor $(\mathbf{F\breve{P}C}_!)^n \to \mathbf{F\breve{P}C}_!$, and, by construction, i is a natural transformation. Moreover, it is clear that H is syntactic. Finally, the bifreeness of $(H(P), i_P)$ derives directly from Theorem 14.2.5. □

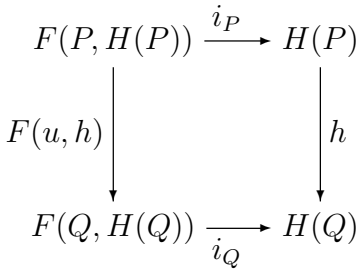
Definition 14.2.7. In Theorem 14.2.6, the functor $H : (\mathbf{F\breve{P}C}_!)^n \to \mathbf{F\breve{P}C}_!$ is constructed out of the functor $F : (\mathbf{F\breve{P}C}_!)^{n+1} \to \mathbf{F\breve{P}C}_!$ as a minimal invariant in the last argument pair X^-, X^+. To indicate this dependence, we write

$$H := \mu F.$$

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

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

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



commutes.

Examples 14.2.8. The syntactic functor acting as minimal X_1-invariant for

$X_1 \to X_2$ is given by

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

Remark 14.2.9. In general, the functor $H := \mu F$ is not symmetric. But we expect that symmetry can be achieved in the form of an operational analogue of Theorem 2.4.4 via Fiore's diagonalisation technique (cf. p.124 of Fiore [15]).

14.3 On the choice of categorical frameworks

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

In the framework of the product category $\mathbf{F\breve{P}C}_!$, it is appropriate to study the class of syntactic functors because all FPC types-in-context can be viewed as syntactic functors. We show how this can be done by induction on the structure of $\Theta \vdash \sigma$. We denote the syntactic functor associated to $\Theta \vdash \sigma$ by $F_{\Theta \vdash \sigma}$, or simply F.

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

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

(2) Product type.
For object P, define

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

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

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

where the component morphisms are defined as follows:

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

(3) Sum type.
For object P, define

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

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

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

where the component morphisms are defined as follows:

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

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

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

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

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

where the component morphisms are defined as follows:

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

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

For object P, define

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

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

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

where the component morphisms are defined as follows:

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

(6) Recursive type.
Let $\Theta, X \vdash \sigma$ be given and F the syntactic functor realising it. Define $F_{\Theta\vdash\mu X.\sigma}$ to be μF as in Definition 14.2.7.

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

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

Proposition 14.3.1. *For every type-in-context $\Theta \vdash \sigma$, the realisable functor $S_{\Theta\vdash\sigma}$ restricts and co-restricts to the syntactic functor $F_{\Theta\vdash\sigma}$, i.e., the diagram*

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

commutes up to natural isomorphism.

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

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

(1) Type variable.
Let $\Theta \vdash X_i$ be given. Define $\eta : F_{\Theta\vdash X_i} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta\vdash X_i}$ as follows.

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

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

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

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

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

(2) Product type.
We define $\eta : F_{\Theta \vdash \tau_1 \times \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_1 \times \tau_2}$ as follows. For every $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$,

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

where

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

(3) Sum type.
We define $\eta : F_{\Theta \vdash \tau_1 + \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_1 + \tau_2}$ as follows. For every $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$,

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

where

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

(4) Function type.
We define $\eta : F_{\Theta \vdash \tau_1 \to \tau_2} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta \vdash \tau_1 \to \tau_2}$ as follows. For every $\vec{\sigma} \in (\mathbf{FPC}_!{}^{\delta})^n$,

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

where

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

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

natural isomorphism

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

We define a natural isomorphism

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

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

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

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

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

We define a natural isomorphism

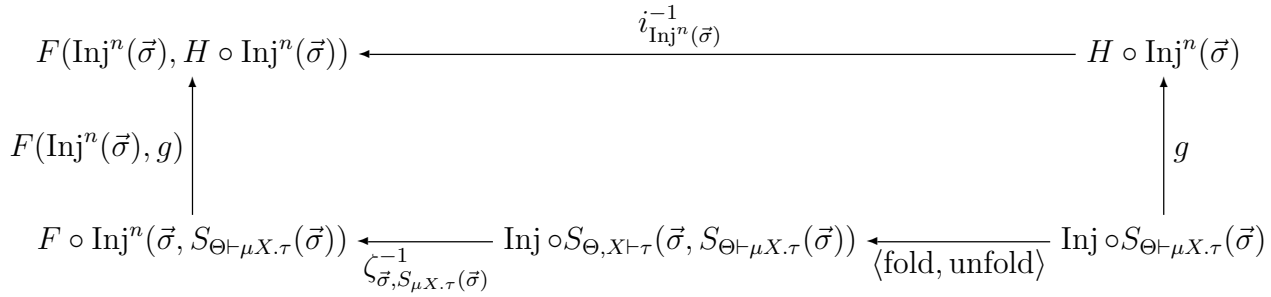
$$\eta : F_{\Theta\vdash\mu X.\tau} \circ \mathrm{Inj}^n \to \mathrm{Inj} \circ S_{\Theta\vdash\mu X.\tau}$$

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

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

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

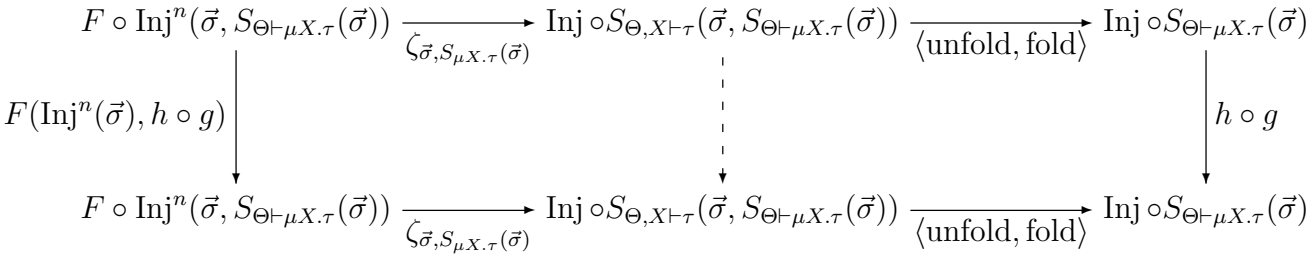
into the following commutative diagram:



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

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

it must be that $g \circ h = \mathrm{id}_{H \circ \mathrm{Inj}^n(\vec{\sigma})}$. For the second equation, we consider the diagram below which is obtained by pasting the above two diagrams: unique g which fits into the following commutative diagram:



where the dotted arrow is the morphism

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

So for the second quadrangle, $(h \circ g)^-$ and $(h \circ g)^+$ are both endomorphisms on $S_{\Theta \vdash \mu X.\tau}(\vec{\sigma})$. By the initiality of

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

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

□

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

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

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

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

In view of Proposition 14.3.1, this three-fold process can be carried out directly in the setting of the diagonal category. More precisely, for each closed type, there is a realisable functor which does the "same" job as its syntactic counterpart restricted and co-restricted to the diagonal. Because realisable functors can cope with variances without having to explicitly distinguish between the positive and negative type variables, the theory developed from using the diagonal category, $\mathbf{FPC}_!^{\delta}$, is clean. For instance, the functoriality of recursive type expression $\mu X.\tau$ can be conveniently defined. This has a strong appeal to the programmer as it requires a relatively little categorical overhead.

However, as a mathematical theory for treating recursive types, the approach via the product category, $\mathbf{F\breve{P}C}_!$, is general and can cope with mathematical notions, such as di-algebras (cf. Freyd [18]), which the diagonal category cannot.

Chapter 15

The Generic Approximation Lemma

In this section, we derive a pre-deflationary structure on the closed FPC types and develop, as a consequence of this, a powerful proof technique known as the *Generic Approximation Lemma*[1]. This lemma was first proposed by G. Hutton and J. Gibbons in [31] in which it was established, via denotational semantics, for *polynomial types* (i.e., types built only from unit, sums and products). In that same reference, the authors have suggested that it is possible to generalise the lemma "to mutually recursive, parameterised, exponential and nested datatypes" (cf. p.4 of [31]). In this chapter, we confirm this by providing a proof based on the operational domain theory we developed in Chapter 13. Also we use some running examples from Pitts [41] and Gibbons & Hutton [20] to demonstrate the power of the Generic Approximation Lemma as a proof technique for establishing program equivalence, where previously many other more complex techniques had been employed.

15.1 Standard FPC pre-deflations

A *pre-deflation* on a type σ is an element of type $(\sigma \to \sigma)$ that is below the identity. Thus, pre-deflations are deflations for which the requirement of finite image is dropped. A *rational pre-deflationary structure* on a closed FPC type σ is a rational chain id_n^σ of idempotent pre-deflations with $\bigsqcup_n \mathrm{id}_n^\sigma = \mathrm{id}_\sigma$. Note that every type has a trivial pre-deflationary structure, given by the constantly identity chain. In what follows, we define for each type a non-trivial pre-deflationary structure.

[1] This is a generalisation of R. Bird's approximation lemma [7], which in turn generalises the well-known take lemma [8].

