# A Universal Realizability Model for Sequential Functional Computation

Vom Fachbereich Mathematik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte

**Dissertation**

von
**Dipl.-Math. Alexander Rohr**
aus Wiesbaden

| | |
|---|---|
| Referent: | Prof. Dr. Thomas Streicher |
| Korreferent: | John Longley, PhD |
| Tag der Einreichung: | 25. April 2002 |
| Tag der mündlichen Prüfung: | 5. Juli 2002 |

Darmstadt 2002
D 17

# Abstract

We construct a universal and even logically fully abstract realizability model for the sequential functional programming language of call-by-name FPC. This model is defined within the category of modest sets over the total combinatory algebra $\mathcal{L}$ of observational equivalence classes of closed terms of the untyped programming language $\lambda$+Error. This language is untyped lazy call-by-name lambda-calculus extended by a single constant $\mathsf{ERR}$ and a conditional construct which distinguishes this constant from any other syntactic value.

Universality and (constructive) logical full abstraction of the model are proved in three steps. Firstly, a canonical universal and logically fully abstract realizability model for FPC over the typed combinatory algebra $\mathcal{T}$ of observational equivalence classes of closed FPC-terms is constructed. Then the recursive type $U := \mu\alpha.\, \mathbf{void} + [\alpha \to \alpha]$ is shown to be universal, i. e. any other type is a definable retract of $U$. Hence this type gives rise to an untyped combinatory algebra $\mathcal{U}$ which is applicatively equivalent to the typed combinatory algebra $\mathcal{T}$. As a consequence, the realizability toposes over $\mathcal{T}$ and $\mathcal{U}$ are equivalent and hence the realizability model for FPC over $\mathcal{U}$ also universal and logically fully abstract. Finally it is shown that every closed FPC-term of type $U$ can be defined in the untyped language $\lambda$+Error. Hence the combinatory algebras $\mathcal{U}$ and $\mathcal{L}$ are applicatively equivalent. It follows that the realizability model over $\mathcal{L}$ is universal and logically fully abstract. As a consequence we prove a variant of the Longley-Phoa Conjecture.

# Zusammenfassung

Wir konstruieren ein universelles und sogar logisch voll abstraktes Realisierbarkeitsmodell für die sequentielle funktionale Programmiersprache Call-by-name-FPC. Das Modell existiert in der Kategorie der Modest Sets über einer totalen kombinatorischen Algebra $\mathcal{L}$ bestehend aus observationalen Äquivalenzklassen von geschlossenen Termen der ungetypten Programmiersprache $\lambda$+Error. Diese Programmiersprache ist eine Erweiterung des ungetypten Call-by-name-lambda-Kalküls um eine einzelne Konstante ERR und ein Konditionalkonstrukt, welches diese Konstante von jedem anderen syntaktischen Wert zu unterscheiden vermag.

Die Universalität und die (konstruktive) logische volle Abstraktheit des Modells werden in drei Schritten gezeigt. Zunächst wird ein kanonisches universelles und logisch voll abstraktes Realisierbarkeitsmodell für FPC über der getypten kombinatorischen Algebra $\mathcal{T}$ der Äquivalenzklassen der geschlossenen FPC-Terme modulo beobachtbarer Gleichheit konstruiert. Anschließend wird gezeigt, daß der rekursive Typ $U := \mu\alpha.\,\mathbf{void} + [\alpha \to \alpha]$ universell ist, daß also jeder andere Typ ein definierbares Retrakt von $U$ ist. Der universelle Typ induziert eine ungetypte kombinatorische Algebra $\mathcal{U}$, welche applikativ äquivalent zur getypten kombinatorischen Algebra $\mathcal{T}$ ist. Folglich sind die Realisierbarkeitstopoi über $\mathcal{T}$ und $\mathcal{U}$ äquivalent und daher ist das Realisierbarkeitsmodell von FPC über $\mathcal{U}$ ebenfalls universell und logisch voll abstrakt. Schließlich wird bewiesen, daß jeder geschlossene FPC-Term vom Typ $U$ in der ungetypten Sprache $\lambda$+Error definierbar ist. Daher sind die kombinatorischen Algebren $\mathcal{U}$ und $\mathcal{L}$ applikativ äquivalent. Es folgt, daß das Realisierbarkeitsmodell über $\mathcal{L}$ universell und logisch voll abstrakt ist. Als Folgerung daraus beweisen wir eine Variante der Longley-Phoa-Vermutung.

# Erklärung

Ich habe die vorliegende Arbeit, abgesehen von den in ihr ausdrücklich genannten Hilfen, selbständig verfaßt.

*Alexander Rohr*

# Contents

# Acknowledgements

In the first place I want to express my gratitude towards my advisor Thomas Streicher for his support and for many helpful explanations, comments and discussions. Further, I am grateful to John Longley who, as a referee, carefully read and examined this thesis and provided many helpful comments. I would also like to thank Hanno Nickau, who came to Darmstadt in April 1998 for some days to present his ideas about realizability models based on games and to discuss them with us.

Peter Lietz, Tobias Löw and Michael Marz deserve special thanks for their careful proofreading and for many helpful suggestions.

Furthermore, I am grateful to our department for the opportunity to teach mathematics to students of many different courses and years. I always enjoyed this very much. Moreover I would like to thank the members of the Hochschuldidaktische Arbeitsstelle and Reiner Liese and especially Michael Heger who helped me to learn a lot about learning, teaching and communication.

This thesis was typeset using teTEX together with LATEX, $\mathcal{AMS}$-LATEX, XY and other macro packages. I would like to thank all people involved in developing and providing all this free software.

I have received financial support from the federal state of Hessen through a grant according to the "Hessisches Gesetz zur Förderung von Nachwuchswissenschaftlern". I have also received support from the Fachbereich Mathematik and the European Community, through the Esprit Working Group on Applied Semantics. I am very grateful to these bodies for their generosity.

Finally, I want to express my gratitude to all the people who have helped me in other ways through the years; to name them would take many pages. In particular my thank goes to my friends from the Fachschaft Mathematik together with whom I experienced and learned so many things. I want to thank my mother, who encouraged me and made many sacrifices for my sake. Very special thanks go to Viola. Without her love and support I would hardly ever have finished this thesis.

# 1 Introduction

The aim of this thesis is to present a universal and (constructively) logically fully abstract realizability model for a typed sequential functional programming language with recursive types. Hence I sketch the full abstraction problem for PCF and discuss different constructions of fully abstract models in Section 1.1 and 1.2. A short overview of the historical context of realizability and topos theory is given in Sections 1.3 and 1.4. In Section 1.5 different realizability models for sequential computation are discussed. An overview of this thesis is given in Section 1.6.

## 1.1 The full abstraction problem for sequential functional languages

In the sixties, Dana Scott defined the language LCF (Logic of Computable Functionals) for reasoning about computable functionals. His paper [Sco93] circulated as an unpublished but most influential manuscript for many years. It was Gordon Plotkin who first studied the properties of this language as a prototypical functional programming language PCF (Programming Computable Functionals) [Plo77]. The language PCF is the simply typed $\lambda$-calculus extended by a base type for natural numbers and constructors for arithmetic operations, a conditional and a fixed point operator. Due to the chosen evaluation strategy of leftmost outermost reduction, PCF-terms denote sequential functions in a strong sense: it follows from Plotkin's Activity Lemma that every PCF-term $t$ contains a uniquely determined sub-term which has to be reduced in the next step of evaluation. In other words, derivation trees for judgements of the form $t \Downarrow v$ ('term $t$ reduces to value $v$') are unique. However, every partial recursive function $\mathbb{N} \to \mathbb{N}$ is PCF-definable.

Scott presented a denotational model for PCF where every type is interpreted as a pointed partially ordered set closed under directed suprema (to be more precise, it is interpreted as a so called Scott-domain), and every term is interpreted as a Scott-continuous function between Scott-domains. This model is adequate, but not all computable continuous functions are definable in PCF, e. g. there is no PCF-term implementing the 'parallel or' function. Moreover, there are two different PCF-terms denoting different functions in the Scott model, which cannot be distinguished by observations in the programming language (the terms are *observationally equivalent*). Thus the question arose, whether there was a *fully abstract* model for PCF, i. e. a model where the interpretations of any two terms were equal if, and only if, the terms were observationally equivalent. Such a fully abstract model would provide a very good semantical representation of the language. However, even in a fully abstract model not every element has to be definable; models with this stronger property are called *universal*.

It turned out to be quite difficult to find a satisfactory fully abstract model for PCF. Robin Milner did find a fully abstract model within the ideal completion of a quotient set of certain PCF-terms. Furthermore, he showed that all order-extensional fully abstract domain models (i. e. cpo-enriched models) of PCF are isomorphic. But the presentation of the fully abstract model as a term model is in some sense unsatisfactory as it does not provide real new insights into the nature of sequentiality. A good general reference for semantics of PCF is [Ong95].

The real aim was to find a satisfactory set-theoretic presentation for the fully abstract model and thus to give a non-operational characterisation of sequential higher type functionals. The problem resulted from demanding that interpretations for PCF-terms should, on the one hand, interact like functions, i. e. in an extensional way, while, on the other hand, they should embody the intensional property of being sequential.

The problem remained unsolved for many years. An intensional model based on so-called *sequential algorithms* was introduced by Berry and Curien in [BC82]. Bucciarelli and Ehrhard presented an extensional model in the category of dI-domains and strongly stable functions [BE91] and Ehrhard showed in [Ehr96] that this model is isomorphic to the extensional collapse of the sequential algorithms. However, it turned out that sequential algorithms describe a different, more liberal notion of sequentiality than PCF does. Thus the strongly stable model is not fully abstract for PCF.

In [Mul87, JS93] it was proved that the extensional collapse of the syntactically definable fragment of the Scott model is fully abstract for PCF. This representation of the fully abstract model is syntax free, but it has the disadvantage that every PCF-term is interpreted as an equivalence class of continuous functions instead of being interpreted as a single function.

A natural way to construct domain models for PCF is to equip complete partial orders (cpo) with additional structure and to consider only those Scott continuous functions which preserve this structure. That way O'Hearn and Riecke found their Kripke relations model [OR95], which is fully abstract. On each cpo O'Hearn and Riecke defined an uncountable family of relations and restricted the morphisms in their model to the continuous functions respecting these relations. Their approach generalized that of [Sie92] by generalizing logical relations (as introduced by Plotkin in [Plo73]) to the more general Kripke logical relations, which had been introduced by Jung and Tiuryn [JT93]. Riecke and Sandholm [RS97b, RS99b] generalized this approach to obtain a fully abstract model for call-by-value FPC. Following the same relational approach, Marz [Mar00] defined a category SD of Sequential Domains and presented a fully abstract model for a very general sequential language SFL incorporating both call-by-name and call-by-value constructs. The disadvantage of this kind of model is the vast number of relations imposed on each domain. On the other hand, according to Loader [Loa01], observational equivalence is undecidable even for finitary PCF and hence one cannot expect to find a simpler model in this way.

## 1.2 Game models for sequential computation

In the years 1994 and 95 several new and independently developed representations of the fully abstract model for PCF were presented, which are based on a game theoretic approach. One of these was described by Abramsky, Malacaria and Jagadeesan [AJM00], another one by Hyland and Ong [HO00] and independently by Nickau [Nic94].

Abramsky, Malacaria and Jagadeesan suggest considering "intensional semantics" as a connection between operational and denotational semantics (cf. [Gir89, Gir90]). Such a model does not have to be extensional, but all morphisms have to respect extensional equality. Abramsky et alii define an intensional model to be "intensionally fully abstract" iff the extensional collapse of it is fully abstract. They define an intensional game semantics for PCF, where types are represented as a special kind of two person games and terms are interpreted as history-free strategies. They consider game semantics to be an expressive and comprehensive theory of intensional semantics for programming languages, which allows one to investigate PCF and non-extensional extensions of it in a unified setting and which is more appropriate for the dynamic character of sequentiality than domain semantics (see also [AHM98] and Jim Laird's PhD Thesis [Lai98]). However, it is still open whether this fully abstract game model is cpo-enriched.

Game semantics looks at an information processing system and its environment as player and opponent, respectively, in a game with questions and answers. The rules of the game correspond to the specification of the possible interactions between a system and its environment. A closed term is interpreted as a strategy specifying how the system should actually behave and an actual play corresponds to a 'run' of a program.

To obtain a model for PCF, Abramsky et al. define a symmetric monoidal closed category of games with an exponential '!'. Objects of this category are two person games and a morphism between two games is a strategy in a new game which is composed from the two given ones. The co-Kleisli-category of this category of games with respect to '!' is a rational cartesian closed category which contains an intensionally fully abstract model for an extension of PCF by a construct for infinite case analysis. Factorizing the sets of morphisms by observational equivalence one obtains a rational cartesian closed category giving rise to a fully abstract model for PCF. Abramsky et al. argue that the collapse is obtained by a continuous homomorphism and that this was an advantage over the extensional collapse of the Scott model obtained by Mulmuley, Stoughton and Jung. However, it is not known whether this model is cpo-enriched.

In [HO00] a similar kind of game semantics is introduced, considering certain games with questions and answers and interpreting PCF-terms as strategies. In contrast to Abramsky et al., this model is not based on linear logic. Hyland and Ong obtain the existence of exponentials by considering *innocent strategies* where moves depend only on a limited view of the previous history. Thus they define a cartesian closed category of games to obtain an intensionally fully abstract model for an extension of PCF. They too get the fully abstract model as the extensional collapse of this model. The model by Hanno Nickau appears as quite similar to the one by Hyland and Ong [Nic94].

From our point of view, the drawback of these fully abstract models for PCF is, that terms have to be interpreted as equivalence classes of strategies rather than single strategies. To us, game semantics seems better suited for non-extensional functional languages like, e. g., $\mu$PCF, an extension of PCF. Jim Laird showed in [Lai01b] that the extensional collapse of certain Hyland-Ong-style game models is isomorphic to the sequential algorithms model for PCF. Thus Laird shows that any two $\mu$PCF-terms are observationally equivalent if, and only if, their interpretations as sequential algorithms are equivalent. Hence the sequential algorithms model is fully abstract and not only 'intensionally fully abstract' for $\mu$PCF.

## 1.3 Realizability

The notion of realizability was introduced by Stephen Cole Kleene in [Kle45], who wanted to explore the connection between Intuitionism and the theory of recursive functions (both theories stressing the importance of extracting information *effectively*). Inspired by certain formulations in Hilbert and Bernays [HB34], he started thinking about it in 1940. His definition specifies in an inductive way what it means that a natural number $n$ realizes a sentence $\varphi$ of the language of arithmetic. Kleene presented an interpretation of intuitionistic logic which can be understood from the point of view of a classical mathematician by giving an interpretation of the intuitionistic connectives in terms of the classical ones.

Kleene's approach is one possible way to make the Brouwer-Heyting-Kolmogorov interpretation of logic more precise, which dates back to the early 1930s and is widely recognized as the intended semantics for intuitionistic logic. This corresponds to the analogy of 'propositions as types': formulae correspond to types and their deductions to terms of this type. In other words, a proposition is considered as the type of its proofs. This analogy, discovered by Howard [How80] in Curry's work, was developed by Dana Scott and Per Martin-Löf [ML84]. See [Kle73] for a retrospective survey of Kleene's work and [vO00] for a survey of realizability in general. A thorough treatment of realizability can be found in [Tro73] and [Tro98].

The first definition of realizability which is based on a general notion of *combinatory algebra* appears in [Sta73]. Feferman, in [Fef75], sets out to code what he calls "explicit mathematics" in a language for partial combinatory algebras. A partial combinatory algebra $A$ is a set $A$ equipped with a partial binary operation (*application*) $x$, $y \mapsto xy$ such that there are elements (*combinators*) $k$ and $s$ satisfying the postulates

> $kx$ and $(kx)y$ are always defined, and $kxy = x$;
> $sxy$ is always defined and $xz(yz)$ is defined iff $sxyz$ is, and in this case $sxyz = xz(yz)$.

The natural numbers with partial recursive function application form a partial combinatory algebra, called the first Kleene algebra. Another example is the set of functions $\mathbb{N}^{\mathbb{N}}$ with application as follows: every function $\alpha$ codes a partial continuous operation $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ (see e. g. [Tro98]). This partial combinatory algebra, the second Kleene algebra, was at the basis of Kleene's function realizability [Kle65, KV65, Kle69] providing an interpretation of "intuitionistic analysis" (a theory validating continuity principles and bar induction which treats numerical functions as well as natural numbers; the functions often being seen as reals).

An example of a total combinatory algebra is the set of closed terms of the $\lambda$-calculus modulo $\beta$-equivalence [Bar84].

Within any partial combinatory algebra $\mathcal{A}$ one can implement the booleans and natural numbers, pairing and projection operators, primitive recursors and fixed-point operators. Moreover, any total recursive function is representable in $\mathcal{A}$. Properties such as this give the substance to the idea that any partial combinatory algebra provides a rich universe of untyped computation [Lon98].

For any partial combinatory algebra $\mathcal{A}$ one can define cartesian closed categories $\mathsf{Ass}(\mathcal{A})$ and $\mathsf{Mod}(\mathcal{A})$ of *assemblies* and *modest sets* over $\mathcal{A}$. An assembly is a set together with a realizability relation specifying, for each element $x \in X$, a non-empty subset of $\mathcal{A}$ thought of *realizers* (codes, algorithms) for $x$. Morphisms in $\mathsf{Ass}(\mathcal{A})$ are those functions which are *tracked* by an element of $\mathcal{A}$.

A modest set is an assembly $X$ where each element of $\mathcal{A}$ realizes at most one element of $X$ (the details are given in Definition 2.4). Modest sets over $\mathcal{A}$ correspond to partial equivalence relations (PERs) on $\mathcal{A}$.

The categories of assemblies and modest sets play a major role in realizability semantics. They can be identified with certain subcategories of realizability toposes.

## 1.4 Toposes and modest sets

Around 1970, Lawvere and Tierney generalized Grothendieck's notion of topos to the notion of *elementary topos* generalizing semantical ideas that had developed in the sixties [Joh77]. Dana Scott, in his foreword to [Bel77], pointed out that toposes provide a natural framework for semantics of higher-order languages. Following ideas of Scott to think about realizability as a kind of truth-value semantics, a completely new type of toposes was discovered around 1979 by Martin Hyland, Peter Johnstone and Andy Pitts [HJP80, Pit81, Hyl82]. Hyland defined the effective topos $\mathcal{E}\!f\!f$ and together with Johnstone and Pitts he described a construction of a standard realizability topos $\mathsf{RT}_{\mathcal{A}}$ over an arbitrary partial combinatory algebra $\mathcal{A}$. In particular, the effective topos is the standard realizability topos over the first Kleene algebra.

Furthermore, Hyland singled out an interesting subcategory of $\mathcal{E}\!f\!f$: the subcategory of what he called 'effective objects'. This subcategory is equivalent to the category of $\omega$-Sets, i. e. the category of modest sets or partial equivalence relations (PERs) over the first Kleene algebra. The idea behind the construction of PERs can be traced back to Kreisel [Kre59] and appears more explicitly in [Gir72].

## 1.5 Realizability models for sequential computation

Around 1985, Moggi and Hyland discovered that a specific internal category in $\mathcal{E}\!f\!f$ was complete in some sense. As a consequence, Moggi found an interesting set-theoretic interpretation of Girard's second-order $\lambda$-calculus 'system $F$' [Gir72] in the category of $\omega$-Sets [LM91]. Subsequently, realizability models for constructive type theories and for System F were studied by Ehrhard, Hyland, Jacobs, Pavlovic, Robinson, Rosolini, Streicher and many others; nowadays realizability models form a standard tool in the semantics of programming languages. John Longley was the first to systematically study and relate realizability models over different partial combinatory algebras [Lon95]. He gives a good introduction to the basic ideas of realizability semantics in [Lon98]:

"We start by choosing some model of data and computations on it which we think of as *primitive* [e. g. a combinatory algebra]. If we like, we can think of this primitive layer as a model for low-level or "machine level" computation. In some cases, this might look quite close to what actually goes on inside a machine; in other cases, it might correspond to a machine only in some quite abstract sense.

In any case, we then build a category of *datatypes* derived from this primitive model of computation [e. g. the corresponding category of modest sets]. We can think of these as high-level or "programming language level" datatypes, and they will include types for natural numbers, functions, lists, streams and many other kinds of data familiar to functional programmers. In our model, these high-level datatypes will be related to the world of low-level computation as follows: each element $x$ of a high-level datatype will have a non-empty set $\|x\|$ of low-level representations or *realizers*, and moreover every function between datatypes in the model will be *realized* by some low-level computation acting on these representations. This corresponds to the idea that, in any high-level programming language, all data values must somehow have representations at a lower machine level, and all run-time computations really happen at this machine level. We might have in mind the representation of data as sequences of bytes, or some intermediate level of description such as the representation of a functional program by closures.

Even in terms of the vague description just given, we may note two typical features of realizability models. Firstly, an element $x$ of a datatype may have more than one realizer. This corresponds to the fact that the "same" value (for instance, the same function) may have several possible machine representations, the differences between them being irrelevant from the point of view of the high-level programmer. (Of course, there might be important differences between representations from the point of view of *efficiency*, but here we are considering only the relation between inputs and outputs.)

Secondly, the fact that we require every function in the high-level model to be realized by a low-level computation means that we obtain a category in which *all morphisms are computable* in the sense determined by the underlying machine level. We should clarify what we mean here by *computable.* In some instances, our low-level model may have some notion of effectivity built into it, and in this case, all morphisms in the high-level model may indeed be computable in some realistic sense. In other instances, the low-level model might admit many non-effective computations, in which case morphisms need not be computable in any genuinely effective sense. Nevertheless, we can still regard the morphisms as computable in an *abstract* sense: we think of the combinatory algebra as defining what we mean by computability, and proceed from there."

Longley [Lon98] also discusses important advantages of realizability models:

"1. Firstly, [...] realizability models offer a very rich categorical structure, and, there are known techniques for modelling phenomena such as polymorphism and recursive types. If one found a good (e. g. universal) realizability model for PCF, one would know in advance, as it were, that all this machinery would be available. For other kinds of model, one might have to work harder to model all these additional language features.

2. Another distinctive feature of realizability models is that they have their own *internal logic*. In other words, realizability toposes can be seen as "universes" within which one can perform various kinds of constructive "set-theoretic" reasoning. Although the precise practical significance of this fact is perhaps not yet clear, the ongoing work of Reus and Streicher [RS97a] explores the possibility of using this logic as a basis for program verification.

3. Many of the combinatory algebras [...] are in some way inspired by the fully abstract models mentioned above; however, it sometimes turns out that the realizability models are technically simpler to construct than the models that inspired them. A good example is Abramsky's model of well-bracketed strategies [...]. The corresponding realizability model [see below] closely resembles the games model of [AJM00], but many of the rather technical conditions appearing in [AJM00] become unnecessary in the realizability model because they actually come for free as consequences of quite simple definitions. This suggests that realizability models may sometimes be [...] useful as ways of simplifying the presentation of other known models. [As a footnote Longley adds:] We have already hinted at the idea that with untyped realizability models one frequently gets a lot out for what one puts in. It is worth mentioning another example of this phenomenon: The [first Kleene algebra] [...] gives rise to the same type structure as the effective Scott model (see e. g. [Lon95]). The construction of the Scott model is based on *continuity*, to which effectivity is then added on, while the realizability model is constructed using only *effectivity*, and continuity follows as a consequence."

In 1996/97 Jaap van Oosten and John Longley independently developed a partial combinatory algebra such that the corresponding realizability model for PCF is equivalent to the strongly stable model of Bucciarelli and Ehrhard.

In [MRS99], Marz, Streicher and the present author present a total combinatory algebra $A$ which is the solution of a recursive domain equation in the category of sequential domains defined in [Mar00]. They show that the category $\mathsf{Mod}\,A$ of modest sets over $A$ contains a fully abstract model for PCF. Moreover, an extension of untyped lazy call-by-name $\lambda$-calculus is defined which can be interpreted in $A$ such that the syntactically definable subalgebra of $A$ gives rise to another fully abstract realizability model for PCF. This leads to a proof of a variant of the Longley-Phoa Conjecture. It is claimed that the latter model is universal for PCF. This is the case if, and only if, any SFL-term of a PCF-type is PCF-definable, i. e. if SFL (or, equivalently, call-by-name FPC) is a conservative extension of PCF. McCusker showed in his PhD Thesis [McC98] that game semantics is conservative over PCF in the sense that every effective strategy in the game model for PCF is definable by a PCF-term.

In [MRS99] it is further announced that using similar methods one could obtain a universal model for a much more general sequential functional language which is equivalent to call-by-name FPC. The aim of the present thesis is to work out the announced ideas.

In [Lon99a] and [Lon99b], Longley defines a typed version of partial combinatory algebras and a 2-category of all typed and untyped combinatory algebras. As observed by Lietz and Streicher in [LS02], a typed partial combinatory algebra contains a universal type if, and only if, it is applicatively equivalent to an untyped partial combinatory algebra.

In this thesis two applicatively equivalent combinatory algebras are presented, a typed and an untyped one, such that the modest sets over each of these contain a universal and logically fully abstract model for call-by-name FPC. The notion of (constructive) logical full abstraction of a realizability model was defined by Longley in [Lon99a]. He shows that logical full abstraction is a stronger criterion than universality.

Note that toposes over different partial combinatory algebras need not be equivalent even if the corresponding finite type hierarchies over the $\mathbb{N}_\perp$ are equivalent. For example, Abramsky and Longley define a partial combinatory algebra $\mathcal{A}_{wb,\mathit{eff}}$ of effective well-bracketed history-free strategies in [LA99] such that the corresponding realizability model is universal for PCF. Hence this model is isomorphic to the model presented in this thesis. However, it is not clear whether the model defined by Abramsky and Longley is logically fully abstract. Moreover, Gernot Belger proved in his Master's Thesis [Bel00] that the realizability topos over $\mathcal{A}_{wb,\mathit{eff}}$ is not equivalent to the topos over the partial combinatory algebra $\mathcal{U}$ presented in this thesis. He showed that the type $(N \to N_\perp) \to N_\perp$ is not projective in the modest sets over $\mathcal{A}_{wb,\mathit{eff}}$ whereas it is projective in $\mathsf{Mod}(\mathcal{U})$. As the combinatory algebra $\mathcal{U}$ is applicatively equivalent to the combinatory algebra of closed FPC-terms, the realizability topos over $\mathcal{U}$ is the best possible choice for a realizability semantics of FPC, as the internal logic of this topos exactly reflects the properties of the language. Thus the realizability model over $\mathcal{U}$ provides the best possible realizability semantics of FPC.

## 1.6 Overview of this thesis

In this thesis a universal and (constructively) logically fully abstract realizability model for call-by-name FPC is presented. This model is constructed within the category of modest sets over the total combinatory algebra of observational equivalence classes of closed $\lambda$+Error-terms. The language $\lambda$+Error is an extension of untyped lazy call-by-name lambda-calculus.

In Chapter 2 the typed combinatory algebra of observational equivalence classes of closed FPC-terms, denoted by $\mathcal{T}$, is defined. It is shown how to obtain a universal model for FPC in the category of modest sets over this combinatory algebra. We shall refer to this model as the *canonical realizability model* for FPC.

Chapter 3 is devoted to the study of universal types in the combinatory algebra $\mathcal{T}$. A necessary and sufficient condition for an FPC-type to be universal is established. Further, the untyped language $\lambda$+Error is defined, which is an extension of lazy untyped $\lambda$-calculus by a single constant ERR and a conditional construct distinguishing this constant from any other syntactic value. This language is a sub-language of FPC. Using this language $\lambda$+Error the type $U := \mu\alpha.\,\mathbf{void} + [\alpha \to \alpha]$ is shown to be universal in the combinatory algebra $\mathcal{T}$, i. e. any other type is a definable retract of $U$. In [LS02] it is shown that for any universal type in a typed combinatory algebra $\mathcal{T}$ there exists an untyped combinatory algebra $\mathcal{U}$ which is applicatively equivalent to $\mathcal{T}$. As a consequence, the realizability toposes over $\mathcal{T}$ and $\mathcal{U}$ are equivalent and hence the realizability model for FPC over $\mathcal{U}$ also universal and logically fully abstract.

In Chapter 4 it is shown that any closed FPC-term of type $U$ can be implemented in our extension of the lambda-calculus. Hence the untyped combinatory algebra $\mathcal{U}$ is applicatively equivalent to the untyped combinatory algebra $\mathcal{L}$ of all observational equivalence classes of closed $\lambda$+Error-terms. Hence the combinatory algebras $\mathcal{U}$ and $\mathcal{L}$ are applicatively equivalent. It follows that the realizability model over $\mathcal{L}$ is universal and logically fully abstract. As a consequence we prove a variant of the Longley-Phoa Conjecture.

As for prerequisites, the reader is expected to have some basic knowledge about category theory (see [Mac71]) and to be familiar with basic ideas of semantics of functional programming languages (see [Gun92]).

## 1.7 Notational conventions

Throughout this thesis, the symbol $\mathbb{N}$ stands for the set of natural numbers including 0. Combinatory algebras are denoted by letters in the calligraphic font, like $\mathcal{A}$, $\mathcal{T}$ etc. Variables for categories are written in the blackboard bold font, $\mathbb{C}$, $\mathbb{D}$ etc. Variables for FPC-types are lowercase Greek letters. Type contexts (i. e. lists of type variables) are denoted by $\Theta$, $\Theta'$ and contexts for FPC-terms by $\Gamma$, $\Gamma'$. The latter symbols are also used for contexts in the untyped language $\lambda$+Error. Terms of the typed language FPC are denoted by $s$, $t$, $t'$ etc. and terms of the untyped language $\lambda$+Error are denoted by $M$, $M'$, $N$ and so on. Names for FPC-constructs (**case**, **fold**) are written in bold type. Capital letters are used for names of $\lambda$+Error-constructs (ERR, IFR). The symbol $\equiv$ is used to indicate literal equality of terms or contexts and $=_{\text{obs}}$ or just $=$ to indicate observational equality.

As usual we use the arrow symbol $\to$ for set-theoretic functions and morphisms in categories. We use $\rightarrowtail$ for monomorphisms, $\twoheadrightarrow$ for epimorphisms and $\rightharpoonup$ for partial functions. Composition of morphisms is written $g \circ f$ which should be read as 'apply $f$ first and then $g$'. Names for special morphisms like fold or up are printed in sans serif. The symbol $\cong$ indicates isomorphy. In commutative diagrams I use the symbol $\lrcorner$ to indicate a pullback. Exponentials in categories are denoted by exponentiation ($Y^X$) or by $X \to Y$ or $[X \to Y]$. For an object $X$ in a category $\mathbb{C}$ we write $\mathbb{C}/X$ for the slice category of $\mathbb{C}$ over $X$. Recall that objects in this category are $\mathbb{C}$-morphisms with co-domain $X$ and a morphism $h$ between two such objects $g\colon W \to X$ and $g'\colon W' \to X$ is a $\mathbb{C}$-morphism $h\colon W \to W'$ such that $g = g' \circ h$ (cf. Diagram 1.1).

$$\begin{tikzcd}
W \arrow[rr, "h"] \arrow[dr, "g"'] & & W' \arrow[dl, "g'"] \\
& X &
\end{tikzcd}$$

Diagram 1.1: $h$ is a morphism from $g$ to $g'$ in $\mathbb{C}/X$

# 2 The canonical realizability model

This chapter is devoted to the construction of the typed combinatory algebra $\mathcal{T}$ of closed FPC-terms modulo observational equivalence and of the canonical realizability model for call-by-name FPC over this algebra.

In Section 2.1 we set up basic definitions and notation for realizability categories over typed combinatory algebras. Section 2.2 gives a precise definition for the sequential typed functional programming of call-by-name FPC. The total combinatory algebra $\mathcal{T}$ of observational equivalence classes of closed FPC-terms is presented in Section 2.3. In Section 2.4 we describe a subcategory $\mathbb{D}$ of domains in the category of modest sets over $\mathcal{T}$. Sections 2.5 and 2.7 are devoted to the construction of lifting, sum types and minimal invariants for FPC-definable functors within $\mathbb{D}$ which are necessary to obtain a model for FPC. In Section 2.8 this canonical realizability model is presented and shown to be universal.

## 2.1 Typed and untyped realizability

In this first section we set up definitions and notation for typed and untyped total combinatory algebras and briefly discuss some basic properties of categories of modest sets. For a detailed introduction into realizability semantics over untyped partial combinatory algebras see [Lon95]. In 1999, Longley first investigated partial combinatory type systems (cf. [Lon99b]). These very much coincide with the typed partial combinatory algebras defined in [LS02]. We restrict ourselves to total combinatory algebras as these suffice for our purposes.

**2.1 Definition (typed combinatory algebra)** A *typed combinatory algebra* is a non-empty set $\mathcal{T}$, the elements of which will be called *types*, together with

- binary operations $\times$ and $\rightarrow$ on $\mathcal{T}$,
- for every type $\sigma \in \mathcal{T}$, a set $|\sigma|$ of so-called *potential realizers of type $\sigma$*,
- for all types $\sigma$ and $\tau$, an application function $|\sigma{\rightarrow}\tau| \times |\sigma| \rightarrow |\tau|$, denoted by mere juxtaposition of its arguments, i. e. $f$ applied to $x$ is written $fx$,

- and, for all types $\varrho$, $\sigma$ and $\tau \in \mathcal{T}$, a collection of combinators

$$\begin{array}{rll} \mathrm{k}_{\sigma,\tau} \in |\sigma{\to}\tau{\to}\sigma| & \mathrm{s}_{\varrho,\sigma,\tau} \in \big|[\varrho{\to}\sigma{\to}\tau] \to [\varrho{\to}\sigma] \to \varrho \to \tau\big| & \\ \mathrm{pair}_{\sigma,\tau} \in |\sigma{\to}\tau{\to}\sigma{\times}\tau| & \mathrm{fst}_{\sigma,\tau} \in |\sigma{\times}\tau{\to}\sigma| & \mathrm{snd}_{\sigma,\tau} \in |\sigma{\times}\tau{\to}\tau|\,, \end{array}$$

  satisfying the following equations

$$\begin{array}{rr} \mathrm{k}ab = a & \mathrm{s}abc = ac(bc) \\ \mathrm{fst}(\mathrm{pair}\,ab) = a & \mathrm{snd}(\mathrm{pair}\,ab) = b. \end{array}$$

As usual, $\to$ associates to the right and $\times$ binds more strongly than $\to$. Application and $\times$ associate to the left. Type annotations usually are omitted. ◇

Note that any typed combinatory algebra is combinatorically complete, i. e. there is an algorithm for function abstraction allowing to define functions just like in the $\lambda$-calculus (cf. [LS02]). Let us list some examples of typed combinatory algebras. More examples can be found in [LS02].

**2.2 Examples** i) Every untyped total combinatory algebra can be viewed as a typed combinatory algebra with exactly one type [Lon99b].

ii) For any untyped partial combinatory algebra $\mathcal{A}$ we define the typed combinatory algebra $\mathcal{P}(\mathcal{A})$ as follows. The underlying set of types is the power set of $\mathcal{A}$ and, for any type $S$, we set $|S| := S$. For types $S$ and $T$ we define

$$\begin{array}{rcl} |S{\times}T| & := & \{a \in \mathcal{A} \mid \mathsf{p}_0 a \in |S| \text{ and } \mathsf{p}_1 a \in |T|\} \\ |S{\to}T| & := & \{a \in \mathcal{A} \mid \forall b \in |S|\,.\, ab \in |T|\}, \end{array}$$

where $\mathsf{p}$, $\mathsf{p}_0$ and $\mathsf{p}_1$ are appropriate combinators for pairing and projections. The typed application operations are defined as restrictions of the application operation in $\mathcal{A}$ to the respective subsets and, therefore, total [LS02].