Recall that we define the vertical natural numbers $\overline{\omega}$ (cf. Section 4.4) by $\overline{\omega} = \mu X.X_\perp$. Using $\overline{\omega}$, we define the programs $\mathrm{d} : \overline{\omega} \to (\sigma \to \sigma)$ by induction on σ as follows:

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

and most crucially for the recursive type $\mu X.\sigma$, the program $\mathrm{d}^{\mu X.\sigma}(n)$ is defined as follows. Let $S : \mathcal{C}^\delta \to \mathcal{C}^\delta$ be the realisable functor associated to the type-in-context $X \vdash \sigma$. We abuse notation by writing $\Pi_2 S(\mathrm{d}^{\mu X.\sigma}(n), \mathrm{d}^{\mu X.\sigma}(n))$ as $S(\mathrm{d}^{\mu X.\sigma}(n))$. Define

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

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

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

15.2 The Generic Approximation Lemma

Theorem 15.2.1. *The rational-chain* $\mathrm{id}^\sigma_n := d^\sigma(n)$ *defines a non-trivial rational pre-deflationary structure on* σ *for every closed type* σ*.*

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

$$\begin{array}{rll}
 & \mathrm{id}^{\mu X.\sigma}_{n+1} \circ \mathrm{id}^{\mu X.\sigma}_{n+1} & \\
= & \mathrm{fold} \circ S(\mathrm{id}^{\mu X.\sigma}_n) \circ \mathrm{unfold} \circ \mathrm{fold} \circ S(\mathrm{id}^{\mu X.\sigma}_n) \circ \mathrm{unfold} & (\text{def. of } \mathrm{id}^{\mu X.\sigma}_{n+1}) \\
= & \mathrm{fold} \circ S(\mathrm{id}^{\mu X.\sigma}_n) \circ S(\mathrm{id}^{\mu X.\sigma}_n) \circ \mathrm{unfold} & (\beta\text{-rule (7.15)}) \\
= & \mathrm{fold} \circ S(\mathrm{id}^{\mu X.\sigma}_n \circ \mathrm{id}^{\mu X.\sigma}_n) \circ \mathrm{unfold} & (S \text{ is a functor.}) \\
= & \mathrm{fold} \circ S(\mathrm{id}^{\mu X.\sigma}_n) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\
= & \mathrm{id}^{\mu X.\sigma}_{n+1}. & (\text{def. of } \mathrm{id}^{\mu X.\sigma}_{n+1})
\end{array}$$

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

$$
\begin{array}{lll}
 & \mathrm{id}_{n+1}^{\mu X.\sigma} & \\
= & \mathrm{fold} \circ S(\mathrm{id}_n^{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{def. of } \mathrm{id}_{n+1}^{\mu X.\sigma}) \\
\sqsubseteq & \mathrm{fold} \circ S(\mathrm{id}_{\mu X.\sigma}) \circ \mathrm{unfold} & (\text{Ind. hyp.}) \\
= & \mathrm{fold} \circ \mathrm{id}_{\sigma[\mu X.\sigma/X]} \circ \mathrm{unfold} & (S\text{is a functor.}) \\
= & \mathrm{id}_{\mu X.\sigma}. & (\eta\text{-rule (7.27)})
\end{array}
$$

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

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

By Lemma 13.3.7, $\mathrm{id}_{\mu X.\sigma}$ is the least solution of the above equation and thus must be below $\mathrm{id}_\infty^{\mu X.\sigma}$. On the other hand, $\mathrm{id}_\infty^{\mu X.\sigma} = \bigsqcup_n \mathrm{id}_n^{\mu X.\sigma}$ so that $\mathrm{id}_\infty^{\mu X.\sigma} \sqsubseteq \mathrm{id}_{\mu X.\sigma}$. Hence $\mathrm{id}_\infty^{\mu X.\sigma} = \bigsqcup_n \mathrm{id}_n^{\mu X.\sigma} = \mathrm{id}_{\mu X.\sigma}$ and thus (3) holds. □

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

Corollary 15.2.2. (The Generic Approximation Lemma)
Let σ be a closed type and $x, y : \sigma$. Then

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

Proof. ($\Longrightarrow$) Trivial.
($\Longleftarrow$) $x = \bigsqcup_n \mathrm{id}_n(x) = \bigsqcup_n \mathrm{id}_n(y) = y$ by Theorem 15.2.1. □

15.3 Sample applications

In this section, we demonstrate the versatility of the generic approximation lemma by using some running examples of programs taken from Pitts [41] and Gibbons & Hutton [20]. For each example, we compare the use of the Generic Approximation Lemma (Corollary 15.2.2) with an alternative method.

15.3.1 List type and some related notations

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

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

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

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

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

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

is written in `Haskell` style:

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

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

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

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

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

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

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

Lemma 15.3.1. *Let $n > 0$ be a natural number and τ be a closed type. Then the program $\text{id}_n : [\tau] \to [\tau]$ satisfies the following equations:*

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

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

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

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

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

□

15.3.2 The map-iterate property

We define two familiar functions map and iterate.

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

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

Proposition 15.3.2. (The map-iterate property)
Let τ be a closed type. For any $f : (\tau \to \tau)$ and any $x : \tau$, it holds that

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

This property had been proven in [20] using *program fusion*. We reproduce their proof here. For this method, one needs to define the program unfd as follows. Let σ and τ be given closed types. Define

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

The unfd function "encapsulates the natural basic pattern of co-recursive definition" (p.9 of [20]). Because several familiar co-recursive functions on lists can be defined in terms of unfd, one can rely on a universal property (which we describe below) of unfd to generate a powerful proof method whenever such co-recursive programs are involved. For example, if we define

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

then we can define the program iterate as follows:

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

Likewise, if we define

$$\begin{array}{lcl} \text{null} : [\tau] \to \texttt{Bool} & & \\ \text{null } [\,] & = & \text{T } (:= \text{inr}(\bot)) \\ \text{null } (x : xs) & = & \text{F } (:= \text{inl}(\bot)) \end{array}$$

then the program map can be defined as follows:

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

The proof method relies on a universal property enjoyed by unfd, which we now describe. Define $q : \sigma \to 1 + \tau \times \sigma$ by

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

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

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

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

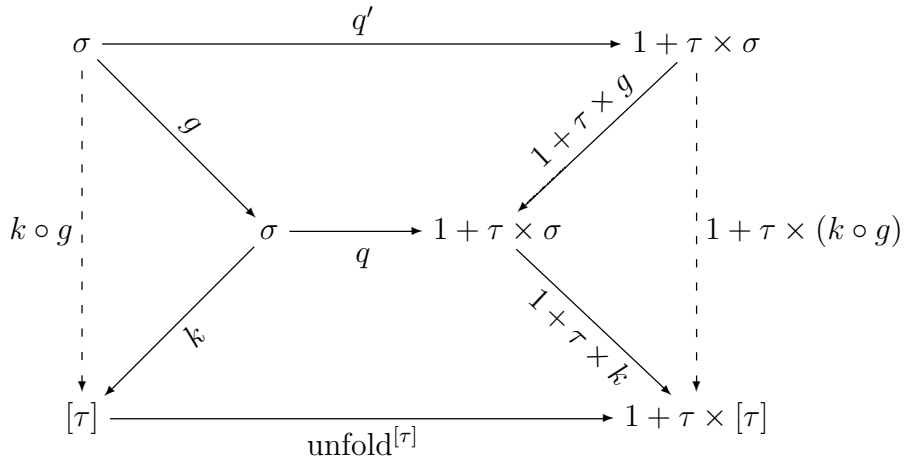
$$\begin{array}{ccc} \sigma & \xrightarrow{\quad q \quad} & 1 + \tau \times \sigma \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle 1 + \text{id}_\tau \times k} \\ [\tau] & \xrightarrow[\text{unfold}^{[\tau]}]{} & 1 + \tau \times [\tau] \end{array}$$

commutes. Moreover $k = \text{unfd } p\ h\ t$ is the unique morphism making the above diagram commute since $\text{unfold}^{[\tau]} : [\tau] \to 1 + \tau \times [\tau]$ is a final $(1 + \tau \times -)$-coalgebra by Theorem 14.1.2. Further suppose that $p' : \sigma \to \texttt{Bool}, h' : \sigma \to \tau$ and $t' : \sigma \to \sigma$ are programs such that

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

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

the upper quadrangle



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

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

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

It follows from the above inference rule that

$$(\text{unfd } p\ h\ t) \circ t = \text{unfd } (p \circ t)\ (h \circ t)\ t \tag{15.1}$$

$$\text{map } f \circ (\text{unfd } p\ h\ t) = \text{unfd } p\ (f \circ h)\ t \tag{15.2}$$

Proof.

$$
\begin{array}{lll}
 & (\text{iterate } f) \circ f & \\
= & (\text{unfd F id } f) \circ f & \text{(def. of iterate)} \\
= & \text{unfd } (\text{F} \circ f)\ (\text{id} \circ f)\ f & \text{(fusion (15.1))} \\
= & \text{unfd F } (f \circ \text{id})\ f & \text{(constant functions, composition)} \\
= & \text{map } f \circ \text{unfd F id } f & \text{(fusion (15.2))} \\
= & \text{map } f \circ \text{iterate } f. & \text{(def. of iterate)}
\end{array}
$$

□

Remark 15.3.3. Program fusion is a high-level method, i.e., it allows proofs to be performed in a purely equational way. However, it is too specialised a method in that programs involved must first be encoded using the unfd function.

We prove the Map-Iterate Proposition 15.3.2 using Corollary 15.2.2.