Moreover, we can restrict the set of types to the set of finite types over some chosen ground type. For appropriate choices of $\mathcal{A}$ this gives rise to HRO (hereditary recursive operations) and ICF (intensional continuous functionals) as discussed in [Tro73].

iii) From a small cartesian closed category $\mathbb{C}$ we obtain a typed combinatory algebra as follows. Define the set of types to be the set of objects of $\mathbb{C}$ and for any type $T$ let $|T|$ be the set $\mathbb{C}(1, T)$ of global sections of $T$. For global sections $f \colon 1 \to T^S$ and $x \colon 1 \to S$ we define application as $fx := \mathsf{eval} \circ \langle f, x\rangle$, where $\mathsf{eval}$ stands for the evaluation morphism.

For instance, the cartesian closed category of algebraic lattices gives rise to a notion of realizability investigated in [BBS98]. Other interesting examples arise from the cartesian closed category $\mathsf{PER}(\mathcal{A})$ of partial equivalence relations on a partial combinatory algebra $\mathcal{A}$ [LS02].

Furthermore, one can restrict to subcategories of finite types over some chosen ground type thus obtaining in particular cases HEO (hereditary effective operations) and ECF (effective continuous functionals) [Tro73].

iv) Fix a set $\mathcal{B}$ of base types and define the set of types as the set freely generated from $\mathcal{B}$ by $\times$ and $\to$. For any type $T$ define $|T|$ to be the set of closed simply typed $\lambda$-terms of type $T$ modulo $\beta$- or $\beta\eta$-equality. Application is induced by the application of $\lambda$-calculus [LS02].

In [Lon99b] a 2-category of typed partial combinatory algebras and applicative morphisms is introduced, containing the 2-category of partial combinatory algebras familiar from [Lon95] as a full subcategory. In [LS02] the induced notion of equivalence for typed partial combinatory algebras is stated as a definition. The following definition is the special case of the latter for typed (total) combinatory algebras.

**2.3 Definition (Applicative equivalence)** Two typed combinatory algebras $\mathcal{T}$ and $\mathcal{S}$ are said to be *applicatively equivalent* whenever there are two functions

$$\begin{aligned} \mathcal{T} \to \mathcal{S}, &\quad \text{written } \sigma \mapsto \sigma' \\ \mathcal{S} \to \mathcal{T}, &\quad \text{written } \sigma \mapsto \sigma^* \end{aligned}$$

and there are two families of functions

$$\Big(u_\sigma \colon\ |\sigma| \to |\sigma'|\Big)_{\sigma\in\mathcal{T}} \qquad \text{and} \qquad \Big(v_\sigma \colon\ |\sigma| \to |\sigma^*|\Big)_{\sigma\in\mathcal{S}}$$

satisfying the following property: for every pair of types $\sigma$, $\tau \in \mathcal{T}$ there is an element $q_{\sigma,\tau}$ in $|(\sigma{\to}\tau)'{\to}\sigma'{\to}\tau'|$ such that, for all elements $a \in |\sigma{\to}\tau|$ and $b \in |\sigma|$,

$$q_{\sigma,\tau}\, u_{\sigma\to\tau}(a)\, u_\sigma(b) = u_\tau(ab).$$

Likewise, the same property holds for the family $(v_\sigma)_{\sigma\in\mathcal{S}}$.

Furthermore, for every $\sigma \in \mathcal{T}$ there are elements $i_\sigma \in |\sigma{\to}\sigma'^*|$ and $j_\sigma \in |\sigma'^*{\to}\sigma|$ satisfying for all $a \in |\sigma|$

$$i_\sigma\, a = v_{\sigma'} \circ u_\sigma(a) \quad \text{and} \quad a = j_\sigma\big(v_{\sigma'} \circ u_\sigma(a)\big)$$

and the same for $u_{\sigma'} \circ v_\sigma$ for any $\sigma \in \mathcal{S}$. $\diamond$

For any typed combinatory algebra one can define the corresponding categories of assemblies and modest sets. A typed combinatory algebra gives rise to a realizability topos iff it is applicatively equivalent to an untyped combinatory algebra (see [LS02]). Since we are not going to use any details about realizability toposes, the description of the construction of the realizability topos over an untyped partial combinatory algebra is omitted. The construction is described in detail in [Lon95].

**2.4 Definition (assemblies and modest sets)**
An *assembly* $X$ over a typed combinatory algebra $\mathcal{T}$ is a set $|X|$ together with a type $\sigma \in \mathcal{T}$ and a relation $\Vdash_X \subseteq |\sigma| \times |X|$ such that, for all $x \in |X|$, there is an element $a \in |\sigma|$ satisfying $a \Vdash_X x$ (in this case $a$ is called a *realizer* for $x$). We write $\mathrm{t}(X)$ to denote the type of an assembly $X$. Note that every element $x$ has at least one realizer $a$, but in general not every element $a$ of $|\sigma|$ is the realizer of some element $x \in |X|$. Further, the same realizer can code more than one element at the same time.

A *modest set* $X$ is an assembly where, for any $a \in |\mathrm{t}(X)|$, there is at most one $x \in |X|$ such that $a \Vdash_X x$, i. e. $\Vdash_X$ is the graph of a partial surjection $|\mathrm{t}(X)| \rightharpoonup\!\!\!\!\rightharpoonup |X|$. This simply means that any element of $\sigma$ can code at most one element of $X$.

Let $X$ be an assembly of type $\sigma \equiv \mathrm{t}(X)$. For any $x \in |X|$, we write $\|x\|_X := \big\{b \in |\sigma| \;\big|\; b \Vdash_X x\big\}$ for the set of realizers of $x$. Note that $\|x\|$ is necessarily a nonempty subset of $|\sigma|$.

Let $f$ be a map from the set $|X|$ to $|Y|$, where $Y$ is an assembly of type $\tau$. An element $a \in |\sigma{\to}\tau|$ is said to *track* (or *realize*) the function $f$ iff, for any element $x \in |X|$ and any realizer $b \in \|x\|_X$, application of $a$ to $b$ yields a realizer $ab$ of the image $f(x)$.

For any typed combinatory algebra $\mathcal{T}$, the assemblies and the modest sets form categories $\mathsf{Ass}(\mathcal{T})$ and $\mathsf{Mod}(\mathcal{T})$, respectively. In these categories, a morphism $f$ from $X$ to $Y$ is a map $f\colon |X| \to |Y|$, which is tracked by some element of $|\sigma{\to}\tau|$.

We define the assembly $[X \to Y]$ of morphisms from $X$ to $Y$ as follows: Its underlying set is the set of all morphisms from $X$ to $Y$ and its type is $\sigma{\to}\tau$. The set of realizers $\|f\|$ of a morphism $f\colon X \to Y$ is defined to be the set of all elements of $|\sigma{\to}\tau|$ tracking $f$. It is easily seen that the assembly $[X \to Y]$ is a modest set if $X$ and $Y$ are modest sets. $\diamond$

The categories of assemblies and modest sets carry a rich categorical structure:

**2.5 Definition (locally cartesian closed category)**
A category $\mathbb{C}$ is called a *locally cartesian closed category (lccc)* iff it has finite limits and, for every morphism $f\colon X \to Y$, the pullback functor between

the corresponding slice categories, written $f^\star\colon \mathbb{C}/Y \to \mathbb{C}/X$, has a right adjoint $\Pi_f$. The pullback functor $f^\star$ is defined according to Diagram 2.1, where the square on the left defines the object part of the functor and the larger right diagram shows how to define $f^\star(h)$ for a morphism $h\colon g \to g'$. $\diamond$

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d, "f^\star(g)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \bullet \arrow[d, "g"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
\bullet \arrow[rr, "h"] \arrow[dr, "g"'] & & \bullet \arrow[dl, "g'"] \\
& Y &
\end{tikzcd}
\quad \overset{f^\star}{\longmapsto} \quad
\begin{tikzcd}
\bullet \arrow[r] \arrow[dr, dotted, "f^\star h"] \arrow[ddr, bend right, "f^\star g"'] \arrow[dr, phantom, "\lrcorner", very near start] & \bullet \arrow[dr, "h"] \arrow[ddr, "g"', near start] & \\
& \bullet \arrow[r] \arrow[d, "f^\star g'"] \arrow[dr, phantom, "\lrcorner", very near start] & \bullet \arrow[d, "g'"] \\
& X \arrow[r, "f"'] & Y
\end{tikzcd}$$

Diagram 2.1: Definition of the pullback functor

Any lccc is a *cartesian closed category (ccc).* For simplicity of notation, we use the letter $X$ both for an object in $\mathbb{C}$ and for the unique morphism $X \to 1$ to the terminal object. Using this notation, exponentials can be derived via $Y^X = \Pi_X X^\star(Y)$, see Diagram 2.2.

$$\begin{tikzcd}
X \times Y \arrow[r, "\pi_2"] \arrow[d, "\pi_1 = X^\star Y"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "Y"] \\
X \arrow[r, "X"'] & 1
\end{tikzcd}$$

Diagram 2.2: Exponentials in a lccc

**2.6 Definition** A monomorphism is called *regular* iff it is an equalizer and an epimorphism is called *regular* iff it arises as a co-equalizer.

For any morphism $f\colon X \to Y$ its kernel pair $(f_1, f_2)$ is defined by the pullback in Diagram 2.3.

$$\begin{tikzcd}
\bullet \arrow[r, "f_2"] \arrow[d, "f_1"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "f"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

Diagram 2.3: $(f_1, f_2)$ is the kernel pair of $f$

A category is *regular* iff it has finite limits and co-equalizers of kernel pairs, and furthermore the pullback of a regular epimorphism along any morphism is a regular epimorphism. ◇

**2.7 Proposition** *The categories of assemblies and modest sets over a typed combinatory algebra are regular lcccs. The obvious inclusion functor from modest sets to assemblies preserves this structure.*

For a proof of regularity see [Lon95, Section 1.2.1]. Note that in regular categories, regular epimorphisms are closed under composition and every morphism can be factored, uniquely up to isomorphism, as a regular epimorphism followed by a mono.

**2.8 Definition** A modest set $X$ is said to be *projective* iff, for any arrow $f\colon X \to Y$ and any regular epimorphism $e\colon Z \twoheadrightarrow Y$, there is a (not necessarily unique) arrow $g\colon X \to Z$ with $e \circ g = f$ (cf. Diagram 2.4). ◇

$$\begin{tikzcd} & X \arrow[dl, dotted, "\exists g"'] \arrow[d, "f"] \\ Z \arrow[r, two heads, "e"'] & Y \end{tikzcd}$$

Diagram 2.4: $X$ is projective

**2.9 Proposition** *A modest set is projective iff it is isomorphic to a modest set $X$ where every element has exactly one realizer.*

For a proof see [Lon95, Prop. 2.4.10].

Note that every type $\sigma \in \mathcal{T}$ induces a projective modest set $\delta\sigma := (|\sigma|, \sigma, \Vdash)$ in $\mathsf{Mod}(\mathcal{T})$ in a canonical way where each element of $|\sigma|$ is realized by itself and by no other element.

**2.10 Proposition** *Let $\mathcal{A}$ be a typed combinatory algebra. Then any regular monomorphism in $\mathsf{Mod}(\mathcal{A})$ is isomorphic to an inclusion map preserving realizers, i. e. to a regular monomorphism $i\colon Y \rightarrowtail Z$ where $Y$ and $Z$ are of the same type, $|Y|$ is a subset of $|Z|$ and, for any $y \in |Y|$, the realizers for $y$ in $Y$ and in $Z$ coincide and $i$ is the associated inclusion map $y \mapsto y\colon |Y| \to |Z|$.*

The easy proof of this proposition is left to the reader.

## 2.2 A very general sequential functional programming language

The call-by-value functional programming language FPC (Fixed Point Calculus) was introduced by Gordon Plotkin in his “Stanford Lecture Notes” [Plo85]. In [Gun92] a call-by-name version of FPC is defined which is restricted to binary sum and product types. In this section we define a call-by-name version of FPC incorporating sum types over an arbitrary finite collection of types. This generalization is necessary as in a call-by-name language finite sums cannot be expressed as iterated binary sums. In the sequel, FPC stands for the language call-by-name FPC.

FPC is the call-by-name fragment of M. Marz's Sequential Functional Language SFL (see [Mar00]). We consider FPC a very general sequential language since all call-by-value constructions can be simulated in this language (see below for details). Hence all known non-polymorphic sequential functional programming languages can be simulated in FPC.

We assume a set of type variables (denoted by $\alpha$, $\alpha'$ and so on) and generate the types of FPC as follows:

$$\sigma ::= \alpha \mid \sigma{\to}\sigma \mid \sigma{\times}\sigma \;\Big|\; \sum_{i=0}^{m}\sigma \;\Big|\; \mu\alpha.\,\sigma \qquad \text{for any } m \in \mathbb{N}$$

A type is called *closed* iff it does not contain free type variables, i. e. if any occurring type variable $\alpha$ is bound under the scope of a recursive type constructor $\mu\alpha$. The terms of FPC are derived according to the following grammar:

$$\begin{aligned} t ::= {} & x \;\big|\; \lambda x{:}\sigma.\, t \;\big|\; tt \;\big|\; \langle t, t\rangle \;\big|\; \mathbf{pl}^{\sigma,\tau}(t) \;\big|\; \mathbf{pr}^{\sigma,\tau}(t) \;\big| \\ & \mathbf{in}_i^{\sigma_0,\dots,\sigma_m}(t) \;\big|\; \mathbf{case}^{\sigma_0,\dots,\sigma_m,\tau}\, t \;\mathbf{of}\; \mathbf{in}_0 x \Rightarrow t \,[\!]\, \dots \,[\!]\, \mathbf{in}_m x \Rightarrow t \;\big| \\ & \mathbf{fold}^{\mu\alpha.\,\sigma}(t) \;\big|\; \mathbf{unfold}^{\mu\alpha.\,\sigma}(t) \end{aligned}$$

for any variable $x$, any number $m \in \mathbb{N}$, any $i \in \{0, \dots, m\}$ and all types $\sigma$, $\sigma_0$, … , $\sigma_m$, $\tau$. Type annotations are merely used for type inference, they are often omitted when they are clear from the context. Terms not containing any free variables are called *closed terms.*

For the typing rules (as given in Table 2.1), we look at terms-in-contexts of the form $\Gamma \vdash M{:}\tau$ where $M$ is a term, $\tau$ is a closed type and $\Gamma \equiv x_1 : \sigma_1, \dots, x_n : \sigma_n$ is a *context* assigning closed types $\sigma_1$, … , $\sigma_n$ to a finite set of variables $x_1$, … , $x_n$.

As the language FPC incorporates recursive types, any inductive proof on the type structure has to deal with types containing free variables. We will use types-in-contexts to formulate such proofs.

$$\frac{}{\Gamma, x{:}\sigma \vdash x{:}\sigma} \qquad \frac{\Gamma, x{:}\sigma \vdash t{:}\tau}{\Gamma \vdash \lambda x{:}\sigma.\, t : \sigma{\to}\tau}$$

$$\frac{\Gamma \vdash s : \sigma{\to}\tau \qquad \Gamma \vdash t{:}\sigma}{\Gamma \vdash st{:}\tau} \qquad \frac{\Gamma \vdash s{:}\sigma \qquad \Gamma \vdash t{:}\tau}{\Gamma \vdash \langle s, t\rangle : \sigma{\times}\tau}$$

$$\frac{\Gamma \vdash t{:}\sigma{\times}\tau}{\Gamma \vdash \mathbf{pl}(t){:}\sigma} \qquad \frac{\Gamma \vdash t{:}\sigma{\times}\tau}{\Gamma \vdash \mathbf{pr}(t){:}\tau}$$

$$\frac{\Gamma \vdash t : \sigma\,[\mu\alpha.\,\sigma\,/\,\alpha]}{\Gamma \vdash \mathbf{fold}^{\mu\alpha.\,\sigma}(t) : \mu\alpha.\,\sigma} \qquad \frac{\Gamma \vdash t : \mu\alpha.\,\sigma}{\Gamma \vdash \mathbf{unfold}^{\mu\alpha.\,\sigma}(t) : \sigma\,[\mu\alpha.\,\sigma\,/\,\alpha]}$$

$$\frac{\Gamma \vdash t{:}\sigma_i}{\Gamma \vdash \mathbf{in}_i^{\sigma_0,\ldots,\sigma_m}(t){:} \sum_{j=0}^{m} \sigma_j}$$

$$\frac{\Gamma \vdash t{:} \sum_{j=0}^{m} \sigma_j}{\Gamma \vdash \mathbf{case}^{\sigma_0,\ldots,\sigma_m,\tau}\, t\ \mathbf{of}\ \mathbf{in}_0 x \Rightarrow t_0 \,[]\, \ldots \mathbf{in}_m x \Rightarrow t_m : \tau}$$

Table 2.1: The typing rules for FPC ($i$ ranging over $\{1, \ldots, m\}$)

$$\frac{}{\lambda x{:}\sigma.\, t \Downarrow \lambda x{:}\sigma.\, t} \qquad \frac{s \Downarrow \lambda x{:}\sigma.\, s' \qquad s'\,[t\,/\,x] \Downarrow v}{s't \Downarrow v}$$

$$\frac{}{\langle s, t\rangle \Downarrow \langle s, t\rangle} \qquad \frac{r \Downarrow \langle s, t\rangle \qquad s \Downarrow v}{\mathbf{pl}(r) \Downarrow v} \qquad \frac{r \Downarrow \langle s, t\rangle \qquad t \Downarrow v}{\mathbf{pr}(r) \Downarrow v}$$

$$\frac{}{\mathbf{in}_i(t) \Downarrow \mathbf{in}_i(t)} \qquad \frac{t \Downarrow \mathbf{in}_i s \qquad t_i\,[s\,/\,x] \Downarrow v}{\mathbf{case}\, t\ \mathbf{of}\ \mathbf{in}_0 x \Rightarrow t_0 \,[]\, \ldots \mathbf{in}_m x \Rightarrow t_m \Downarrow v}$$

$$\frac{t \Downarrow v}{\mathbf{fold}(t) \Downarrow \mathbf{fold}(v)} \qquad \frac{t \Downarrow \mathbf{fold}(v)}{\mathbf{unfold}\, t \Downarrow v}$$

Table 2.2: Big-step reduction for FPC ($i$ ranging over $\{1, \ldots, m\}$)

**2.11 Definition** A *type context* $\Theta$ is a finite list of type variables $\Theta \equiv \alpha_1, \dots, \alpha_n$. Let $\sigma$ be a not necessarily closed type. We use the *type judgement* $\Theta \vdash \sigma$ to indicate that all type variables occurring freely in $\sigma$ belong to $\Theta$. $\diamond$

The operational semantics is defined via big-step reduction in Table 2.2 and syntactic values of FPC are

$$v ::= \langle t, t\rangle \mid \mathbf{in}_i^{\sigma_0,\dots,\sigma_m}(t) \mid \lambda x{:}\sigma.\, t \mid \mathbf{fold}^{\mu\alpha.\,\sigma}(t).$$

To simplify notation we introduce the following abbreviations for the type **void** possessing no value, for lifted types and for the single valued type.

$$\mathbf{void} := \mu\alpha.\, \alpha \qquad \sigma_\perp := \sum_{i=0}^{0} \sigma \qquad \mathbf{unit} := \mathbf{void}_\perp$$

For any closed type $\sigma$ we define the following abbreviations for certain terms:

$$\begin{array}{rcl}
\mathbf{Y}_\sigma & :\equiv & \lambda f : \sigma\to\sigma.\, k\big(\mathbf{fold}^\tau(k)\big) \\
 & & \text{with } \tau :\equiv \mu\alpha.[\alpha\to\sigma] \\
 & & \text{and } k :\equiv \lambda x{:}\tau.\, f\big(\mathbf{unfold}^\tau(x)x\big) \\
\mathbf{id}_\sigma & :\equiv & \lambda x{:}\sigma.\, x \\
\Omega_\sigma & :\equiv & \mathbf{Y}_\sigma\, \mathbf{id}_\sigma \\
\mathbf{up}(t) & :\equiv & \mathbf{in}_0^\sigma(t) \\
\mathbf{down}(t) & :\equiv & \mathbf{case}^{\sigma,\sigma}\, t \,\mathbf{of}\, \mathbf{in}_0 x \Rightarrow x
\end{array}$$

In Section 2.8 we are going to define the interpretation of a context $\Gamma \equiv x_1{:}\sigma_1, \dots, x_n{:}\sigma_n$ to be the interpretation of the product type $\sigma_1\times \dots \times\sigma_n$. Hence we need some notation for arbitrary finite products. Hence we write $\sigma_1\times \dots \times\sigma_n$ for the left associative product type $\big(\dots(\sigma_1\times\sigma_2)\times \dots \times\sigma_n\big)$ and $\mathbf{pr}_i : \sigma_1\times \dots \times\sigma_n\to\sigma_i$ for the obvious closed term doing the projection onto the $i$th component. Further we set

$$\sigma^n :\equiv \underbrace{\sigma \times \cdots \times \sigma}_{n \text{ times}}.$$

**2.12 Definition (observational equivalence)** Let $\sigma$ be a closed FPC-type. Two FPC-programs $s$ and $t$ are called *observationally equivalent*, written $s =_{\text{obs}} t$ or simply $s = t$, iff, for any closed term $p : \sigma\to\mathbf{unit}$, the closed terms $ps$ and $pt$ either both reduce to a syntactic value or they both do not.

The notion of observational equivalence can be generalized to terms: Let $\Gamma$ be a non-empty context, $\Gamma \equiv x_1{:}\sigma_1, \dots, x_n{:}\sigma_n$, and assume $\Gamma \vdash s{:}\tau$ and $\Gamma \vdash t{:}\tau$

are terms-in-context. Then $s$ and $t$ are called observationally equivalent with respect to $\Gamma$, written $\Gamma \vdash s =_{\text{obs}} t$, iff the corresponding closed terms are equivalent, i. e.

$$\lambda x_1{:}\sigma_1.\ldots\lambda x_n{:}\sigma_n.\,s \quad =_{\text{obs}} \quad \lambda x_1{:}\sigma_1.\ldots\lambda x_n{:}\sigma_n.\,t.$$

For any closed FPC-term $t$ the *observational class* of $t$ is defined to be the class of all observationally equivalent closed terms. It is denoted by $[t]_{\text{obs}}$ or simply by $[t]$. If no confusion can arise, by abuse of notation, we often write the letter $t$ to denote the observational class of a closed FPC-term $t$. $\diamond$

**2.13 Examples** i) For any closed FPC-type $\sigma$ we have

$$\lambda x{:}\sigma.\,\mathbf{down}\big(\mathbf{up}(x)\big) =_{\text{obs}} \mathsf{id}_\sigma\,.$$

ii) It is easy to see that $\mathbf{Y}_\sigma$ is a fixed point combinator satisfying $\mathbf{Y}_\sigma\,t =_{\text{obs}} t(\mathbf{Y}_\sigma\,t)$ for any closed type $\sigma$ and any closed term $t : \sigma{\to}\sigma$. The proof that the interpretation of $\mathbf{Y}_\sigma$ in the $\mathsf{SD}$-model is a least fixed point operator is a standard exercise, which is left to the reader.

There are essentially two different known order-extensional domain models for call-by-name FPC. One is the well known model pioneered by Dana Scott and codified and studied in [SP82]. This model is computationally adequate, but not fully abstract. The second model is the fully abstract model defined by Marz [Mar00] within the category $\mathsf{SD}$ of Sequential Domains. Objects of $\mathsf{SD}$ are domains together with large families of Kripke logical relations and morphisms are continuous functions preserving these relations. Many domain theoretic constructions carry over from the Scott domains. In particular, directed suprema of morphisms are calculated pointwise. Marz showed that this category contains a fully abstract model for the sequential functional language SFL. SFL is a very general sequential language incorporating both call-by-name and call-by-value constructs. However, any computational SFL-type can be represented as a retract of an FPC-type and any SFL-construct can be simulated in FPC. We illustrate this for the coalesced sum type $\sigma_0 \oplus \sigma_1$ (cf. Diagram 2.5).

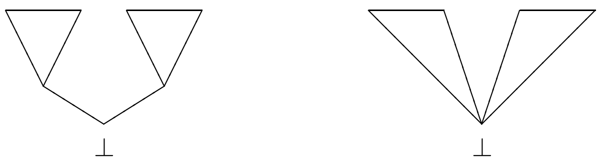

Diagram 2.5: Separated sum and coalesced sum of two types

Assume $\sigma_0$ and $\sigma_1$ are closed FPC-types and there are closed FPC-terms $t_i\colon \sigma_i{\to}\sigma_i$ such that $t_i\bot_{\sigma_i}$ diverges and $t_i x =_{\text{obs}} x$ for any terminating $x{:}\sigma_i$.

Such types are called computational types or SSB-types in SFL, SSB standing for sequentially separable bottom. Then there is an obvious retraction from the coalesced sum $\sigma \oplus \tau$ to the separated sum $\sigma + \tau$. Informally spoken, the embedding maps $\bot_{\sigma\oplus\tau}$ to $\bot_{\sigma+\tau}$ and any other element of $\sigma \oplus \tau$ to the corresponding element of $\sigma + \tau$. The projection maps each of the elements $\bot_{\sigma+\tau}$, $\mathbf{in}_0(\bot_\sigma)$ and $\mathbf{in}_1(\bot_\tau)$ to $\bot_{\sigma\oplus\tau}$ and any other element of $\sigma + \tau$ to the corresponding element of $\sigma \oplus \tau$. Obviously, projection after embedding gives the identity on $\sigma \oplus \tau$. The retraction, i. e. embedding after projection, is given by the following closed FPC-term:

$$\begin{aligned} &r \colon \sigma{+}\tau{\to}\sigma{+}\tau, \\ &r :\equiv \lambda x{:}\sigma{+}\tau.\, \mathbf{case}\, x\, \mathbf{of}\, \mathbf{in}_0 w \Rightarrow t_0\mathbf{in}_0(w) \,[]\, \mathbf{in}_1 w \Rightarrow t_1\mathbf{in}_1(w) \end{aligned}$$

In this sense, the computational SFL-type $\sigma{\oplus}\tau$ can be simulated in FPC by the retraction $r$. Since any computational SFL-type can be simulated this way, the SD-model is also fully abstract for call-by-name FPC. However, our results do not depend on full abstraction of the SD-model but only on adequacy. In the following we will thus deliberately use the fact that adequate domain models for FPC are well known to show observational equivalence of closed FPC-terms. We do this by proving that their interpretations in some adequate model coincide. However, we are convinced that in principle it should be possible — although quite tedious and not very illustrative — to formulate all proofs in this thesis in a purely syntactic way without any reference to a domain model.

The aim of this thesis is to present a universal realizability model for call-by-name FPC. In [MRS99] we announced a similar result for call-by-value FPC. Both languages possess a universal type $U$ such that all other types appear as retracts of $U$. However, for our construction it is essential that for any type $\sigma$, there is an *FPC-definable* embedding and projection between $U$ and $\sigma$. In Section 3.1 we will see that the projection can always be chosen to be a strict function, but the embedding in general cannot. However, in call-by-value FPC any definable function is strict. Hence not all relevant embeddings are definable in call-by-value FPC. To avoid this problem, we decided to concentrate on the call-by-name version of FPC defined above. Anyway, this is not a severe restriction since call-by-value FPC can be simulated in call-by-name FPC.

## 2.3 The algebra of closed FPC-terms

We are now going to define a combinatory algebra consisting of equivalence classes of closed FPC-terms. For every closed FPC-type $\sigma$, let $\sigma/_{=_{\mathrm{obs}}}$ denote the set of equivalence classes of observationally equivalent closed terms of type $\sigma$. Further, we write $[t]_{\mathrm{obs}}$ or just $[t]$ for the class of a closed term $t$.

**2.14 Proposition** *Equipping the set of FPC-types with the following structure yields a typed combinatory algebra $\mathcal{T}$:*

- *for every type $\sigma$, the set $|\sigma|$ of realizers of type $\sigma$ is defined to be the quotient set $\sigma/_{=_{obs}}$;*
- *product and function types are those of FPC;*
- *application is defined in the obvious manner: given terms $t{:}\sigma{\to}\tau$ and $s{:}\sigma$ we put $[t][s] := [ts]$;*
- *each of the combinators is defined to be the class of the evidently corresponding FPC-term.*

The straightforward proof of this proposition is left to the reader.

**2.15 Definition** From now on we write $\mathcal{T}$ for the particular typed combinatory algebra as defined above and call it the *typed combinatory algebra of closed FPC-terms.*

Furthermore, we write $\mathbb{M}$ to denote the category $\mathsf{Mod}(\mathcal{T})$ of modest sets over this algebra of closed terms.

By abuse of notation, we will often write $t \Vdash f$ for $[t]_{\text{obs}} \Vdash f$ and call $t$ a realizer of $f$. $\diamond$

The category $\mathbb{M}$ is the category within which we are going to construct a universal model for FPC. According to Proposition 2.7 it is locally cartesian closed.

## 2.4 A subcategory of domains

We are going to prove that the combinatory algebra $\mathcal{T}$ carries the structure of a cartesian closed category and can be embedded into $\mathbb{M}$ as a full sub-ccc $\mathbb{D}$. In order to interpret FPC in $\mathbb{M}$ we have to come up with a category of domains in $\mathbb{M}$ allowing the construction of sum types and recursive types. We will show in the subsequent sections that we can regard the subcategory $\mathbb{D}$ as such a category of domains within $\mathbb{M}$.

**2.16 Definition** The class of objects of $\mathbb{D}$ is defined to be the set of all closed FPC-types. For objects $\sigma$ and $\tau$, the set of morphisms $\mathbb{D}(\sigma,\tau)$ is simply $[\sigma{\to}\tau]/_{=_{\text{obs}}}$, the set of observational equivalence classes of closed terms of type $\sigma{\to}\tau$. For closed terms $t{:}\sigma{\to}\tau$ and $s{:}\varrho{\to}\sigma$, composition of the corresponding morphisms is defined as $[t]_{\text{obs}} \circ [s]_{\text{obs}} := \big[\lambda x{:}\varrho.\, t(sx)\big]_{\text{obs}}$. The identity morphism on $\sigma/_{=_{\text{obs}}}$ is $[\lambda x{:}\sigma.\, x]_{\text{obs}}$.

For any object $\sigma$ the morphism $[\lambda x{:}\sigma.\, \Omega_{\mathbf{void}}]_{\mathrm{obs}}$ is the only morphism going to the terminal object **void**.

The product of $\sigma$ and $\tau$ is the type $\sigma{\times}\tau$ with the projections

$$\big[\lambda x{:}\sigma{\times}\tau.\, \mathbf{pl}(x)\big]_{\mathrm{obs}} \quad \text{and} \quad \big[\lambda x{:}\sigma{\times}\tau.\, \mathbf{pr}(x)\big]_{\mathrm{obs}}.$$

The exponential $\tau^{\sigma}$ is the type $[\sigma{\to}\tau]$ with the evaluation morphism

$$\big[\lambda x{:}[\sigma{\to}\tau]{\times}\sigma. \big(\mathbf{pl}(x)\big)\big(\mathbf{pr}(x)\big)\big]_{\mathrm{obs}}.$$

It is easy to verify that $\mathbb{D}$ is a cartesian closed category. It embeds into $\mathbb{M}$ as a full sub-ccc in the following way:

- Every type $\sigma$ canonically induces a projective modest set $\delta\sigma$ of type $\sigma$. Its underlying set is the quotient set $|\sigma| = \sigma/_{=_{\mathrm{obs}}}$. Realizability is trivial, i. e. each class $[t] \in |\sigma|$ is realized by itself and by nothing else.
- For every class $[t]$ of a closed term $t : \sigma{\to}\tau$ there is a unique function $\delta t$ from $\sigma/_{=_{\mathrm{obs}}}$ to $\tau/_{=_{\mathrm{obs}}}$ which is tracked by $[t]$. It is given by $[s] \mapsto [ts]$. On the other hand, any $\mathbb{M}$-morphism $\delta\sigma \to \delta\tau$ is of this kind.

**2.17 Proposition** *The above embedding $\delta\colon \mathbb{D} \to \mathbb{M}$ respects the cartesian closed structure.*

*Proof:* We have to show that $\delta$ preserves terminal objects, binary products and exponentials.

*Terminals:* The set $|\delta\mathbf{void}|$ is a singleton, hence it is terminal in $\mathsf{Set}$. For any $X$ in $\mathbb{M}$ the unique function $|X| \to |\delta\mathbf{void}|$ is tracked by $\lambda x{:}\,\mathrm{t}(X).\,\Omega_{\mathbf{void}}$, where $t(X)$ stands for the type of the modest set $X$. Hence $\delta\mathbf{void}$ is terminal in $\mathbb{M}$.

*Products:* Let $\sigma$ and $\tau$ be types, $X$ an arbitrary $\mathbb{M}$-object of type $\varrho$ and $[p]$ and $[q]$ realizers of $\mathbb{M}$-morphisms $f$ and $g$, respectively, which make the outer square of the following diagram commute:

$$\begin{tikzcd}
X \arrow[r, "{[q]\Vdash g}"] \arrow[d, "{[p]\Vdash f}"'] \arrow[dr, dotted, "{[t]\Vdash m}" description] & \delta\tau \\
\delta\sigma & \delta(\sigma{\times}\tau) \arrow[l, "\lambda x.\,\mathbf{pl}(x)"] \arrow[u, "\lambda x.\,\mathbf{pr}(x)"']
\end{tikzcd}$$

Obviously the whole diagram commutes if we choose $t := \lambda x{:}\varrho.\, \langle px, qx\rangle$. We are left to show that the morphism $m$ realized by $[t]$ is the only one making this diagram commute.

Assume there is another such morphism, $m'$, realized by $[t']$. Then the interpretations of $t$ and $t'$ in some adequate domain model, like Marz's $\mathsf{SD}$-Model, make the corresponding diagram of domains commute. Hence these interpretations coincide and we conclude $t =_{\mathrm{obs}} t'$.

Exponentials: Suppose $\sigma$ and $\tau$ are types, $X$ an arbitrary $\mathbb{M}$-object of type $\varrho$ and $[t]$ a realizer of a morphism $f\colon X \times \delta(\sigma) \to \delta(\tau)$.