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

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

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

$$
\begin{array}{lll}
 & \text{id}_{n+1}(\text{map } f\ (\text{iterate } f\ x)) & \\
= & \text{id}_{n+1}(\text{map } f\ (x : \text{iterate } f\ f(x))) & \text{(def. of iterate)} \\
= & \text{id}_{n+1}(f(x) : \text{map } f\ (\text{iterate } f\ f(x))) & \text{(def. of map)} \\
= & f(x) : \text{id}_n(\text{map } f\ (\text{iterate } f\ f(x))) & \text{(Lemma 15.3.1)} \\
= & f(x) : \text{id}_n(\text{iterate } f\ f(f(x))) & \text{(Ind. hyp.)} \\
= & \text{id}_{n+1}(f(x) : \text{iterate } f\ f(f(x))) & \text{(Lemma 15.3.1)} \\
= & \text{id}_{n+1}(\text{iterate } f\ f(x)) & \text{(def. of iterate)}
\end{array}
$$

Thus the result holds by Corollary 15.2.2. □

15.3.3 Zipping two natural number lists

Let us define some programs.

$$
\begin{array}{l}
\text{zip} : [\sigma] \to [\tau] \to [\sigma \times \tau] \\
\text{zip } [\,]\ l \quad = \quad [\,] \\
\text{zip } (x : xs)\ [\,] \quad = \quad [\,] \\
\text{zip } (x : xs)\ (y : ys) \quad = \quad (x, y) : \text{zip } xs\ ys \\
\text{from} : \texttt{Nat} \to \texttt{Nat} \to [\texttt{Nat}] \\
\text{from } x\ y \quad = \quad (x : \text{from } (x + y)\ y)
\end{array}
$$

$$
\begin{array}{l}
\text{succ} : \texttt{Nat} \to \texttt{Nat} \\
\text{succ } x \quad = \quad x + 1
\end{array}
$$

We use the following notation:

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

plus : $(\mathtt{Nat} \times \mathtt{Nat}) \to \mathtt{Nat}$
plus (x, y) = if $x == 0$ then y else $1 + \mathrm{plus}(x - 1, y)$
Note that $\mathrm{plus}(x, y) = x + y$.

For each natural number k, define:
nats_k : $[\mathtt{Nat}]$
nats_k = $(k : \mathrm{map\ succ\ nats}_k)$

Proposition 15.3.4. *For any positive integer k, it holds that:*

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

In Pitts [41], the above proposition is established using Kleene equivalence and list-bisimulations. Before we reproduce his proof, let us recall the definition of list-bisimulation (cf. Pitts [41]), a technique used for proving contextual equivalence of lists.

Proposition 15.3.5. (List-bisimulation, Proposition 3.10 of [41])
For any type τ, a binary relation $\mathcal{R} \subseteq \mathrm{Exp}_{[\tau]} \times \mathrm{Exp}_{[\tau]}$ is called a $[\tau]$-bisimulation if whenever $l\ \mathcal{R}\ l'$

$$l \Downarrow [\,] \implies l' \Downarrow [\,] \tag{15.3}$$

$$l' \Downarrow [\,] \implies l \Downarrow [\,] \tag{15.4}$$

$$l \Downarrow (x : xs) \implies \exists x', xs'. \quad (l' \Downarrow (x' : xs') \wedge x =_\tau x' \wedge xs\ \mathcal{R}\ xs') \tag{15.5}$$

$$l' \Downarrow (x' : xs') \implies \exists x, xs. \quad (l \Downarrow (x : xs) \wedge x =_\tau x' \wedge xs\ \mathcal{R}\ xs'). \tag{15.6}$$

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

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

Proof. ($\Longrightarrow$): Since contextual equivalence is a bisimulation, it follows from Theorem 7.4.4 that $=_{[\tau]}$ satisfies the condition (bis 5). Using this, we can prove that

$$\{(l, l') | l =_{[\tau]} l'\}$$

is a $[\tau]$-bisimulation. The details are as follows:

(1) Suppose $l \Downarrow [\,]$ and $l =_{[\tau]} l'$. We must show that $l' \Downarrow [\,]$. Recall that $[\,] := \text{fold}(\text{inl}(*))$. By the evaluation rule ($\Downarrow$ unfold), we deduce that $\text{unfold}(l) \Downarrow \text{inl}(*)$. Now by (bis 5) we have that $\text{unfold}(l) =_{1+\tau\times[\tau]} \text{unfold}(l')$. It then follows from (bis 3a)that $\text{unfold}(l') \Downarrow \text{inl}(*')$ for where $* =_1 *'$. Thus by ($\Downarrow$ unfold) again, we have $l' \Downarrow \text{fold}(\text{inl}(*'))$, i.e., $l' \Downarrow [\,]$. Hence (15.3) holds. Similarly, (15.4) holds.

(2) Suppose that $l \Downarrow (x : xs)$ and $l =_{[\tau]} l'$. We must show that there exist $x' : \tau$ and $xs' : [\tau]$ such that $x =_\tau x'$ and $xs =_{[\tau]} xs'$. Recall that $(x : xs) := \text{fold}(\text{inr}(x, xs))$. It follows from ($\Downarrow$ unfold) that $\text{unfold}(l) \Downarrow \text{inr}(x, xs)$. By (bis 5), it follows that $\text{unfold}(l) =_{1+\tau\times[\tau]} \text{unfold}(l')$. It then follows that from (bis 3b) that $\text{unfold}(l') \Downarrow \text{inr}(x', xs')$ for some x', xs' with $x =_\tau x'$ and $xs =_{[\tau]} xs'$. By ($\Downarrow$ unfold), we have that $l' \Downarrow \text{fold}(\text{inr}(x', xs'))$. Thus (15.5) holds. Similarly, so does (15.6).

($\Longleftarrow$): Given a $[\tau]$-bisimulation $\mathcal{R}$, construct a relation $\mathcal{R}' \subseteq \text{Exp}_{1+\tau\times[\tau]} \times \text{Exp}_{1+\tau\times[\tau]}$ as follows:

$$\begin{aligned} t\ \mathcal{R}'\ t' \ \overset{\text{def}}{\Leftrightarrow}\ & (t \Downarrow \text{inl}(*) \implies t' \Downarrow \text{inl}(*)) \wedge \\ & (t' \Downarrow \text{inl}(*) \implies t \Downarrow \text{inl}(*)) \wedge \\ & (\forall x : \tau, xs : [\tau].t \Downarrow \text{inr}(x, xs) \implies \\ & \quad \exists x' : \tau, xs' : [\tau].t' \Downarrow \text{inr}(x', xs') \wedge x =_\tau x' \wedge xs\ \mathcal{R}\ xs') \wedge \\ & (\forall x' : \tau, xs' : [\tau].t' \Downarrow \text{inr}(x', xs') \implies \\ & \quad \exists x : \tau, xs : [\tau].t \Downarrow \text{inr}(x, xs) \wedge x =_\tau x' \wedge xs\ \mathcal{R}\ xs') \end{aligned}$$

The fact that contextual equivalence satisfies the conditions of a bisimulation and that $\mathcal{R}$ satisfies (15.3) - (15.6) together imply that

$$\mathcal{B}_\sigma := \begin{cases} \mathcal{R} \cup \{(l, l') | l =_{[\tau]} l'\} & \text{if } \sigma = [\tau] \\ \mathcal{R}' \cup \{(t, t') | t =_{1+\tau\times[\tau]} t'\} & \text{if } \sigma = 1 + \tau \times [\tau] \\ \{(t, t') | t =_\sigma t'\} & \text{otherwise} \end{cases}$$

defines an FPC bisimulation. Thus if $l\ \mathcal{R}\ l'$, then $l\ \mathcal{B}_{[\tau]}\ l'$ and so $l =_{[\tau]} l'$ by Theorem 7.4.4. □

We reproduce Pitts' proof of Proposition 15.3.4 which uses Proposition 15.3.5.

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

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

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

$$\text{from } e_m\ 2 \Downarrow (e_m : \text{from } e_{m+1}\ 2) \tag{15.7}$$

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

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

By applying the zip function, we have that

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

and from the definition of map that

$$\text{map plus } (\text{zip } l_m\ l_m) \Downarrow (\text{plus } (n_m, n_m) : \text{map plus } (\text{zip } l_{m+1}\ l_{m+1})) \tag{15.8}$$

One then establishes routinely by induction on m that

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

and thus by Proposition 7.5.1 we have

$$\text{plus } (n_m, n_m) =_{\texttt{Nat}} e_m. \tag{15.9}$$

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

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

Then properties (15.7) - (15.9) together implies that $\mathcal{R}$ satisfies all the conditions (15.3)-(15.6) and hence is a $[\texttt{Nat}]$-bisimulation. Thus the proof is complete by virtue of Proposition 15.3.5. □

Remark 15.3.6. Notice that in order to apply Proposition 15.3.5 one must come up with a suitable list-bisimulation.

Before using Corollary 15.2.2 to prove Proposition 15.3.4, we establish a useful property.

Proposition 15.3.7. *For any $i, k \in \mathbb{N}$, it holds that*

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

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

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

and the desired result follows from Corollary 15.2.2. The base case is trivial and the inductive step is justified by

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

□

We now prove Proposition 15.3.4 using Corollary 15.2.2.

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

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

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

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

The desired result then follows from Corollary 15.2.2. □

15.3.4 The 'take' lemma

Let us now define the take function of Bird & Wadler [8].

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

Proposition 15.3.8. (The 'take' lemma)
Let τ be a closed type and $l, l' : [\tau]$.

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

We reproduce Pitts' proof (cf. [41] of Proposition 15.3.8 which uses Proposition 15.3.5.

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

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

We prove that $\mathcal{R}$ satisfies conditions (15.3) - (15.6). First of all, by the evaluation rules and Kleene equivalence, the following properties hold: For all $n \in \mathbb{N}, x : \tau$ and $l, xs : [\tau]$,

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

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

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

(1) To establish condition (15.3), we suppose that $l \Downarrow [\ ]$. Then (a) implies that $\text{take } 1\ l \Downarrow [\ ]$. Since $l\ \mathcal{R}\ l'$, by definition of $\mathcal{R}$, $\text{take } 1\ l =_{[\tau]} \text{take } 1\ l'$. Since contextual equivalence is an FPC bisimulation, it follows that $\text{take } 1\ l' \Downarrow [\ ]$. Hence by (a) again, it holds that $l' \Downarrow [\ ]$.

(2) A symmetrical argument shows that $\mathcal{R}$ satisfies condition (15.4).

(3) To see that it satisfies condition (15.5), suppose $l \Downarrow (x : xs)$. Then by (b) for any $n \in \mathbb{N}$ we have $\text{take } n+1\ l \Downarrow (x : \text{take } n\ xs)$. Since $l\ \mathcal{R}\ l'$, by definition of $\mathcal{R}$, $\text{take } n+1\ l =_{[\tau]} \text{take } n+1\ l'$. So since contextual equivalence is an FPC bisimulation, it follows that there are terms x' and xs'' with

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

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

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

(4) A symmetrical argument shows that $\mathcal{R}$ also satisfies condition (15.6).

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

Let us now provide an alternative proof of Proposition 15.3.8 by using Corollary 15.2.2.

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

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

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

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

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

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

This implies that

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

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

15.3.5 The filter-map property

The next sample application involves the filter function, which we define below.

$$\begin{array}{l} \text{filter} : (\tau \to \texttt{Bool}) \to ([\tau] \to [\tau]) \\ \text{filter}\ u\ [\ ] \quad = \quad [\ ] \\ \text{filter}\ u\ (x : xs) \quad = \quad \text{if}\ u(x)\ \text{then}\ (x : \text{filter}\ u\ xs)\ \text{else filter}\ u\ xs \end{array}$$

Proposition 15.3.9. *For any* $u : (\tau \to \texttt{Bool})$, $v : (\tau \to \tau)$ *and* $l : [\tau]$, *it holds that*

$$\text{filter } u \text{ (map } v \ l) =_{[\tau]} \text{map } v \text{ (filter } (u \circ v) \ l).$$

This proposition was established in Pitts [41] based on an induction on the *depths of proofs of evaluation*. Here we elaborate. Define the nth level evaluation relation $\Downarrow^n$ (written as $x \Downarrow^n v$) as follows. Replace in the axioms and rule regarding $\Downarrow$ (see Figure 4.3) each occurrence of $\Downarrow$ by $\Downarrow^n$ in an axiom or the premise of a rule and replacing $\Downarrow$ by $\Downarrow^{n+1}$ in the conclusion of each rule. Then of course we have:

$$x \Downarrow v \Leftrightarrow \exists n \in \mathbb{N}.(x \Downarrow^n v) \tag{15.10}$$

It suffices to show that there is a list bisimulation that relates filter u (map $v \ l$) and map v (filter $(u \circ v)$ l). Usually it is Hobson's choice.

Proof. Define

$$\mathcal{R} := \{(\text{filter } u \text{ (map } v \ l), \text{map } v \text{ (filter } (u \circ v) \ l)) | l : [\tau]\}.$$

Instead of proving the three conditions of a list bisimulation directly, we deduce them via (15.10), using the properties of $\Downarrow^n$:

(1) $\forall l.(\text{filter } u \text{ (map } v \ l) \Downarrow^n [\ ] \implies \text{map } v \text{ (filter } (u \circ v) \ l \Downarrow [\ ])$.

(2) $\forall l.(\text{map } v \text{ (filter } (u \circ v) \ l \Downarrow^n [\ ] \implies \text{filter } u \text{ (map } v \ l) \Downarrow [\ ])$.

(3) $\forall l, x, xs.(\text{filter } u \text{ (map } v \ l) \Downarrow^n (x : xs) \implies$
$\exists xs'.(\text{map } v \text{ (filter } (u \circ v) \ l) \Downarrow (x : xs') \wedge xs \ \mathcal{R} \ xs')$.

(4) $\forall l, x, xs'.(\text{map } v \text{ (filter } (u \circ v) \ l) \Downarrow^n (x : xs') \implies$
$\exists xs.(\text{filter } u \text{ (map } v \ l) \Downarrow (x : xs) \ \& \ xs \ \mathcal{R} \ xs')$.

The proofs of (1) - (4) are by induction on n. □

We now prove Proposition 15.3.9 by using Corollary 15.2.2.

Proof. Given any $l : [\tau]$, we have two possibilities:

(1) There is $n \in \mathbb{N}$ such that $\text{tl}^{(n)}(l) =_{[\tau]} [\ ]$.

(2) For all $n \in \mathbb{N}$, $\text{tl}^{(n)}(l) \neq_{[\tau]} [\ ]$.

Here $\mathrm{tl}^{(0)}(l) := l$ and $\mathrm{tl}^{(n+1)}(l) := \mathrm{tl}(\mathrm{tl}^{(n)}(l))$.
Those lists which satisfy (1) are called finite lists. For a finite list, we define its length to be $n \in \mathbb{N}$ for which $\mathrm{tl}^{(n)}(l) =_{[\tau]} [\ ]$. Those lists which satisfy (2) are called infinite lists.
We prove Proposition 15.3.9 for each of these cases.

(1) Finite lists
We prove by induction on the length of finite lists that

$$\text{filter } u \text{ (map } v \ l) =_{[\tau]} \text{map } v \text{ (filter } (u \circ v) \ l).$$

Base case: $n = 0$.
In this case, $l =_{[\tau]} [\ ]$. On one hand, we have:

$$\begin{aligned} \text{filter } u \text{ (map } v \ l) &\equiv \text{filter } u \text{ (map } v \ [\ ]) \\ &=_{[\tau]} \text{filter } u \ [\ ] \\ &=_{[\tau]} [\ ] \end{aligned}$$

On the other hand, we have:

$$\begin{aligned} \text{map } v \text{ (filter } (u \circ v) \ l) &\equiv \text{map } v \text{ (filter } (u \circ v) \ [\ ]) \\ &=_{[\tau]} \text{map } v \ [\ ] \\ &=_{[\tau]} [\ ] \end{aligned}$$

Hence the statement holds.
Inductive step:
Assume that the statement holds for all finite lists of length n. We want to prove that the statement holds for all finite lists of length $n+1$. We write $l = (x : xs)$.

$$\begin{aligned} \text{filter } u \text{ (map } v \ l) &\equiv \text{filter } u \text{ (map } v \text{ } (x : xs)) \\ &=_{[\tau]} \text{filter } u \text{ } (v(x) : \text{map } v \ xs) \\ &=_{[\tau]} \begin{cases} \text{filter } u \text{ (map } v \ xs) & \text{if } u \circ v(x) = \mathrm{F} \\ (v(x) : \text{filter } u \text{ (map } v \ xs)) & \text{if } u \circ v(x) = \mathrm{T} \end{cases} \\ &\overset{\text{Ind. hyp.}}{=_{[\tau]}} \begin{cases} \text{map } v \text{ (filter } (u \circ v) \ xs) & \text{if } u \circ v(x) = \mathrm{F} \\ (v(x) : \text{map } v \text{ (filter } (u \circ v) \ xs)) & \text{if } u \circ v(x) = \mathrm{T} \end{cases} \end{aligned}$$

(2) Infinite lists

We prove by induction on m that for all infinite lists l

$$\text{filter } u \text{ (map } v \text{ } l) =_m \text{map } v \text{ (filter } (u \circ v) \text{ } l).$$

Base case: $m = 0$. This holds trivially.
Inductive step: There are two possibilities:
(i) $u \circ v(\text{hd}(\text{tl}^{(n)}(l))) =_{[\tau]} \text{F}$ for all $n \in \mathbb{N}$.
Since the evaluations of filter u (map v l) and map v (filter $(u \circ v)$ l) involve infinite unfoldings, it follows that both diverges. Hence the statement holds.
(ii) There is a minimum $n \in \mathbb{N}$ such that $u \circ v(\text{hd}(\text{tl}^{(n)}(l))) =_{[\tau]} \text{T}$.
Then we have:

	$\text{id}_{m+1}(\text{filter } u \text{ (map } v \text{ } l))$	
$=_{[\tau]}$	$\text{id}_{m+1}(\text{filter } u \text{ (map } v \text{ } \text{tl}^{(n)}(l)))$	(Kleene equiv.)
$=_{[\tau]}$	$\text{id}_{m+1}(v(\text{hd}(\text{tl}^{(n)}(l))) : \text{filter } u \text{ (map } v \text{ } \text{tl}^{n+1}(l)))$	(map & filter)
$=_{[\tau]}$	$(v(\text{hd}(\text{tl}^{(n)}(l))) : \text{id}_m(\text{filter } u \text{ (map } v \text{ } \text{tl}^{n+1}(l))))$	(Lemma 15.3.1)
$=_{[\tau]}$	$(v(\text{hd}(\text{tl}^{(n)}(l))) : \text{id}_m(\text{map } v \text{ (filter } (u \circ v) \text{ } \text{tl}^{n+1}(l))))$	(Ind. hyp.)
$=_{[\tau]}$	$\text{id}_{m+1}(v(\text{hd}(\text{tl}^{(n)}(l))) : \text{map } v \text{ (filter } (u \circ v) \text{ } \text{tl}^{n+1}(l)))$	(Lemma 15.3.1)
$=_{[\tau]}$	$\text{id}_{m+1}(\text{map } v \text{ filter } (u \circ v) \text{ } \text{tl}^{(n)}(l))$	(map & filter)
$=_{[\tau]}$	$\text{id}_{m+1}(\text{map } v \text{ (filter } (u \circ v) \text{ } l))$.	(Kleene equiv.)

The desired result then follows from Corollary 15.2.2.

□

Part V

Conclusion

In this part, we make some concluding remarks regarding the present research. In Chapter 16, we state some problems which we have encountered during the course of the research and which at the time of writing still remains open. Also we describe some possible work which may be carried out in the future. In Chapter 17, we give a summary of the contributions made in this thesis.

Chapter 16

Open problems and future work

In this chapter, we describe some open problems encountered in the course of our present work. In addition, we briefly explore future areas of work.

16.1 An operational proof of the minimal invariance property

In Chapter 13, we have employed a domain-theoretic denotational semantics to establish an operational minimal invariance property, i.e., Lemma 13.3.7. A natural question is whether this can be proven by purely operational methods. In the existing literature, Lassen [33] and Birkedal & Harper [9] contain an operational proof of a 'syntactic minimal invariance property'. However, as noted in Chapter 1, the language considered in these works only deal with one top-level recursive type. In what follows, we give an outline of one possible[1] adaption of Birkedal & Harper's method.