Let $s := \lambda x{:}[\sigma{\to}\tau]{\times}\sigma.\big(\mathbf{pl}(x)\big)\big(\mathbf{pr}(x)\big)$ and $[q] \Vdash e\colon \delta(\sigma{\to}\tau){\times}\delta(\sigma) \to \delta(\tau)$.

$$\begin{tikzcd}
\delta(\sigma\to\tau) & \delta(\sigma\to\tau) \times \delta(\sigma) \arrow[r, "{[q]\Vdash e}"] & \delta(\tau) \\
X \arrow[u, dotted, "{[r]_{\mathrm{obs}}\Vdash\widetilde{f}}"] & X \times \delta(\sigma) \arrow[u, dotted, "{[r']\Vdash\widetilde{f}\times\mathsf{id}}"] \arrow[ur, "{[t]\Vdash m}"'] &
\end{tikzcd}$$

We have to show that there is a unique morphism $\widetilde{f}$ such that the above diagram commutes. To prove its existence we set $r := \lambda x{:}\varrho.\, \lambda y{:}\sigma.\, t\,\langle x, y\rangle$. Then the observational equivalence class of

$$r' := \lambda y{:}\varrho{\times}\sigma.\, \big\langle r\big(\mathbf{pl}(y)\big), \mathbf{pr}(y)\big\rangle$$

realizes $\widetilde{f} \times \mathsf{id}$ and hence we obtain

$$\begin{aligned} qr' &=_{\mathrm{obs}} \lambda y{:}\varrho{\times}\sigma.\big(r(\mathbf{pl}(y))\big)(\mathbf{pr}(y)) \\ &=_{\mathrm{obs}} \lambda y{:}\varrho{\times}\sigma.\, t\,\langle \mathbf{pl}(y), \mathbf{pr}(y)\rangle =_{\mathrm{obs}} \lambda y{:}\varrho{\times}\sigma.\, ty =_{\mathrm{obs}} t. \end{aligned}$$

As for uniqueness, suppose there is another morphism $[s] \Vdash d\colon X \to \delta(\sigma{\to}\tau)$ making the above diagram commute. Then again the interpretations of $r$ and $s$ in some adequate domain model have to coincide since they both make the corresponding diagram in $\mathsf{SD}$ commute. Hence $r =_{\mathrm{obs}} s$. □

**2.18 Remark** *A typed combinatory algebra $\mathcal{A}$ is called* extensional *iff, for all arrow types $\sigma{\to}\tau$, two realizers $f$, $g \in |\sigma{\to}\tau|$ coincide iff $fx = gx$ for all $x \in |\sigma|$.*

*If $\mathcal{A}$ is extensional, one can define a full subcategory $\mathbb{C}$ of $\mathsf{Mod}(\mathcal{A})$ consisting of all modest sets $(|\sigma|, \sigma, \Vdash)$ with trivial realizability relations. Note that $\mathbb{C}$ always is a sub-ccc, which in general is not closed under isomorphisms.*

*For the algebra $\mathcal{T}$ of closed FPC-terms, this subcategory is equivalent to the subcategory $\mathbb{D}$ defined above.*

Maybe the definition for our category $\mathbb{D}$ seems a bit ad hoc. Of course one could use the machinery of synthetic domain theory to describe an internal category of domains (see, e. g. [RS99a, LS97, vOS00]). However, in the end this would probably lead to the same model for FPC. Furthermore, our $\mathbb{D}$ is easily seen to be a sub-category of the category of well-complete objects as defined in [LS97].

If no confusion can arise, from now on we will identify a closed FPC-term $t$ with the corresponding $\mathbb{D}$-morphism $[t]_{\mathrm{obs}}$. In particular in diagrams in $\mathbb{D}$ we will label arrows with closed FPC-terms instead of their observational classes.

## 2.5 Liftings

Our next aim is to present a realizability model for FPC in the category $\mathbb{D}$, which we will call the canonical realizability model for FPC. The interpretation of product and function types will turn out to be straightforward. It will be more difficult to define the interpretation of liftings, sum types and recursive types. Hence we will discuss them extensively in this section and the following.

Let $t$ be a closed FPC-term. Whenever no confusion can arise, we will use the letter $t$ from now on not only to denote this term, but also to denote the class $[t]_{\mathrm{obs}}$ of terms observationally equivalent to $t$. It will be clear from the context, whether $t$ refers to the term or the class of terms.

In order to interpret lifted types and sum types we have to define a lifting monad on $\mathbb{M}$. We do this by choosing a dominance $\Sigma$ in $\mathbb{M}$ which, in a natural way, gives rise to such a monad. The following definitions are taken from [Ros86, Chapter 2] and [Lon95, Section 4].

**2.19 Definition (Dominion)** A *dominion* in a category $\mathbb{C}$ is a class $\mathcal{D}$ of monomorphisms, which contains all identity morphisms and is closed under composition and has the property that, for any $\mathbb{C}$-morphisms $f$ and any monomorphism $m \in \mathcal{D}$, a pullback of $m$ along $f$ exists and all such pullbacks belong to $\mathcal{D}$. $\diamond$

**2.20 Definition (Partial Morphism)** Let $\mathcal{D}$ be a dominion in $\mathbb{C}$. A *partial morphism $X \rightharpoonup Y$ with domain of definition in the dominion $\mathcal{D}$* (also called a *$\mathcal{D}$-partial morphism*) is given by a pair $(m, f)$ consisting of a monomorphism $m\colon X' \rightarrowtail X$ in $\mathcal{D}$ and a $\mathbb{C}$-morphism $f\colon X' \to Y$. We regard $(m\colon X' \rightarrowtail X, f)$ and $(\widetilde{m}\colon \widetilde{X}' \rightarrowtail X, \widetilde{f})$ as giving the same partial morphism iff there is an isomorphism $i\colon X' \cong \widetilde{X}'$ such that the left square of Diagram 2.6 commutes.

$$\begin{tikzcd}
X' \arrow[r, "f"] \arrow[d, tail, "m"'] \arrow[dr, "i", "\cong"'] & Y \\
X & \widetilde{X}' \arrow[l, tail, "\widetilde{m}"] \arrow[u, "\widetilde{f}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
X'' \arrow[r, "\varphi'"] \arrow[d, tail, "m'"] \arrow[rr, dotted, bend left, harpoon, "\vartheta"] \arrow[dd, dotted, bend right=60, tail, harpoon, "k"'] \arrow[dr, phantom, "\lrcorner", very near start] & Y' \arrow[r, "\psi"] \arrow[d, tail, "n"] & Z \\
X' \arrow[r, "\varphi"'] \arrow[d, tail, "m"] & Y & \\
X & &
\end{tikzcd}$$

Diagram 2.6: Composition of partial morphisms

For any two partial morphisms given by $X \overset{m}{\leftarrowtail} X' \xrightarrow{\varphi} Y$ and $Y \overset{n}{\leftarrowtail} Y' \xrightarrow{\psi} Z$, the *composition* $(k, \vartheta) = (m, \varphi) \circ (n, \psi)$ is defined by the pullback given in the right diagram in Diagram 2.6.

For any class $\mathcal{D}$ of monomorphisms in $\mathbb{C}$, we write $\mathcal{D}(X, Y)$ for the set of partial morphisms $X \overset{m}{\leftarrowtail} X' \xrightarrow{f} Y$ with $m \in \mathcal{D}$.

The *category of $\mathcal{D}$-partial morphisms corresponding to* $\mathbb{C}$, denoted by $\mathsf{Part}_{\mathcal{D}}$, consists of all objects of $\mathbb{C}$ together with all $\mathcal{D}$-partial morphisms. ◇

The category $\mathbb{C}$ embeds into $\mathsf{Part}_{\mathcal{D}}$ via the functor $J\colon \mathbb{C} \to \mathsf{Part}_{\mathcal{D}}$ with $JX = X$ and $Jf = (\mathsf{id}, f)$ for $f\colon X \to Y$. This functor is faithful and reflects isomorphisms.

**2.21 Definition (Pre-dominance)**
For any object $\Sigma$, a monomorphism $\top\colon 1 \rightarrowtail \Sigma$ is called a *pre-dominance* iff, for any two morphisms $f$, $g\colon X \to \Sigma$,

$$f^\star(\top) \cong g^\star(\top) \quad \text{implies} \quad f = g.$$

Hence $f$ and $g$ have to coincide, if the pullback functors $f^\star$ and $g^\star$ map $\top$ to isomorphic objects. ◇

**2.22 Definition** A class $\mathcal{D}$ of monomorphisms is called *representable* iff, for any object $Y$, there is a morphism $\mathsf{up}_Y\colon Y \to Y_\perp$ such that, for every $X$, we have a bijective correspondence $\mathcal{D}(X, Y) \cong \mathbb{C}(X, Y_\perp)$ where $X \overset{m}{\leftarrowtail} X' \xrightarrow{f} Y$ corresponds to $g$ iff Diagram 2.7 is a pullback.

$$\begin{tikzcd}
X' \arrow[r, "f"] \arrow[d, tail, "m"'] \arrow[dr, phantom, "\lrcorner", very near start] & Y \arrow[d, "\mathsf{up}_Y"] \\
X \arrow[r, "g"'] & Y_\perp
\end{tikzcd}$$

Diagram 2.7: $g$ classifies $(m, f)$

In this case we call $g$ the *classifier* of $(m, f)$. We will abbreviate $\mathsf{up}_Y$ to $\mathsf{up}$ when no confusion can arise. ◇

In particular, a dominion is representable iff there is a bijective correspondence between $\mathsf{Part}_{\mathcal{D}}(X, Y)$ and $\mathbb{C}(X, Y_\perp)$.

**2.23 Proposition** *For any pre-dominance $\top\colon 1 \to \Sigma$ in a locally cartesian closed category $\mathbb{C}$, the class $\mathcal{D}$ of all pullbacks of $\top$ is representable. Further, the class $\mathcal{D}$ has the following properties:*

- *it contains all isomorphisms in* $\mathbb{C}$*;*
- *it contains* $\mathsf{up}_Y$ *for any* $Y$*;*
- *for any* $Y$*, the object* $Y_\perp$ *and morphism* $\mathsf{up}_Y$ *are uniquely determined up to isomorphism;*
- $\mathcal{D}$ *is closed under pullbacks;*
- *the mapping* $Y \mapsto Y_\perp$ *extends to a functor* $(\_)_\perp \colon \mathbb{C} \to \mathbb{C}$*, called the* lift functor associated with $\mathcal{D}$*, and the morphisms* $\mathsf{up}_Y$ *form a natural transformation.*

A proof for the representability of $\mathcal{D}$ can be found in [Lon95, Proposition 4.1.7]. The rest of the above proposition is straightforward.

**2.24 Definition (Dominance)** A pre-dominance $\top$ is said to be a *dominance* iff the associated class $\mathcal{D}$ of pullbacks of $\top$ is a dominion. In this case, $\top$ is called a *classifier* for $\mathcal{D}$. ◇

**2.25 Proposition** *For a pre-dominance* $\top$ *and its associated class* $\mathcal{D}$ *of pullbacks the following are equivalent:*

- $\top$ *is a dominance;*
- $\mathcal{D}$ *is closed under composition;*
- *the composite morphism* $\mathsf{up} \circ \top \colon 1 \to \Sigma_\perp$ *is a pullback of* $\top$*.*

*If* $\top$ *is a dominance then the associated lift functor can be equipped with the structure of a monad. Specifically, there is a right adjoint* $L \colon \mathsf{Part}_{\mathcal{D}} \to \mathbb{C}$ *of the inclusion functor* $J$ *such that the lift monad is* $L \circ J$*.*

For a proof see [Lon95, Section 4.1].

**2.26 Example** Let $K_1$ denote the first Kleene algebra, i. e. the set of natural numbers together with the application $m \bullet n := \{m\}n$, the value of the $m$th partial recursive function at the argument $n$. Let $\mathsf{Mod}(K_1)$ be the category of $\omega$-sets and $N$ the canonical natural numbers object. Define $\Sigma$ by

$$|\Sigma| = \{\top, \perp\}, \quad \|\top\| = \{n \mid \{n\}(0)\downarrow\}, \quad \|\perp\| = \{n \mid \{n\}(0)\uparrow\}.$$

Then the morphism $\top \colon 1 \to \Sigma$ which picks out the element $\top$ is a dominance. The monomorphisms into $N$ belonging to the associated dominion $\mathcal{D}$ correspond precisely to the recursively enumerable subsets of $\mathbb{N}$. Further, the partial morphisms $N \rightharpoonup N$ defined within $\mathcal{D}$ are just the partial recursive functions. For a proof that $\Sigma$ indeed is a dominance see [Lon95, Example 4.2.9(ii)].

The following proposition establishes a sufficient criterion for a morphism in the category $\mathbb{M} = \mathsf{Mod}(\mathcal{T})$ of modest sets over the typed combinatory algebra of FPC-terms to be a pre-dominance. Moreover, it gives a concrete description of the associated lifting functor. It is a variant of Proposition 4.2.4 and Remark 4.2.5 in [Lon95] for modest sets over the typed combinatory algebra $\mathcal{T}$.

**2.27 Proposition (Lifting in $\mathsf{Mod}(\mathcal{T})$)** *A monomorphism $\top\colon 1 \to \Sigma$ in the category $\mathbb{M} = \mathsf{Mod}(\mathcal{T})$ is a pre-dominance if $\Sigma$ has exactly two elements. In this case, for any modest set $X$ of type $\sigma := \mathrm{t}(X)$, the object $X_\perp$ and the morphism $\mathsf{up}$ are given as follows (up to isomorphism):*

$$|X_\perp| = |X| \uplus \{\perp_X\}, \quad \mathrm{t}(X_\perp) = \sigma_\perp, \quad \|x \in X\| = \big\{\mathbf{up}(a) \bigm| a \Vdash_X x\big\},$$
$$\|\perp_X\| = \big\{\Omega_{\sigma_\perp}\big\}, \quad \mathsf{up}(x) = x \text{ for } x \in |X|\,, \quad \lambda x{:}\sigma.\,\mathbf{up}(x) \Vdash \mathsf{up}.$$

*Furthermore, the lift functor maps any morphism $f\colon X \to Y$ of type $\sigma{\to}\tau$ to*

$$f \mapsto (f_\perp\colon X_\perp \to Y_\perp) \qquad f_\perp(x) = \begin{cases} f(x) & \text{if } x \in |X| \\ \perp_Y & \text{if } x = \perp_X \end{cases}$$

*If $f$ is tracked by $t$, then the lifted morphism $f_\perp$ is tracked by*

$$\lambda x{:}\sigma_\perp.\,\mathbf{case}\,x\,\mathbf{of}\;\mathbf{in}_0 w \Rightarrow \mathbf{up}(tw).$$

*Proof:* Suppose $\Sigma$ has exactly two elements $b$ and $t$ and let $\top\colon 1 \to \Sigma$ be a monomorphism in $\mathbb{M}$ such that the image of $\top$ is $t$. Then, for any $\mathbb{M}$-morphism $f\colon X \to \Sigma$, the image of $\top$ under the pullback functor $f^\star$ is the inclusion map $f^{-1}\{t\} \hookrightarrow X$ (cf. Diagram 2.8). Given $f$ and $g\colon X \to \Sigma$ such

$$\begin{tikzcd}
f^{-1}\{t\} \arrow[r, "!"] \arrow[d, hook, "f^\star(\top)"'] & 1 \arrow[d, "\top"] \\
X \arrow[r, "f"'] & \Sigma
\end{tikzcd}$$

Diagram 2.8: Pullback of $\top$ along $f$

that $f^\star(\top) \cong g^\star(\top)$, we conclude $f^{-1}\{t\} = g^{-1}\{t\}$ and since $\Sigma$ has exactly two elements this yields $f = g$. Hence $\Sigma$ is a pre-dominance.

The morphism $\mathsf{up}$ is tracked by $\lambda x{:}\sigma.\,\mathbf{up}(x)$. It remains to be shown that every partial morphism $X \overset{m}{\hookleftarrow} X' \xrightarrow{f} Y$ with $m \in \mathcal{D}$ has a unique classifier $g\colon X \to Y_\perp$. As it is a pullback of the regular monomorphism $\top$, the monomorphism $m$ is again regular. According to Proposition 2.10, $m$ is isomorphic to an inclusion which preserves realizers. Thus we can assume, without loss of generality, $|X'| \subseteq |X|$ with $a \Vdash_{X'} x \iff a \Vdash_X x$ and $m(x) = x$. Let $\sigma := \mathrm{t}(X)$,

$$\begin{tikzcd}
X' \arrow[r, "!"] \arrow[d, hook, "m"'] & 1 \arrow[d, tail, "\top"] \\
X \arrow[r, "\overline{m}"'] & \Sigma.
\end{tikzcd}$$

Diagram 2.9: Choice of $\overline{m}$

$\tau := \mathrm{t}(Y)$ and let $\overline{m}$ be the morphism making Diagram 2.9 a pullback. Let $t$ and $s$ be realizers of $\overline{m}$ and $f$, respectively. We show that the morphism $g\colon X \to Y_\perp$ tracked by $\lambda b{:}\sigma.\,\mathbf{case}\,tb\,\mathbf{of}\ \mathbf{in}_0 w \Rightarrow \mathbf{up}(sb)$ classifies $f$. Let $m$ and $f'$ be morphisms such that the outer square of Diagram 2.10 commutes. Then the morphism $h\colon X'' \to X'$ realized by $t'$ obviously is the unique mediating morphism and therefore Diagram 2.10 is a pullback.

$$\begin{tikzcd}
X'' \arrow[drr, bend left, "f'"] \arrow[dr, dotted, "h"] \arrow[ddr, bend right, "{[t']_{\mathrm{obs}} \Vdash m'}"'] & & \\
& X' \arrow[r, "f"] \arrow[d, hook, "m"'] & X \arrow[d, tail, "\mathsf{up}"] \\
& X \arrow[r, "g"'] & X_\perp,
\end{tikzcd}$$

Diagram 2.10: Construction of a classifier $g$ for $(m, f)$

We are left to show that $g$ is unique. The above pullback in $\mathbb{M}$ induces a pullback in $\mathsf{Set}$ and $g$ is the only function making that diagram a pullback. Thus $g$ also has to be unique in $\mathbb{M}$. $\Box$

**2.28 Definition** Let $\Sigma$ denote the modest set in $\mathbb{D}$ given by

$$|\Sigma| = \{\perp, \top\}, \quad \mathrm{t}(\Sigma) = \mathbf{void}_\perp, \quad \mathbf{up}(\Omega_{\mathbf{void}}) \Vdash \top, \quad \Omega_{\mathbf{void}_\perp} \Vdash \perp$$

and let, by abuse of notation, $\top\colon 1 \to \Sigma$ also denote the morphism mapping the only element of $1$ to the element $\top \in |\Sigma|$. This morphism is tracked by $\lambda x{:}\mathbf{void}.\,\mathbf{up}(x)$. $\diamond$

**2.29 Proposition** *The morphism $\top\colon 1 \to \Sigma$ defined in Definition 2.28 is a dominance in $\mathbb{M}$.*

*Proof:* Due to Proposition 2.25 it suffices to show that $\mathsf{up} \circ \top\colon 1 \to \Sigma_\perp$ is a pullback of $\top$. This is a fact as Proposition 2.27 shows that this morphism is the pullback of $\top$ along the morphism $g$ realized by $\lambda x{:}(\mathbf{void}_\perp)_\perp.\,\mathbf{down}(x)$. $\Box$

$$\begin{tikzcd}
1 \arrow[r, equal] \arrow[d, tail, "\top"'] \arrow[dr, phantom, "\lrcorner", very near start] & 1 \arrow[dd, tail, "\top"] \\
\Sigma \arrow[d, "\mathsf{up}"'] & \\
\Sigma_\perp \arrow[r, "g"'] & \Sigma
\end{tikzcd}$$

Diagram 2.11: $\top$ is a dominance

## 2.6 Sum types

Based on lifting we define the notion of separated sum in $\mathbb{M}$ as the lifted co-product of objects and show that it has the expected universal property.

**2.30 Notation** Let $X_0, \dots, X_m$ be a finite collection of objects of $\mathbb{M}$ with types $\sigma_i := t(X_i)$. We write $\coprod_{i=0}^m X_i$ to denote the co-product of these objects and we use $\mathsf{copr}_j \colon X_j \to \coprod_{i=0}^m X_i$ to denote the $j$th co-projection, $j \in \{0, \dots, m\}$. Up to isomorphism, the underlying set of $\coprod_{i=0}^m X_i$ is the disjoint union $\bigcup_{i=0}^m \{i\} \times |X_i|$. The type of the co-product is $\sum_{i=0}^m \sigma_i$ and the set of realizers of an element $(i, x) \in \left|\coprod_{i=0}^m X_i\right|$ is $\left\{\mathbf{in}_i(t) \;\middle|\; t \Vdash_{X_i} x\right\}$. The $j$th co-projection, mapping $x \in X_j$ to $\mathsf{copr}_j(x) = (j, x)$ is realized by $\lambda x{:}\sigma_j.\, \mathbf{in}_j(x)$. It is easy to verify the universal property. $\diamond$

Diagram 2.12: Co-product and separated sum of two objects

**2.31 Definition** If $X_0, \dots, X_m$ are objects of types $\sigma_0, \dots, \sigma_m$ in $\mathbb{M}$, the *separated sum* $\sum_{i=0}^m X_i$ of these objects is defined to be the object with underlying set $\{\perp_{\sum_{i=0}^m X_i}\} \uplus \bigcup_{i=0}^m \{i\} \times X_i$ and type $t\left(\sum_{i=0}^m X_i\right) = \sum_{i=0}^m \sigma_i$ where realizability is defined as

$$\left\|\perp_{\sum_{i=0}^m X_i}\right\| := \Omega_{\sum_{i=0}^m \sigma_i} \quad \text{and} \quad \|(j, x)\| := \lambda x{:}\sigma_j.\, \mathbf{in}_j(x).$$

The inclusion map $\mathsf{in}_j \colon X_j \to \sum_{i=0}^m X_i$, $x \mapsto (j, x)$ is realized by $\lambda x{:}\sigma_j.\, \mathbf{in}_j(x)$.

For any object $Y$ of type $\tau$ in $\mathbb{M}$, we define a function $\mathsf{case}$:

$$\mathsf{case}\colon \left(\sum_{i=0}^{m} X_i\right) \times [X_0 \to Y] \times \ldots \times [X_m \to Y] \longrightarrow Y$$

$$\mathsf{case}(x, f_0, \ldots, f_m) := \begin{cases} f_i(w) & \text{iff } x = \mathsf{in}_i(w) \text{ for some } i \in \{0, \ldots, m\} \\ & \qquad\qquad \text{and } w \in X_i \\ \perp_Y & \text{otherwise,} \end{cases}$$

realized by the observational equivalence class of

$$\lambda p : \left(\sum_{i=0}^{m} \sigma_i\right) \times [\sigma_0 \to \tau] \times \ldots \times [\sigma_m \to \tau].$$
$$\mathbf{case}\,\mathbf{pr}_0(p)\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow \big(\mathbf{pr}_1(p)\big) w \ [] \ldots [] \ \mathbf{in}_m w \Rightarrow \mathbf{pr}_{m+1}\big(\mathbf{pr}(p)\big) w.$$

◇

To formulate the usual universal property of separated sums, we first have to define the notion of strictness of morphisms. It is impossible to do this for arbitrary $\mathbb{M}$-morphisms. However, there is a canonical notion of strictness of morphisms in the subcategory $\mathbb{D}$ of domains in $\mathbb{M}$. Recall that objects of $\mathbb{D}$ are just the closed FPC-types and the morphisms are observational classes of closed terms.

**2.32 Definition (strict morphism)** For any closed FPC-type $\sigma$ we define the closed term $\mathbf{down}_\sigma :\equiv \lambda x{:}\sigma_\perp.\,\mathbf{down}(x)$.

Let $\sigma$ and $\tau$ be closed FPC-types. A $\mathbb{D}$-morphism $\sigma \xrightarrow{t} \tau$ , i. e. a closed term $t : \sigma \to \tau$, is called *strict* iff Diagram 2.13 commutes. We write $\sigma \overset{t}{\circ\!\!\longrightarrow} \tau$ to

$$\begin{tikzcd}
\sigma_\perp \arrow[r, "t_\perp"] \arrow[d, "\mathbf{down}_\sigma"'] & \tau_\perp \arrow[d, "\mathbf{down}_\tau"] \\
\sigma \arrow[r, "t"'] & \tau
\end{tikzcd}$$

Diagram 2.13: $t$ is strict

indicate that $t$ is strict. ◇

**2.33 Proposition** *Let $t : \sigma \to \tau$ be a closed FPC-term. Then $t$ is strict iff it satisfies $t\,\Omega_\sigma =_{obs} \Omega_\tau$.*

*In $\mathbb{D}$ the separated sum $\sum_{i=0}^{m} X_i$ has the following universal property:*

*For any object $Z$ and any collection of strict morphisms $f_i\colon X_i \circ\!\!\longrightarrow Z$ with $i \in \{0,\ldots,m\}$ there is a unique strict morphism $h\colon \sum_{i=0}^{m} X_i \circ\!\!\longrightarrow Z$ such that $h \circ \mathsf{in}_j = f_j$ for any $j \in \{0,\ldots,m\}$. (See Diagram 2.14 for the binary case.)*

$$\begin{tikzcd}
 & Z & \\
 & \sum_{i=0}^{1} X_i \arrow[u, dotted, "\exists_1 h", "\text{strict}"'] & \\
X_0 \arrow[uur, bend left, "f_0"] \arrow[ur, "\mathsf{in}_0"'] & & X_1 \arrow[uul, bend right, "f_1"'] \arrow[ul, "\mathsf{in}_1"]
\end{tikzcd}$$

Diagram 2.14: The universal property of sums

*In particular, the separated sum $\sum_{i=0}^{m} X_i$ thus is isomorphic to the lifted coproduct $(\coprod_{i=0}^{m} X_i)_\perp$ via an isomorphism which takes the inclusion map $\mathsf{in}_j$ to $\mathsf{up} \circ \mathsf{copr}_j$.*

*Proof:* The first claim holds since $t\,\Omega_\sigma =_{\mathrm{obs}} \Omega_\tau$ is equivalent to

$$t(\lambda x{:}\sigma_\perp.\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow w) = \lambda x{:}\sigma_\perp.\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow tw$$

and hence

$$\begin{aligned}
t \circ \mathbf{down}_\sigma &= \lambda x{:}\sigma_\perp.\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow tw \\
&= \lambda x{:}\sigma_\perp.\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow \mathbf{down}\big(\mathbf{up}(tw)\big) \\
&= \lambda x{:}\sigma_\perp.\,\mathbf{down}\big(\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow \mathbf{up}(tw)\big) \\
&= \mathbf{down}_\tau \circ t_\perp.
\end{aligned}$$

As the morphisms $f_j$ are assumed to be strict, in $\mathsf{Set}$ there is a unique strict function $h\colon \big|\sum_{i=0}^{m} X_i\big| \to |Z|$ such that $h \circ \mathsf{in}_j = f_j$. If $t_j \Vdash f_j$ for any $j$, then $h$ is realized by $\lambda x{:}\sum_{i=0}^{m}\sigma_i.\,\mathbf{case}\,x\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow t_0 w \,[]\, \ldots \,[]\, \mathbf{in}_m w \Rightarrow t_m w$. □

## 2.7 Minimal invariants of functors

As the language FPC includes recursive types and the interpretation of types in $\mathbb{M}$ has to be defined inductively on the type structure, we cannot restrict ourselves to closed types, but we have to give an interpretation of types containing free type variables.

There is a natural way to interpret a type-in-context $\alpha_1, \dots, \alpha_n \vdash \sigma$ as a functor $(\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$, where $\mathbb{D}$ is the subcategory of domains in $\mathbb{M}$. For example, the type $\alpha \vdash \alpha{\to}\alpha$ corresponds to the Hom-functor, which is contravariant in the first and covariant in the second argument:

$$\begin{aligned} F(X,Y) &:= [X \to Y] \text{ for objects } X, Y \in \mathbb{D} \\ F(f,g) &:= [f \to g] \text{ for morphisms } f\colon X' \to X,\ g\colon Y \to Y', \\ &\phantom{:=} \text{ where } [f \to g]\colon [X{\to}Y] \to [X'{\to}Y'],\ [f \to g](h) = g \circ h \circ f \end{aligned}$$

It is straightforward to define such functors for finite (i. e. non-recursive) types. Things are not that easy for recursive types. In the categories of Scott domains or Sequential Domains recursive types are interpreted as solutions of recursive domain equations, which can be constructed as a bilimit (see [SP82, Mar00]). This construction can be carried over to a subcategory of domains of the category $\mathbb{M} = \mathsf{Mod}(\mathcal{T})$ using methods of synthetic domain theory (cf. [RS99a, Sim92]). In the special case of our syntactically defined subcategory $\mathbb{D}$ there is a simpler way to show that so-called syntactic functors have unique minimal invariants which we will use to interpret recursive types. However, we are convinced that synthetic domain theory leads to essentially the same model for FPC. Our approach is related to Freyd's results on initial algebras and final co-algebras, see [Fre91] and [Fre92].

Following the notational convention introduced in [Pit96] we will decorate variable names with superscripts $+$ and $-$ to distinguish between 'co- and contravariant arguments'. If $F\colon \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$ is a functor, we will use variable names $x^-$, $y^-$, ... to denote its first, 'contravariant', argument and $x^+$, $y^+$ and so on for its second, 'covariant', argument. Further, we will abbreviate pairs like $(\tau^-, \tau^+)$ to $\tau^\mp$ and $(\tau^+, \tau^-)$ to $\tau^\pm$ and type variable substitutions like $\sigma\,[\tau^- / \alpha^-, \tau^+ / \alpha^+]$ to $\sigma\,[\tau^\mp / \alpha^\mp]$. Hence the superscript $\mp$ is used to indicate 'first minus, then plus' and $\pm$ stands for 'first plus, then minus'.

**2.34 Definition (minimal invariant)** Let $F\colon \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$ be a functor. An object $\iota \in \mathbb{D}$ together with an isomorphism $\mathsf{fold}\colon F(\iota,\iota) \cong \iota$ is called a *minimal invariant* of $F$ in $\mathbb{D}$ if it has the following universal property (with $\mathsf{unfold} := \mathsf{fold}^{-1}$):

> For any two objects $\tau^-$ and $\tau^+$ and any morphisms $f^-\colon \tau^- \to F(\tau^+, \tau^-)$ and $f^+\colon F(\tau^-, \tau^+) \to \tau^+$ there is a unique pair of morphisms $k^-\colon \tau^- \to \iota$ and $k^+\colon \iota \to \tau^+$ making Diagram 2.15 commute.

Minimal invariants of functors are easily seen to be unique up to isomorphism.

**2.35 Remark** *If $(\iota, \mathsf{fold})$ is a minimal invariant of $F\colon \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$, then the identity on $\iota$ is the only morphism $f\colon \iota \to \iota$ making the Diagram 2.16 commute:*

$$\begin{tikzcd}
F(\iota,\iota) & \iota \arrow[l, "\mathsf{unfold}"'] \\
F(\tau^+,\tau^-) \arrow[u, "F(k^+,k^-)"] & \tau^- \arrow[l, "f^-"] \arrow[u, "k^+"']
\end{tikzcd}
\qquad
\begin{tikzcd}
F(\iota,\iota) \arrow[r, "\mathsf{fold}"] \arrow[d, "F(k^-,k^+)"'] & \iota \arrow[d, "k^-"] \\
F(\tau^-,\tau^+) \arrow[r, "f^+"'] & \tau^+
\end{tikzcd}$$

Diagram 2.15: Universal property of the minimal invariant

$$\begin{tikzcd}
F(\iota,\iota) \arrow[d, "F(f,f)"'] & \iota \arrow[l, "\mathsf{fold}"'] \arrow[d, "f"] \\
F(\iota,\iota) & \iota \arrow[l, "\mathsf{fold}"']
\end{tikzcd}$$

Diagram 2.16: $f = \mathsf{id}$ is the only morphism making this diagram commute

The above definition of minimal invariants suffices to handle recursive types which are not nested. In order to handle nested recursive types, we have to generalize this notion. Given a syntactic functor, we have to form a minimal invariant in, say, the $i$th argument pair $\tau_i^-$, $\tau_i^+$ of $F$, regarding the other arguments $\tau_j^-$, $\tau_j^+$ $(j \neq i)$ of $F$ as parameters. For simplicity of notation, we gather these parameters of $F$ in an 'argument vector' denoted by $\vec{\tau}$. Because of the mixed variant nature of $F$, we not only have to consider $F(\tau_1^-, \tau_1^+, \dots)$ but also $F(\tau_1^+, \tau_1^-, \dots)$. We again use superscripts $\mp$ and $\pm$ to indicate this reversal of order of 'contravariant' and 'covariant' arguments: for any sequence $\vec{\tau} := (\tau_1^-, \tau_1^+, \dots, \tau_n^-, \tau_n^+)$ of FPC-types we put $\vec{\tau}^{\mp} := \vec{\tau}$ and $\vec{\tau}^{\pm} := (\tau_1^+, \tau_1^-, \dots, \tau_n^+, \tau_n^-)$.

**2.36 Definition (syntactic functor)** A functor $F\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ is said to be *syntactic* iff there is an *associated* type-in-context $\alpha_1^{\mp}, \dots, \alpha_n^{\mp} \vdash \sigma$ satisfying the following properties for all collections of closed types $\vec{\tau} = \tau_1^-, \tau_1^+, \dots, \tau_n^-, \tau_n^+$ and all types $\varrho^-$ and $\varrho^+$:

1. $F(\vec{\sigma}) = \sigma\,[\vec{\tau} \,/\, \vec{\alpha}]$

2. for any $i \in \{1, \dots, n\}$, the restriction of the morphism part of the functor $F$ in the $i$th argument pair to the product of Hom-sets $\mathbb{D}(\varrho^-, \tau_i^-) \times \mathbb{D}(\tau_i^+, \varrho^+)$ can be *implemented* in FPC. This means there is a closed term

$$m : [\varrho^- {\to} \tau_i^-] {\times} [\tau_i^+ {\to} \varrho^+] \to \Big[\sigma\,[\vec{\tau} \,/\, \vec{\alpha}] \to \sigma\,\big[\varrho^{\mp} \,/\, \alpha_i^{\mp}, \tau_j^{\mp} \,/\, \alpha_j^{\mp}, j \neq i\big]\Big]$$

such that, for all morphisms $[s^-]_{\mathrm{obs}} : \varrho^- \to \tau_i^-$ and $[s^+]_{\mathrm{obs}} : \tau_i^+ \to \varrho^+$,

$$F\Big(\mathsf{id}, \dots, \mathsf{id}, [s^-]_{\mathrm{obs}}, [s^+]_{\mathrm{obs}}, \mathsf{id}, \dots, \mathsf{id}\Big) = [t\langle s^-, s^+\rangle]_{\mathrm{obs}}$$

**2.37 Definition (minimal invariant, generalized)** Let $F\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ be a functor. A functor $H\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^{n-1} \to \mathbb{D}$ together with a natural isomorphism $\mathsf{fold}_{\vec{\tau}}\colon F\big(H(\vec{\tau}^{\pm}), H(\vec{\tau}^{\mp}), \vec{\tau}^{\mp}\big) \cong H(\vec{\tau}^{\mp})$ with $\vec{\tau}$ ranging over all sequences $(\tau_2^-, \tau_2^+, \dots, \tau_n^-, \tau_n^+)$ of types is called a *minimal invariant in the first argument pair* of $F$ if it has the following universal property:

> For any two objects $\varrho^-$ and $\varrho^+$, any sequence of types $\vec{\tau}$ and any two morphisms $f^-\colon \varrho^- \to F(\varrho^+, \varrho^-, \vec{\tau}^{\mp})$ and $f^+\colon F(\varrho^-, \varrho^+, \vec{\tau}^{\pm}) \to \varrho^+$ there is a unique pair of morphisms $k^-\colon \varrho^- \to H(\vec{\tau}^{\mp})$ and $k^+\colon H(\vec{\tau}^{\pm}) \to \varrho^+$ making Diagram 2.17 commute. Here $\mathsf{unfold}_{\vec{\tau}}$ stands for the inverse of $\mathsf{fold}_{\vec{\tau}}$.

$$\begin{tikzcd}
F\big(H(\vec{\tau}^{\pm}), H(\vec{\tau}^{\mp}), \vec{\tau}^{\mp}\big) & H(\vec{\tau}^{\mp}) \arrow[l, "\mathsf{unfold}_{\vec{\tau}}"'] & F\big(H(\vec{\tau}^{\mp}), H(\vec{\tau}^{\pm}), \vec{\tau}^{\pm}\big) \arrow[r, "\mathsf{fold}_{\vec{\tau}}"] \arrow[d, "F(k^-{,}k^+{,}\vec{\mathsf{id}})"'] & H(\vec{\tau}^{\pm}) \arrow[d, "k^+"] \\
F(\varrho^+, \varrho^-, \vec{\tau}^{\mp}) \arrow[u, "F(k^+{,}k^-{,}\vec{\mathsf{id}})"] & \varrho^- \arrow[l, "f^-"] \arrow[u, "k^-"'] & F(\varrho^-, \varrho^+, \vec{\tau}^{\pm}) \arrow[r, "f^+"'] & \varrho^+
\end{tikzcd}$$

Diagram 2.17: Universal property of the generalized minimal invariant

Similarly one can define the notion of minimal invariant in any other of the argument pairs of $F$. ◇

Definition 2.34 is obviously a special case of Definition 2.37: For $F\colon \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$, the functor $H$ becomes a constant functor, the natural isomorphism becomes one single isomorphism $\mathsf{fold}$ and the diagrams in the general definition boils down to the simpler ones in Definition 2.34.

Using the fact that Marz's $\mathsf{SD}$-model is an adequate model for FPC, a purely syntactical argument suffices to show that every syntactic functor has a minimal invariant in each argument pair. A similar proposition for categories of domains can be found in [Pit96, Section 7] and [Mar00, Proposition 2.3].

**2.38 Proposition** *For every syntactic functor $F\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ and any number $i \in \{1, \dots, n\}$ there exists a minimal invariant in the $i$th argument pair consisting of a* syntactic *functor $H$ and a natural isomorphism*

$$\mathsf{fold}_{\vec{\tau}}\colon F\big(H(\vec{\tau}^{\pm}), H(\vec{\tau}^{\mp}), \vec{\tau}^{\mp}\big) \cong H(\vec{\tau}^{\mp}).$$

*Proof:* Let $F\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ be a syntactic functor and $\alpha_1^{\mp}, \dots, \alpha_n^{\mp} \vdash \sigma^{\mp}$ be an associated type-in-context. Without loss of generality assume $i = 1$ and let $\sigma := \sigma^{\mp}\,[\alpha \,/\, \alpha^-,\ \alpha \,/\, \alpha^+]$ be the type obtained by replacing the first two arguments by a fresh type variable $\alpha$.

For any sequence of objects $\vec{\tau} = (\tau_2^-, \tau_2^+, \dots, \tau_n^-, \tau_n^+)$ let

$$\begin{aligned} H(\vec{\tau}^{\mp}) &:= \mu\alpha.\,\sigma\left[\tau_i^{\mp} \,/\, \alpha_i^{\mp}\right] \quad \text{and} \\ \mathsf{fold}_{\vec{\tau}} &:= \left[\lambda x : \sigma\left[\mu\alpha.\,\sigma/\alpha\right]\left[\tau_i^{\mp} \,/\, \alpha_i^{\mp}\right].\,\mathbf{fold}(x)\right]_{\text{obs}}. \end{aligned}$$

Before defining the morphism part of $H$ we verify the universal property. So assume $\varrho^-$, $\varrho^+ \in \mathbb{D}$, $f^- : \varrho^- \to F(\varrho^+, \varrho^-, \vec{\tau}^{\mp})$ and $f^+ : F(\varrho^-, \varrho^+, \vec{\tau}^{\pm}) \to \varrho^+$.

*Existence of $k^-$ and $k^+$:* The two diagrams in Definition 2.37 are equivalent to the equations

$$\begin{aligned} k^- &= \mathsf{fold}_{\vec{\tau}^{\mp}} \circ F(k^+, k^-, \vec{\mathsf{id}}) \circ f^- \\ k^+ &= f^+ \circ F(k^-, k^+, \vec{\mathsf{id}}) \circ \mathsf{unfold}_{\vec{\tau}^{\pm}}. \end{aligned}$$

As $F$ is syntactic, these mutually recursive equations in $\mathbb{D}$ can be translated into equations of closed terms and by forming pairs and applying the appropriate fixed point combinator one obtains a closed term $s$ such that $k^- := [\mathbf{pl}(s)]$ and $k^+ := [\mathbf{pr}(s)]$ make the diagrams commute. Specifically, for $f^- = [q^-]$ and $f^+ = [q^+]$ and $t$ as in Definition 2.36, we set

$$\begin{aligned} s :\equiv \mathbf{Y}\,\lambda y : &\left(\varrho^- {\to} H(\vec{\tau}^{\mp})\right) \times \left(H(\vec{\tau}^{\pm}) {\to} \varrho^+\right). \\ &\Big\langle \lambda x{:}\varrho^-.\,\mathbf{fold}\big(t\langle \mathbf{pr}(y), \mathbf{pl}(y)\rangle(q^- x)\big), \lambda x{:}H(\vec{\tau}^{\pm}).\,q^+\big(ty\,(\mathbf{unfold}(x))\big)\Big\rangle. \end{aligned}$$

*Uniqueness of $k^-$ and $k^+$:* If $h^-$ and $h^+$ satisfy the above equations, both the interpretation of the closed terms for $k^-$ and $k^+$ and for $h^-$ and $h^+$ in some adequate domain model, like the SD-model, make the diagrams from Definition 2.37 in SD commute. Thus these interpretations have to coincide and we obtain $k^- = h^-$ and $k^+ = h^+$.

Now we are able to define the morphism part of the functor $H$ applying the universal property. Given a collection of morphisms $u_i^- : \varrho_i^- \to \tau_i^-$ and $u_i^+ : \tau_i^+ \to \varrho_i^+$, $i \in \{2, \dots, n\}$, we define $H(\vec{u}^{\mp}) : H(\vec{\tau}^{\mp}) \to H(\vec{\varrho}^{\mp})$ to be the first component of the pair $(k^-, k^+)$ given by the universal property in Definition 2.37 for the morphisms

$$\begin{aligned} f^- &:= F(\mathsf{id}_{H(\vec{\tau}^{\pm})}, \mathsf{id}_{H(\vec{\tau}^{\mp})}, \vec{u}^{\mp}) \circ \mathsf{unfold}_{\vec{\tau}^{\mp}} \\ f^+ &:= \mathsf{fold}_{\vec{\tau}^{\mp}} \circ F(\mathsf{id}_{H(\vec{\tau}^{\mp})}, \mathsf{id}_{H(\vec{\tau}^{\pm})}, \vec{u}^{\pm}). \end{aligned}$$

Hence $H(\vec{u}^{\mp})$ is uniquely determined by Diagram 2.18.

Uniqueness of $k^-$ and $k^+$ ensures that this action on morphisms preserves identities and composition.

It is easily seen that $H$ is a syntactic functor with associated type $\mu\alpha.\,\sigma$. We leave it to the reader to explicitly describe the FPC implementation of the morphism part of $H$ (see also Proposition 2.41 below). □

$$\begin{tikzcd}
F\big(H(\vec\varrho^{\pm}), H(\vec\varrho^{\mp}), \vec\varrho^{\mp}\big) & H(\vec\varrho^{\mp}) \arrow[l, "\mathsf{unfold}_{\vec\varrho}"'] \\
F\big(H(\vec\tau^{\pm}), H(\vec\tau^{\mp}), \vec\varrho^{\mp}\big) \arrow[u, "F(k^{+},k^{-},\vec{\mathsf{id}})"] & H(\vec\tau^{\mp}) \arrow[l, "f^{-}"'] \arrow[u, dotted, "k^{-}=H(\vec u^{\mp})"'] \arrow[dl, "\mathsf{unfold}_{\vec\tau}"] \\
F\big(H(\vec\tau^{\pm}), H(\vec\tau^{\mp}), \vec\tau^{\mp}\big) \arrow[u, "F(\mathsf{id},\mathsf{id},\vec u^{\mp})"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
F\big(H(\vec\varrho^{\mp}), H(\vec\varrho^{\pm}), \vec\varrho^{\pm}\big) \arrow[r, "\mathsf{fold}_{\vec\varrho}"] \arrow[d, "F(k^{-},k^{+},\vec{\mathsf{id}})"'] & H(\vec\varrho^{\pm}) \arrow[d, dotted, "k^{+}"] \\
F\big(H(\vec\tau^{\mp}), H(\vec\tau^{\pm}), \vec\varrho^{\pm}\big) \arrow[r, "f^{+}"] \arrow[d, "F(\mathsf{id},\mathsf{id},\vec u^{\pm})"'] & H(\vec\tau^{\pm}) \\
F\big(H(\vec\tau^{\mp}), H(\vec\tau^{\pm}), \vec\tau^{\pm}\big) \arrow[ur, "\mathsf{fold}_{\vec\tau}"'] &
\end{tikzcd}$$

Diagram 2.18: The morphism part of $H$

**2.39 Notation** In Proposition 2.38 the functor $H\colon \mathbb{D}^{\mathrm{op}} \times \mathbb{D} \to \mathbb{D}$ is constructed out of the functor $F\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D}) \to \mathbb{D}$ as a minimal invariant in the $i$th argument pair. To indicate this dependence, we write

$$H =: \mathrm{rec}_i\, F.$$

**2.40 Definition** Let $\Theta \equiv \alpha_1, \ldots, \alpha_n$ be a type context. For any type-in-context $\Theta \vdash \sigma$, we define the corresponding syntactic functor $F_{\Theta\vdash\sigma}\colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ by induction on the structure of $\sigma$. For any collection $\vec{x}^{\mp} = (x_1^-, x_1^+, \ldots, x_n^-, x_n^+)$ of either objects or morphisms in $\mathbb{D}^{\mathrm{op}}$ and $\mathbb{D}$, respectively, we put

$$\begin{aligned}
F_{\Theta\vdash\alpha_i}\vec{x}^{\mp} &:= x_i^+ \\
F_{\Theta\vdash\sigma_1\times\sigma_2}\ \vec{x}^{\mp} &= F_{\Theta\vdash\sigma_1}\ \vec{x}^{\mp} \times F_{\Theta\vdash\sigma_2}\ \vec{x}^{\mp} \\
F_{\Theta\vdash\sigma_1\to\sigma_2}\ \vec{x}^{\mp} &= F_{\Theta\vdash\sigma_1}\ \vec{x}^{\pm} \to F_{\Theta\vdash\sigma_2}\ \vec{x}^{\mp} \\
F_{\Theta\vdash\sum_{i=0}^{n}\sigma_i}\ \vec{x}^{\mp} &= \sum_{i=0}^{n} F_{\Theta\vdash\sigma_i}\ \vec{x}^{\mp} \\
F_{\Theta\vdash\mu\alpha.\,\sigma}\ \vec{x}^{\mp} &= \mathrm{rec}_1\, F_{\alpha,\Theta\vdash\sigma}\ \vec{x}^{\mp} \text{ for } \alpha \notin \{\alpha_1, \ldots, \alpha_n\}
\end{aligned}$$

**2.41 Proposition** *For any type-in-context $\Theta \vdash \sigma$, the functor $F_{\Theta\vdash\sigma}$ is a syntactic functor. For later use in Section 4.2 we give an explicit description of a special case of its FPC-implementation.*

*For any closed FPC-type $\varrho$, for any collection $\vec\tau := (\tau_1, \ldots, \tau_n)$ of closed FPC-types and any number $i \in \{1, \ldots, n\}$ we define*

$$\sigma^{\vec\tau} := F_{\Theta\vdash\sigma}(\vec\tau) = \sigma\big[\tau_i \,/\, \alpha_i\big]$$
$$\sigma^{\varrho} := F_{\Theta\vdash\sigma}(\tau_1, \ldots, \tau_{i-1}, \varrho, \varrho, \tau_{i+1}, \ldots, \tau_n) = \sigma\big[\varrho \,/\, \alpha_i, \tau_j \,/\, \alpha_j \text{ where } j \neq i\big].$$

*Further, let $\pi_i := [\tau_i{\to}\varrho]{\times}[\varrho{\to}\tau_i]$. Then the functor $F_{\Theta\vdash\sigma}$ induces a function of type $\pi_i \to [\sigma^{\varrho}{\to}\sigma^{\vec\tau}]$ mapping $g^- : \tau_i \to \varrho$ and $g^+ : \varrho \to \tau_i$ to*

$$F_{\Theta\vdash\sigma}(\mathit{id}, \ldots, \mathit{id}, g^-, g^+, \mathit{id}, \ldots, \mathit{id}).$$

*As $F$ is syntactic, this function is implementable, i. e. there is a closed FPC-term $t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i \to \left[\sigma^{\varrho}\to\sigma^{\vec{\tau}}\right]$ such that, for all closed terms $k^- : \tau_i\to\varrho$ and $k^+ : \varrho\to\tau_i$, we get*

$$F_{\Theta\vdash\sigma}\left(\mathit{id},\ldots,\mathit{id},\left[k^-\right]_{obs},\left[k^+\right]_{obs},\mathit{id},\ldots,\mathit{id}\right) = \left[t^{\vec{\varrho},\tau,i}_{\Theta\vdash\sigma}k^-k^+\right]_{obs}.$$

*For $j \in \{1,\ldots,n\}$ with $j \neq i$ explicit descriptions of these FPC-terms are given in Table 2.3. For later use in Chapter 4 we further define*

$$t^{\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i \to \left[\sigma^{\vec{\tau}}\to\sigma^{\vec{\tau}}\right], \qquad t^{\vec{\tau},i}_{\Theta\vdash\sigma} :\equiv t^{\tau_i,\vec{\tau},i}_{\Theta\vdash\sigma}.$$

| $\sigma$ | $t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma}$ |
|---|---|
| $\mathbf{void}$ | $\Omega_{\pi_i\to[\mathbf{void}\to\mathbf{void}]}$ |
| $\alpha_i$ | $\lambda x : \pi_i.\, \mathbf{pr}(x)$ |
| $\alpha_j$ | $\lambda x : \pi_i.\, \mathbf{id}_{\tau_j}$ |
| $\sigma_1\times\sigma_2$ | $\lambda x : \pi_i.\, \lambda y : \sigma_1^{\vec{\tau}}\times\sigma_2^{\vec{\tau}}.\langle t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_1}x(\mathbf{pl}(y)), t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_2}x(\mathbf{pr}(y))\rangle$ |
| $\sigma_1\to\sigma_2$ | $\lambda x : \pi_i.\, \lambda y : [\sigma_1^{\vec{\tau}}\to\sigma_2^{\vec{\tau}}].\, \lambda z{:}\sigma_1^{\vec{\tau}}.\, t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_2}x\Big(y\big(t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_1}\langle\mathbf{pr}(x),\mathbf{pl}(x)\rangle z\big)\Big)$ |
| $\sum_{i=0}^{m}\sigma_i$ | $\lambda x : \pi_i.\, \lambda y : \sum_{j=0}^{m}\sigma_j^{\vec{\tau}}.\, \mathbf{case}\, y\, \mathbf{of}\, \mathbf{in}_0 w \Rightarrow \mathbf{in}_0(t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_0}xw) \,[]\, \ldots \,[]\, \mathbf{in}_m w \Rightarrow \mathbf{in}_m(t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma_m}xw)$ |

Table 2.3: Implementation of syntactic functors in FPC

Recursive types are discussed in the following lemma. We leave it to the reader to verify the details of the above proposition.

**2.42 Lemma** *Consider type contexts $\Theta :\equiv \alpha_1,\ldots,\alpha_n$ and $\Theta' :\equiv \alpha_2,\ldots,\alpha_n$ and let $\varrho$ be a closed type and $\vec{\tau}_2 := (\tau_2,\ldots,\tau_n)$ be a sequence of closed types. Further, let $\Theta \vdash \sigma$ be a type-in-context and let $\tau_1 := \mu\alpha_1.\sigma$ and $\vec{\tau} := (\tau_1,\ldots,\tau_n)$.*

*Let $i \in \{2,\ldots,n\}$ and $\pi_i := [\tau_i\to\varrho]\times[\varrho\to\tau_i]$. Assume $t^{\varrho,\vec{\tau},1}_{\Theta\vdash\sigma} : \pi_1 \to [\sigma^{\vec{\tau}}\to\sigma^{\vec{\tau}}]$ and $t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i \to [\sigma^{\vec{\tau}}\to\sigma^{\vec{\tau}}]$ are FPC-implementations as described in Proposition 2.41 for the syntactic functor $F_{\Theta\vdash\sigma}$.*

*Then we claim that*

$$\begin{aligned} t^{\varrho,\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma} &\equiv \lambda p{:}\pi_i.\,\mathbf{pl}\big(\mathbf{Y}_{\pi_1}\,\lambda q{:}\pi_1.\langle l,r\rangle\big) \quad \textit{where} \\ l &:\equiv \lambda x{:}\tau_1.\,\mathbf{fold}\Big(t^{\varrho,\vec{\tau},1}_{\Theta\vdash\sigma}\langle\mathbf{pr}(q).\,\mathbf{pl}(q)\rangle\big(t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma}p(\mathbf{unfold}(x))\big)\Big) \\ r &:\equiv \lambda x{:}\tau_1.\,\mathbf{fold}\Big(t^{\varrho,\vec{\tau},i}_{\Theta\vdash\sigma}\langle\mathbf{pr}(p),\mathbf{pl}(p)\rangle\big(t^{\varrho,\vec{\tau},1}_{\Theta\vdash\sigma}q(\mathbf{unfold}(x))\big)\Big) \end{aligned}$$

*is an FPC-implementation of the function*

$$(g^-,g^+)\mapsto F_{\Theta'\vdash\mu\alpha_1.\,\sigma}(\mathit{id},\dots,\mathit{id},g^-,g^+,\mathit{id},\dots,\mathit{id}).$$

*Proof:* Let $g^-:\tau_i\to\varrho$ and $g^+:\varrho\to\tau_i$ be morphisms. By definition we have $F_{\Theta'\vdash\mu\alpha_1.\,\sigma}=\mathrm{rec}_1\,F_{\Theta\vdash\sigma}$. The proof of Proposition 2.38 shows that $\mathrm{rec}_1\,F_{\Theta\vdash\sigma}$ applied to $\vec{u}^{\mp}:=(\mathsf{id},\dots,\mathsf{id},g^-,g^+,\mathsf{id},\dots,\mathsf{id})$ is the first component of the pair $(k^-,k^+)$ of morphisms which is uniquely determined by Diagram 2.19. Replacing functors by their implementations completes the proof. $\square$

$$\begin{tikzcd} \sigma^{\vec{\tau}} \arrow[r, "\mathsf{fold}"] & \tau_1 \\ \sigma^{\vec{\tau}} \arrow[u, "F_{\Theta\vdash\sigma}(k^+,k^-,\vec{\mathsf{id}})"] & \tau_1 \arrow[u, dotted, "k^-"'] \arrow[dl, "\mathsf{unfold}"] \\ \sigma^{\vec{\tau}} \arrow[u, "F_{\Theta\vdash\sigma}(\mathsf{id},\mathsf{id},\vec{u}^{\mp})"] & \end{tikzcd} \qquad \begin{tikzcd} \sigma^{\vec{\tau}} \arrow[d, "F_{\Theta\vdash\sigma}(k^-,k^+,\vec{\mathsf{id}})"'] & \tau_1 \arrow[l, "\mathsf{unfold}"'] \arrow[d, dotted, "k^+"] \\ \sigma^{\vec{\tau}} \arrow[d, "F_{\Theta\vdash\sigma}(\mathsf{id},\mathsf{id},\vec{u}^{\pm})"'] & \tau_1 \\ \sigma^{\vec{\tau}} \arrow[ur, "\mathsf{fold}"'] & \end{tikzcd}$$

Diagram 2.19: Implementation of a syntactic functor for a recursive type

## 2.8 Properties of the canonical realizability model

Applying the results about lifting and minimal invariants in $\mathbb{D}$ we define the canonical realizability model for FPC in $\mathbb{D}$. Any type-in-context $\Theta\vdash\sigma$ is interpreted as the corresponding syntactic functor $F_{\Theta\vdash\sigma}$. The interpretation of a closed type $\sigma$ is a constant functor. Hence it can be identified with the value of this functor, which is just $\sigma$ viewed as an object of $\mathbb{D}$.

A context $\Gamma\equiv x_1{:}\sigma_1,\dots,x_n{:}\sigma_n$ is interpreted as the product $[\![\Gamma]\!]:=[\![\sigma_1]\!]\times\cdots\times[\![\sigma_n]\!]$. A term-in-context $\Gamma\vdash t{:}\sigma$ is interpreted as a morphism $[\![\Gamma\vdash t]\!]:[\![\Gamma]\!]\to[\![\sigma]\!]$. Let $\Gamma$ denote an arbitrary context. We define the interpretation of terms in context according to Table 2.4.

Given a realizability model for FPC, the question arises how good this model fits the programming language. There are some standard criteria to classify

$$\begin{aligned}
[\![x_1{:}\sigma_1, \dots, x_n{:}\sigma_n \vdash x_i]\!] &:= \mathsf{pr}_i \\
[\![\Gamma \vdash \langle s, t\rangle]\!] &:= \big\langle [\![\Gamma \vdash s]\!], [\![\Gamma \vdash t]\!] \big\rangle \\
[\![\Gamma \vdash \mathbf{pl}(t)]\!] &:= \mathsf{pr}_1 \circ [\![\Gamma \vdash t]\!] \\
[\![\Gamma \vdash \mathbf{pr}(t)]\!] &:= \mathsf{pr}_2 \circ [\![\Gamma \vdash t]\!] \\
[\![\Gamma \vdash ts]\!] &:= \mathsf{eval} \circ \big\langle [\![\Gamma \vdash t]\!], [\![\Gamma \vdash s]\!] \big\rangle \\
[\![\Gamma \vdash \lambda x{:}\sigma.\, t]\!] &:= \mathsf{curry}\, [\![\Gamma, x{:}\sigma \vdash t]\!] \\
[\![\Gamma \vdash \mathbf{in}_i(t)]\!] &:= \mathsf{in}_i \circ [\![\Gamma \vdash t]\!] \\
\left[\!\!\left[ \begin{matrix} \Gamma \vdash \mathbf{case}\, t\, \mathbf{of}\ \mathbf{in}_0 w \Rightarrow t_0 \,[]\, \dots \,[] \\ \mathbf{in}_m w \Rightarrow t_m \end{matrix} \right]\!\!\right] &:= \mathsf{case} \circ \big\langle [\![\Gamma \vdash t]\!], [\![\Gamma, W{:}\tau_0 \vdash t_0]\!], \dots, \\
&\qquad\qquad [\![\Gamma, w{:}\tau_m \vdash t_m]\!] \big\rangle \\
[\![\Gamma \vdash \mathbf{fold}(t)]\!] &:= \mathsf{fold} \circ [\![\Gamma \vdash t]\!] \\
[\![\Gamma \vdash \mathbf{unfold}(t)]\!] &:= \mathsf{unfold} \circ [\![\Gamma \vdash t]\!]
\end{aligned}$$

Table 2.4: Interpretation of terms-in-contexts

models with respect to this aspect, see e. g. [Str02]. Any reasonable model has to be computationally adequate with respect to the operational semantics of the language. A much stronger criterion is full abstraction, stating that observational and denotational equality coincide. The construction of a satisfactory fully abstract model for the sequential functional language of PCF had been an open problem for many years when O'Hearn and Riecke [OR95] presented their Kripke relations model. A similar fully abstract model for SFL, a sequential language of the same expressivity as FPC, was constructed by Marz [Mar00]. However, Kripke relations models are not universal, i. e. not all elements of the model are definable in the language. For realizability models Longley stated the even stronger criterion of (constructive) logical full abstraction.

From the construction of the combinatory algebra $\mathcal{T}$ of closed FPC-terms we expect the canonical model to represent the programming language as exact as can be. Indeed it satisfies any of these quality criteria.

**2.43 Definition** A model for FPC is *correct* with respect to the operational semantics iff $t \Downarrow v$ implies $[\![t]\!] = [\![v]\!]$ for any closed term $t$ and any syntactic value $v$.

The operational semantics is complete with respect to a model iff $[\![t]\!] \neq \bot$ implies $t \Downarrow \mathbf{up}(\Omega_{\mathbf{void}})$ for any closed term $t{:}\mathbf{unit}$.

A model is called *computationally adequate* for the operational semantics iff it is correct and the semantics is complete and $[\![t]\!] = [\![\mathbf{up}\,\Omega_{\mathbf{void}}]\!]$ implies $t \Downarrow \mathbf{up}\,\Omega_{\mathbf{void}}$ for any closed term $t$ of type $\mathbf{unit}$. As a consequence, $[\![s]\!] = [\![t]\!]$ implies $s =_{\text{obs}} t$ for closed terms $s$, $t$ of the same type.

An adequate model is *fully abstract* iff, for any closed terms $s$, $t$ of the same type,

$$s =_{\mathrm{obs}} t \iff [\![s]\!] = [\![t]\!] \,.$$

An adequate model is *universal* iff, for any closed FPC-type $\sigma$ and any element $x \in [\![\sigma]\!]$, there is a term $t{:}\sigma$ such that $[\![t]\!] = x$.

The definition of logical full abstraction is given below. $\diamond$

**2.44 Theorem** *The canonical model for FPC is computationally adequate, fully abstract and universal.*

This theorem is a trivial consequence of the following lemma.

**2.45 Lemma** *For any context $\Gamma = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ and any term-in-context $\Gamma \vdash t{:}\tau$ let $\overline{t}$ denote the closed term $\lambda x{:}\sigma_1 \times \ldots \times \sigma_n.\, t\,[\mathbf{pr}_i\, x \,/\, x_i]$.*
*Then $[\![\Gamma \vdash t]\!] = \left[\overline{t}\right]_{obs}$, i. e. the interpretation of a term-in-context in the canonical model is just the observational class of the corresponding closed term. In particular, $[\![t]\!] = [t]_{obs}$ for any closed term $t$.*

*Proof:* The proof is a straightforward induction on the typing rules. We illustrate it for function application and leave the other cases to the reader. Hence assume $[\![\Gamma \vdash s : \sigma{\to}\tau]\!] = [\overline{s}]_{\mathrm{obs}}$ and $[\![\Gamma \vdash t{:}\sigma]\!] = \left[\overline{t}\right]_{\mathrm{obs}}$ and consider the term-in-context $\Gamma \vdash st{:}\tau$.

$$\begin{array}{lll}
[\![\Gamma \vdash st]\!] & = & \mathsf{eval} \circ \left\langle [\![\Gamma \vdash t]\!] \,, [\![\Gamma \vdash s]\!] \right\rangle \\
& \ni & \lambda x{:}\sigma_1 \times \ldots \times \sigma_n.\Big(\lambda p{:}[\sigma{\to}\tau]{\times}\sigma.\big(\mathbf{pl}(p)\big)\big(\mathbf{pr}(p)\big)\Big) \left\langle \overline{s}x, \overline{t}x \right\rangle \\
& =_{\mathrm{obs}} & \lambda x{:}\sigma_1 \times \ldots \times \sigma_n.\, (\overline{s}x)\,\big(\overline{t}x\big) \\
& =_{\mathrm{obs}} & \lambda x{:}\sigma_1 \times \ldots \times \sigma_n.\, st\,[\mathbf{pr}_i\, x \,/\, x_i] \\
& =_{\mathrm{obs}} & \overline{st}
\end{array}$$

The notion of (constructive) logical full abstraction was first sketched in Chapter 8 of John Longley's PhD Thesis [Lon95] and defined and studied for untyped partial combinatory algebras in [Lon99a]. In order to generalize his definition to typed combinatory algebras, we consider the class $\mathbf{J}$ of logical formulae given by the following grammar:

$$\varphi \quad ::= \quad s =_\sigma t \;\big|\; r{\downarrow} \;\big|\; \varphi \wedge \varphi \;\big|\; \varphi \Rightarrow \varphi \;\big|\; \exists x{:}\sigma.\, \varphi \;\big|\; \forall x{:}\sigma.\, \varphi$$

where $\sigma$ is a closed FPC-type, $s$, $t{:}\sigma$ are closed terms of this type and $r{:}\mathbf{unit}$ is a closed term. One should think about $=_\sigma$ being an equality predicate at type $\sigma$ and $\downarrow$ being a termination predicate at the unit type; we usually omit the subscript in equality formulae. Any formula $\varphi$ in $\mathbf{J}$ is assigned a closed FPC-type $\mathrm{t}(\varphi)$ as follows:

$$\begin{array}{rcl} \mathrm{t}(s=t) & := & \mathbf{unit} \quad =: \quad \mathrm{t}(r\!\downarrow) \\ \mathrm{t}(\varphi \wedge \psi) & := & \mathrm{t}(\varphi) \times \mathrm{t}(\psi) \\ \mathrm{t}(\varphi \Rightarrow \psi) & := & \mathrm{t}(\varphi) \rightarrow \mathrm{t}(\psi) \\ \mathrm{t}(\exists x{:}\sigma.\, \psi) & := & \sigma \times \mathrm{t}(\varphi) \\ \mathrm{t}(\forall x{:}\sigma.\, \varphi) & := & \sigma \rightarrow \mathrm{t}(\varphi) \end{array}$$

The categorical structure of $\mathbb{M} = \mathsf{Mod}(\mathcal{T})$ is rich enough to provide a realizability interpretation for the formulae of $\mathbf{J}$ within the canonical model. However, there is another natural notion of realizability, defined purely in terms of the programming language FPC and without reference to any particular model. In this definition, realizers are closed FPC-terms rather than elements of a combinatory algebra. Inductively we define a relation $t\,\mathbf{r}\,\varphi$ between a closed FPC-term $t$ and a closed formula $\varphi$ of $\mathbf{J}$. Assume $\varrho$ is a closed type and $\Gamma \vdash s{:}\varrho$ and $\Gamma \vdash s'{:}\varrho$ are terms-in-context. Further let $\varphi$ and $\psi \in \mathbf{J}$ be formulae and let $\sigma := \mathrm{t}(\varphi)$ and $\tau := \mathrm{t}(\psi)$.

- $t\,\mathbf{r}\,s =_{\varrho} s'$ for any $t{:}\mathbf{unit}$ iff $s =_{\mathrm{obs}} s'$.
- $t\,\mathbf{r}\,r\!\downarrow$ for any $t{:}\mathbf{unit}$ iff $r{:}\mathbf{unit}$ terminates.
- $t\,\mathbf{r}\,\varphi \wedge \psi$ iff $t : \sigma{\times}\tau$ and $\mathbf{pl}(t)\,\mathbf{r}\,\varphi$ and $\mathbf{pr}(t)\,\mathbf{r}\,\psi$.
- $t\,\mathbf{r}\,\varphi \Rightarrow \psi$ iff $t : \sigma{\rightarrow}\tau$ and $tt'\,\mathbf{r}\,\psi$ whenever $t'\,\mathbf{r}\,\varphi$ for $t'{:}\sigma$.
- $t\,\mathbf{r}\,\exists x{:}\sigma.\,\varphi$ iff $t : \sigma{\times}\tau$ and $\mathbf{pr}(t)\,\mathbf{r}\,\varphi\,[\mathbf{pl}(t)\;/\;x]$.
- $t\,\mathbf{r}\,\forall x{:}\sigma.\,\varphi$ iff $t : \sigma{\rightarrow}\tau$ and $tt'\,\mathbf{r}\,\varphi\,[t'\;/\;x]$, whenever $t'{:}\sigma$.

If there exists a closed FPC-term $t$ such that $t\,\mathbf{r}\,\varphi$, we say that $\varphi$ is realizable in FPC.

**2.46 Definition** A realizability model of FPC is called is *(constructively) logically fully abstract* iff a closed formula $\varphi \in \mathbf{J}$ is realizable in the model iff it is realizable in FPC. $\diamond$

**2.47 Theorem** *The canonical model is (constructively) logically fully abstract.*

This theorem is again a consequence of Lemma 2.45.

Thus the canonical model is the 'best fitting' realizability model for FPC. On the other hand, this model is derived from the syntax of FPC. Hence it does not provide new insights into the nature of sequential programming languages, which is in some sense unsatisfactory. Therefore this thesis aims at constructing a realizability model for FPC which is not simply defined in terms of the syntax of FPC, but which is isomorphic to the canonical model.

# 3 The universal type

In [LS02] it is shown that a typed partial combinatory algebra $\mathcal{T}$ is applicatively equivalent to an untyped one iff there is a *universal type* $U$ in $\mathcal{T}$ such that every type of $\mathcal{T}$ is a partial retract of $U$. In this case the category of modest sets over $\mathcal{T}$ can be embedded into a realizability topos. We will state and discuss the special case of universal types for the combinatory algebra $\mathcal{T}$ of FPC in Section 3.1. In Section 3.2 we present the untyped language $\lambda$+Error, an extension of call-by-name $\lambda$-calculus. Further we will consider the object $D$ in $\mathbb{D}$ which is given by the interpretation of the FPC-type $U :\equiv \mu\alpha.\,\mathbf{void} + [\alpha{\to}\alpha]$ in the canonical realizability model. We will show that the untyped language $\lambda$+Error can be interpreted in this object $D$. In Section 3.3 we exploit this interpretation to show that the FPC-type $U$ is universal. The final section of this chapter, Section 3.4, is devoted to presenting a fixed point operator $\mathbf{fix} : [U{\to}U]{\to}U$ in FPC which can easily be implemented in the untyped language $\lambda$+Error.

## 3.1 Universal types in general

An FPC-type $U$ is a universal type in the sense of [LS02] of the combinatory algebra $\mathcal{T}$ of closed FPC-terms iff any closed FPC-type is an FPC-definable retract of $U$ (see Definition 3.1 below). The main part of this section is devoted to a nice necessary and sufficient criterion for universality: an FPC $U$ is universal iff the types $U{\to}U$, $U{\times}U$ and finite sum types constructed of $U$s are definable retracts of $U$.

**3.1 Definition** A closed FPC-type $\sigma$ is called an *FPC-definable retract* of an FPC-type $\tau$ (denoted $\sigma \lhd \tau$) iff there exist closed FPC-terms $e : \sigma{\to}\tau$ and $p : \tau{\to}\sigma$ such that, for any closed FPC-term $t{:}\sigma$,

$$p(et) =_{\text{obs}} t,$$

i. e. $t$ is observationally equivalent to $p(et)$.

The pair $e$, $p$ is said to be a *retraction pair*, $e$ is called *embedding*, $p$ is called *projection.* Note that we do not impose any condition on $e \circ p$. Thus $p$ is not neccessarily a projection in the domain theoretic sense. $\diamond$

Being a definable retract is transitive, i. e. $\sigma \lhd \tau \lhd \varrho$ implies $\sigma \lhd \varrho$.

**3.2 Definition** A type $U$ of FPC is called *universal* iff every closed FPC-type is an FPC-definable retract of $U$. ◇

**3.3 Example** One of the main results of this thesis is that the FPC-type

$$U :\equiv \mu\alpha.\, \mathbf{void} + [\alpha{\to}\alpha]$$

is universal. We prove this in Section 3.3 using the subsequent lemma.

Obviously, every type which has a universal type as a definable retract is itself universal.

Note that the following useful characterisation for unversality relies on the universal canonical realizability model introduced in Section 2.8.

**3.4 Lemma** *An FPC-type $U$ is universal iff the following types are definable retracts of $U$:*

$$[U{\to}U], \qquad U{\times}U, \qquad \sum_{i=0}^{m} U \textit{ for all } m \in \mathbb{N}.$$

*Proof:* One of the implications is trivial. In a proof for the other implication we have to handle recursive types. Hence it does not suffice to look at closed types only. In order to deal with types containing free variables we will use syntactic functors.

Assume $\Theta \equiv \alpha_1, \ldots, \alpha_n$ is a type context and $\Theta \vdash \sigma$ is an type-in-context. Throughout this proof we will call a sequence $\vec{\tau} := (\tau_1, \ldots, \tau_n)$ of closed FPC-types *suitable* iff the types $\tau_1, \ldots, \tau_n$ are FPC-definable retracts of $U$. We further denote by $\sigma^{\vec{\tau}}$ the closed FPC-type $F_{\Theta\vdash\sigma}(\tau_1, \tau_1, \ldots, \tau_n, \tau_n)$.

We have to show that, for any suitable sequence $\vec{\tau}$, the closed type $\sigma^{\vec{\tau}}$ is an FPC-definable retract of $U$. We prove this by induction on the typing rules.