16.1.1 Functoriality

Recall that in Chapter 13, we can view FPC expressions as realisable functors. Since we wish to establish a purely operational proof of the operational minimal invariance property, we do not assume that FPC expressions preserve identity morphisms.

By the same construction in Section 13.3 , all FPC type expressions *almost* define realisable functors: all properties of functors hold except that we are

[1]The other possibility that may work is to adapt Lassen's relational reasoning about contexts [33]. Unfortunately, this reference is only known to the author very near the completion of the present writing.

not able to establish preservation of identities. We refer to such gadgets as *semi-functors*.

Although we have yet to show that the semi-functors associated to FPC types-in-context are indeed functors, we can say something about their action on the identities.

Lemma 16.1.1. *For every type-in-context $\Theta \vdash \tau$, the realisable semi-functor $T_{\Theta\vdash\tau}$ associated to it satisfies the following property:*
For every sequence of closed types $\vec{\sigma}$, it holds that

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

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

What we have been unable to show at the time of writing is that the opposite inequality also holds. We elaborate a possible proof strategy in Section 16.1.6. We first need some technical material developed in the next section.

16.1.2 Pre-deflations revisited

Recall that in Chapter 15, we defined a type-indexed family of pre-deflations $\mathrm{d}^\sigma : \overline{\omega} \to (\sigma \to \sigma)$. In particular, for the clause concerning recursive types, we abuse notation by writing $\Pi_2 S(\mathrm{d}^{\mu X.\sigma}(n), \mathrm{d}^{\mu X.\sigma}(n))$ as $S(\mathrm{d}^{\mu X.\sigma}(n))$ and define

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

Note that:

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

If one define the program $\Phi : (\mu X.\sigma \to \mu X.\sigma) \to (\mu X.\sigma \to \mu X.\sigma)$ by $\Phi(u, v) := (\mathrm{unfold}, \mathrm{fold}) \circ T_{X\vdash\sigma} \circ (\mathrm{fold}, \mathrm{unfold})$, then it is clear that

$$\Phi^{(n)}(\bot, \bot) = (\mathrm{d}^{\mu X.\sigma}(n), \mathrm{d}^{\mu X.\sigma}(n)).$$

Notice that establishing the operational minimal invariance property is equivalent to establishing that

$$\mathrm{fix}(\Phi) = (\mathrm{id}_{\mu X.\sigma}, \mathrm{id}_{\mu X.\sigma}).$$

In order to prove this, we propose to show that

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

It is easy to show that $d^{\mu X.\sigma}(\infty) \sqsubseteq \mathrm{id}_{\mu X.\sigma}$. In fact, we can do better.

Lemma 16.1.2. *For every closed type σ and every $n : \overline{\omega}$, the standard pre-deflation $\mathrm{d}^{\sigma}(n) : \sigma \to \sigma$ is*

(1) below the identity id_{σ}, and

(2) idempotent.

Proof. By induction on the structure of σ. The proof is fairly routine but the reader may want to note how part (1) is established for the recursive types. To do this, we have to prove by a further induction on n that

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

(n = 0): This is trivial since $d^{\mu X.\sigma}(0) = \bot \sqsubseteq \mathrm{id}$.
(n+1): Abusing notations, one may proceed as follows:

$$\begin{aligned} \mathrm{d}^{\mu X.\sigma}(n+1) &= \mathrm{fold} \circ T_{X \vdash \sigma}(\mathrm{d}^{\mu X.\sigma}(n)) \circ \mathrm{unfold} \\ &\sqsubseteq \mathrm{fold} \circ T_{X \vdash \sigma}(\mathrm{id}_{\mu X.\sigma}) \circ \mathrm{unfold} \\ &\sqsubseteq \mathrm{fold} \circ \mathrm{id}_{\mu X.\sigma} \circ \mathrm{unfold} \\ &= \mathrm{id}_{\mu X.\sigma} \end{aligned}$$

where the first $\sqsubseteq$ holds by induction hypothesis and the second $\sqsubseteq$ holds because of Lemma 16.1.1. □

So in order to prove the operational minimal invariance property, it suffices to show that

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

Before we proceed to the next subsection, we establish the following result.

Lemma 16.1.3. *Let $\vec{X} \vdash \sigma$ be a type-in-context. Then the semi-functor $S_{\vec{X} \vdash \sigma}$ satisfies the following (contextual) inequality.:*

$$S_{\vec{X} \vdash \sigma}(\mathrm{d}^{\tau_1}(\infty), \ldots, \mathrm{d}^{\tau_n}(\infty)) \sqsubseteq \mathrm{d}^{\sigma[\vec{\tau}/\vec{X}]}(\infty)$$

for any sequence of closed types $\vec{\tau}$.

Proof. By induction on the structure of $\vec{X} \vdash \sigma$. □

In order that the proof strategy of Birkedal & Harper [9] works for our language, we invoke the following conjecture.

Conjecture 16.1.4. *For every type-in-context of the form $X \vdash \sigma$, it holds that*

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

We have yet to establish this conjecture, though in many instances (e.g. list-type) this conjecture holds trivially. Notice that if we can establish the other inequality in Lemma 16.1.3, i.e.,

$$S_{\vec{X} \vdash \sigma}(\mathrm{d}^{\tau_1}(\infty), \ldots, \mathrm{d}^{\tau_n}(\infty)) \sqsupseteq \mathrm{d}^{\sigma[\vec{\tau}/\vec{X}]}(\infty)$$

holds for any sequence of closed types $\vec{\tau}$, then Conjecture 16.1.4 follows. However, a proof of the above inequality evades us at the time of writing.

In what follows, we show how an operational proof of Lemma 13.3.7 can be obtained assuming this conjecture. This conjecture is invoked only once and we indicate clearly where this takes place.

16.1.3 Compilation relation

One important tool which Birkedal & Harper use in their operational proof of the 'syntactic minimal invariance' (Theorem 3.66 of [9]) is the compilation relation $\Rightarrow$. In this section, we define and prove several elementary properties regarding this relation.

The *compilation relation* on $\mathrm{Exp}_\sigma(\Gamma)$ is defined by induction on the derivation of $\Gamma \vdash t : \sigma$, given by the axioms and rules in Figure 16.1.

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

Proposition 16.1.5. *If $\Gamma \vdash t : \sigma$, then $\Gamma \vdash t : \sigma \Rightarrow |t|$ for some unique $|t| \in \mathrm{Exp}_\sigma(\Gamma)$.*

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

Lemma 16.1.6. *If $\Gamma \vdash t : \sigma \Rightarrow |t|$, then $\Gamma \vdash \mathrm{d}^\sigma(\infty)(|t|) =_\sigma |t|$.*

Proof. By induction on the derivation of $\Gamma \vdash t : \sigma \Rightarrow |t|$.
The cases for ($\Rightarrow$ var), ($\Rightarrow$ pair), ($\Rightarrow$ inl), ($\Rightarrow$ inr), ($\Rightarrow$ abs), ($\Rightarrow$ up) and ($\Rightarrow$ fold) rely on the idempotence of $\mathrm{d}(\infty)$ (cf. Lemma 16.1.2 without having to invoke the induction hypothesis. We show the case for ($\Rightarrow$ var) here.
Given that $\Gamma \vdash x : \sigma \Rightarrow |x|$. By definition, $|x| = \mathrm{d}^\sigma(\infty)(x)$. We are to show that $\Gamma \vdash \mathrm{d}^\sigma(\infty)|x| = |x|$. But this follows from the idempotence of $d^\sigma(\infty)$, i.e., $\Gamma \vdash \mathrm{d}^\sigma(\infty)|x| =_\sigma \mathrm{d}^\sigma(\mathrm{d}^\sigma(x)) =_\sigma \mathrm{d}^\sigma(x) = |x|$.

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 16.1: Definition of $\Gamma \vdash t : \sigma \Rightarrow |t|$

The rest of the cases are fairly routine except for the case ($\Rightarrow$ unfold) which we now show.
Let $\Gamma \vdash \mathrm{unfold}(t) : \sigma[\mu X.\sigma/X] \Rightarrow |\mathrm{unfold}(t)|$ be given. We must show that

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

The inference rule ($\Rightarrow$ unfold)

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

guarantees that $|\mathrm{unfold}(t)| \equiv \mathrm{unfold}(|t|)$. The induction hypothesis asserts that $\Gamma \vdash \mathrm{d}^{\mu X.\sigma}(|t|) =_{\mu X.\sigma} |t|$. It then follows that

$$\begin{array}{lll}
 & \Gamma \vdash \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)|\mathrm{unfold}(t)| & \\
\equiv & \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)(\mathrm{unfold}(|t|)) & \\
\sqsupseteq & S_{X\vdash\sigma}(d^{\mu X.\sigma}(\infty)) \circ \mathrm{unfold}(|t|) & \text{(Lemma 16.1.3)} \\
= & \mathrm{unfold} \circ \mathrm{d}^{\mu X.\sigma}(\infty)(|t|) & \text{(def. of } \mathrm{d}^{\mu X.\sigma(\infty)}\text{)} \\
= & \mathrm{unfold}(|t|) & \text{(Ind. hyp.)} \\
= & |\mathrm{unfold}(t)|. &
\end{array}$$

But already, by Lemma 16.1.2, we always have $\mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)(|\mathrm{unfold}(t)|) \sqsubseteq |\mathrm{unfold}(t)|$ so that

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

□

Lemma 16.1.7. *If* $\Gamma \vdash t : \sigma \Rightarrow |t|$, *then* $\Gamma \vdash |t| \sqsubseteq_\sigma t$.

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

16.1.4 Compilation of a context

One last technical gadget is to compile a context $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$. For a given context $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$, we define a compiled context $|C|[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ using the axioms and rules similar to those for defining $\Gamma \vdash t : \sigma \Rightarrow |t|$. The axioms and rules for defining $|C|$ is given in Figure 16.2.