Let the retraction pairs of $[U{\to}U]$, $U{\times}U$ and $\sum_{i=0}^{n} U$ be denoted by $\mathbf{e}_{[U\to U]}$, $\mathbf{p}_{[U\to U]}$ and so on. Further let $\Theta \vdash \sigma$, $\Theta \vdash \sigma_0, \ldots, \Theta \vdash \sigma_m$ be types-in-context such that, for any suitable sequence $\vec{\tau}$, the closed types $\sigma^{\vec{\tau}}$, $\sigma_0^{\vec{\tau}}, \ldots, \sigma_m^{\vec{\tau}}$ are definable retracts of $U$ with retraction pairs $\mathbf{e}_{\sigma_i^{\vec{\tau}}}$, $\mathbf{p}_{\sigma_i^{\vec{\tau}}}$. Let $\vec{\tau}$ be a suitable sequence of types.

We have to show that the following types are definable retracts of $U$:

i) $\mathbf{void}^{\vec{\tau}}$ ii) $\alpha_i^{\vec{\tau}}$ iii) $\sigma_0^{\vec{\tau}}{\to}\sigma_1^{\vec{\tau}}$

iv) $\sigma_0^{\vec{\tau}}{\times}\sigma_1^{\vec{\tau}}$ v) $\displaystyle\sum_{j=0}^{m} \sigma_j^{\vec{\tau}}$ vi) $F_{\Theta'\vdash\mu\alpha_1.\,\sigma}(\tau_2, \tau_2, \ldots, \tau_n, \tau_n)$

where $i \in \{1, \dots, n\}$ and $\Theta' \equiv \alpha_2, \dots, \alpha_n$. For notational reasons we restrict to recursion over $\alpha_1$; we may do this without loss of generality.

Claim (i) is obvious and (ii) trivially holds as $\tau_i$ is a definable retract of $U$. As $\lhd$ is transitive, for Claims (iii) to (v) it suffices to show

$$\text{(iii)}\quad \sigma_0^{\vec{\tau}} \to \sigma_1^{\vec{\tau}} \lhd [U \to U] \qquad \text{(iv)}\quad \sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}} \lhd [U \times U] \qquad \text{(v)}\quad \sum_{j=0}^{m} \sigma_j^{\vec{\tau}} \lhd \sum_{j=0}^{m} U$$

In order to prove these claims, we choose the following retraction pairs:

$$\text{(iii)}\quad \widetilde{e}_{\sigma_0 \to \sigma_1} : [\sigma_0^{\vec{\tau}} \to \sigma_1^{\vec{\tau}}] \to [U \to U] \qquad \widetilde{p}_{\sigma_0 \to \sigma_1} : [U \to U] \to [\sigma_0^{\vec{\tau}} \to \sigma_1^{\vec{\tau}}]$$

$$\widetilde{e}_{\sigma_0 \to \sigma_1} \equiv \lambda f : \sigma_0 \to \sigma_1.\, \lambda x{:}U.\, \mathbf{e}_{\sigma_1^{\vec{\tau}}}\big(f(\mathbf{p}_{\sigma_0^{\vec{\tau}}}\, x)\big)$$

$$\widetilde{p}_{\sigma_0 \to \sigma_1} \equiv \lambda f : U \to U.\, \lambda x{:}\sigma_0^{\vec{\tau}}.\, \mathbf{p}_{\sigma_1^{\vec{\tau}}}\big(f(\mathbf{e}_{\sigma_0^{\vec{\tau}}}\, x)\big)$$

$$\text{(iv)}\quad \widetilde{e}_{\sigma_0 \times \sigma_1} : \sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}} \to U \times U \qquad \widetilde{p}_{\sigma_0 \times \sigma_1} : U \times U \to \sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}}$$

$$\widetilde{e}_{\sigma_0 \times \sigma_1} \equiv \lambda x : \sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}}.\, \Big\langle \mathbf{e}_{\sigma_0^{\vec{\tau}}}\big(\mathbf{pl}(x)\big), \mathbf{e}_{\sigma_1^{\vec{\tau}}}\big(\mathbf{pr}(x)\big) \Big\rangle$$

$$\widetilde{p}_{\sigma_0 \times \sigma_1} \equiv \lambda x : U \times U.\, \Big\langle \mathbf{p}_{\sigma_0^{\vec{\tau}}}\big(\mathbf{pl}(x)\big), \mathbf{p}_{\sigma_1^{\vec{\tau}}}\big(\mathbf{pr}(x)\big) \Big\rangle$$

$$\text{(v)}\quad \widetilde{e}_{\sum_{j=0}^{m} \sigma_j} : \sum_{j=0}^{m} \sigma_j^{\vec{\tau}} \to \sum_{j=0}^{m} U \qquad \widetilde{p}_{\sum_{j=0}^{m} \sigma_j} : \sum_{j=0}^{m} U \to \sum_{j=0}^{m} \sigma_j^{\vec{\tau}}$$

$$\begin{aligned}\widetilde{e}_{\sum_{j=0}^{m} \sigma_j} \equiv \lambda x : \sum_{j=0}^{m} \sigma_j^{\vec{\tau}}.\, \mathbf{case}\, x\, \mathbf{of}\ & \mathbf{in}_0 w \Rightarrow \mathbf{in}_0(\mathbf{e}_{\sigma_0^{\vec{\tau}}}\, w) \,[]\, \dots \,[] \\ & \mathbf{in}_m w \Rightarrow \mathbf{in}_m(\mathbf{e}_{\sigma_m^{\vec{\tau}}}\, w)\end{aligned}$$

$$\begin{aligned}\widetilde{p}_{\sum_{j=0}^{m} \sigma_j} \equiv \lambda x : \sum_{j=0}^{m} U.\, \mathbf{case}\, x\, \mathbf{of}\ & \mathbf{in}_0 w \Rightarrow \mathbf{in}_0(\mathbf{p}_{\sigma_0^{\vec{\tau}}}\, w) \,[]\, \dots \,[] \\ & \mathbf{in}_m w \Rightarrow \mathbf{in}_m(\mathbf{p}_{\sigma_n^{\vec{\tau}}}\, w)\end{aligned}$$

It is left to the reader to verify that these pairs of closed terms indeed are retraction pairs, which is not difficult.

Finally we have to consider the case of the recursive type $\Theta' \vdash \mu\alpha_i.\, \sigma$. We define

$$\begin{aligned}\tau_1 &:= F_{\Theta' \vdash \mu\alpha_1.\,\sigma}(\tau_2, \tau_2, \dots, \tau_n, \tau_n) \ = \mu\alpha_1.\, \sigma\big[\tau_i \,/\, \alpha_i \text{ where } i > 1\big] \\ \sigma^U &:= F_{\Theta \vdash \sigma}(U, U, \tau_2, \tau_2, \dots, \tau_n, \tau_n) = \sigma\big[U \,/\, \alpha_1, \tau_i \,/\, \alpha_i \text{ where } i > 1\big] \\ \vec{\tau}_U &:= U, \tau_2, \dots, \tau_n\end{aligned}$$

As $U \lhd U$, the sequence $\vec{\tau}_U$ is suitable. Hence, by induction hypothesis, $\sigma^U$ is a definable retract of $U$ with a retraction pair $\mathbf{e}_{\sigma^U}$, $\mathbf{p}_{\sigma^U}$.

For any $i \in \{1, \dots n\}$ let $t_{\tau_1,U} := t^{\vec{\tau},\vec{\tau}_U,i}_{\Theta\vdash\sigma}$ and $t_{U,\tau_1} := t^{\vec{\tau}_U,\vec{\tau},i}_{\Theta\vdash\sigma}$ be the FPC-implementations defined in Proposition 2.41 for the functions

$$\begin{aligned}(g^-, g^+) &\mapsto F_{\Theta\vdash\sigma}(g^-, g^+, \mathsf{id}, \dots, \mathsf{id}) &:\quad& [\tau_1{\to}U]{\times}[U{\to}\tau_1] \to [\sigma^U{\to}\sigma^{\vec{\tau}}]\\ (g^-, g^+) &\mapsto F_{\Theta\vdash\sigma}(g^-, g^+, \mathsf{id}, \dots, \mathsf{id}) &:\quad& [U{\to}\tau_1]{\times}[\tau_1{\to}U] \to [\sigma^{\vec{\tau}}{\to}\sigma^U].\end{aligned}$$

We claim that the following recursive equations define a retraction pair for the recursive type $\tau_1$:

$$\begin{aligned}\mathbf{e}_{\tau_1} &=_{\text{obs}} \lambda x{:}\tau_1.\, \mathbf{e}_{\sigma^U}\big(t_{\tau_1,U}\ \langle \mathbf{p}_{\tau_1}, \mathbf{e}_{\tau_1}\rangle\ (\mathbf{unfold}(x))\big)\\ \mathbf{p}_{\tau_1} &=_{\text{obs}} \lambda x{:}U.\, \mathbf{fold}\big(t_{U,\tau_1}\ \langle \mathbf{e}_{\tau_1},\ \mathbf{p}_{\tau_1}\rangle\ (\mathbf{p}_{\sigma^U}\ x)\big)\end{aligned}$$

Using the fixed point combinator on the product type $\varrho := [\tau_1{\to}U]{\times}[U{\to}\tau_1]$ we obtain as an explicit definition of the retraction pair $\mathbf{e}_{\tau_1} :\equiv \mathbf{pl}(t)$ and $\mathbf{p}_{\tau_1} :\equiv \mathbf{pr}(t)$ where

$$\begin{aligned}t :\equiv \mathbf{Y}_\varrho\, \lambda y{:}\varrho.\Big\langle &\lambda x{:}\tau_1.\, \mathbf{e}_{\sigma^U}\big(t_{\tau_1,U}\ \langle \mathbf{pr}(y), \mathbf{pl}(y)\rangle\ (\mathbf{unfold}(x))\big),\\ &\qquad\qquad \lambda x{:}U.\, \mathbf{fold}\big(t_{U,\tau_1} y\, (\mathbf{p}_{\sigma^U}\ x)\big)\Big\rangle.\end{aligned}$$

$$\begin{tikzcd}
\sigma^U \arrow[r, "\mathbf{e}_{\sigma^U}"] & U \\
\sigma^{\vec{\tau}} \arrow[u, "F(\mathbf{p}_{\tau_1}{,}\mathbf{e}_{\tau_1}{,}\vec{\mathsf{id}})"] & \tau_1 \arrow[u, "\mathbf{e}_{\tau_1}"'] \arrow[l, "\mathsf{unfold}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^U \arrow[d, "F(\mathbf{e}_{\tau_1}{,}\mathbf{p}_{\tau_1}{,}\vec{\mathsf{id}})"'] & U \arrow[l, "\mathbf{p}_{\sigma^U}"'] \arrow[d, "\mathbf{p}_{\tau_1}"] \\
\sigma^{\vec{\tau}} \arrow[r, "\mathsf{fold}"'] & \tau_1
\end{tikzcd}$$

Diagram 3.1: A retraction pair for a recursive type

Abbreviating $F_{\Theta\vdash\sigma}$ to $F$, the above recursive equations correspond to Diagram 3.1 in $\mathsf{Mod}(\mathcal{T})$. Combining the squares in Diagram 3.1 and applying

$$F(\mathbf{e}_{\tau_1}, \mathbf{p}_{\tau_1}, \mathsf{id}_B, \mathsf{id}_B) \circ F(\mathbf{p}_{\tau_1}, \mathbf{e}_{\tau_1}, \mathsf{id}_B, \mathsf{id}_B) = F(\mathbf{p}_{\tau_1} \circ \mathbf{e}_{\tau_1}, \mathbf{p}_{\tau_1} \circ \mathbf{e}_{\tau_1}, \mathsf{id}_B, \mathsf{id}_B)$$

yields Diagram 3.2. According to Definition 2.40, $\tau_1$ is the minimal invariant of the syntactic functor $F(\_, \_, \tau_2, \tau_2, \dots, \tau_n, \tau_n)$. Hence commutativity of the latter diagram implies $\mathbf{p}_{\tau_1} \circ \mathbf{e}_{\tau_1} = \mathsf{id}_{\tau_1}$, as explained in Remark 2.35. Consequently, in the SD-model the equation $[\![\mathbf{p}_{\tau_1}(\mathbf{e}_{\tau_1}\, t)]\!] = [\![t]\!]$ holds for any closed term $t{:}\tau_1$. As the realizability model for FPC in $\mathsf{Mod}(\text{FPC}/_{=_{\text{obs}}})$ is fully abstract, this implies $\mathbf{p}_{\tau_1}(\mathbf{e}_{\tau_1}\, t) =_{\text{obs}} t$. Thus the terms $\mathbf{e}_{\tau_1}$ and $\mathbf{p}_{\tau_1}$ form a retraction pair. □

**3.5 Remark** *The retraction $\mathbf{r}_{\tau_1} := \mathbf{e}_{\tau_1} \circ \mathbf{p}_{\tau_1}$ is the unique morphism making Diagram 3.3 commute.*

$$\begin{tikzcd}
\sigma^{\vec{\tau}} \arrow[d, "F(\mathbf{p}_{\tau_1},\mathbf{e}_{\tau_1},\vec{\mathsf{id}})"'] & \tau_1 \arrow[l, "\mathsf{unfold}"'] \arrow[d, "\mathbf{e}_{\tau_1}"] \\
\sigma^{U} \arrow[r, shift left, "\mathbf{e}_{\sigma^U}"] \arrow[d, "F(\mathbf{e}_{\tau_1},\mathbf{p}_{\tau_1},\vec{\mathsf{id}})"'] & U \arrow[l, shift left, "\mathbf{p}_{\sigma^U}"] \arrow[d, "\mathbf{p}_{\tau_1}"] \\
\sigma^{\vec{\tau}} \arrow[r, "\mathsf{fold}"'] & \tau_1
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^{\vec{\tau}} \arrow[r, "\mathrm{fold}"] \arrow[d, "F(\mathbf{p}_{\tau_1}\circ\mathbf{e}_{\tau_1},\mathbf{p}_{\tau_1}\circ\mathbf{e}_{\tau_1},\mathsf{id}_B,\mathsf{id}_B)"'] & \tau_1 \arrow[d, "\mathbf{p}_{\tau_1}\circ\mathbf{e}_{\tau_1}"] \\
\sigma^{\vec{\tau}} \arrow[r, "\mathrm{fold}"'] & \tau_1 .
\end{tikzcd}$$

Diagram 3.2: $\mathbf{p}_{\tau_1} \circ \mathbf{e}_{\tau_1} = \mathsf{id}$

$$\begin{tikzcd}
F(U,U,B,B) \arrow[d, "F(\mathbf{r}_{\tau_1},\mathbf{r}_{\tau_1},\mathsf{id}_B,\mathsf{id}_B)"'] & U \arrow[l, "\mathbf{p}_{\sigma[\varphi_U]}"'] \arrow[d, "\mathbf{r}_{\tau_1}"] \\
F(U,U,B,B) \arrow[r, "\mathbf{e}_{\sigma[\varphi_U]}"] & U
\end{tikzcd}$$

Diagram 3.3: The universal property of $\mathbf{r}_{\tau_1}$

Recall that a closed FPC-term $p : \sigma \to \tau$ is called *strict* iff $p\,\Omega_\sigma =_{\text{obs}} \Omega_\tau$.

**3.6 Lemma** *If $U$ is a universal type and there are strict projections for the types $U \to U$, $U \times U$ and $\sum_{i=0}^{n} U$, then, for any closed FPC-type $\sigma$, the projection $\mathbf{p}_\sigma$ constructed in the proof of Lemma 3.4 is also strict.*

*Proof:* The proof is by induction on the structure of $\sigma$. Assume $\vec{\tau}$ is a sequence of types with strict projections. For the non-recursive cases it is easy to verify that the constructions from Lemma 3.4 lead to strict projections.

From Proposition 2.41 and Lemma 2.42 it is readily seen that the closed FPC-terms $t_{\tau_1,U}$ and $t_{U,\tau_1}$ in the same proof are strict. Hence $F\left(\mathbf{e}_{\tau_1}, \mathbf{p}_{\tau_1}, \vec{\mathsf{id}}\right) : \sigma^U \to \sigma^{\vec{\tau}}$ is strict. The morphism $\mathsf{fold}$ is also strict. Hence $\mathbf{p}_{\tau_1}$ has to be strict. □

## 3.2 The untyped language $\lambda$+Error

The language $\lambda$+Error is an extension of untyped lazy call-by-name lambda-calculus (cf. [Abr90]) by a single constant $\mathsf{ERR}$ and a conditional construct $\mathsf{IFR}$. The constant $\mathsf{ERR}$ is thought of as an error element and the term $\mathsf{IFR}(M, N_0, N_1)$ evaluates to $N_0$ iff $M$ evaluates to $\mathsf{ERR}$ and to $N_1$ if $M$ evaluates to some $\lambda$-abstraction.

We will show that the FPC-type $U = \mu\alpha.\,\mathbf{void} + [\alpha{\to}\alpha]$ contains $U{\to}U$ as a definable retract. Hence the language $\lambda$+Error can be interpreted in $\mathbb{D}$ in a canonical way. In particular, any closed $\lambda$+Error-term corresponds to a closed FPC-term of type $U$. We will use this interpretation in 3.3 to show that the type $U$ is universal. In Section 4.4 we will consider the category of modest sets over the combinatory algebra obtained by factorizing the set of closed $\lambda$+Error-terms by observational equivalence and show that it is a universal model for the typed language FPC.

**3.7 Definition** We define the language $\lambda$+Error as follows:

**Contexts** $\Gamma \equiv x_1, \dots, x_n$

**Terms** $M ::= x \mid \lambda x.\, M \mid MM \mid \mathsf{ERR} \mid \mathsf{IFR}(M, M, M)$

**Syntactic values** $V ::= \mathsf{ERR} \mid \lambda x.\, M$

**Operational semantics**

$$\frac{}{\lambda x.\, M \Downarrow \lambda x.\, M} \qquad \frac{}{\mathsf{ERR} \Downarrow \mathsf{ERR}} \qquad \frac{M \Downarrow \lambda x.\, M' \quad M'[N/x] \Downarrow V}{MN \Downarrow V}$$

$$\frac{M \Downarrow \mathsf{ERR} \quad N_1 \Downarrow V}{\mathsf{IFR}(M, N_1, N_2) \Downarrow V} \qquad \frac{M \Downarrow \lambda x.\, M' \quad N_2 \Downarrow V}{\mathsf{IFR}(M, N_1, N_2) \Downarrow V}$$

In order to interpret this language in the FPC-type $U = \mu\alpha.\,\mathbf{void} + [\alpha{\to}\alpha]$, we have to show that the function type $[U{\to}U]$ is a definable retract of $U$.

**3.8 Lemma** $[U{\to}U] \lhd U$

*Proof:* Unfolding $U$ gives the type $\mathbf{void} + [U{\to}U]$, which obviously contains $[U{\to}U]$ as a retract. A straightforward computation shows that

$$\begin{aligned} \mathbf{e}_{U\to U} &\equiv \lambda f{:}[U{\to}U].\, \mathbf{fold}\big(\mathbf{in}_1(f)\big) \\ \mathbf{p}_{U\to U} &\equiv \lambda r{:}U.\, \mathbf{case}\, \mathbf{unfold}(r)\ \mathbf{of}\ \mathbf{in}_0 f \Rightarrow \Omega_{U\to U} \,[\!]\, \mathbf{in}_1 f \Rightarrow f \end{aligned}$$

defines a retraction pair. Note that the projection $\mathbf{p}_{U\to U}$ is strict. □

We write **err** for the closed FPC-term $\mathbf{fold}\big(\mathbf{in}_0(\Omega_{\mathbf{void}})\big)$ of type $U$.

**3.9 Definition** Inductively we define a syntactic translation of a $\lambda$+Error-term-in-context $x_1, \dots, x_n \vdash M$ into an FPC-term-in-context $x_1{:}U, \dots, x_n{:}U \vdash t{:}U$. For any $\lambda$+Error-context $\Gamma \equiv x_1, \dots, x_n$ we will, by abuse of notation, use $\Gamma$ also to denote the corresponding FPC-context $x_1{:}U, \dots, x_n{:}U$.

**Variables:** For any $i \in \{1, \dots, n\}$ define $(\!|\Gamma \vdash x_i|\!) :\equiv \Gamma \vdash x_i$.

**Abstraction:** If $\Gamma, y \vdash M$ is translated to $\Gamma, y \vdash t$, then $(\!|\Gamma \vdash \lambda y.\, M|\!) :\equiv \Gamma \vdash \mathbf{e}_{U \to U}\, \lambda x{:}U.\, t$

**Application:** If $(\!|\Gamma \vdash M|\!)$ is $\Gamma \vdash s$ and $(\!|\Gamma \vdash N|\!)$ is $\Gamma \vdash t$ , then we let $(\!|\Gamma \vdash MN|\!) :\equiv \Gamma \vdash \mathbf{p}_{U \to U}\, s\, t$

**Error element:** $(\!|\Gamma \vdash \mathsf{ERR}|\!) :\equiv \Gamma \vdash \mathbf{err}$

**Conditional:** If $t \equiv (\!|\Gamma \vdash M|\!)$ and $s_i \equiv (\!|\Gamma \vdash N_i|\!)$, for $i \in \{0, 1\}$, then

$$\Big(\!\!\Big|\Gamma \vdash \mathsf{IFR}(M, N_0, N_1)\Big|\!\!\Big) :\equiv \\ \Gamma \vdash \mathbf{case}\,\mathbf{unfold}(t)\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow s_0 \,[\!]\, \mathbf{in}_1 w \Rightarrow s_1.$$

Using the syntactic translation we can define an interpretation of λ+Error-terms in the category $\mathbb{D}$. A λ+Error-term-in-context $\Gamma \vdash M$ is interpreted as the observational class of the corresponding FPC-term-in-context, i. e. as the $\mathbb{D}$-morphism $[\![(\!|\Gamma \vdash M|\!)]\!] : U^n \to U$. ◇

**3.10 Definition** Two closed λ+Error-terms $M$ and $N$ are said to be *observationally equivalent*, written $M =_{\text{obs}} N$ or simply $M = N$, iff, for any closed λ+Error-term $P$, the term $PM$ reduces to a syntactic value if, and only if, the term $PN$ does.

Similar to Definition 2.12, the notion of observational equivalence can be generalized to terms: Let $\Gamma \equiv x_1, \ldots, x_n$ be a non-empty context and assume $\Gamma \vdash M$ and $\Gamma \vdash N$ are λ+Error-terms-in-context. Then $M$ and $N$ are called observationally equivalent with respect to $\Gamma$, written $\Gamma \vdash M =_{\text{obs}} N$, iff $\lambda x_1. \ldots \lambda x_n.\, M =_{\text{obs}} \lambda x_1. \ldots \lambda x_n.\, N$, i. e. the corresponding closed terms are equivalent. ◇

**3.11 Lemma** *If $M$ is a closed λ+Error-term and $V$ is a syntactic value such that $M \Downarrow V$, then the interpretations of $M$ and $V$ in $\mathbb{D}$ are observationally equivalent FPC-terms.*

*If $M$ is a closed λ+Error-term that does not reduce to a syntactic value, then the interpretation of $M$ in $\mathbb{D}$ is observationally equivalent to the diverging FPC-term $\Omega_U$.*

*In particular, two closed λ+Error-terms $M$ and $N$ are observationally equivalent iff the corresponding FPC-terms $(\!|M|\!)$ and $(\!|N|\!)$ are, i. e. iff their interpretations in $\mathbb{D}$ coincide.*

The proof of this lemma is straightforward. In any adequate domain model for FPC, the interpretations of two $\beta$-equivalent closed λ+Error-terms $M$ and $N$ obviously coincide and hence we obtain the following corollary.

**3.12 Corollary** *Two closed λ+Error-terms $M$ and $N$ which are $\beta$-equivalent are also observationally equivalent. In particular, $(\lambda x.\, M)N =_{obs} M\,[N\,/\,x]$ for all closed terms $M$ and $N$.*

The language λ+Error is powerful enough to express arithmetic and to encode products.

**3.13 Definition** Inductively we define numerals in λ+Error as

$$\overline{0} :\equiv \mathsf{ERR} \qquad \overline{n+1} :\equiv \lambda x.\, \overline{n}.$$

A test for zero is simply given by $\mathsf{IFR}$. Terms for the successor and predecessor function are

$$\mathsf{SUCC} :\equiv \lambda x.\, \lambda y.\, x \qquad \mathsf{PRED} :\equiv \lambda x.\, \mathsf{IFR}(x, \mathsf{ERR}, x\overline{0}).$$

Pairs can be encoded using the closed terms

$$\mathsf{PAIR} :\equiv \lambda x, y, z.\, \mathsf{IFR}(z, x, y) \qquad \mathsf{FST} :\equiv \lambda x.\, x\,\overline{0} \qquad \mathsf{SND} :\equiv \lambda x.\, x\,\overline{1}.$$

Note that pairing not is surjective, i. e. we get the equations

$$\begin{array}{rcl} \mathsf{PRED}(\mathsf{SUCC}\,x) = x & \text{and} & \\ \mathsf{FST}(\mathsf{PAIR}\,x\,y) = x & \text{and} & \mathsf{SND}(\mathsf{PAIR}\,x\,y) = y, \end{array}$$

but the following equations in general do not hold:

$$\mathsf{SUCC}(\mathsf{PRED}\,x) = x \qquad \text{and} \qquad \mathsf{PAIR}\,(\mathsf{FST}\,x)\,(\mathsf{SND}\,x) = x.$$

Letting $\mathsf{ID} :\equiv \lambda x.\, x$ we define a fixed point combinator $\mathsf{FIX}$ and an always diverging term $\Omega$ as

$$\begin{array}{rcl} \mathsf{FIX} & :\equiv & \lambda f.\, K\big(\lambda y.\, Ky\big), \quad \text{where } K :\equiv \lambda x.\, f(xx) \\ \Omega & :\equiv & \mathsf{FIX}\,\mathsf{ID}\,. \end{array}$$

It is easy to check that $\mathsf{FIX}$ indeed is a fixed point combinator, i. e. $\mathsf{FIX}\,f = f(\mathsf{FIX}\,f)$. The proof of Lemma 4.3(*xiv*) shows why we choose this $\eta$-expanded variant of the usual Church-style fixed point combinator. Finally, we inductively define a case construct for any natural number $n \in \mathbb{N}$:

$$\mathsf{CASE}_0\, M \;\mathsf{OF}\; M_0 :\equiv \mathsf{IFR}(M, M_0, \Omega)$$

$$\mathsf{CASE}_{n+1}\, M \;\mathsf{OF}\; M_0 \,[]\, \ldots \,[]\, M_{n+1} :\equiv$$
$$\mathsf{IFR}\Big(M, M_0, \big(\mathsf{CASE}_n(\mathsf{PRED}\,M)\;\mathsf{OF}\; M_1 \,[]\, \ldots \,[]\, M_{n+1}\big)\Big)$$

## 3.3 Construction of a universal type

In this section we show that $U = \mu\alpha.\, \mathbf{void} + [\alpha{\to}\alpha]$ is a universal type and we choose a canonical retraction pair $\mathbf{e}_\sigma$, $\mathbf{p}_\sigma$ for each closed type $\sigma$. According to Lemma 3.4 and 3.8, we can prove universality of $U$ simply by presenting retraction pairs for the types $U{\times}U$ and $\sum_{i=0}^{n} U$. Let us first define some auxiliary FPC-terms.

**3.14 Definition** Let $n$ be a natural number and $t$, $t_0$ ... , $t_n$, $M$, $M_0$, ... , $M_n$ be closed FPC-terms of type $U$ and closed λ+Error-terms, respectively, such that $t \equiv (\!|M|\!)$ and $t_i \equiv (\!|M_i|\!)$ for $i \in \{0, \dots .n\}$. Write $\mathbf{e}_U$, $\mathbf{p}_U$ for the trivial retraction pair $\mathbf{e}_U :\equiv \mathbf{p}_U :\equiv \mathbf{id}_U$. Recall that, according to Lemma 3.8 and Lemma 3.4, the projections $\mathbf{p}_{U\to U}$, $\mathbf{p}_{U\to U\to U}$ and $\mathbf{p}_{U\to U\to U\to U}$ are defined as

$$\begin{aligned} \mathbf{p}_{U\to U} &\equiv \lambda r{:}U.\, \mathbf{case}\, \mathbf{unfold}(r)\ \mathbf{of}\ \mathbf{in}_0 f \Rightarrow \Omega_{U\to U} \,[\!]\, \mathbf{in}_1 f \Rightarrow f \\ \mathbf{p}_{U\to U\to U} &\equiv \lambda x, y{:}U.\, \mathbf{p}_{U\to U}(\mathbf{p}_{U\to U}\, x\, y) \\ \mathbf{p}_{U\to U\to U\to U} &\equiv \lambda x, y{:}U.\, \mathbf{p}_{U\to U\to U}(\mathbf{p}_{U\to U}\, x\, y). \end{aligned}$$

We define FPC-terms for arithmetic, pairing and case analysis in $U$ according to Table 3.1. ◇

It is an easy exercise to verify the equalities postulated in Table 3.1.

Our next aim is to show that the FPC-type $U = \mu\alpha.\, \mathbf{void} + [\alpha{\to}\alpha]$ is indeed universal. To apply Lemma 2.4 we have to show that the product type $U{\times}U$ and all sum types $\sum_{i=0}^{n} U$ are definable retracts of $U$.

**3.15 Lemma** $U{\times}U \lhd U$

*Proof:* The product $U{\times}U$ can be identified as a retract of $U$ by encoding a pair $(x, y)$ as a function from $U$ to $U$ mapping $\underline{0}$ to $x$ and $\underline{1}$ to $y$. It is easy to verify that

$$\mathbf{e}_{U\times U} :\equiv \lambda x{:}U{\times}U.\, \mathbf{pair}\, \big(\mathbf{pl}(x)\big)\, \big(\mathbf{pr}(x)\big) \qquad \mathbf{p}_{U\times U} :\equiv \lambda x{:}U.\, \langle \mathbf{fst}\, x, \mathbf{snd}\, x\rangle\,.$$

is a retraction pair and that the projection $\mathbf{p}_{U\times U}$ is strict. □

**3.16 Lemma** $\displaystyle\sum_{i=0}^{n} U \lhd U$

*Proof:* An embedding of a finite sum of $U$'s into $U$ can be achieved by replacing $\mathbf{in}_j(x)$ by the pair $(j, x)$. Hence we will encode $\mathbf{in}_j(x)$ as $\mathbf{pair}\ \underline{j}\, x$. This

$$\begin{aligned}
\underline{0} &:\equiv (\!|\, \overline{0} \,|\!) \\
&\equiv \mathbf{fold}\big(\mathbf{in}_0(\Omega_{\mathbf{void}})\big) \\
\underline{n+1} &:\equiv (\!|\, \overline{n+1} \,|\!) \\
&\equiv \mathbf{e}_{U\to U}(\lambda x{:}U.\, \underline{n}) \\
\mathbf{ifz} &:\equiv \mathbf{p}_{U\to U\to U\to U}\, \big(\!\big|\lambda x,y,z.\, \mathsf{IFR}(x,y,z)\big|\!\big) \\
&\equiv \lambda x,y,z{:}U.\, \mathbf{case}\, \mathbf{unfold}(x)\, \mathbf{of}\, \mathbf{in}_0 w \Rightarrow y \,[]\, \mathbf{in}_1 w \Rightarrow z \\
\mathbf{succ} &:\equiv \mathbf{p}_{U\to U}\, (\!|\mathsf{SUCC}|\!) \\
&\equiv \lambda x{:}U.\, \mathbf{e}_{U\to U}\, \lambda y{:}U.\, x \\
\mathbf{pred} &:\equiv \mathbf{p}_{U\to U}\, (\!|\mathsf{PRED}|\!) \\
&\equiv \lambda x{:}U.\, \mathbf{ifz}\, x\; \mathbf{err}\; (\mathbf{p}_{U\to U}\, x\, \underline{0}) \\
\mathbf{pair} &:\equiv \mathbf{p}_{U\to U\to U}\, (\!|\mathsf{PAIR}|\!) \\
&\equiv \lambda x,y{:}U.\, \mathbf{e}_{U\to U}\; (\lambda z{:}U.\, \mathbf{ifz}\, zxy) \\
\mathbf{fst} &:\equiv \mathbf{p}_{U\to U}\, (\!|\mathsf{FST}|\!) \\
&\equiv \lambda x{:}U.\, \mathbf{p}_{U\to U}\, x\underline{0} \\
\mathbf{snd} &:\equiv \mathbf{p}_{U\to U}\, (\!|\mathsf{SND}|\!) \\
&\equiv \lambda x{:}U.\, \mathbf{p}_{U\to U}\, x\underline{1} \\
\mathbf{case}_n\, t\; \mathbf{of}\; t_0 \,[]\, \ldots \,[]\, t_n &:\equiv \big(\!\big|\mathsf{CASE}_n\, M\; \mathsf{OF}\; M_0 \,[]\, \ldots \,[]\, M_n\big|\!\big)
\end{aligned}$$

Table 3.1: Auxiliary FPC-terms

yields the following closed terms, which are easily seen to form a retraction pair. Note that the projection is strict.

$$\begin{aligned}
\mathbf{e}_{\sum_{i=0}^n U} &\equiv \lambda x{:}\sum_{i=0}^{n} U.\, \mathbf{case}\, x\; \mathbf{of}\;\; \mathbf{in}_0 w \Rightarrow \mathbf{pair}\; \underline{0}\, w \,[]\, \ldots \,[] \\
&\qquad\qquad\qquad\qquad\quad \mathbf{in}_n w \Rightarrow \mathbf{pair}\; \underline{n}\, w \\
\mathbf{p}_{\sum_{i=0}^n U} &\equiv \lambda x{:}U.\, \mathbf{case}_n\, \mathbf{fst}\, x\; \mathbf{of}\; \mathbf{in}_0\, \mathbf{snd}\, x \,[]\, \ldots \,[]\, \mathbf{in}_n\, \mathbf{snd}\, x
\end{aligned}$$

□

Lemma 3.8, 3.15 and 3.16 ensure that Lemma 3.4 can be applied. Since the projections given in these lemmas are easily seen to be strict, we obtain the following theorem.

**3.17 Theorem** *The FPC-type $U = \mu\alpha.\, \mathbf{void} + [\alpha\to\alpha]$ is universal. Moreover, for any closed type $\sigma$, we can choose a canonical retraction pair $\mathbf{e}_\sigma$, $\mathbf{p}_\sigma$, such that $\mathbf{p}_\sigma$ is strict.*

The concrete definition for the retraction pairs $\mathbf{e}_\sigma$, $\mathbf{p}_\sigma$ is given in Lemma 3.8, 3.15 and 3.16 and in the proof of 3.4. Note that the embedding $\mathbf{e}_\sigma$ in general can not be expected to be strict, since the embeddings in Lemma 3.8, 3.15 and 3.16 are not strict.

**3.18 Definition** From now on, $U$ shall denote the universal FPC-type

$$U :\equiv \mu\alpha.\, \mathbf{void} + [\alpha{\to}\alpha].$$

For any closed type $\sigma$ we write $\mathbf{p}_\sigma : U{\to}\sigma$ and $\mathbf{e}_\sigma : \sigma{\to}U$ for the *canonical retraction pair.* Further $\mathbf{r}_\sigma : U{\to}U$ stands for the *canonical retraction* $r_\sigma :\equiv \lambda x{:}U.\, \mathbf{e}_\sigma(\mathbf{p}_\sigma\, x)$. $\diamond$

Given a type-in-context $\Theta \vdash \sigma$ with a type context $\Theta :\equiv \alpha_1, \dots, \alpha_n$ and given a collection $\vec{\tau} := \tau_1, \dots, \tau_n$ of closed types and given any index $j \in \{1, \dots, n\}$, we consider the following closed types:

$$\tau_j \qquad \sigma^{\vec{\tau}} := \sigma\,[\tau_i \,/\, \alpha_i] \qquad \sigma^U := \sigma\,[U \,/\, \alpha_j, \tau_i \,/\, \alpha_i \text{ where } i \neq j]$$

It is natural to ask how the canonical retraction pairs of these types are related. The answer is given by the following lemma, which we are going to apply in order to find a $\lambda$+Error-implementation of **fold** and **unfold** for every recursive FPC-type in Lemma 4.3.

**3.19 Lemma** *Under the above assumptions, Diagram 3.4 in $\mathbb{D}$ commutes. Here and subsequently, $F := F_{\Theta\vdash\sigma} \colon (\mathbb{D}^{op} \times \mathbb{D})^n \to \mathbb{D}$ denotes the syntactic*

$$\begin{tikzcd}
\sigma^U \arrow[r, "\mathbf{e}_{\sigma^U}"] & U \\
\sigma^{\vec{\tau}} \arrow[u, "F(\vec{\mathsf{id}}{,}\mathbf{p}_{\tau_j}{,}\mathbf{e}_{\tau_j}{,}\vec{\mathsf{id}})"] \arrow[r, "\mathbf{e}_{\sigma^{\vec{\tau}}}"'] & U \arrow[u, "\mathbf{r}_{\sigma^U}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^U \arrow[d, "F(\vec{\mathsf{id}}{,}\mathbf{e}_{\tau_j}{,}\mathbf{p}_{\tau_j}{,}\vec{\mathsf{id}})"'] & U \arrow[l, "\mathbf{p}_{\sigma^U}"'] \arrow[d, "\mathbf{r}_{\sigma^U}"] \\
\sigma^{\vec{\tau}} & U \arrow[l, "\mathbf{p}_{\sigma^{\vec{\tau}}}"]
\end{tikzcd}$$

Diagram 3.4: Relation between canonical retractions

*functor defined by $\Theta \vdash \sigma$ and $\vec{\mathsf{id}}$ stands for a sequence $\mathsf{id}, \dots, \mathsf{id}$ of appropriate length.*

*Proof:* The proof is by induction on the structure of the type $\sigma$. First observe that the claim is equivalent to commutativity of Diagram 3.5.

In case $\sigma = \mathbf{void}$ this is trivial since we have $F_{\Theta\vdash\mathbf{void}}\big(\vec{\mathsf{id}}, \mathbf{p}_{\tau_j}, \mathbf{e}_{\tau_j}, \vec{\mathsf{id}}\big) = \mathsf{id}_{\mathbf{void}} = F_{\Theta\vdash\mathbf{void}}\big(\vec{\mathsf{id}}, \mathbf{e}_{\tau_j}, \mathbf{p}_{\tau_j}, \vec{\mathsf{id}}\big)$. For the same reason the claim is also trivial if $\sigma$ is a closed type.

$$\begin{tikzcd}
\sigma^U & U \arrow[l, "\mathbf{p}_{\sigma^U}"'] \\
\sigma^{\vec{\tau}} \arrow[u, "F(\vec{\mathsf{id}},\mathbf{p}_{\tau_1},\mathbf{e}_{\tau_1},\vec{\mathsf{id}})"] \arrow[ur, "\mathbf{e}_{\sigma^{\vec{\tau}}}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^U \arrow[r, "\mathbf{e}_{\sigma^U}"] \arrow[d, "F(\vec{\mathsf{id}},\mathbf{e}_{\tau_1},\mathbf{p}_{\tau_1},\vec{\mathsf{id}})"'] & U \arrow[dl, "\mathbf{p}_{\sigma^{\vec{\tau}}}"] \\
\sigma^{\vec{\tau}}
\end{tikzcd}$$

Diagram 3.5: Equivalent diagrams to Diagram 3.4

In case $\sigma = \alpha_i$, for some $i \in \{1, \ldots, n\}$, the claim is also trivial: If $i = j$ then $\sigma^U = U$ and $\mathbf{p}_{\sigma^U} = \mathsf{id}_U = \mathbf{e}_{\sigma^U}$ whereas $F(\vec{\mathsf{id}}, \mathbf{p}_{\tau_1}, \mathbf{e}_{\tau_1}, \vec{\mathsf{id}}) = \mathbf{e}_{\tau_1}$ and $F(\vec{\mathsf{id}}, \mathbf{e}_{\tau_1}, \mathbf{p}_{\tau_1}, \vec{\mathsf{id}}) = \mathbf{p}_{\tau_1}$. If $i \neq j$ then $\sigma^{\vec{\tau}} = \tau_i = \sigma^U$ and $F(\mathbf{p}_{\tau_1}, \mathbf{e}_{\tau_1}, \vec{\mathsf{id}}) = \mathsf{id}_{\tau_i} = F(\mathbf{e}_{\tau_1}, \mathbf{p}_{\tau_1}, \vec{\mathsf{id}})$.

We are left to discuss the cases $\sigma = \sigma_0 \times \sigma_1$, $\sigma = \sigma_0 \to \sigma_1$ and $\sigma = \sum_{i=0}^{n} \sigma_i$ and the case of recursive types.

Assume $\sigma = \sigma_0 \times \sigma_1$ and the claim holds for $\sigma_0$ and $\sigma_1$. Since the associated functors satisfy $F_\sigma = F_{\sigma_0} \times F_{\sigma_1}$, we obtain Diagram 3.6. In this diagram

$$\begin{tikzcd}
\sigma_0^U \times \sigma_1^U \arrow[r, "\mathbf{e}_{\sigma_0^U} \times \mathbf{e}_{\sigma_1^U}"] \arrow[rr, bend left, "\mathbf{e}_{\sigma_0^U \times \sigma_1^U}"] & U \times U \arrow[r, "\mathbf{e}_{U \times U}"] & U \\
\sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}} \arrow[u, "F_{\Theta \vdash \sigma_0 \times \sigma_1}(\vec{\mathsf{id}}, p_{\tau_1}, \mathbf{e}_{\tau_1}, \vec{\mathsf{id}})"] \arrow[ur, "\mathbf{e}_{\sigma_0^{\vec{\tau}}} \times \mathbf{e}_{\sigma_1^{\vec{\tau}}}"] \arrow[urr, "\mathbf{e}_{\sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}}}"']
\end{tikzcd}$$

Diagram 3.6: The case of product types

the top triangle and the lower right triangle commute by definition of $\mathbf{e}_{\sigma_0^U \times \sigma_1^U}$ and $\mathbf{e}_{\sigma_0^{\vec{\tau}} \times \sigma_1^{\vec{\tau}}}$ and the lower left triangle commutes by induction hypothesis. Hence the whole diagram commutes. Commutativity of the corresponding diagram for $\mathbf{p}_\sigma$ is obtained in analogous manner. The cases $\sigma = \sigma_0 \to \sigma_1$ and $\sigma = \sum_{i=0}^{n} \sigma_i$ work similarly and are left to the reader.

Now for the case of recursive types. Assume the claim of the lemma holds for some type $\Theta \vdash \sigma$. Then, without loss of generality, it suffices to prove the claim for $\Theta' \vdash \mu\alpha_1.\,\sigma$ where $\Theta' :\equiv \alpha_2, \ldots, \alpha_n$. If $n = 1$ then the syntactic functor $F_{\Theta' \vdash \mu\alpha_1.\,\sigma}$ is constant and the claim is therefore trivial. Otherwise, without loss of generality, we assume $j = 2$.

Let $\tau_2, \ldots, \tau_n$ be closed FPC-types and let $F := F_{\Theta \vdash \sigma}$ and $H := \mathrm{rec}_1\, F_{\Theta \vdash \sigma} = F_{\Theta' \vdash \mu\alpha_1.\,\sigma}$ be the syntactic functor associated with the types-in-context under consideration. Further define

$$\begin{aligned}
\vec{\tau} &:= \tau_2, \tau_2, \dots, \tau_n, \tau_n \\
\tau_1 &:= H(\vec{\tau}) = \mu\alpha_1.\, \sigma\big[\tau_i \,/\, \alpha_i \text{ where } i > 1\big] \\
\pi &:= H(U, U, \tau_3, \tau_3 \dots, \tau_n, \tau_n) = \mu\alpha_1.\, \sigma\big[U \,/\, \alpha_2, \tau_i \,/\, \alpha_i \text{ where } i > 2\big]
\end{aligned}$$

To simplify notation we are going to ignore the type variables $\alpha_3, \dots\ \alpha_n$ in the following. Thus we write $F(\gamma, \delta)$ instead of $F(\gamma, \gamma, \delta, \delta, \tau_3, \tau_3, \dots, \tau_n, \tau_n)$ for any closed FPC-types $\gamma$ and $\delta$ and we write $F(f_1, \dots, f_4)$ instead of $F(f_1, \dots, f_4, \vec{\mathsf{id}})$ and $H(f_3, f_4)$ instead of $H(f_3, f_4, \vec{\mathsf{id}})$ for any $\mathbb{D}$-morphisms $f_1, \dots, f_4$.

$$\begin{tikzcd}
\pi & U \arrow[l, "\mathbf{p}_\pi"'] & & \pi \arrow[r, "\mathbf{e}_\pi"] \arrow[d, "H(\mathbf{e}_{\tau_2}, \mathbf{p}_{\tau_2})"'] & U \arrow[dl, "\mathbf{p}_{\tau_1}"] \\
\tau_1 \arrow[u, "H(\mathbf{p}_{\tau_2}, \mathbf{e}_{\tau_2})"] \arrow[ur, "\mathbf{e}_{\tau_1}"'] & & & \tau_1 &
\end{tikzcd}$$

Diagram 3.7: The case of recursive types

$$\begin{tikzcd}
F(U,U) & U \arrow[l, "\mathbf{p}_{F(U,U)}"'] & & F(U,U) \arrow[r, "\mathbf{e}_{F(U,U)}"] \arrow[d, "F(\mathsf{id}, \mathsf{id}, \mathbf{e}_{\tau_2}, \mathbf{p}_{\tau_2})"'] & U \arrow[dl, "\mathbf{p}_{F(U,\tau_2)}"] \\
F(U,\tau_2) \arrow[u, "F(\mathsf{id}, \mathsf{id}, \mathbf{p}_{\tau_2}, \mathbf{e}_{\tau_2})"] \arrow[ur, "\mathbf{e}_{F(U,\tau_2)}"'] & & & F(U,\tau_2) &
\end{tikzcd}$$

Diagram 3.8: Induction hypothesis

Using this simplified notation the claim reads as commutativity of Diagram 3.7. By induction hypothesis we obtain Diagram 3.8 and hence Diagram 3.9. In this diagram, the square containing fold and the one containing unfold commute according to the definition of the retraction pairs for $\tau_1$ and $\pi$ in the proof of Lemma 3.4, respectively. The lower left square commutes since $F$ is a functor and the middle left square commutes by induction hypothesis. Hence the outer square commutes, as shown in Diagram 3.10.

Analogously one shows that the dual square commutes, see Diagram 3.11. On the other hand, due to Proposition 2.38, the two morphisms $k^- := H(\mathbf{p}_{\tau_2}, \mathbf{e}_{\tau_2})$ and $k^+ := H(\mathbf{e}_{\tau_2}, \mathbf{p}_{\tau_2})$ are the unique morphisms making Diagram 3.12 commute. Therefore we get $H(\mathbf{p}_{\tau_2}, \mathbf{e}_{\tau_2}) = \mathbf{p}_\pi \circ \mathbf{e}_{\tau_1}$ and $H(\mathbf{e}_{\tau_2}, \mathbf{p}_{\tau_2}) = \mathbf{p}_{\tau_1} \circ \mathbf{e}_\pi$, which is the desired conclusion. $\square$

**3.20 Corollary** *Let $\Theta \vdash \sigma$ be a type-in-context. Let $\tau_2, \dots, \tau_n$ be closed*

$$\begin{tikzcd}
F(\pi, U) \arrow[rr, "\mathsf{fold}"] & & \pi \\
F(U, U) \arrow[u, "F(\mathbf{e}_\pi{,}\mathbf{p}_\pi{,}\mathsf{id}{,}\mathsf{id})"] \arrow[d, equal] & U \arrow[l, "\mathbf{p}_{F(U,U)}"'] \arrow[ur, "\mathbf{p}_\pi"] & \\
F(U, U) & F(U, \tau_2) \arrow[l, "F(\mathsf{id}{,}\mathsf{id}{,}\mathbf{p}_{\tau_2}{,}\mathbf{e}_{\tau_2})"'] \arrow[u, "\mathbf{e}_{F(U,\tau_2)}"] & \\
F(\tau_1, U) \arrow[u, "F(\mathbf{p}_{\tau_1}{,}\mathbf{e}_{\tau_1}{,}\mathsf{id}{,}\mathsf{id})"] & F(\tau_1, \tau_1, \tau_2, \tau_2) \arrow[l, "F(\mathsf{id}{,}\mathsf{id}{,}\mathbf{p}_{\tau_2}{,}\mathbf{e}_{\tau_2})"] \arrow[u, "F(\mathbf{p}_{\tau_1}{,}\mathbf{e}_{\tau_1}{,}\mathsf{id}{,}\mathsf{id})"] & \tau_1 \arrow[l, "\mathsf{unfold}"] \arrow[ul, "\mathbf{e}_{\tau_1}"] \arrow[uuu, "\mathbf{p}_\pi \circ \mathbf{e}_{\tau_1}"']
\end{tikzcd}$$

Diagram 3.9: Combining hypotheses

$$\begin{tikzcd}
F(\pi, U) & & \pi \arrow[ll, "\mathsf{unfold}"'] \\
F(\tau_1, U) \arrow[u, "F(\mathbf{p}_{\tau_1} \circ \mathbf{e}_\pi{,}\mathbf{p}_\pi \circ \mathbf{e}_{\tau_1}{,}\mathsf{id}{,}\mathsf{id})"] & F(\tau_1, \tau_2) \arrow[l, "F(\mathsf{id}{,}\mathsf{id}{,}\mathbf{p}_{\tau_2}{,}\mathbf{e}_{\tau_2})"] & \tau_1 \arrow[l, "\mathsf{unfold}"] \arrow[u, "\mathbf{p}_\pi \circ \mathbf{e}_{\tau_1}"']
\end{tikzcd}$$

Diagram 3.10: The outer square of Diagram 3.9

$$\begin{tikzcd}
F(\pi, U) \arrow[rr, "\mathsf{fold}"] \arrow[d, "F(\mathbf{p}_\pi \circ \mathbf{e}_{\tau_1}{,}\mathbf{p}_{\tau_1} \circ \mathbf{e}_\pi{,}\mathsf{id}{,}\mathsf{id})"'] & & \pi \arrow[d, "\mathbf{p}_{\tau_1} \circ \mathbf{e}_\pi"] \\
F(\tau_1, U) \arrow[r, "F(\mathsf{id}{,}\mathsf{id}{,}\mathbf{e}_{\tau_2}{,}\mathbf{p}_{\tau_2})"'] & F(\tau_1, \tau_2) \arrow[r, "\mathsf{fold}"'] & \tau_1
\end{tikzcd}$$

Diagram 3.11: The dual square

*types. Define*

$$\begin{aligned}
\tau_1 &:= \mu\alpha_1.\, \sigma\big[\tau_i \,/\, \alpha_i \text{ where } i > 1\big] \\
\vec{\tau} &:= (\tau_1, \ldots, \tau_n) \\
\sigma^{\vec{\tau}} &:= \sigma\big[\tau_i \,/\, \alpha_i\big].
\end{aligned}$$

*Then Diagram 3.13 commutes.*

*Proof:* Let $F := F_{\Theta \vdash \sigma}$. In both squares shown in Diagram 3.14 the outer square commutes due to the definition of $\mathbf{e}_{\tau_1}$ and $\mathbf{p}_{\tau_1}$ in the proof of Lemma 3.4. Commutativity of the lower left triangle of each square is shown in Lemma 3.19. Hence the upper right triangle commutes. For $\mathsf{unfold}$ this finishes the proof.

$$\begin{tikzcd}
F(\pi, U) & & & \pi \arrow[lll, "\mathsf{unfold}"'] \\
F(\tau_1, U) \arrow[u, "{F(k^-,k^+,\mathsf{id},\mathsf{id})}"] & & F(\tau_1, \tau_1, \tau_2, \tau_2) \arrow[ll, "{F(\mathsf{id},\mathsf{id},\mathbf{p}_{\tau_2},\mathbf{e}_{\tau_2})}"] & \tau_1 \arrow[l, "\mathsf{unfold}"] \arrow[u, "k^-"']
\end{tikzcd}$$

$$\begin{tikzcd}
F(\pi, U) \arrow[rrr, "\mathsf{fold}"] \arrow[d, "{F(k^-,k^+,\mathsf{id},\mathsf{id})}"'] & & & \pi \arrow[d, "k^+"] \\
F(\tau_1, U) \arrow[rr, "{F(\mathsf{id},\mathsf{id},\mathbf{e}_{\tau_2},\mathbf{p}_{\tau_2})}"'] & & F(\tau_1, \tau_2) \arrow[r, "\mathsf{fold}"'] & \tau_1
\end{tikzcd}$$

Diagram 3.12: $k^-$ and $k^+$ are the unique morphisms making these squares commute

$$\begin{tikzcd}
\sigma^{\vec{\tau}} \arrow[rr, "\mathsf{fold}"] & & \tau_1 \\
U \arrow[u, "\mathbf{p}_{\sigma^{\vec{\tau}}}"] \arrow[r, "\mathbf{r}_{\sigma^{\vec{\tau}}}"'] & U \arrow[r, "\mathbf{r}_{\sigma^U}"'] & U \arrow[u, "\mathbf{p}_{\tau_1}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^{\vec{\tau}} & \tau_1 \arrow[l, "\mathsf{unfold}"'] \\
U \arrow[u, "\mathbf{p}_{\sigma^{\vec{\tau}}}"] & U \arrow[l, "\mathbf{r}_{\sigma^U}"] \arrow[u, "\mathbf{p}_{\tau_1}"']
\end{tikzcd}$$

Diagram 3.13: Claim of Corollary 3.20

$$\begin{tikzcd}
\sigma^{\vec{\tau}} \arrow[rr, "\mathsf{fold}"] \arrow[dd, "{F(\mathbf{p}_{\tau_1},\mathbf{e}_{\tau_1},\vec{\mathsf{id}})}"'] \arrow[dr, "\mathbf{e}_{\sigma^{\vec{\tau}}}"] & & \tau_1 \arrow[dd, "\mathbf{e}_{\tau_1}"] \\
& U \arrow[dr, "\mathbf{r}_{\sigma^U}"] & \\
\sigma^U \arrow[rr, "\mathbf{e}_{\sigma^U}"'] & & U
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma^{\vec{\tau}} & & \tau_1 \arrow[ll, "\mathsf{unfold}"'] \\
& U \arrow[ul, "\mathbf{p}_{\sigma^{\vec{\tau}}}"] & \\
\sigma^U \arrow[uu, "{F(\mathbf{e}_{\tau_1},\mathbf{p}_{\tau_1},\vec{\mathsf{id}})}"] & & U \arrow[ll, "\mathbf{p}_{\sigma^U}"] \arrow[uu, "\mathbf{p}_{\tau_1}"'] \arrow[ul, "\mathbf{r}_{\sigma^U}"']
\end{tikzcd}$$

Diagram 3.14: Proof of Corollary 3.20

For $\mathsf{fold}$ we conclude

$$\begin{aligned}
\mathbf{e}_{\tau_1} \circ \mathsf{fold} &= \mathbf{r}_{\sigma^U} \circ \mathbf{e}_{\sigma^{\vec{\tau}}} \\
\mathsf{fold} &= \mathbf{p}_{\tau_1} \circ \mathbf{r}_{\sigma^U} \circ \mathbf{e}_{\sigma^{\vec{\tau}}} \\
\mathsf{fold} \circ \mathbf{p}_{\sigma^{\vec{\tau}}} &= \mathbf{p}_{\tau_1} \circ \mathbf{r}_{\sigma^U} \circ \mathbf{r}_{\sigma^{\vec{\tau}}},
\end{aligned}$$

which completes the proof. $\square$

## 3.4 A fixed point combinator

The closed term **fix** of type $[U{\to}U]{\to}U$ introduced in the following lemma plays an important role in Chapter 4. Exploiting the fact that the SD-model is an adequate model for FPC we show that **fix** is a least fixed point combinator. The proof follows the line of [Bar84, Exercise 18.4.20].

**3.21 Lemma** *The closed FPC-term* $\mathbf{fix} : [U{\to}U]{\to}U$,

$$\mathbf{fix} :\equiv \lambda f{:}U{\to}U.\big(\lambda x{:}U.\, f(\mathbf{p}_{U\to U}\, vv)\big)\Big(\mathbf{e}_{U\to U}\big(\lambda x{:}U.\, f(\mathbf{p}_{U\to U}\, vv)\big)\Big)$$

*is observationally equivalent to* $\mathbf{Y}_U$*.*

*Proof:* It suffices to show that the interpretation $[\![\mathbf{fix}]\!]$ of **fix** in some adequate domain model, like Marz's SD-model, is a least fixed point operator. Throughout this proof we will identify closed FPC-types and their interpretation in SD and closed FPC-terms and their interpretation, respectively. E. g. we will write $U$ both for the FPC-type $\mu\alpha.\,\mathbf{void} + [\alpha{\to}\alpha]$ and for its SD interpretation. This does not cause any problems since the SD-model is fully abstract and directed suprema in SD are calculated pointwise.

Let $f\colon U \to U$ be a morphism in SD. We first show that $\mathbf{fix}\, f$ is a fixed point of $f$. Define $t := \lambda x{:}U.\, f(\mathbf{p}_{U\to U}\, vv)$.

$$\begin{aligned}\mathbf{fix}\, f &= t(\mathbf{e}_{U\to U}\, t) = f\big(\mathbf{p}_{U\to U}(\mathbf{e}_{U\to U}\, t)(\mathbf{e}_{U\to U}\, t)\big) = f\big(t(\mathbf{e}_{U\to U}\, t)\big)\\ &= f(\mathbf{fix}\, f).\end{aligned}$$

Interpreting the universal type $U$ in SD we obtain that the least fixed point of $f$ is the directed supremum $\bigvee_{n\in\mathbb{N}} f^n(\bot)$. It remains to prove that $\mathbf{fix}\, f$ equals the least fixed point, i. e.

$$\mathbf{fix}\, f \le \bigvee_{n\in\mathbb{N}} f^n(\bot_U).$$

According to [Mar00], the type $U$ is constructed in SD as a bilimit over Diagram 3.15. where 1 is the singleton domain and $F\colon \mathsf{SD}^{\mathrm{op}} \times \mathsf{SD} \to \mathsf{SD}$, $F(x, y) = 1 + [x \to y]$ is the functor corresponding to the type $U = \mu\alpha^-, \alpha^+.\, \mathbf{void} +$

$$\begin{tikzcd}
& & U \arrow[r, shift left, "\text{unfold}"] \arrow[dll, shift left, "q_0"] \arrow[dl, shift left, "q_1"] \arrow[d, shift left, "q_2"] & 1+[U\to U] \arrow[l, shift left, "\text{fold}"] \\
1 \arrow[r, shift left, "v_0"] \arrow[urr, shift left, "i_0"] & F(1,1) \arrow[l, shift left, "w_0"] \arrow[r, shift left, "v_1"] \arrow[ur, shift left, "i_1"] & F\big(F(1,1),F(1,1)\big) \arrow[l, shift left, "w_1"] \arrow[r, shift left, "v_2"] \arrow[u, shift left, "i_2"] & \cdots \arrow[l, shift left, "w_2"]
\end{tikzcd}$$

Diagram 3.15: Bilimit construction

$[\alpha^-{\to}\alpha^+]$ and for any $n \in \mathbb{N}$ the morphisms in the diagram are defined by

$$\begin{aligned}
v_0(x) &= \bot_{F(1,1)} & i_0(x) &= \bot_U \\
v_1(x) &= \bot_1 & q_0(x) &= \bot_1 \\
v_{n+1} &= F(w_n, v_n) & i_{n+1} &= \text{fold} \circ F(q_n, i_n) \\
w_{n+1} &= F(v_n, w_n) & q_{n+1} &= F(i_n, q_n) \circ \text{unfold}
\end{aligned}$$

For any $n \in \mathbb{N}$ the morphisms $v_n \circ w_n$ and $r_n := i_n \circ q_n \colon U{\to}U$ are idempotents and $w_n \circ v_n = \mathsf{id}$ and $q_n \circ i_n = \mathsf{id}$. Further $q_n = w_n \circ q_{n+1}$ and $i_n = i_{n+1} \circ v_n$. Thus $r_n \circ r_{n+1} = r_n = r_{n+1} \circ r_n$. Any element $x \in U$ satisfies $x = \bigvee_n r_n x$. For any morphisms $g$, $r \colon U \to U$ it is easy to see that $F(r,r) \colon 1 + [U \to U] \to 1 + [U \to U]$ satisfies

$$F(r,r)\left(\mathbf{in}_1^{\mathbf{void},[U\to U]}(g)\right) = \mathbf{in}_1^{\mathbf{void},[U\to U]}(r \circ g \circ r).$$

Hence we conclude

$$\begin{aligned}
\mathbf{fix}\, f = t(\mathbf{e}_{U\to U}\, t) &= \bigvee_n r_n \left(t\left(\bigvee_m r_m(\mathbf{e}_{U\to U}\, t)\right)\right) \\
&= \bigvee_{m,n} r_n\Big(t\big(r_m(\mathbf{e}_{U\to U}\, t)\big)\Big) = \bigvee_n (r_n \circ t)\big(r_n(\mathbf{e}_{U\to U}\, t)\big)
\end{aligned}$$

For any $n \in \mathbb{N}$ let $t_n := r_n(\mathbf{e}_{U\to U}\, t)$ and $f_n := r_n(\mathbf{e}_{U\to U}\, f) \in U$. We shall have established the lemma if we prove, for any $n \in \mathbb{N}$,

$$(r_n \circ t)\, t_n = \mathbf{p}_{U\to U}\, t_{n+1} t_n \le f^{n+1}(\bot_U).$$

We apply $F(q_n, i_n) \circ F(i_n, q_n) = F(r_n, r_n)$ to conclude, for any $g \colon U \to U$,

$$\begin{aligned}
r_{n+1}(\mathbf{e}_{U\to U}\, g) &= \text{fold}\Big(F(r_n, r_n)\big(\text{unfold}(\mathbf{e}_{U\to U}\, g)\big)\Big) \\
&= \text{fold}\big(F(r_n, r_n)(\mathbf{in}_1 g)\big) = \text{fold}\big(\mathbf{in}_1(r_n \circ g \circ r_n)\big) \\
&= \mathbf{e}_{U\to U}(r_n \circ g \circ r_n).
\end{aligned}$$

Consequently, $r_n \circ t \circ r_n = \mathbf{p}_{U\to U}\, t_{n+1}$ and hence $(r_n \circ t)\, t_n = p_{U\to U} t_{n+1} t_n$. On the other hand we get $r_n \circ f \circ r_n = \mathbf{p}_{U\to U}\, f_{n+1}$. Accordingly, we have $\mathbf{p}_{U\to U}\, t_{n+1} t_n = \mathbf{p}_{U\to U}\, f_{n+1}\big(\mathbf{p}_{U\to U}\, t_{n+1} t_n\big)$ for any $n \geq 1$ since

$$\begin{aligned}
\mathbf{p}_{U\to U}\, t_{n+1} t_n &= r_n(t\, t_n) \\
&= r_n\big(f(\mathbf{p}_{U\to U}\, t_n t_n)\big) \\
&= (r_n \circ f)\big((r_{n-1} \circ t \circ r_{n-1})\, t_n\big) \\
&= (r_n \circ f)\big((r_n \circ r_{n-1} \circ t \circ r_{n-1} \circ r_{n-1})\, t_n\big) \\
&= (r_n \circ f \circ r_n)\big((r_{n-1} \circ t \circ r_{n-1})(r_{n-1} t_n)\big) \\
&= \mathbf{p}_{U\to U}\, f_{n+1}(\mathbf{p}_{U\to U}\, t_n t_{n-1}).
\end{aligned}$$

In case $n = 0$ we get $\mathbf{p}_{U\to U}\, t_1 t_0 = \mathbf{p}_{U\to U}\, f_1 \bot_U$ since

$$\mathbf{p}_{U\to U}\, t_1 t_0 = (r_0 \circ t \circ r_0)\, t_0 = \bot_U = (r_0 \circ f \circ r_0)\, \bot_U = \mathbf{p}_{U\to U}\, f_1 \bot_U.$$

From $f_n = r_n(\mathbf{e}_{U\to U}\, f) \leq \mathbf{e}_{U\to U}\, f$ we conclude $f'_n := \mathbf{p}_{U\to U}\, f_n \leq f$ and hence

$$\mathbf{p}_{U\to U}\, t_{n+1} t_n = f'_{n+1}\Big(f'_n\big(\ldots(f'_1 \bot_U)\ldots\big)\Big) \leq f^{n+1} \bot_U$$

which completes the proof. $\square$

# 4 Equivalence of the languages

This chapter constitutes the heart of this thesis. It is devoted to the construction of a realizability model for FPC over the combinatory algebra of closed $\lambda$+Error-terms factorized by observational equivalence. We will show that this model is applicatively equivalent to the canonical model. As a consequence, any sequential functional program is realized by a program in the minimalistic untyped language $\lambda$+Error.

In Section 4.1 we introduce a realizability relation between FPC-terms-in-context and $\lambda$+Error-terms-in-context. We show that all term forming operations of FPC can be implemented in $\lambda$+Error, provided that the canonical retractions of FPC-types can be implemented in $\lambda$+Error.

In Section 4.2 we will show that any FPC-type-in-context has a $\lambda$+Error representation, i. e. its canonical retraction is implementable in $\lambda$+Error. To show this for recursive types we construct $\lambda$+Error-implementations for the morphism parts of the corresponding syntactic functors. Note that the FPC-implementations of the latter were given in Section 2.7. Combining this result with the results of the previous section, we conclude in Section 4.3 that any FPC-term-in-context is implemented by some $\lambda$+Error-term-in-context.

In Section 4.4 we define an untyped combinatory algebra $\mathcal{L}$ of $\lambda$+Error-terms and show this algebra to be applicatively equivalent to the typed combinatory algebra $\mathcal{T}$ defined in Section 2.3. Thus we derive the main result of this thesis: the realizability model for FPC over this algebra $\mathcal{L}$ of $\lambda$+Error-terms is universal. As an application of this result we prove of a variant of the Longley-Phoa Conjecture in Section 4.5.

## 4.1 Untyped terms as realizers

According to Section 2.8, the interpretation of an FPC-term-in-context $x{:}\sigma \vdash s$ is a morphism $f\colon \sigma \to \tau$ in the cartesian closed category $\mathbb{D}$ introduced in Section 2.4. On the other hand, in Section 3.2 we defined the interpretation of a $\lambda$+Error-term-in-context $\vec{x} \vdash M$ to be the observational class of a term-in-context $\vec{x}{:}\vec{U} \vdash t{:}U$, i. e. a $\mathbb{D}$-morphism $g\colon U^n \to U$. Hence obviously one cannot expect the interpretation of an arbitrary FPC-term to coincide with the

$$\begin{tikzcd}
\sigma \arrow[r, "\Gamma\vdash t"] & \tau \\
U^n \arrow[u, "\mathbf{p}_{\sigma_1}\times\cdots\times\mathbf{p}_{\sigma_n}"] \arrow[r, "|\Gamma|\vdash M"'] & U \arrow[u, "\mathbf{p}_\tau"']
\end{tikzcd}$$

Diagram 4.1: $M$ realizes $t$ in context $\Gamma$

interpretation of some $\lambda$+Error-term. However, by composing both morphisms with the canonical projections $\mathbf{p}_\sigma$ and $\mathbf{p}_\tau$, we obtain morphisms $f \circ \mathbf{p}_\sigma$ and $\mathbf{p}_\tau \circ g \colon U \to \tau$. Hence we can relate FPC-terms-in-context and $\lambda$+Error-terms-in-context by the following realizability relation:

**4.1 Definition** If $\Gamma \equiv x_1{:}\sigma_1, \dots, x_n{:}\sigma_n$ is an FPC-context we write $|\Gamma|$ for the FPC-context $x_1{:}U, \dots, x_n{:}U$ and, for simplicity of notation, also for the $\lambda$+Error-context $x_1, \dots, x_n$. Assume $\Gamma \vdash t{:}\tau$ is an FPC-term-in-context and $|\Gamma| \vdash M$ is a $\lambda$+Error-term-in-context. We say $|\Gamma| \vdash M$ realizes $\Gamma \vdash t{:}\tau$ and write

$$M \Vdash_\Gamma t \qquad \text{iff} \qquad |\Gamma| \vdash \mathbf{p}_\tau \llparenthesis M \rrparenthesis =_{\text{obs}} t\left[\mathbf{p}_{\sigma_i}\, x_i \,\big/\, x_i\right],$$

i. e. iff Diagram 4.1 in $\mathbb{D}$ commutes. In the sequel we will use $t^U$ to denote the term $t\left[\mathbf{p}_{\sigma_i}\, x_i \,\big/\, x_i\right]$ and we will write $M \Vdash_\Gamma t{:}\tau$ to indicate that $\Gamma \vdash t{:}\tau$ and $M \Vdash_\Gamma t$. Note that $t^U$ satisfies the type judgement $|\Gamma| \vdash t^U{:}\tau$. $\diamond$

**4.2 Example** For the trivial case $\Gamma \vdash x_i{:}\sigma_i$ of a single variable we get $x_i \Vdash_\Gamma x_i$, since $|\Gamma| \vdash x_i^U = \mathbf{p}_{\sigma_i}\, x_i = \mathbf{p}_{\sigma_i} \llparenthesis M \rrparenthesis$ for $M :\equiv x_i$.

If $M$ is a closed $\lambda$+Error-term and $t{:}\tau$ is a closed FPC-term then $t = t^U$ and therefore $M \Vdash t$ iff $p_\tau \llparenthesis M \rrparenthesis = t$.

A nontrivial example can be found in Example 4.4 below.

The following lemma describes how the term forming operations from FPC can be implemented in $\lambda$+Error. Note that functional application in FPC cannot simply be implemented as functional application in $\lambda$+Error; rather it involves a certain retraction. As we shall see below, if $M \Vdash_\Gamma s : \varrho{\to}\tau$ and $N \Vdash_\Gamma t{:}\varrho$, then in general we cannot conclude $MN \Vdash_\Gamma st$. However, if the canonical retraction $\mathbf{r}_\varrho$ is realized by a closed $\lambda$+Error-term $\mathbf{R}_\varrho$, i. e. $\mathbf{R}_\varrho \Vdash \mathbf{r}_\varrho : U{\to}U$, then $M(\mathbf{R}_\varrho N) \Vdash_\Gamma st$.