Lemma 16.1.8. *If* $\Gamma \vdash t : \sigma \Rightarrow |t|$ *and* $C[-_\sigma] \in \mathrm{Ctx}(\Gamma)$, *then*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 16.2: Definition of $\Gamma \vdash C[-_\sigma] : \tau \Rightarrow |C|[-_\sigma]$

Lemma 16.1.9. *Let* $C[-_\sigma] \in \mathrm{Ctx}_\tau(\Gamma)$ *and* $t \in \mathrm{Exp}_\sigma$. *Then*

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

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

16.1.5 A crucial lemma

The following lemma relies on Conjecture 16.1.4.

Lemma 16.1.10. *If Conjecture 16.1.4 holds, then*

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

Note that only the case ($\Downarrow$ unfold) uses Conjecture 16.1.4.

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

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

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

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

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

$$\begin{array}{lcll}
\emptyset \vdash |\mathrm{fst}(p)| & \equiv & \mathrm{fst}(|p|) & (\text{def. of } |\mathrm{fst}(p)|) \\
& =_\sigma & \mathrm{fst}(|(s,t)|) & (\text{Ind. hyp.}) \\
& =_\sigma & \mathrm{fst}(\mathrm{d}^\sigma(\infty)(|s|), \mathrm{d}^\tau(\infty)(|t|)) & (\text{def. of } |(s,t)|) \\
& =_\sigma & \mathrm{d}^\sigma(\infty)(|s|) & (\beta\text{-rule (7.11)}) \\
& =_\sigma & |s| & (\text{Lemma 16.1.6}) \\
& =_\sigma & |v|. & (\text{Ind. hyp.})
\end{array}$$

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

(3) ($\Downarrow$ app):
Given that $\emptyset \vdash s(t) \Rightarrow |s(t)|$ and $s(t) \Downarrow v$. We must show that $\emptyset \vdash |s(t)| =_\tau |v|$. The only derivation of $s(t) \Downarrow v$ is via an application of the evaluation rule ($\Downarrow$ app) whose premise is given by

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

The induction hypothesis asserts that $\emptyset \vdash |s| =_{\sigma\to\tau} |\lambda x.r|$ and $\emptyset \vdash r[t/x] =_\sigma |v|$. Then the desired result follows from:

$$\begin{array}{lcll}
\emptyset \vdash |s(t)| & \equiv & |s|(|t|) & \text{(def. of } |s(t)|) \\
& =_\tau & |\lambda x.r|(|t|) & \text{(Ind. hyp.)} \\
& \equiv & (\mathrm{d}^{\sigma\to\tau}(\infty)(\lambda x.|r|))(|t|) & \text{(def. of } |\lambda x.r|) \\
& \equiv & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r| \circ \mathrm{d}^\sigma(\infty))(|t|) & \text{(def. of } \mathrm{d}^{\sigma\to\tau}) \\
& =_\tau & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r|)(\mathrm{d}^\sigma(\infty)(|t|)) & \\
& =_\tau & (\lambda x.\mathrm{d}^\tau(\infty) \circ |r|)(|t|) & \text{(Lemma 16.1.6)} \\
& =_\tau & \mathrm{d}^\tau(\infty)(|r|[|t|/x]) & (\beta\text{-rule 7.10}) \\
& =_\tau & \mathrm{d}^\tau(\infty)(|r[t/x]|) & \text{(Lemma 16.1.8)} \\
& =_\tau & \mathrm{d}^\tau(\infty)(|v|) & \text{(Ind. hyp.)} \\
& =_\tau & |v|. & \text{(Lemma 16.1.6)}
\end{array}$$

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

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

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

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

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

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

The induction hypothesis asserts that

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

It then follows that

$$\begin{array}{lll}
\emptyset \vdash |\mathrm{case}(s)\ \text{of}\ \mathrm{inl}(x).t_1\ \text{or}\ \mathrm{inr}(y).t_2| & & \\
\equiv \quad \mathrm{case}(|s|)\ \text{of}\ \mathrm{inl}(x).|t_1|\ \text{or}\ \mathrm{inr}(y).|t_2| & & \text{(by def.)} \\
=_\rho \quad \mathrm{case}(\mathrm{inl}(|t|))\ \text{of}\ \mathrm{inl}(x).|t_1|\ \text{or}\ \mathrm{inr}(y).|t_2| & & \text{(Ind. hyp.)} \\
=_\rho \quad |t_1|[|t|/x] & & \text{(Kleene equivalence)} \\
=_\rho \quad |t_1[t/x]| & & \text{(Lemma 16.1.8)} \\
=_\rho \quad |v|. & & \text{(Ind. hyp.)}
\end{array}$$

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

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

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

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

The induction hypothesis asserts that

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

The desired result then follows from

$$\begin{array}{lll}
 & \emptyset \vdash |\text{case}(s)\ \text{of}\ \text{up}(x).t| & \\
\equiv & \text{case}(|s|)\ \text{of}\ \text{up}(x).|t| & (\text{def. of } |\text{case}(s)\ \text{of}\ \text{up}(x).t|) \\
=_\rho & \text{case}(|\text{up}(t')|)\ \text{of}\ \text{up}(x).|t| & (\text{Ind. hyp.}) \\
=_\rho & \text{case}(\text{up}(|t'|))\ \text{of}\ \text{up}(x).|t| & (\text{def. of } |\text{up}(t')|) \\
=_\rho & |t|[|t'|/x] & (\text{Kleene equivalence}) \\
=_\rho & |t[t'/x]| & (\text{Lemma 16.1.8}) \\
=_\rho & |v|. & (\text{Ind. hyp.})
\end{array}$$

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

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

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

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

The induction hypothesis asserts that

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

The desired result follows from

$$\begin{array}{lll}
 & \emptyset \vdash |\mathrm{unfold}(t)| & \\
\equiv & \mathrm{unfold}(|t|) & (\text{def. of } |\mathrm{unfold}(t)|) \\
=_{\sigma[\mu X.\sigma/X]} & \mathrm{unfold}(|\mathrm{fold}(s)|) & (\text{Ind. hyp.}) \\
\equiv & \mathrm{unfold}(\mathrm{d}^{\mu X.\sigma}(\infty)(\mathrm{fold}(|s|))) & (\text{def. of } |\mathrm{fold}(s)|) \\
=_{\sigma[\mu X.\sigma/X]} & \mathrm{unfold} \circ \mathrm{fold} \circ \mathrm{d}^{\sigma[\mu X.\sigma/X]}(\infty)(|s|) & (\text{Conjecture 16.1.4}) \\
=_{\sigma[\mu X.\sigma/X]} & \mathrm{unfold}(\mathrm{fold}(|s|)) & (\text{Lemma 16.1.6}) \\
=_{\sigma[\mu X.\sigma/X]} & |s| & (\beta\text{-rule (7.15)}) \\
=_{\sigma[\mu X.\sigma/X]} & |v|. & (\text{Ind. hyp.})
\end{array}$$

□

16.1.6 Incomplete proof of functoriality

In this subsection, we present an operational proof of Lemma 13.3.7 assuming that Conjecture 16.1.4 holds. Note that the only use of Conjecture 16.1.4 occurs in Lemma 16.1.10 above.

Lemma 16.1.11. *Let $f, g \in \mathrm{Exp}_{\mu X.\sigma \to \mu X.\sigma}$ be given. Suppose that for all $t \in \mathrm{Exp}_{\sigma[\mu X.\sigma/X]}$ and for all contexts of the form $C[-_{\mu X.\sigma \to \mu X.\sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$ it holds that*

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

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

Proof. By the extensionality property (7.22), in order to prove that $f \sqsubseteq g$, it suffices to prove that for all $s \in \mathrm{Exp}_{\mu X.\sigma}$, $f(s) \sqsubseteq_{\mu X.\sigma} g(s)$ holds. Let $s \in \mathrm{Exp}_{\mu X.\sigma}$ be given and suppose $C[-_{\mu X.\sigma}] \in \mathrm{Ctx}_\Sigma$ is such that $C[f(s)] \Downarrow \top$. Because of η-rule (7.27), it follows from the definition of $\sqsubseteq$ that

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

Thus by assumption that $C[f(\mathrm{fold}(t))] \sqsubseteq_\Sigma C[g(\mathrm{fold}(t))]$ for all $t \in \mathrm{Exp}_{\sigma[\mu X.\sigma/X]}$, it follows (by defining $t := \mathrm{unfold}(s)$) that $C[g(\mathrm{fold}(\mathrm{unfold}(s)))] \Downarrow \top$. Again invoking η-rule (7.27) and the definition of $\sqsubseteq$, we have that $C[g(s)] \Downarrow \top$, as required. □

Lemma 16.1.12. *If Conjecture 16.1.4 holds, then for any type-in-context of the form $X \vdash \sigma$, we have*

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

The conjecture is invoked via Lemma 16.1.10.

Proof. By Lemma 16.1.11, it suffices to show that for any $t \in \mathrm{Exp}_{\sigma[\mu X.\sigma/X]}$ and for any context $C[-_{\mu X.\sigma \to \mu X.\sigma}(\mathrm{fold}(t))] \in \mathrm{Ctx}_\Sigma$, it holds that

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

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

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

By Lemma 16.1.7, it suffices to show that

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

But by Lemma 16.1.6, it suffices to show that

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

By Lemma 16.1.10, $C[\mathrm{fold}(t)] \Downarrow \top$ implies that $|C[\mathrm{fold}(t)]| = |\top| =_\Sigma \top$. It then follows that

$$\begin{array}{rcll} C[\mathrm{fold}(t)] \Downarrow \top & \Longrightarrow & |C[\mathrm{fold}(t)]| \Downarrow \top & \text{(Lemma 16.1.10)} \\ & \Longrightarrow & |C|[|\mathrm{fold}(t)|] \Downarrow \top & \text{(Lemma 16.1.8)} \\ & \Longrightarrow & C[|\mathrm{fold}(t)|] \Downarrow \top. & \text{(Lemma 16.1.9)} \end{array}$$

which is what we aim to show. □

This and Lemma 16.1.2 establish functoriality, assuming Conjecture 16.1.4.