**4.3 Lemma** *Let $\Gamma \equiv x_1{:}\sigma_1, \dots, x_n{:}\sigma_n$ be an FPC-context, let $m$ be a natural number and let $\varrho$, $\varrho_0$, ... , $\varrho_m$ and $\tau$ be closed FPC-types. Further let $M$, $M_0$, ... , $M_m$ and $N$ be $\lambda$+Error-terms-in-context $|\Gamma|$ and let $s$, $t$, $t_0$, ... , $t_m$ be FPC-term-in-context $\Gamma$ (unless explicitly stated otherwise).*

*i)* *Realizability is preserved under extension and retraction of the context: if $M \Vdash_\Gamma t{:}\tau$ and $\Gamma' \vdash t{:}\tau$, then $M \Vdash_{\Gamma'} t$.*

*ii)* *If $M \Vdash_\Gamma t{:}U$, then $M \Vdash_\Gamma \mathbf{p}_\tau\, t : \tau$.*

*iii)* *If $\mathbf{R}_\tau \Vdash \mathbf{r}_\tau : U{\to}U$ and $M \Vdash_\Gamma t{:}\tau$, then $\mathbf{R}_\tau\, M \Vdash_\Gamma \mathbf{e}_\tau\, t{:}U$.*

*iv)* *If we define $\mathbf{R}_{U\to U} :\equiv \lambda x.\, \lambda y.\, xy$, then $\mathbf{R}_{U\to U} \Vdash \mathbf{r}_{U\to U} : U{\to}U$.*

*v)* *Assume $M \Vdash_\Gamma s : \varrho{\to}\tau$ and $N \Vdash_\Gamma t{:}\varrho$. If $\mathbf{R}_\varrho \Vdash \mathbf{r}_\varrho : U{\to}U$, then we get $M(\mathbf{R}_\varrho\, N) \Vdash_\Gamma st{:}\tau$.*

*vi)* *If $\Gamma' \equiv \Gamma, x{:}\varrho$ and $M \Vdash_{\Gamma'} t{:}\tau$, then $\lambda x.\, M \Vdash_\Gamma \lambda x{:}\varrho.\, t : \varrho{\to}\tau$.*

*vii)* *Folding and unfolding of recursive types are realized as follows: if $\alpha \vdash \sigma$ is an FPC-type-in-context and $M \Vdash_\Gamma s{:}\, \mu\alpha.\, \sigma$ and $N \Vdash_{\Gamma'} t : \sigma\,[\mu\alpha.\, \sigma \,/\, \alpha]$, then*

$$\mathbf{R}_{\sigma^U}\, M \Vdash_\Gamma \mathbf{unfold}(s) : \sigma\,[\mu\alpha.\, \sigma \,/\, \alpha] \qquad \textit{and}$$
$$\mathbf{R}_{\sigma^U}\left(\mathbf{R}_{\sigma^{\vec{\tau}}}\, N\right) \Vdash_{\Gamma'} \mathbf{fold}(t) : \mu\alpha.\, \sigma.$$

*viii)* *The arithmetic operations in FPC defined in Definition 3.14 are realized by those of λ+Error:*

$$\overline{0} \Vdash \underline{0} : U \qquad \overline{n+1} \Vdash \underline{n+1} : U$$
$$\mathsf{SUCC} \Vdash \mathbf{succ} : U{\to}U \qquad \mathsf{PRED} \Vdash \mathbf{pred} : U{\to}U$$
$$\mathsf{FST} \Vdash \mathbf{fst} : U{\to}U \qquad \mathsf{SND} \Vdash \mathbf{snd} : U{\to}U$$
$$\mathsf{PAIR} \Vdash \mathbf{pair} : U{\to}U{\to}U \qquad \mathsf{IFR} \Vdash \mathbf{ifz} : U{\to}U{\to}U{\to}U$$

*For any natural number $n$ and any closed FPC-terms $t$, $t_0 \,\ldots\,, t_n$ and closed λ+Error-terms $M$, $M_0, \ldots\,, M_n$ such that $M \Vdash t{:}U$ and $M_i \Vdash t_i{:}U$ for all $i \in \{0, \ldots .n\}$ we get*

$$\mathsf{CASE}_n\, M\, \mathsf{OF}\, M_0 \,[]\, \ldots \,[]\, M_n \quad \Vdash \quad \mathbf{case}_n\, t\, \mathbf{of}\, t_0 \,[]\, \ldots \,[]\, t_n : U$$

*ix)* $\mathsf{ID} \Vdash_\Gamma \mathbf{id}_\tau : \tau$

*x)* *If $M \Vdash_\Gamma s{:}\varrho{\times}\tau$ then $\mathsf{FST}\, M \Vdash_\Gamma \mathbf{pl}(s)$ and $\mathsf{SND}\, M \Vdash_\Gamma \mathbf{pr}(s)$.*

*xi)* *If $M \Vdash_\Gamma s{:}\varrho$ and $N \Vdash_\Gamma t{:}\tau$ then $\mathsf{PAIR}\; M\, N \Vdash_\Gamma \langle s, t\rangle$.*

*xii)* *The λ+Error-constructs for case analysis realize the corresponding FPC-constructs: if $M \Vdash_\Gamma t{:}\varrho_i$ for some $i \in \{0, \ldots, m\}$, then*

$$\mathsf{PAIR}\; \overline{i}\, M \Vdash_\Gamma \mathbf{in}_i\, t : \sum_{j=0}^{m} \varrho_j.$$

*xiii)* *Assume* $M_j \Vdash_\Gamma t_j : \varrho_j \to \tau$ *and* $\mathbf{R}_{\varrho_j} \Vdash \mathbf{r}_{\varrho_j}$, *for any* $j \in \{0, \ldots, m\}$. *Then, for* $\varrho :\equiv \sum_{j=0}^{m} \varrho_j$ *and* $\Gamma' :\equiv \Gamma, y{:}\varrho$, *we get*

$$\begin{aligned} &\mathsf{CASE}_m\,\mathsf{FST}\,y\,\mathsf{OF}\,M_0\big(\mathbf{R}_{\varrho_0}(\mathsf{SND}(y))\big)\,[]\ldots[] \\ &\qquad\qquad\qquad M_m\big(\mathbf{R}_{\varrho_m}(\mathsf{SND}\,y)\big) \\ &\Vdash_{\Gamma'} \mathbf{case}\,y\,\mathbf{of}\,\mathbf{in}_0 w \Rightarrow t_0 w\,[]\ldots[]\,\mathbf{in}_m w \Rightarrow t_m w. \end{aligned}$$

*xiv)* $\mathsf{FIX} \Vdash \mathbf{Y}_U : [U \to U] \to U$

*xv)* $\Omega \Vdash \Omega_\tau$

*xvi)* $\mathsf{FIX} \Vdash \mathbf{Y}_\tau : [\tau \to \tau] \to \tau$

The proof of this lemma can be found at the end of this section. To show how to apply this lemma we discuss the following example in detail. We will use this result in the proof of Theorem 4.7 to implement the retraction $\mathbf{r}_{\sigma\to\tau}$ of a function type in $\lambda$+Error.

**4.4 Example** Let $\Gamma \equiv x{:}U, y{:}U$. If $R_1 \Vdash r_1 : U \to U$ and $R_2 \Vdash r_2 : U \to U$, then

$$\lambda x, y.\, R_2\big(x(R_1 y)\big) \Vdash \lambda x{:}U.\, \mathbf{e}_{U\to U}\Big(\lambda y{:}U.\, r_2\big(\mathbf{p}_{U\to U}\, x(r_1 y)\big)\Big) : U \to U.$$

This can be proved as follows: From Lemma 4.3(*ii*) we know

$$x \Vdash_\Gamma \mathbf{p}_{U\to U}\, x : U \to U.$$

Applying Lemma 4.3($v$) twice yields, since $\mathsf{ID} \Vdash \mathbf{r}_U : U \to U$,

$$\begin{aligned} x(R_1 y) &\Vdash_\Gamma \mathbf{p}_{U\to U}\, x(r_1 y) : U \quad \text{and} \\ R_2\big(x(R_1 y)\big) &\Vdash_\Gamma r_2\big(\mathbf{p}_{U\to U}\, x(r_1 y)\big) : U. \end{aligned}$$

Now we can apply Lemma 4.3(*vi*) to get

$$\lambda y.\, R_2\big(x(R_1 y)\big) \Vdash_\Gamma \lambda y{:}U.\, r_2\big(\mathbf{p}_{U\to U}\, x(r_1 y)\big) : U \to U.$$

From Lemma 4.3(*iii*) we conclude

$$\lambda y.\, R_2\big(x(R_1 y)\big) \Vdash_\Gamma \mathbf{e}_{U\to U}\Big(\lambda y{:}U.\, r_2\big(\mathbf{p}_{U\to U}\, x(r_1 y)\big)\Big) : U.$$

Finally, application of Lemma 4.3(*vi*) yields the conclusion.

*Proof of Lemma 4.3:*

*i)* This is obvious.

*ii)* $M \Vdash_\Gamma t{:}U$ implies $|\Gamma| \vdash (\!|M|\!) = t^U$, since $\mathbf{p}_U = \mathsf{id}_U$.
Hence $|\Gamma| \vdash (\mathbf{p}_\tau\, t)^U = \mathbf{p}_\tau\, t^U = \mathbf{p}_\tau (\!|M|\!)$.

*iii)* We conclude from $M \Vdash_\Gamma t{:}\tau$ that $|\Gamma| \vdash \mathbf{e}_\tau\, t^U = \mathbf{r}_\tau (\!| M |\!)$. Since $|\Gamma| \vdash (\!| \mathbf{R}_\tau\, M |\!) = \mathbf{p}_{U\to U} (\!| \mathbf{R}_\tau |\!) (\!| M |\!) = \mathbf{r}_\tau (\!| M |\!)$ we obtain $|\Gamma| \vdash (\!| \mathbf{R}_\tau\, M |\!) = \mathbf{e}_\tau\, t^U$.

*iv)* $$\begin{aligned} \mathbf{p}_{U\to U} (\!| \mathbf{R}_{U\to U} |\!) &= \lambda x{:}U.\, \mathbf{e}_{U\to U} (\lambda y{:}U.\, \mathbf{p}_{U\to U}\, xy) \\ &= \lambda x{:}U.\, \mathbf{e}_{U\to U} (\mathbf{p}_{U\to U}\, x) = \mathbf{r}_{U\to U}\,. \end{aligned}$$

*v)* Since $\mathbf{R}_\varrho \Vdash \mathbf{r}_\varrho : U{\to}U$, we have $|\Gamma| \vdash (\!| \mathbf{R}_\varrho\, N |\!) = \mathbf{r}_\varrho (\!| N |\!)$. According to the definition of $\mathbf{p}_{\varrho\to\tau}$ in the proof of Lemma 3.4 we obtain

$$\begin{aligned} |\Gamma| \vdash (st)^U = s^U t^U &= \mathbf{p}_{\varrho\to\tau} (\!| M |\!) \big(\mathbf{p}_\varrho (\!| N |\!)\big) \\ &= \mathbf{p}_\tau \Big(\mathbf{p}_{U\to U} (\!| M |\!) \big(\mathbf{e}_\varrho (\mathbf{p}_\varrho (\!| N |\!))\big)\Big) \\ &= \mathbf{p}_\tau \Big(\mathbf{p}_{U\to U} (\!| M |\!) \big(\mathbf{r}_\varrho (\!| N |\!)\big)\Big) \\ &= \mathbf{p}_\tau \big(\mathbf{p}_{U\to U} (\!| M |\!) (\!| \mathbf{R}_\varrho\, N |\!)\big) = \mathbf{p}_\tau \big(\!| M (\mathbf{R}_\varrho\, N) |\!\big) \end{aligned}$$

*vi)* By hypothesis we have $|\Gamma|, x{:}U \vdash \mathbf{p}_\tau (\!| M |\!) = t\, [\mathbf{p}_{\sigma_i}\, x_i \,/\, x_i, \mathbf{p}_\sigma \,/ x]$ and hence

$$\begin{array}{rrcll} |\Gamma|, x{:}\sigma \vdash & \mathbf{p}_\tau (\!| M |\!)\, [\mathbf{e}_\sigma\, x \,/\, x] & = & t\, [\mathbf{p}_{\sigma_i}\, x_i \,/\, x_i] & \text{and} \\ |\Gamma| \vdash & \lambda x{:}\sigma.\, \mathbf{p}_\tau (\!| M |\!)\, [\mathbf{e}_\sigma\, x \,/\, x] & = & \lambda x{:}\sigma.\, t\, [\mathbf{p}_{\sigma_i}\, x_i \,/\, x_i]\,. & \end{array}$$

We have to show $|\Gamma| \vdash \mathbf{p}_{\sigma\to\tau} (\!| \lambda x.\, M |\!) = \lambda x{:}\sigma.\, t\, [\mathbf{p}_{\sigma_i}\, x_i \,/\, x_i]$. By definition, $|\Gamma| \vdash \mathbf{p}_{U\to U} (\!| \lambda x.\, M |\!) = \mathbf{p}_{U\to U} \big(\mathbf{e}_{U\to U}\, \lambda x{:}U. (\!| M |\!)\big) = \lambda x{:}U. (\!| M |\!)$. Hence

$$\begin{aligned} |\Gamma| \vdash \mathbf{p}_{\sigma\to\tau} (\!| \lambda x.\, M |\!) &= \lambda x{:}\sigma.\, \mathbf{p}_\tau \big(\mathbf{p}_{U\to U} (\!| \lambda x.\, M |\!) (\mathbf{e}_\sigma\, x)\big) \\ &= \lambda x{:}\sigma.\, \mathbf{p}_\tau \big(\lambda x{:}U. (\!| M |\!) (\mathbf{e}_\sigma\, x)\big) \\ &= \lambda x{:}\sigma.\, \mathbf{p}_\tau (\!| M |\!)\, [\mathbf{e}_\sigma\, x \,/\, x] \\ &= \lambda x{:}\sigma.\, t\, [\mathbf{p}_{\sigma_i}\, x_i \,/\, x_i] \end{aligned}$$

*vii)* This is an immediate consequence of Corollary 3.20.

*viii)* These relations are immediate consequences of Definition 3.14.

*ix)* This is trivial.

*x)* $M \Vdash_\Gamma s{:}\varrho{\times}\tau$ implies

$$|\Gamma| \vdash s^U = \mathbf{p}_{\varrho\times\tau} (\!| M |\!) = \big\langle \mathbf{p}_\varrho \big(\mathbf{fst} (\!| M |\!)\big), \mathbf{p}_\tau \big(\mathbf{snd} (\!| M |\!)\big) \big\rangle.$$

It follows $|\Gamma| \vdash \mathbf{pl}(s^U) = \mathbf{p}_\varrho \big(\mathbf{fst} (\!| M |\!)\big)$, hence $\mathsf{FST}\, M \Vdash_\Gamma \mathbf{pl}(s)$ and the same for $\mathsf{SND}\, M$.

*xi)* $|\Gamma| \vdash \mathbf{p}_{\varrho\times\tau} (\!| \mathsf{PAIR}\; M\, N |\!) = \langle \mathbf{p}_\varrho (\!| M |\!), \mathbf{p}_\tau (\!| N |\!) \rangle = \big\langle s^U, t^U \big\rangle$

*xii)* From $|\Gamma| \vdash (\!|\mathsf{PAIR}\ \bar{i}\, M|\!) = \mathbf{pair}\ \underline{i}\,(\!|M|\!)$ we obtain

$$|\Gamma| \vdash \mathbf{p}_{\sum_{j=0}^{m}\tau_j}(\!|\mathsf{PAIR}\ \bar{i}\, M|\!) = \mathbf{in}_i\big(\mathbf{p}_{\tau_i}(\!|M|\!)\big) = \mathbf{in}_i t^U.$$

*xiii)* The proof is by induction on $m$.
Assume $m = 0$ and $\Gamma \vdash M_0 \Vdash_\Gamma t_0 : \varrho_0{\to}\tau$. Writing $\varrho := \sum_{j=0}^{0} \varrho_j$ and $H :\equiv \mathsf{IFR}\big(\mathsf{FST}\, y, M_0\big(\mathbf{R}_{\varrho_0}(\mathsf{SND}\, y)\big), \Omega\big)$ and $h :\equiv \mathbf{case}^{\varrho_0,\tau}\ \mathbf{p}_\varrho\, y\, \mathbf{of}$ $\mathbf{in}_0 w \Rightarrow t_0^U\, w : \tau$, we have to show

$$|\Gamma'| \vdash \mathbf{p}_\tau(\!|H|\!) = h.$$

By definition,

$$\begin{aligned}
|\Gamma'| \vdash \mathbf{p}_\tau(\!|H|\!) = \mathbf{p}_\tau\Big(&\ \mathbf{case}^{\mathbf{void},U\to U,U}\ \mathbf{fst}\, y\ \mathbf{of}\\
&\quad \mathbf{in}_0 w \Rightarrow \mathbf{p}_{U\to U}(\!|M_0|\!)\big(\mathbf{r}_{\varrho_0}(\mathbf{snd}\, y)\big) \,\square\, \mathbf{in}_1 w \Rightarrow \Omega_U\Big)\\
= &\ \mathbf{case}^{\mathbf{void},U\to U,U}\ \mathbf{fst}\, y\ \mathbf{of}\\
&\ \mathbf{in}_0 w \Rightarrow \mathbf{p}_\tau\Big(\mathbf{p}_{U\to U}(\!|M_0|\!)\big(\mathbf{e}_{\varrho_0}(\mathbf{p}_{\varrho_0}(\mathbf{snd}\, y))\big)\Big) \,\square\\
&\ \mathbf{in}_1 w \Rightarrow \Omega_\tau\\
= &\ \mathbf{case}^{\mathbf{void},U\to U,U}\ \mathbf{fst}\, y\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow t_0^U\big(\mathbf{p}_{\varrho_0}(\mathbf{snd}\, y)\big) \,\square\\
&\qquad\qquad \mathbf{in}_1 w \Rightarrow \Omega_\tau
\end{aligned}$$

On the other hand,

$$\begin{aligned}
|\Gamma'| \vdash \mathbf{p}_\varrho\, y = &\ \mathbf{case}^{\mathbf{void},U\to U,\varrho}\ \mathbf{fst}\, y\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow \mathbf{in}_0^{\varrho_0}\big(\mathbf{p}_{\varrho_0}(\mathbf{snd}\, y)\big) \,\square\\
&\qquad \mathbf{in}_1 w \Rightarrow \Omega_\varrho.
\end{aligned}$$

and hence

$$\begin{aligned}
|\Gamma'| \vdash h = &\ \mathbf{case}^{\mathbf{void},U\to U,\tau}\ \mathbf{fst}\, y\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow t_0^U\big(\mathbf{p}_{\varrho_0}(\mathbf{snd}\, y)\big) \,\square\\
&\qquad \mathbf{in}_1 w \Rightarrow \Omega_\tau.
\end{aligned}$$

We leave it to the reader to verify the inductive step, which is quite similar.

*xiv)* Let $\Gamma :\equiv f{:}U{\to}U$ and $\Gamma' :\equiv \Gamma, x{:}U$. Define $K :\equiv \lambda x.\, f(xx)$ in $\lambda$+Error and $k :\equiv \lambda x{:}U.\, f(\mathbf{p}_{U\to U}\, xx)$ in FPC. Then $\Gamma \vdash k : U{\to}U$ is an FPC-term-in-context and the term $\mathbf{fix} : [U{\to}U]{\to}U$ introduced in Lemma 3.21 is just $\mathbf{fix} \equiv \lambda f{:}U{\to}U.\, k(\mathbf{e}_{U\to U}\, k)$. We conclude

$$\begin{array}{rl}
x \Vdash_{\Gamma'} x{:}U & \text{from Example 4.2,} \\
x \Vdash_{\Gamma'} \mathbf{p}_{U\to U}\, x : U{\to}U & \text{from Lemma 4.3}(ii), \\
xx \Vdash_{\Gamma'} \mathbf{p}_{U\to U}\, xx : U & \text{from Lemma 4.3}(v), \\
f(xx) \Vdash_{\Gamma'} f(\mathbf{p}_{U\to U}\, xx) : U & \text{from Lemma 4.3}(v), \\
K \Vdash_{\Gamma} k : U{\to}U & \text{from Lemma 4.3}(vi), \\
\mathbf{R}_{U\to U}\, K \Vdash_{\Gamma} \mathbf{e}_{U\to U}\, k : U & \text{from Lemma 4.3}(iii), \\
\lambda y.\, Ky \Vdash_{\Gamma} \mathbf{e}_{U\to U}\, k : U & \text{from Lemma 4.3}(iv), \\
 & \text{since } \Gamma \vdash (\!|\mathbf{R}_{U\to U}\, K|\!) = (\!|\lambda y.\, Ky|\!) \\
K(\lambda y.\, Ky) \Vdash_{\Gamma} k(\mathbf{e}_{U\to U}\, k) : U & \text{from Lemma 4.3}(v), \\
\mathsf{FIX} \Vdash \mathbf{fix} : [U{\to}U]{\to}U & \text{from Lemma 4.3}(vi).
\end{array}$$

Lemma 3.21 shows $\mathbf{fix} =_{\text{obs}} \mathbf{Y}_U$. Hence FIX also realizes $\mathbf{Y}_U$.

*xv)* This is obvious.

*xvi)* We have to show

$$\mathbf{Y}_\tau = \lambda f : \tau{\to}\tau.\, \mathbf{p}_\tau\Big(\mathbf{p}_{U\to U}(\!|\mathsf{FIX}|\!)\big(\mathbf{e}_{U\to U}\, \lambda x{:}U.\, \mathbf{e}_\tau(f(\mathbf{p}_\tau\, x))\big)\Big).$$

According to Lemma 4.3(*xiv*),

$$\mathbf{Y}_U = \lambda g : U{\to}U.\, \mathbf{p}_{U\to U}(\!|\mathsf{FIX}|\!)(\mathbf{e}_{U\to U}\, g).$$

Hence it suffices to show

$$\mathbf{Y}_\tau =_{\text{obs}} \lambda f{:}\tau{\to}\tau.\Big(\mathbf{Y}_U\big(\lambda x{:}U.\, \mathbf{e}_\tau(f(\mathbf{p}_\tau\, x))\big)\Big).$$

Similarly to Lemma 3.21 we may consider the SD-interpretations of these terms. For these we have to show

$$\mathbf{Y}_\tau\, f = \mathbf{Y}_U\big(x \mapsto \mathbf{e}_\tau \circ f \circ \mathbf{p}_\tau(x)\big)$$

for any morphism $f\colon \tau \to \tau$. Since $\mathbf{Y}_\tau\, f = \bigvee_n f^n \bot_\tau$ and the same for $\mathbf{Y}_U$, the observation

$$\mathbf{p}_\tau\Big(\big(x \mapsto \mathbf{e}_\tau \circ f \circ \mathbf{p}_\tau(x)\big)^n \bot_U\Big) = (\mathbf{p}_\tau \circ \mathbf{e}_\tau \circ f)^n(\mathbf{p}_\tau\, \bot_U) = f^n \bot_\tau$$

for any $n \in \mathbb{N}$ proves the claim.

□

## 4.2 Representability of retractions

As we have seen in the previous section, realizability of retractions is necessary to implement FPC functional abstraction in $\lambda$+Error. Hence this section aims at proving the following theorem:

**4.5 Theorem** *For any closed type $\sigma$, the canonical retraction $\mathbf{r}_\sigma : U{\to}U$ can be implemented in $\lambda$+Error, i. e. there is a closed $\lambda$+Error-term $\mathbf{R}_\sigma$ such that $\mathbf{R}_\sigma \Vdash \mathbf{r}_\sigma$.*

According to Lemma 3.4, the definition of the canonical retraction pairs of recursive types involves the FPC-implementation of the corresponding syntactic functors. Thus to implement the retractions in $\lambda$+Error we hence have to implement the corresponding syntactic functors as well. Hence we define an FPC-type-in-context to be representable in $\lambda$+Error iff both its canonical retraction and the FPC-implementation of the corresponding syntactic functor are implementable in $\lambda$+Error. Then we prove the above theorem by showing that any FPC-type-in-context is representable.

**4.6 Definition (representable types)** Let $\Theta :\equiv \alpha_1, \ldots, \alpha_n$ be a type context and let $\vec{\tau} := \tau_1, \ldots, \tau_n$ be a sequence of closed types with $\lambda$+Error-implementable retractions, i. e. for any $i \in \{1, \ldots, n\}$, there exists a closed $\lambda$+Error-term $\mathbf{R}_{\tau_i}$ satisfying $\mathbf{R}_{\tau_i} \Vdash \mathbf{r}_{\tau_i} : U{\to}U$.

Write $\sigma^{\vec{\tau}}$ for the closed type $\sigma\big[\tau_i \,/\, \alpha_i\big]$ and let $F_{\Theta\vdash\sigma} \colon (\mathbb{D}^{\mathrm{op}} \times \mathbb{D})^n \to \mathbb{D}$ denote the syntactic functor corresponding to the type $\Theta \vdash \sigma$. For any $i \in \{1, \ldots, n\}$ let $\pi_i := [\tau_i{\to}\tau_i]{\times}[\tau_i{\to}\tau_i]$ and let

$$t^{\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i \to \Big[\sigma^{\vec{\tau}}{\to}\sigma^{\vec{\tau}}\Big]$$

denote a closed FPC-term according to Proposition 2.41 such that, for all closed terms $k^-$, $k^+ : \tau_i{\to}\tau_i$, we get

$$F_{\Theta\vdash\sigma}\left(\mathsf{id}, \ldots, \mathsf{id}, \left[k^-\right]_{\mathrm{obs}}, \left[k^+\right]_{\mathrm{obs}}, \mathsf{id}, \ldots, \mathsf{id}\right) = \left[t^{\vec{\tau},i}_{\Theta\vdash\sigma} k^- k^+\right]_{\mathrm{obs}}.$$

Under these assumptions we say that the type-in-context $\Theta \vdash \sigma$ *has a $\lambda$+Error representation* iff, for any such sequence $\vec{\tau}$ and any $i \in \{1, \ldots, n\}$, there are closed $\lambda$+Error-terms $\mathbf{R}_{\sigma^{\vec{\tau}}}$ and $M^{\vec{\tau},i}_{\Theta\vdash\sigma}$ such that

$$\mathbf{R}_{\sigma^{\vec{\tau}}} \Vdash \mathbf{r}_{\sigma^{\vec{\tau}}} : U{\to}U \quad \text{and} \quad M^{\vec{\tau},i}_{\Theta\vdash\sigma} \Vdash t^{\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i \to [\sigma^{\vec{\tau}}{\to}\sigma^{\vec{\tau}}].$$

**4.7 Theorem** *Any FPC-type-in-context has a $\lambda$+Error representation.*

This theorem is proved by induction over the type structure of FPC.

*Proof:* Let $\Theta :\equiv \alpha_1, \ldots, \alpha_n$ be a type context and let $\vec{\tau} := \tau_1, \ldots, \tau_n$ be a sequence of closed types with $\lambda$+Error-implementable retractions.

*Case 1:* The FPC-type $\Theta \vdash \mathbf{void}$ has a $\lambda$+Error representation.

By definition, $\mathbf{void}^{\vec{\tau}} \equiv \mathbf{void}$ and by Proposition 2.41 we have

$$\mathbf{r}_{\mathbf{void}} = \Omega_{U\to U} \quad \text{and} \quad t^{\vec{\tau},i}_{\Theta\vdash\mathbf{void}} = \Omega_{\pi_i\to[\mathbf{void}\to\mathbf{void}]}$$

We put $\mathbf{R}_{\mathbf{void}} :\equiv M^{\vec{\tau},i}_{\Theta\vdash\mathbf{void}} :\equiv \Omega$. Since $\Omega \Vdash \Omega_{\pi_i\to[\mathbf{void}\to\mathbf{void}]}$ by Lemma 4.3($xv$), we conclude

$$\mathbf{r}_{\mathbf{void}} = \mathbf{p}_{U\to U}(\!|\mathbf{R}_{\mathbf{void}}|\!) \quad \text{and} \quad t^{\vec{\tau},i}_{\Theta\vdash\mathbf{void}} = \mathbf{p}_{\pi_i\to[\sigma^{\vec{\tau}}\to\sigma^{\vec{\tau}}]}(\!|M^{\vec{\tau},i}_{\Theta\vdash\mathbf{void}}|\!).$$

*Case 2:* For any $j \in \{1, \ldots, n\}$, the FPC-type $\Theta \vdash \alpha_j$ has a $\lambda$+Error representation.

For $\sigma :\equiv \alpha_j$ we get $\sigma^{\vec{\tau}} \equiv \tau_j$ which, by hypotheses, has a $\lambda$+Error-implementable retraction. Hence $\mathbf{R}_{\sigma^{\vec{\tau}}} :\equiv \mathbf{R}_{\tau_j} \Vdash \mathbf{r}_{\sigma^{\vec{\tau}}}$.

Further $F_{\Theta\vdash\sigma}\left(f_1^-, f_1^+, \ldots, f_n^-, f_n^+\right) = f_j^+$. Hence, for $i \in \{1, \ldots, n\}$, we have $t^{\vec{\tau},i}_{\Theta\vdash\alpha_j} = \mathbf{id}_{\tau_j}$, for $i \neq j$, and $t^{\vec{\tau},j}_{\Theta\vdash\alpha_j} = \lambda x : \pi_j.\, \mathbf{pr}\, x$, which, as an immediate consequence of Lemma 4.3($ix$) and ($x$), are implemented in $\lambda$+Error by $M^{\vec{\tau},i}_{\Theta\vdash\alpha_j} :\equiv \mathsf{ID}$ and $M^{\vec{\tau},j}_{\Theta\vdash\alpha_j} :\equiv \mathsf{SND}$.

*Case 3:* If FPC-types $\Theta \vdash \sigma_1$ and $\Theta \vdash \sigma_2$ have $\lambda$+Error representations, so does $\Theta \vdash \sigma_1{\times}\sigma_2$.

Let $\sigma :\equiv \sigma_1{\times}\sigma_2$. Since $\sigma^{\vec{\tau}} \equiv \sigma_1^{\vec{\tau}}{\times}\sigma_2^{\vec{\tau}}$, we get from Lemmas 3.4 and 3.15 and from Proposition 2.41:

$$\begin{aligned}\mathbf{r}_{\sigma^{\vec{\tau}}} &= \lambda x{:}U.\, \mathbf{pair}\left(\mathbf{r}_{\sigma_1^{\vec{\tau}}}(\mathbf{fst}\, x)\right)\left(\mathbf{r}_{\sigma_2^{\vec{\tau}}}(\mathbf{snd}\, x)\right)\\ t^{\vec{\tau},i}_{\Theta\vdash\sigma} &= \lambda x{:}\pi_i.\, \lambda y{:}\sigma^{\vec{\tau}}.\left\langle t^{\vec{\tau},i}_{\Theta\vdash\sigma_1} x\big(\mathbf{pl}(y)\big), t^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x\big(\mathbf{pr}(y)\big)\right\rangle\end{aligned}$$

We choose

$$\begin{aligned}\mathbf{R}_{\sigma^{\vec{\tau}}} &:\equiv \lambda x.\, \mathsf{PAIR}\left(\mathbf{R}_{\sigma_1^{\vec{\tau}}}(\mathsf{FST}\, x)\right)\left(\mathbf{R}_{\sigma_2^{\vec{\tau}}}(\mathsf{SND}\, x)\right)\\ M^{\vec{\tau},i}_{\Theta\vdash\sigma} &:\equiv \lambda x.\, \lambda y.\, \mathsf{PAIR}\left(\mathbf{R}_{\sigma_1^{\vec{\tau}}}\big(M^{\vec{\tau},i}_{\Theta\vdash\sigma_1} x(\mathsf{FST}\, y)\big)\right)\left(\mathbf{R}_{\sigma_2^{\vec{\tau}}}\big(M^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x(\mathsf{SND}\, y)\big)\right).\end{aligned}$$

Applying Lemma 4.3($viii$), ($v$) and ($vi$) we obtain $\vdash \mathbf{R}_{\sigma^{\vec{\tau}}} \Vdash \mathbf{r}_{\sigma^{\vec{\tau}}} : U{\to}U$.

Let $\Gamma :\equiv x{:}\pi_i, y{:}\sigma^{\vec{\tau}}$. Applying Lemma 4.3($x$) and ($v$) to the hypothesis, we obtain

$$\begin{aligned}M^{\vec{\tau},i}_{\Theta\vdash\sigma_1} x(\mathsf{FST}\, y) &\Vdash_\Gamma t^{\vec{\tau},i}_{\Theta\vdash\sigma_1} x\big(\mathbf{pl}(y)\big) : \sigma_1^{\vec{\tau}}\\ M^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x(\mathsf{SND}\, y) &\Vdash_\Gamma t^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x\big(\mathbf{pr}(y)\big) : \sigma_2^{\vec{\tau}}\end{aligned}$$

Application of Lemma 4.3($xi$), ($v$) and ($vi$) completes the proof.

*Case 4:* If FPC-types $\Theta \vdash \sigma_1$ and $\Theta \vdash \sigma_2$ have $\lambda$+Error representations, so does $\Theta \vdash [\sigma_1 \to \sigma_2]$.

Putting $\sigma :\equiv \sigma_1 \to \sigma_2$, we get $\sigma^{\vec{\tau}} \equiv [\sigma_1^{\vec{\tau}} \to \sigma_2^{\vec{\tau}}]$. Lemmas 3.4 and 3.15 and Proposition 2.41 yield

$$\begin{aligned}\mathbf{r}_{\sigma^{\vec{\tau}}} &= \lambda x{:}U.\, \mathbf{e}_{U\to U}\Big(\lambda y{:}U.\, \mathbf{r}_{\sigma_2^{\vec{\tau}}}\big(\mathbf{p}_{U\to U}\, x(\mathbf{r}_{\sigma_1^{\vec{\tau}}}\, y)\big)\Big)\\ t^{\vec{\tau},i}_{\Theta\vdash\sigma_1\to\sigma_2} &= \lambda x{:}\pi_i.\, \lambda y : [\sigma_1^{\vec{\tau}} \to \sigma_2^{\vec{\tau}}].\, \lambda z{:}\sigma_1^{\vec{\tau}}.\, t^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x\Big(y\big(t^{\vec{\tau},i}_{\Theta\vdash\sigma_1}\langle \mathbf{pr}(x), \mathbf{pl}(x)\rangle z\big)\Big).\end{aligned}$$

According to Example 4.4, $\mathbf{r}_{\sigma^{\vec{\tau}}}$ is realized by

$$\mathbf{R}_{\sigma^{\vec{\tau}}} :\equiv \lambda x, y.\, \mathbf{R}_{\sigma_2^{\vec{\tau}}}\big(x(\mathbf{R}_{\sigma_1^{\vec{\tau}}}\, y)\big)$$

By Case 3 it follows that, for $\pi :\equiv [\tau_i \to \tau_i] \times [\tau_i \to \tau_i]$, there is a closed $\lambda$+Error-term $\mathbf{R}_{\pi_i}$ such that $\mathbf{R}_{\pi_i} \Vdash r_{\pi_i}$. Letting $\Gamma :\equiv x{:}\pi_i, z{:}\sigma_1^{\vec{\tau}}$ we thus obtain, according to Lemma 4.3($x$) and ($xi$) and ($v$),

$$\begin{aligned}H :\equiv M^{\vec{\tau},i}_{\Theta\vdash\sigma_1}\Big(\mathbf{R}_{\pi_i}\Big(\mathsf{PAIR}\,(\mathsf{SND}\, x)\,(\mathsf{FST}\, x)(\mathbf{R}_{\sigma_1^{\vec{\tau}}}\, z)\Big)\Big)&\\ \Vdash_\Gamma t^{\vec{\tau},i}_{\Theta\vdash\sigma_1}\langle \mathbf{pr}(x), \mathbf{pl}(x)\rangle z&\end{aligned}$$

Lemma 4.3($v$) and ($vi$) now yields

$$M^{\vec{\tau},i}_{\Theta\vdash\sigma_1\to\sigma_2} :\equiv \lambda x, y, z.\, M^{\vec{\tau},i}_{\Theta\vdash\sigma_2} x\Big(\mathbf{R}_{\sigma_2^{\vec{\tau}}}\Big(y\Big(\mathbf{R}_{\sigma_1^{\vec{\tau}}}\, H\Big)\Big)\Big) \Vdash t^{\vec{\tau},i}_{\Theta\vdash\sigma_1\to\sigma_2}.$$

*Case 5:* Let $m \in \mathbb{N}$. If FPC-types $\Theta \vdash \sigma_j$, for $j \in \{0, \dots, m\}$, have $\lambda$+Error representations, so does $\Theta \vdash \sum_{j=0}^m \sigma_j$.