16.2 SFP structure on FPC closed types

The careful reader would have noticed that the pre-deflationary structure is not an SFP-structure in that the standard pre-deflations do not have finite images modulo contextual equivalence. To see this, just take the case of list-types, e.g. $[\tau]$. The pre-deflation at precision n is given by

$$\mathrm{d}_n^{[\tau]}(x : xs) = (x : \mathrm{d}_{n-1}^{[\tau]}(xs)).$$

Because τ may have an infinite number of elements (mod contextual equivalence), it is clear that $\mathrm{d}^{[\tau]}$ does not have a finite image modulo contextual equivalence. The question is whether it is possible to derive an SFP-structure on FPC closed types by defining families of SFP-deflations. At the time of writing, it is not known whether FPC types are SFP.

One possible way is to replace in the definition of standard pre-deflations the clause concerning recursive types by:

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

This is not a recursive definition of the form $\mathrm{fix}(t)$ for some suitable term t because it involves the type-index $\sigma[\mu X.\sigma]$ in the right-hand term. However, it can be shown, by finite number of unfoldings of $\mu X.\sigma$, that $\mathrm{d}^{\mu X.\sigma}$ can be defined via a finite system of mutual recursions. We omit the details here.

Notice that for list-types, the program

$$\mathrm{d}^{[\tau]}(n+1)(x:xs) = (\mathrm{d}^{\tau}(n)(x) : \mathrm{d}^{[\tau]}(n)(xs))$$

and thus $\mathrm{d}^{[\tau]}(n+1)$ has a finite image if $\mathrm{d}^{\tau}(n)$ and $\mathrm{d}^{[\tau]}(n)$ have finite images.

With this definition of $\mathrm{d} : \overline{\omega} \to (\sigma \to \sigma)$, one observes that Conjecture 16.1.4 holds by definition. So the good news is that we can prove operationally that

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

using the method outlined in the previous section. But the bad news is that one cannot use induction on types to establish that $\mathrm{d}_\sigma(n)$ is below id_σ and idempotent for all $n : \overline{\omega}$ because of the way $\mathrm{d}^{\mu X.\sigma}$ is defined.

16.3 Relational properties of recursive types

In Pitts [41], the relational properties of domains have been studied extensively. One application of this is a new proof technique for establishing computational adequacy. A natural question to ask is whether the present work can be developed to give an relational interpretation of recursive types in an operational setting. This question is answered by Birkedal & Harper [9] for a fragment of ML with one top-level recursive type. In that reference, the relational properties of types are exploited to prove correctness of CPS transformations. However, the study of relational properties of types has not been carried out for the language FPC. Though we are not sure of the implications of such a study, we believe that it should yield a better understanding of recursive types.

16.4 Non-determinism and probability

One real challenge is to extend our work to cope with non-determinism and probability. One possible direction is to work with the non-deterministic

typed λ-calculus considered by Hennessy & Ashcroft [26]. In this paper, the authors added a non-deterministic control structure to a typed λ-calculus and defined a computationally adequate domain-theoretic model for the language. Here we suggest two parts to this investigation: (1) To understand the theory of powerdomains (cf. Plotkin [44] and Smyth [54]) directly in terms of the operational semantics, and (2) to develop bisimulation and coinductive techniques for this non-deterministic language. To achieve these goals, one should first understand the relational techniques extensively employed by S.B. Lassen in Part II of his Ph.D. dissertation [34] to study contextual equivalence (both the may and must modalities) of programs of a non-deterministic functional language.

Chapter 17

Summary of work done

Recall that this thesis consists of the following four parts:

I Background

II Operational Toolkit

III Operational Domain Theory for PCF

IV Operational Domain Theory for FPC

17.1 Operational domain theory for PCF

The operational domain theory and topology developed for the language PCF reported in Part III of this thesis consists of joint work with Martín Escardó [14].

17.1.1 Rational completeness

In Chapter 6 of Part II, we identify rational-chain completeness as the salient completeness condition in the study of the contextual preorder of PCF terms. Note that we have to work with rational-chain completeness because Dag Normann [39] exhibited an example of an ω-chain with no least upper bound in the contextual order. We characterise these rational chains as programs defined on a "vertical natural numbers" type $\overline{\omega}$. Note that a proof of rational-completeness for FPC has been provided in Section 7.6 of Part IV. In the development of operational domain theory, we take the crucial step of replacing the directed sets by rational chains. In Part III, we demonstrate how, with this modification, many of classical definitions and theorems go

through. For instance, in Chapter 9 of Part III we show that programs of functional type preserve suprema of rational chains.

17.1.2 Operational topology

In Chapters 9 and 11 of Part III, we build upon two main ideas of Escardó's work [13]: (1) Open sets are defined via "Sierpinski-valued" programs. (2) Compact sets are defined via "Sierpinski-valued" universal quantification programs. We show how these notions, together with rational-chain completeness, gives rise to many familiar topological properties. In particular, we prove in Chapter 9 of Part III that (1) the open sets of any type are closed under the formation of finite intersections and rational unions, (2) open sets are rationally Scott open and in Chapter 11 of Part III that (3) compact sets satisfy the rational Heine-Borel property. In Chapter 11 of Part III, we introduce an operational notion of saturated set and well-filtered subspace. Also we study the properties of compact saturated sets and their relationship with finiteness and well-filteredness. We continue to show that familiar topological notions continue to hold the operational setting. For instance, (1) a set is compact iff its saturation is, and (2) every Hausdorff subspace is well-filtered. In particular, we produce uniform-continuity principles with which to reason about PCF programs.

17.1.3 Operational finiteness

In Chapter 10 of Part III, we define an operational notion of finiteness and establish an SFP-style characterisation of finiteness using rational chains of deflations. In passing, we established that (1) every element (closed term) of any type is the supremum of a rational chain of finite elements, and (2) two programs of functional type are contextually equivalent iff they produce a contextually equivalent result for every finite input. Another contribution in Chapter 10 of Part III is a topological characterisation of finiteness. In Chapter 10, we also formulate a number of continuity principles based on finite elements. Additionally, we see how the standard SFP-deflations give rise to an ultrametric on PCF. With respect to this metric, every operationally open set is metrically open. In Chapter 10, we also study dense subsets in connection with finiteness and obtain a Kleene-Kreisel density theorem for total elements. In Chapter 11 of Part III, we study the properties of compact saturated sets in connection with finiteness. In particular, we establish that every compact saturated set is the intersection of upper parts of finite sets of finite elements.

17.1.4 Data language

In order to be able to formulate certain specifications of higher-type programs without invoking a denotational semantics, we work, in Chapter 12 of Part III, with a "data language" for our programming language PCF, which consists of the latter extended with first-order "oracles" (Escardó [13]). The idea is to have a more powerful environment in order to get stronger program specifications. In particular, we establish some folkloric results in an operational setting. For instance, we prove, using standard SFP-deflations, that program equivalence defined by ground data contexts coincides with program equivalence defined by ground program contexts.

17.1.5 Program correctness

In Chapter 12 of Part III, we illustrate the versatility of the operational domain theory we developed to prove the correctness of non-trivial programs that manipulate infinite data. One such example is taken from Simpson [52]. We show that the given specification and proof in the Scott model can be directly understood in our operational setting.

17.2 Operational domain theory for FPC

Part of the operational domain theory developed for the language FPC reported in Part IV consists of work that appeared in Ho [28], and is my own work.

17.2.1 Type expressions as functors

In Chapter 13 of Part IV, we show how FPC types-in-context can be viewed as n-ary functors. We work with a syntactic category - the diagonal category, $\mathbf{FPC}_!^{\delta}$, i.e., the full subcategory of $\mathbf{F\breve{P}C}_!$ where $\mathbf{FPC}_!$ is the category of closed FPC types and strict programs. In this categorical setting, we define the class of realisable functors and showed how FPC types-in-context can be defined as realisable functors. Because the object part of a realisable functor is realised by a type-in-context and the morphism part by terms-in-contexts, any realisable functor is monotone and locally continuous. In order to establish the functoriality of type expressions, we prove operational analogues of useful domain-theoretic results such as the Plotkin's uniformity principle and the minimal invariance property. We employ a computationally adequate domain-theoretic model to establish the operational minimal

invariance property. Although we do not have a purely operational proof of this result, we outline in Chapter 16 a possible proof strategy.

17.2.2 Operational algebraic compactness

In Chapter 14, we show how the notion of algebraic completeness and compactness can be understood directly in an operational setting. We make use of the functorial status of type-in-contexts (established in Chapter 13 to provides a sound basis for the definitions of operational algebraic completeness and compactness). We also introduce parametrised algebraic completeness and compactness to cope with n-ary realisable functors. In the same chapter, we prove that the diagonal category is parametrised algebraically compact. We also consider an alternative choice of syntactic category, i.e., the product category $\mathbf{F\check{P}C}_!$ and show that this category is also parametrised algebraically compact. We briefly discuss the relationship between these categories.

17.2.3 Generic approximation lemma

In Chapter 15 of Part IV, we derive a pre-deflationary structure on the closed FPC types. This gives rise to a powerful proof technique for establishing program equivalence in FPC. This proof technique, known as the "Generic Approximation Lemma", was first developed by Hutton & Gibbons [31], via denotational semantics, for polynomial types (i.e., types built only from unit, sums and products). They suggested it would be possible to generalise the lemma "to mutually recursive, parametrised, exponential and nested datatypes" (cf. p.4 of Hutton & Gibbons [31]). In Chapter 15, we confirm this by using the operational domain theory developed in Chapter 13. Additionally, we use some running examples taken from [41] and [20] to demonstrate the power of the "Generic Approximation Lemma" as a technique for proving program equivalence by simple inductions, where previously various other more complex methods had been employed.

Appendix A

Improvements to Ho [28]

Since the publication of Ho [28], materials contained therein have been improved on and included in the various chapters of Part IV of this thesis. In addition, mistakes in the same reference have also been rectified. The various improvements are listed below:

(1) Materials on operational algebraic completeness and compactness of the diagonal category $\mathbf{FPC}_!^{\delta}$ have been re-organised in the form of Theorems 14.1.1, 14.1.2 and 14.1.3.

(2) Materials on operational algebraic completeness and compactness of the product category $\mathbf{F\breve{P}C}_!$ are new (see Theorems 14.2.4, 14.1 and 14.2.6).

(3) In Ho [28], it has been thought that the syntax somehow forbids us from using the product category $\mathbf{F\breve{P}C}_!$ as a categorical framework for our theory. We show, in this thesis, that this is not the case. In particular, we demonstrate how an operational domain theory for FPC can be worked out based on the category $\mathbf{F\breve{P}C}_!$. In addition, we compare the two approaches, i.e., the diagonal category $\mathbf{FPC}_!^{\delta}$ and the product category $\mathbf{F\breve{P}C}_!$. These are presented in Sections 14.2 and 14.3.

The following rectifications have been made:

(1) In Chapter 15, a pre-deflationary structure is derived on the closed types. This rectifies the mistake in Ho [28] that these standard "deflations" have finite images.

(2) The typographical error in the definition of $q'(x)$ in the proof of the Map-Iterate property in Ho [28] has been corrected. See Proposition 15.3.2.

(3) The proof of the Filter-Map property in Ho [28] is incomplete and in this thesis, we provide a complete proof. See Proposition 15.3.9.

Index

Bibliography

[1] M. Abadi and M.P. Fiore. Syntactic considerations on recursive types. In *Proceedings of the 11th Annual IEEE Symposium on Logic In Computer Science*, pages 242–252. IEEE Computer Society Press, 1996.

[2] S. Abramsky. The Lazy Lambda Calculus. In D. Turner, editor, *Research Topics in Functional Programming*, chapter 4, pages 65 – 117. Addison Wesley, December 1990.

[3] S. Abramsky and A. Jung. *Domain Theory*, volume 3 of *Handbook of Logic in Computer Science*. Clarendon Press, Oxford, 1994.

[4] M.J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

[5] U. Berger. *Totale Objekte und Mengen in der Bereichstheorie*. PhD thesis, Mathsmatisches Institut der Universitat Munchen, 1990.

[6] U. Berger. Computability and totality in domains. *Mathematical Structures in Computer Science*, 12(3):281–294, 2002.

[7] R. Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall International, 2nd edition, 1998.

[8] R. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice-Hall, 1988.

[9] L. Birkedal and R. Harper. Relational Interpretations of Recursive Types in an Operational Setting. *Information and Computation*, (155):3 – 63, 1999.

[10] N. Bourbaki. *General Topology*, volume 1,2. Addison-Wesley, London, 1966.

[11] P. Dybjer and H.P. Sander. A functional programming approach to the specification and verification of concurrent systems. *Formal Aspects of Computing 1*, pages 303–319, 1989.

[12] M.H. Escardó. Injective locales over perfect embeddings and algebras of the upper powerlocale monad. *Applied General Topology*, 4(1):193–200, 2003.

[13] M.H. Escardó. Synthetic topology of data types and classical spaces. *Electronic Notes in Theoretic Computer Science*, 87, 2004.

[14] M.H. Escardo and W.K. Ho. Operational domain theory and topology of a sequential language. In *Proceedings of the 20th Annual IEEE Symposium on Logic In Computer Science*, pages 427 – 436. IEEE Computer Society Press, 2005.

[15] M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1996. Distinguished Dissertations in Computer Science.

[16] M.P. Fiore and G.D. Plotkin. An Axiomatisation of Computationally Adequate Domain-Theoretic Models of FPC. In *Proceedings of the 10th Annual IEEE Symposium on Logic In Computer Science*, pages 92 – 102. IEEE Computer Society Press, 1994.

[17] P.J. Freyd. Recursive types reduced to inductive types. In *Proceedings of the 5th Annual IEEE Symposium on Logic In Computer Science*, pages 498–507. IEEE Computer Society Press, 1990.

[18] P.J. Freyd. Algebraically complete categories. In *Lecture Notes in Mathematics*, volume 1488, pages 95 – 104. Springer Verlag, 1991.

[19] P.J. Freyd. Remarks on algebraically compact categories. In *Applications of Categories in Computer Science*, volume 177, pages 95 – 106. Cambridge University Press, 1992. Lecture Notes in Mathematics.

[20] J. Gibbons and G. Hutton. Proof Methods for Corecursive Programs. *Fundamentae Informaticae*, 20:1 – 14, 2005.

[21] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M.W. Mislove, and D.S. Scott. *Continuous Lattices and Domains*. Number 93 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 2003.

[22] A. D. Gordon. Functional programming and input/output. In *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1994.

[23] A.D. Gordon. Bisimilarity as a theory of functional programming. *Notes Series: BRICS-NS-95-3, BRICS*, 1995. Department of Computer Science, University of Aarhus.

[24] C.A. Gunter. *Semantics of Programming Languages - Structures and Techniques*. The MIT Press, London, 1992.

[25] C.A. Gunter and D.S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretic Computer Science*, volume B, pages 635 – 674. Elsevier, 1990.

[26] M.C.B. Hennessy and E.A. Ashcroft. A mathematical semantics for a nondeterministic typed lambda-calculus. *Theoretical Computer Science*, 11(3):227–245, July 1980.

[27] W.K. Ho. Theory of Frames. Master's thesis, National Institute of Education, Nanyang Technological University, 2001.

[28] W.K. Ho. An Operational Domain-theoretic Treatment of Recursive Types. In M. Mislove and S. Brookes, editors, *Proceedings of the 22nd Conference on Mathematical Foundations in Programming Semantics*, number 158 in Electronic Notes in Theoretic Computer Science, pages 237–259, 2006.

[29] D.J. Howe. Equality in lazy computation systems. In *Proceedings of the 4th Annual Symposium on Logic In Computer Science*, pages 198–203. IEEE Computer Society Press, 1989.

[30] D.J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, February 1996.

[31] G. Hutton and J. Gibbons. The Generic Approximation Lemma. *Information Processing Letters*, 79(4):197 – 201, 2001.

[32] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 2nd edition, 1998.

[33] S.B. Lassen. Relational Reasoning about Contexts. *Higher Order Operational Techniques in Semantics*, 1998.

[34] S.B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, University of Aarhus, 1998.

[35] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266(1-2):341–364, September 2001.

[36] I.A. Mason, S.F. Smith, and C.L. Talcott. From operational semantics to domain theory. *Information and Computation*, 128(1):26–47, 1996.

[37] G. McCusker. Games and Full Abstraction for FPC. *Information and Computation*, (160):1–61, 2000.

[38] R. Milner. Fully abstract models of typed lambda-calculi. *Theoretical Computer Science*, 4:1–22, 1977.

[39] D. Normann. On sequential functionals of type 3. *Mathematical Structures for Computer Science*, 16:279–289, 2006.

[40] C.-H.L. Ong. Operational and Denotational Semantics of PCF. *Notes Series: BRICS-NS-99-51, BRICS*, 1999. Notes for a course of the Summer School on Semantics of Computation at BRICS, University of Aarhus.

[41] A.M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A.M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

[42] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(1):223–255, 1977.

[43] G. Plotkin. Pisa notes on domains. Department of Computer Science, University of Edinburgh, 1983. Available from `http://www.dcs.ed.ac.uk/home/gdp/publications/`.

[44] G.D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5:452–487, 1976.

[45] G.D. Plotkin. Lectures on predomains and partial functions, 1985. Notes for a course given at the Center for the Study of Languages and Information, Stanford.

[46] A. Poigné. Basic category theory. In *Handbook of logic in computer science*, volume 1, pages 413–640. Oxford University Press, New York, 1992.

[47] A. Rohr. *A Universal Realizability Model for Sequential Functional Computation*. PhD thesis, Technischen Universitat Darmstadt, July 2002.

[48] D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Transactions on Programming Languages and Systems*, 18(2):175 – 234, March 1996.

[49] D.S. Scott. Continuous lattices. In *Toposes, Algebraic Geometry and Logic*, Lecture Notes in Mathematics, pages 97–136. Springer-Verlag.

[50] D.S. Scott. Data types as lattices. *SIAM Journal on Computing*, 5(3):522–587, 1976.

[51] D.S. Scott. Domains for denotational semantics. In M. Neilson and E.M. Schmidt, editors, *Automata, Languages and Programming, Proceedings*, volume 140. Springer-Verlag, Berlin, 1982.

[52] A. Simpson. Lazy functional algorithms for exact real functionals. *Lecture Notes in Computer Science*, (1450):323–342, 1998.

[53] A.K. Simpson. Recursive Types in Kleisli Categories. Available from `http://homepages.inf.ed.ac.uk/als/Research`, 1992.

[54] M.B. Smyth. Powerdomains. *Journal of Computer and System Sciences*, 16:23–36, 1978.

[55] M.B. Smyth. *Topology*, volume 1 of *Handbook of Logic in Computer Science*. Clarendon Press, Oxford, 1992.

[56] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11:761–783, 1982.

[57] A. Stoughton. Interdefinability of Parallel Operations In PCF. *Theoretical Computer Science*, 79:357–358, 1991.

[58] T. Streicher. Mathematical foundations of functional programming. Department of Mathematics, University of Darmstadt, 2003. Available from `http://www.mathematik.tu-darmstadt.de/~streicher/`.

[59] T. Streicher. *Domain-theoretic foundations of functional programming*. To appear.

[60] W.A. Sutherland. *Introduction to metric and topological spaces*. Clarendon Press, Oxford, 1986.

[61] S. Vickers. *Topology via Logic*. Cambridge University Press, Cambridge, 1989.