For $\sigma := \sum_{j=0}^m \sigma_j$ we have $\sigma^{\vec{\tau}} = \sum_{j=0}^m \sigma_j^{\vec{\tau}}$. Hence we get, according to Lemmas 3.4 and 3.15 and Proposition 2.41,

$$\begin{aligned}\mathbf{r}_{\sigma^{\vec{\tau}}} &= \lambda x{:}U.\, \mathbf{case}_m\, \mathbf{fst}\, x\ \mathbf{of}\ \mathbf{pair}\ \underline{0}\,\big(\mathbf{r}_{\sigma_0^{\vec{\tau}}}(\mathbf{snd}\, x)\big) \mathbin{[]} \dots \mathbin{[]}\\ &\qquad\qquad\qquad\qquad \mathbf{pair}\ \underline{m}\,\big(\mathbf{r}_{\sigma_m^{\vec{\tau}}}(\mathbf{snd}\, x)\big)\\ t^{\vec{\tau},i}_{\Theta\vdash\sigma} &= \lambda x{:}\pi_i.\, \lambda y : \sigma^{\vec{\tau}}.\ \mathbf{case}\, y\ \mathbf{of}\ \mathbf{in}_0 w \Rightarrow \mathbf{in}_0(t^{\vec{\tau},i}_{\Theta\vdash\sigma_0} x w) \mathbin{[]} \dots \mathbin{[]}\\ &\qquad\qquad\qquad\qquad \mathbf{in}_m w \Rightarrow \mathbf{in}_m(t^{\vec{\tau},i}_{\Theta\vdash\sigma_m} x w)\end{aligned}$$

We choose

$$\begin{aligned}\mathbf{R}_{\sigma^{\vec{\tau}}} &:\equiv \lambda x.\, \mathsf{CASE}_m\, \mathsf{FST}\, x\ \mathsf{OF}\ \mathsf{PAIR}\ \overline{0}\,\big(\mathbf{R}_{\sigma_0^{\vec{\tau}}}(\mathsf{SND}\, x)\big) \mathbin{[]} \dots \mathbin{[]}\\ &\qquad\qquad\qquad\qquad \mathsf{PAIR}\ \overline{m}\,\big(\mathbf{R}_{\sigma_m^{\vec{\tau}}}(\mathsf{SND}\, x)\big)\\ M^{\vec{\tau},i}_{\Theta\vdash\sigma} &:\equiv \lambda x, y.\ \mathsf{CASE}_m\, \mathsf{FST}\, y\ \mathsf{OF}\ \mathsf{PAIR}\ \overline{0}\,\big(M^{\vec{\tau},i}_{\Theta\vdash\sigma_0} x(\mathsf{SND}\, y)\big) \mathbin{[]} \dots \mathbin{[]}\\ &\qquad\qquad\qquad\qquad \mathsf{PAIR}\ \overline{m}\,\big(M^{\vec{\tau},i}_{\Theta\vdash\sigma_m} x(\mathsf{SND}\, y)\big)\end{aligned}$$

Let $\Gamma :\equiv x{:}U$ and assume $j \in \{0,\dots,m\}$. By assumption, $\mathbf{R}_{\sigma_j^{\vec\tau}} \Vdash \mathbf{r}_{\sigma_j^{\vec\tau}}$. Hence $\mathbf{R}_{\sigma_j^{\vec\tau}}(\mathsf{SND}\,x) \Vdash_\Gamma \mathbf{r}_{\sigma_j^{\vec\tau}}(\mathbf{snd}\,x) : U$ according to Lemma 4.3(*viii*) and (*v*). Since $\overline{j} \Vdash \underline{j}{:}U$, the same lemma yields

$$\mathsf{PAIR}\ \overline{j}\ \Big(\mathbf{R}_{\sigma_j^{\vec\tau}}(\mathsf{SND}\,x)\Big) \Vdash_\Gamma \mathbf{pair}\ \underline{j}\ \Big(\mathbf{r}_{\sigma_j^{\vec\tau}}(\mathbf{snd}\,x)\Big){:}U.$$

Application of Lemma 4.3(*viii*) leads to $\mathbf{R}_{\sigma^{\vec\tau}} \Vdash \mathbf{r}_{\sigma^{\vec\tau}} : U{\to}U$.

Now let $k \in \{0,\dots,m\}$ and put $\Gamma :\equiv x{:}\pi_i$ and $\Gamma' :\equiv \Gamma, z{:}\sigma_k$. Application of Lemma 4.3(*iii*) and (*v*) to the assumption $M^{\vec\tau,i}_{\Theta\vdash\sigma_k} \Vdash t^{\vec\tau,i}_{\Theta\vdash\sigma_k} : \pi_i \to [\sigma_k^{\vec\tau}{\to}\sigma_k^{\vec\tau}]$ leads to

$$H :\equiv M^{\vec\tau,i}_{\Theta\vdash\sigma_k}\,(\mathbf{R}_{\pi_i}\,x)\,\Big(\mathbf{R}_{\sigma_k^{\vec\tau}}\,z\Big) \Vdash_{\Gamma'} t^{\vec\tau,i}_{\Theta\vdash\sigma_k}xz : \sigma_k^{\vec\tau}.$$

Lemma 4.3(*xii*) and (*vi*) yields

$$\lambda z.\,\mathsf{PAIR}\ \overline{k}\,H \Vdash_\Gamma \lambda x{:}\sigma_k.\,\mathbf{in}_k t^{\vec\tau,i}_{\Theta\vdash\sigma_k}xz : \sigma_k{\to}\sum_{j=0}^{m}\sigma_j^{\vec\tau}.$$

Hence Lemma 4.3(*xiii*) and (*vi*) shows the desired conclusion $M^{\vec\tau,i}_{\Theta\vdash\sigma} \Vdash t^{\vec\tau,i}_{\Theta\vdash\sigma} : \pi_i \to [\sigma^{\vec\tau}{\to}\sigma^{\vec\tau}]$.

*Case 6:* If the FPC-type $\Theta\vdash\sigma$ has a $\lambda$+Error representation, so does the type $\alpha_2,\dots,\alpha_n \vdash \mu\alpha_1.\,\sigma$.

Let $\tau_2, \dots, \tau_n$ be closed FPC-types with $\lambda$+Error-implementable retractions and define

$$\begin{aligned}
\tau_1 &:= \mu\alpha_1.\,\sigma\big[\tau_i \,/\, \alpha_i\big] & & \\
\vec\tau &:= \tau_1,\dots,\tau_n & \sigma^{\vec\tau} &:= \sigma\big[\tau_i \,/\, \alpha_i\big] \\
\vec\tau_U &:= U,\tau_2,\dots,\tau_n & \sigma^{U} &:= \big[U \,/\, \alpha_1, \tau_i \,/\, \alpha_i \text{ where } i>1\big]
\end{aligned}$$

Due to Remark 3.5, the canonical retraction of $\tau_1$ (cf. Diagram 4.2) is

$$\mathbf{r}_{\tau_1} :\equiv \mathbf{Y}_{U\to U}\,\lambda f{:}U{\to}U.\,\lambda x{:}U.\,e_{\sigma^U}\left(t^{\vec\tau_U,1}_{\Theta\vdash\sigma}\langle f,f\rangle(\mathbf{p}_{\sigma^U}\,x)\right).$$

We claim that $\mathbf{r}_{\tau_1}$ is $\lambda$+Error-implementable. Let $\Gamma :\equiv f{:}U{\to}U, x{:}U$. According to Lemma 4.3(*xi*) we obtain

$$\mathsf{PAIR}\ f\ f \Vdash_\Gamma \langle f,f\rangle : [U{\to}U]{\times}[U{\to}U].$$

By Lemma 4.3(*ii*), (*v*) and (*vi*) we conclude

$$\begin{aligned}
&\lambda f.\,\lambda x.\,M^{\vec\tau_U,1}_{\Theta\vdash\sigma}\big(\mathbf{R}_{\pi_1}\,(\mathsf{PAIR}\ f\ f)\big)x \\
&\qquad\qquad \Vdash \lambda f{:}U{\to}U.\,\lambda x{:}U.\,t^{\vec\tau_U,1}_{\Theta\vdash\sigma}\langle f,f\rangle(\mathbf{p}_{\sigma^U}\,x) : U.
\end{aligned}$$

$$\begin{tikzcd}
\sigma^U \arrow[d, "t^{\vec{\tau}_U,1}_{\Theta\vdash\sigma}\langle\mathbf{r}_{\tau_1},\mathbf{r}_{\tau_1}\rangle"'] & U \arrow[l, "\mathbf{p}_{\sigma^U}"'] \arrow[d, "\mathbf{r}_{\tau_1}"] \\
\sigma^U \arrow[r, "\mathbf{e}_{\sigma^U}"'] & U
\end{tikzcd}$$

Diagram 4.2: The universal property of $\mathbf{r}_{\tau_1}$

Applying Lemma 4.3(*xvi*) and Lemma 4.3(*v*) we get $R_{\tau_1} \Vdash \mathbf{r}_{\tau_1}$ for

$$R_{\tau_1} :\equiv \mathsf{FIX}\,\lambda f.\,\lambda x.\, M^{\vec{\tau}_U,1}_{\Theta\vdash\sigma}\mathsf{PAIR}\,(\,\mathbf{R}_{U\to U}f)(\mathbf{R}_{U\to U}\,f)(R_{\sigma^U}x).$$

The task is now to find, for any number $i \in \{2,\ldots,n\}$, a closed $\lambda$+Error-term $M^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma}$ which implements the closed FPC-term $t^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma}$. Here $\Theta'$ stands for the type context $\Theta' \equiv \alpha_2,\ldots,\alpha_n$ and $\vec{\tau}_2$ stands for the sequence $(\tau_2,\ldots,\tau_n)$.

Writing $\pi_1 := [\tau_1{\to}\tau_1]{\times}[\tau_1{\to}\tau_1]$ and $\pi_i := [\tau_i{\to}\tau_i]{\times}[\tau_i{\to}\tau_i]$ and

$$l :\equiv \lambda x{:}\tau_1.\,\mathbf{fold}\Big(t^{\vec{\tau},1}_{\Theta\vdash\sigma}\big\langle\mathbf{pr}(q),\mathbf{pl}(q)\big\rangle\Big(t^{\vec{\tau},i}_{\Theta\vdash\sigma}p\big(\mathbf{unfold}(x)\big)\Big)\Big)$$
$$r :\equiv \lambda x{:}\tau_1.\,\mathbf{fold}\Big(t^{\vec{\tau},i}_{\Theta\vdash\sigma}\big\langle\mathbf{pr}(p),\mathbf{pl}(p)\big\rangle\Big(t^{\vec{\tau},1}_{\Theta\vdash\sigma}q\big(\mathbf{unfold}(x)\big)\Big)\Big)$$

we obtain

$$t^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma} \equiv \lambda p{:}\pi_i.\,\mathbf{pl}\big(\mathbf{Y}_{\pi_1}\,\lambda q{:}\pi_1.\langle l,r\rangle\big).$$

Let $\Gamma :\equiv p{:}\pi_i$, $\Gamma' :\equiv \Gamma, q{:}\pi_1$ and $\Gamma'' :\equiv \Gamma', x{:}\tau_1$. Lemma 4.3(*vii*) yields $\mathbf{R}_{\sigma^U}\,x \Vdash \mathbf{unfold}(x) : \sigma^{\vec{\tau}}$. By hypothesis, $M^{\vec{\tau},1}_{\Theta\vdash\sigma} \Vdash t^{\vec{\tau},1}_{\Theta\vdash\sigma} : \pi_1{\to}[\sigma^{\vec{\tau}}{\to}\sigma^{\vec{\tau}}]$ and $M^{\vec{\tau},i}_{\Theta\vdash\sigma} \Vdash t^{\vec{\tau},i}_{\Theta\vdash\sigma} : \pi_i{\to}[\sigma^{\vec{\tau}}{\to}\sigma^{\vec{\tau}}]$. Lemma 4.3(*x*), (*xi*) and (*v*) leads to $L_1 \Vdash_{\Gamma'} l_1 : \sigma^{\vec{\tau}}{\to}\sigma^{\vec{\tau}}$ and $L_i \Vdash_{\Gamma''} l_i : \sigma^{\vec{\tau}}$ for

$$L_1 :\equiv M^{\vec{\tau},1}_{\Theta\vdash\sigma}\Big(\mathbf{R}_{\pi_1}\big(\mathsf{PAIR}\,(\mathsf{SND}\,q)\,(\mathsf{FST}\,q)\big)\Big) \qquad l_1 :\equiv t^{\vec{\tau},1}_{\Theta\vdash\sigma}\big\langle\mathbf{pr}(q),\mathbf{pl}(q)\big\rangle$$
$$L_i :\equiv M^{\vec{\tau},i}_{\Theta\vdash\sigma}\,(\mathbf{R}_{\pi_i}\,p)\,\Big(\mathbf{R}_{\sigma^{\vec{\tau}}}\,(\mathbf{R}_{\sigma^U}\,x)\Big) \qquad l_i :\equiv t^{\vec{\tau},i}_{\Theta\vdash\sigma}p\big(\mathbf{unfold}(x)\big)$$

Further application of Lemma 4.3(*v*), (*vi*) and (*vii*) yields

$$L_1(\mathbf{R}_{\sigma^{\vec{\tau}}}\,L_i) \Vdash_{\Gamma''} l_1 l_i : \sigma^{\vec{\tau}}$$
$$\mathbf{R}_{\sigma^U}\,\Big(\mathbf{R}_{\sigma^{\vec{\tau}}}\,\Big(L_1(\mathbf{R}_{\sigma^{\vec{\tau}}}\,L_i)\Big)\Big) \Vdash_{\Gamma''} \mathbf{fold}(l_1 l_i) : \tau_1$$
$$L :\equiv \lambda x.\,\mathbf{R}_{\sigma^U}\,\Big(\mathbf{R}_{\sigma^{\vec{\tau}}}\big(L_1(\mathbf{R}_{\sigma^{\vec{\tau}}}\,L_i)\big)\Big) \Vdash_{\Gamma'} l : \tau_1{\to}\tau_1$$

Similarly one shows $R :\equiv \lambda x.\, \mathbf{R}_{\sigma^U} \Big(\mathbf{R}_{\sigma^{\vec{\tau}}} \big(R_i(\mathbf{R}_{\sigma^{\vec{\tau}}}\, R_1)\big)\Big) \Vdash_{\Gamma'} r : \tau_1{\to}\tau_1$ where

$$R_i :\equiv M^{\vec{\tau},i}_{\Theta\vdash\sigma} \Big(\mathbf{R}_{\pi_i} \big(\mathsf{PAIR}\,(\mathsf{SND}\,p)\,(\mathsf{FST}\,p)\big)\Big) \quad \text{and}$$

$$R_1 :\equiv M^{\vec{\tau},1}_{\Theta\vdash\sigma}\,(\mathbf{R}_{\pi_1}\, q)\, \Big(\mathbf{R}_{\sigma^{\vec{\tau}}}\,(\mathbf{R}_{\sigma^U}\, x)\Big)\,.$$

The proof is completed by applying Lemma 4.3(*xi*), (*vi*), (*v*) and (*xvi*) to obtain

$$\begin{aligned} H :\equiv \mathsf{PAIR}\; L\, R \Vdash_{\Gamma'} \langle l, r\rangle : \pi_1 \\ \lambda q.\, H \Vdash_{\Gamma} \lambda q{:}\pi_1.\, \langle l, r\rangle : \pi_1{\to}\pi_1 \\ \mathsf{FIX}(\mathbf{R}_{\pi_1\to\pi_1}\, \lambda q.\, H) \Vdash_{\Gamma} \mathbf{Y}_{\pi_1}\, \lambda q{:}\pi_1.\, \langle l, r\rangle : \pi_1 \\ \mathsf{FST}\Big(\mathbf{R}_{\pi_1}\big(\mathsf{FIX}(\mathbf{R}_{\pi_1\to\pi_1}\, \lambda q.\, H)\big)\Big) \Vdash_{\Gamma} \mathbf{pl}\big(\mathbf{Y}_{\pi_1}\, \lambda q{:}\pi_1.\, \langle l, r\rangle\big) : \tau_1 \\ M^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma} \Vdash t^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma} : \pi_1{\to}\tau_1 \end{aligned}$$

where $M^{\vec{\tau}_2,i}_{\Theta'\vdash\mu\alpha_1.\,\sigma} :\equiv \lambda p.\, \mathsf{FST}\Big(\mathbf{R}_{\pi_1}\big(\mathsf{FIX}(\mathbf{R}_{\pi_1\to\pi_1}\, \lambda q.\, H)\big)\Big)$. □

## 4.3 Implementation of terms

In the light of the previous two sections it is clear that any FPC-term-in-context can be implemented in $\lambda$+Error.

**4.8 Theorem** *For any FPC-term-in-context $\Gamma \vdash t{:}\tau$ there is a $\lambda$+Error-term $|\Gamma| \vdash M$ in the corresponding untyped context such that $M \Vdash_\Gamma t{:}\tau$.*

*Proof:* Since the retraction $\mathbf{r}_\sigma$ is implementable in $\lambda$+Error for any FPC-type $\sigma$, Lemma 4.3($v$) yields $\Gamma \vdash M(\mathbf{R}_\varrho\, N) \Vdash_\Gamma mn{:}\tau$ for any terms satisfying $M \Vdash_\Gamma m : \varrho{\to}\tau$ and $N \Vdash_\Gamma n{:}\varrho$.

Hence Lemma 4.3 can be applied to show by induction on the structure of FPC-terms that any FPC-term-in-context $\Gamma \vdash t{:}\tau$ is implemented by some $\lambda$+Error-term $|\Gamma| \vdash M$, where $|\Gamma| := x_1, \ldots, x_n$ is the corresponding $\lambda$+Error-context. □

## 4.4 The universal $\lambda$+Error-model

We define an untyped combinatory algebra $\mathcal{L}$ of classes of closed $\lambda$+Error-terms and show that it is applicatively equivalent to the combinatory algebra $\mathcal{T}$ of observational classes of closed FPC-terms. Thus we conclude that the canonical realizability model for FPC is isomorphic to the realizability model over $\mathcal{L}$

and hence the latter is universal. Using this result we prove a variant of the Longley-Phoa Conjecture.

**4.9 Proposition** *Let $|\mathcal{L}|$ denote the set of all observational equivalence classes of closed $\lambda$+Error-terms. Further let $[K]$ and $[S]$ be the classes of the closed terms*

$$K :\equiv \lambda x.\, \lambda y.\, x \qquad S :\equiv \lambda x.\, \lambda y.\, \lambda z.\, xz(yz)$$

*By defining an application operation on classes in $|\mathcal{L}|$ to be functional application of their representatives we obtain an untyped combinatory algebra $\mathcal{L}$.*

The proof is straightforward.

For untyped (total) combinatory algebras, Definition 2.3 yields the following sufficient criterion for equivalence.

**4.10 Lemma** *Two untyped combinatory algebras $\mathcal{A}$ and $\mathcal{B}$ are equivalent if there are mutually inverse bijections $f\colon\ |\mathcal{A}| \to |\mathcal{B}|$ and $g\colon\ |\mathcal{B}| \to |\mathcal{A}|$ such that for any $a$, $a' \in |\mathcal{A}|$ and $b$, $b' \in |\mathcal{B}|$*

$$f(aa') = f(a)f(a') \quad \text{and} \quad g(bb') = g(b)g(b').$$

The following proposition is a special case of [LS02, Prop. 2.3].

**4.11 Proposition** *The universal type $U$ in the typed combinatory algebra $\mathcal{T}$ gives rise to an untyped combinatory algebra $\mathcal{U}$ as in particular $U{\to}U$ is a retract of $U$. The combinatory algebra $\mathcal{U}$ is applicatively equivalent to the typed combinatory algebra $\mathcal{T}$.*

*The underlying set of $\mathcal{U}$ is $|U|$, the set of observational classes of closed FPC-terms of type $U$. For $[s]$, $[t] \in |U|$, application is defined as $[p_{U\to U}\, s\, t]$, where $\mathbf{p}_{U\to U}$ is the canonical projection from $U$ to $U{\to}U$.*

The proof is a straightforward exercise. Thus we obtain the following theorem:

**4.12 Theorem** *The untyped combinatory algebra $\mathcal{L}$ of closed $\lambda$+Error-terms is equivalent to the typed combinatory algebra $\mathcal{T}$ of closed FPC-terms. Hence* $\mathsf{Mod}(\mathcal{L})$ *and* $\mathsf{Mod}(\mathcal{T})$ *are equivalent as cartesian closed categories.*

*Proof:* Due to Proposition 4.11 it suffices to prove that $\mathcal{L}$ is equivalent to $\mathcal{U}$, which can be shown using Lemma 4.10. For any closed FPC-term $t{:}U$ define $M := f\big([t]_{\text{obs}}\big)$ to be a closed $\lambda$+Error-term realizing $t$, i. e. $M \Vdash t{:}U$, which is equivalent to $(\!|M|\!) =_{\text{obs}} t$. In particular the realizers for $t$ are unique up to observational equivalence.

For any closed $\lambda$+Error-term $M$ we define $g(M) := [(\!|M|\!)]_{\mathrm{obs}}$ to be the interpretation of $M$ in $\mathbb{D}$. It is obvious that $f$ and $g$ are mutually inverse. Now let $s$, $t{:}U$ be closed FPC-terms and let $M$ and $N$ be closed $\lambda$+Error-terms such that $(\!|M|\!) =_{\mathrm{obs}} s$ and $(\!|N|\!) =_{\mathrm{obs}} t$. Then $\mathbf{p}_{U\to U}(\!|M|\!)(\!|N|\!) =_{\mathrm{obs}} (\!|MN|\!)$ implies $g(MN) = \mathbf{p}_{U\to U}\, g(M)g(N)$ and $f\big([\mathbf{p}_{U\to U}\, s\, t]_{\mathrm{obs}}\big) = f\big([s]_{\mathrm{obs}}\big)f\big([t]_{\mathrm{obs}}\big)$, which completes the proof. $\square$

Since the applicative equivalence is easily seen to respect the obvious dominances on $\mathcal{T}$ and $\mathcal{L}$, we obtain the following corollary, which expresses the main result of this thesis.

**4.13 Theorem** *The realizability toposes over the typed combinatory algebra $\mathcal{T}$ and the untyped combinatory algebra $\mathcal{L}$ are equivalent. In particular the model for FPC in $\mathsf{Mod}\,\mathcal{L}$ is universal and logically fully abstract.*

## 4.5 A variant of the Longley-Phoa Conjecture

The Longley-Phoa Conjecture states that the category of modest sets over a combinatory algebra consisting of equivalence classes of untyped closed $\lambda$-terms is universal. This was implicitly conjectured by Phoa in [Pho91] and explicitly stated by Longley in [Lon95, Section 7.4].

Let $\Lambda^0$ denote the set of all closed terms of the untyped $\lambda$-calculus and let $T$ be a semi-sensible $\lambda$-theory, i. e. a $\lambda$-theory that does not equate a solvable and an unsolvable term. (See [Bar84, Chapter 4] for details about $\lambda$-theories.) Then the set $\Lambda^0/T$ of equivalence classes of closed terms with respect to the theory $T$ in a natural way carries the structure of a total combinatory algebra. (cf. [Lon95])

**4.14 Conjecture (Longley-Phoa)** *The model for PCF in $\mathsf{Mod}(\Lambda^0/T)$ is universal.*

Although we do not see how to prove this very conjecture, it is easily seen that Theorem 4.13 implies a variant of it, since observational equivalence for $\lambda$+Error is a $\lambda$-theory.

**4.15 Corollary** *The model for PCF in $\mathsf{Mod}(\mathcal{L})$ is universal.*

Thus the Longley-Phoa Conjecture holds for our untyped language $\lambda$+Error, which is an extension of the $\lambda$-calculus by a single constant $\mathsf{ERR}$ and a conditional construct which distinguishes this constant from any other syntactic value.

# 5 Remarks and future work

Although we have proved a variant of the Longley-Phoa Conjecture, we have to admit that the original conjecture still remains unsettled. Further we do not think that our methods can be generalized to solve it. Out proof strongly depends on the fact that the 'flat booleans' are a definable retract of the combinatory algebra $\mathcal{U}$. This does not hold true if we replace the language $\lambda$+Error by untyped call-by-name $\lambda$-calculus itself.

In an unpublished note [Lai01a], Jim Laird shows that the category of bidomains and stable continuous functions yields a fully abstract model for unary call-by-value FPC, i. e. Plotkin's FPC with the constructor for sum types being restricted to its unary form, i. e. lifting. As a consequence, the canonical models in the category of bidomains for the untyped call-by-value $\lambda$-calculus and for the untyped call-by-name $\lambda$-calculus extended by a sequential "convergence test" construct are fully abstract. Note that the lack of binary sums in unary FPC is a substantial restriction — in particular it rules out the standard representation of ground types (containing more than one value) as "flat domains". However, booleans and numerals can, of course, be encoded in Church style in the untyped $\lambda$-calculus.

One should investigate how this result relates to the material presented in this thesis. However, simple restriction of our construction to unary FPC does not suffice to show that the combinatory algebra of observational equivalence classes of closed terms of the untyped call-by-name $\lambda$-calculus plus "convergence test" gives rise to a fully abstract model for unary FPC. The reason for this is that in our approach not only sum types in FPC but also product types are implemented in $\lambda$+Error using the constant ERR and the conditional.

The most interesting open question is whether and how our results can be generalized to obtain a universal model for a language incorporating polymorphism in the style of Girard's system $F$. Of course, it is known as folklore that there can be no universal realizability model for system $F$ itself, since such a model would contain all computable endofunctions of the type $\forall X.\, X{\to}(X{\to}X){\to}X$ of polymorphic integers, but not all of these functions can be defined in system $F$. However, for system $F$ extended by general recursion this problem does not occur. Hence it makes sense to ask whether our main result generalizes to an extension of call-by-name FPC by system $F$ style polymorphism. So the question is whether factorizing closed $\lambda$+Error-terms by a suitable equivalence relation gives rise to a universal model for such an extension of FPC. Anyway, we think that this problem is quite hard as it is related to questions of parametricity of realizability models.

# Bibliography

[Abr90] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.

[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings of the Thirteenth International Symposium on Logic in Computer Science*, pages 334–344. Computer Society Press of the IEEE, 1998.

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, December 2000.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, revised edition, 1984.

[BBS98] A. Bauer, L. Birkedal, and D.S. Scott. Equilogical spaces. Revised February 2001. To appear in Theoretical Computer Science, September 1998.

[BC82] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.

[BE91] A. Bucciarelli and T. Ehrhard. Sequentiality and strong stability. In *6th Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1991.

[Bel77] J.L. Bell. *Boolean-valued Models and Independence Proofs in Set Theory.* Clarendon Press, Oxford, 1977.

[Bel00] K.G. Belger. The combinatory algebras $\mathcal{U}$ and $\mathcal{A}_{wb,\mathit{eff}}$ are not equivalent. Master's thesis, Technische Universität Darmstadt, September 2000. Available from `http://www.mathematik.tu-darmstadt.de/~streicher/THESES/belger.ps.gz`.

[Ehr96] T. Ehrhard. Projecting sequential algorithmes on strongly stable functions. *Annals of Pure and Applied Logic*, 77(3):201–244, 1996.

[Fef75] S. Feferman. A language and axioms for explicit mathematics. In J.N. Crossley, editor, *Algebra and Logic*, pages 87–139. Springer-Verlag, 1975.

[Fre91] P.J. Freyd. Algebraically complete categories. In A. Carboni, M.C. Pedicchio, and G. Rosolini, editors, *Category Theory, Proc. Int. Conf. in Como, Italy, July 1990*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer Verlag, 1991.

[Fre92] P.J. Freyd. Remarks on algebraically compact categories. In M.P. Fourman, P.T. Johnstone, and A.M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *L.M.S. Lecture Notes*, pages 95–106. Cambridge University Press, 1992.

[Gir72] J.-Y. Girard. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur, 1972. Thèse d'Etat, Université Paris VII.

[Gir89] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In C. Bonotto, R. Ferro, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, pages 221–260. North-Holland, 1989.

[Gir90] J.-Y. Girard. Geometry of interaction II: Deadlock-free algorithms. In P. Martin-Löf and G. Mints, editors, *COLOG-88*, pages 76–93. Springer-Verlag LNCS 417, 1990.

[Gun92] C. Gunter. *Semantics of Programming Languages. Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[HB34] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I.* Springer Verlag, 1934.

[HJP80] J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. Camb. Phil. Soc.*, 88:205–232, 1980.

[HO00] J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163(2):285–408, December 2000.

[How80] W.A. Howard. The formulae-as-type notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[Hyl82] J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 165–216. North-Holland, 1982.

[Joh77] P.T. Johnstone. *Topos Theory.* Number 10 in LMS Mathematical Monographs. Academic Press, London, 1977.

[JS93] A. Jung and A. Stoughton. Studying the fully abstract model of PCF within its continuous function model. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 230–244. Springer Verlag, 1993.

[JT93] A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, 1993.

[Kle45] S.C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[Kle65] S.C. Kleene. Logical calculus and realizability. *Acta Philosophica Fennica*, 18:71–80, 1965.

[Kle69] S.C. Kleene. *Formalized Recursive Functionals and Formalized Relizability*, volume 89 of *Memoirs of the American Mathematical Society.* American Mathematical Society, 1969.

[Kle73] S.C. Kleene. Realizability: a retrospective survey. In A.R.D. Mathias and H. Rogers, editors, *Cambridge Summer School in Mathematical Logic*, volume 337 of *Lecture Notes in Mathematics*, pages 95–112. Springer-Verlag, 1973.

[Kre59] G. Kreisel. Interpretation of analysis by means of functionals of finite type. In *Constructivity in Mathematics.* North Holland, Amsterdam, 1959.

[KV65] S.C. Kleene and R.E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions.* North-Holland Publishing Company, 1965.

[LA99] J.R. Longley and S. Abramsky. Realizability models based on history-free strategies. Unpublished note, September 1999.

[Lai98] J. Laird. *A semantic analysis of control.* PhD thesis, University of Edinburgh, 1998. Available from `http://www.cogs.susx.ac.uk/users/jiml/thesis.ps.gz`.

[Lai01a] J. Laird. Fully abstract bidomain models of the $\lambda$-calculus. Unpublished manuscript. Available from `http://www.cogs.susx.ac.uk/users/jiml/ufpc.ps.gz`, October 2001.

[Lai01b] J. Laird. Games and sequential algorithms. To appear in Theoretical Computer Science. Available from `http://www.cogs.susx.ac.uk/users/jiml/seqa.ps.gz`, August 2001.

[LM91] G. Longo and E. Moggi. Constructive natural deduction and its "$\omega$-set" interpretation. *Mathematical Structures in Computer Science*, 1(2):215–254, 1991.

[Loa01] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266(1–2):341–364, 2001.

[Lon95] J. Longley. *Realizability Toposes and Language Semantics.* PhD thesis, University of Edinburgh, 1995. Published as Technical Report ECS-LFCS-95-332, available from `http://www.lfcs.informatics.ed.ac.uk/reports/95/ECS-LFCS-95-332`.

[Lon98] J. Longley. Realizability models for sequential computation. Unfinished draft, available from http://www.dcs.ed.ac.uk/home/jrl/pisa.ps.gz, September 1998.

[Lon99a] J. Longley. Matching typed and untyped realizability. In L. Birkedal, J. van Oosten, G. Rosolini, and D.S. Scott, editors, *Workshop on Realizability Semantics and Applications*, volume 23 of *Electronic Notes in Computer Science.* Elsevier, 1999. Available from http://www.elsevier.nl/gej-ng/31/29/23/92/27/show/Products/notes/index.htt.

[Lon99b] J. Longley. Unifying typed and untyped realizability. Manuscript (available from `http://www.dcs.ed.ac.uk/home/jrl/unifying.txt`), 1999.

[LS97] J.R. Longley and A.K. Simpson. A uniform approach to domain theory in realizability models. *Mathematical Structures in Computer Science*, 7:469–506, 1997.

[LS02] P. Lietz and T. Streicher. Impredicativity entails untypedness. *Mathematical Structures in Computer Science*, 2002. To appear.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician.* Springer Verlag, Berlin, 1971.

[Mar00] M. Marz. *A Fully Abstract Model for Sequential Computation.* PhD thesis, Technische Universität Darmstadt, Fachbereich Mathematik, January 2000. Logos Verlag, 2000. Available from `http://www.mathematik.tu-darmstadt.de/~streicher/THESES/famsc.ps.gz`.

[McC98] G. McCusker. *Games and Full Abstraction for a Functional Metalanguage with Recursive Types.* Distinguished Dissertations in Computer Science. Springer Verlag, 1998. Ph.D. thesis, Department of Computing, Imperial College, University of London, 1996.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Naples, 1984.

[MRS99] M. Marz, A. Rohr, and T. Streicher. Full abstraction and universality via realisability. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 174–182. IEEE Computer Society Press, 1999.

[Mul87] K. Mulmuley. *Full Abstraction and Semantic Equivalence.* The MIT Press, 1987.

[Nic94] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, Lecture Notes in Computer Science. Springer Verlag, 1994.

[Ong95] C.-H. L. Ong. Correspondence between operational and denotational semantics: the full abstraction problem for PCF. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 269–356. cp, 1995.

[OR95] P.W. O'Hearn and J.G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995.

[Pho91] W. Phoa. *Domain Theory in Realizability Toposes.* PhD thesis, University of Cambridge, 1991.

[Pit81] A.M. Pitts. *The Theory of Triposes.* PhD thesis, Cambridge University, 1981.

[Pit96] A.M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[Plo73] G. D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, October 1973.

[Plo77] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo85] G.D. Plotkin. Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.

[Ros86] G. Rosolini. *Continuity and Effectiveness in Topoi.* PhD thesis, Oxford University, 1986. Available from `http://www.disi.unige.it/ftp/person/RosoliniG/papers/coneit.dvi.gz`.

[RS97a] B. Reus and T. Streicher. General synthetic domain theory – a logical approach. In *Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*, pages 293–313. Springer Verlag, 1997.

[RS97b] J.G. Riecke and A. Sandholm. A relational account of call-by-value sequentiality. In *Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS'97*, pages 258–267, 1997.

[RS99a] B. Reus and T. Streicher. General synthetic domain theory – a logical approach. *Mathematical Structures in Computer Science*, 9(2):177–223, 1999.

[RS99b] J.G. Riecke and A. Sandholm. A relational account of call-by-value sequentiality. *Information and Computation*, 1999. To appear.

[Sco93] D.S. Scott. A type theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. Reprint of a manuscript written in 1969.

[Sie92] K. Sieber. Reasoning about sequential functions via logical relations. In M.P. Fourman, P.T. Johnstone, and A.M. Pitts, editors, *Proc. LMS Symposium on Applications of Categories in Computer Science, Durham 1991*, volume 177 of *LMS Lecture Note Series*, pages 258–269. Cambridge University Press, 1992.

[Sim92] A. K. Simpson. Recursive types in Kleisli categories. Manuscript, available from `http://www.dcs.ed.ac.uk/home/als/Research/kleisli.ps.gz`, 1992.

[SP82] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Computing*, 11(4):761–783, 1982.

[Sta73] J. Staples. Combinator realizability of constructive finite type analysis. In A.R.D. Mathias and H. Rogers, editors, *Cambridge Summer School in Mathematical Logic*, pages 253–273. Springer, 1973.

[Str02] T. Streicher. Mathematical foundations of functional programming. Unpublished lecture notes, available from http://www.mathematik.tu-darmstadt.de/~streicher/MGFP/mgfp.ps.gz, February 2002.

[Tro73] A. S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis.* Springer Verlag, 1973. With contributions by A.S. Troelstra, C.A. Smoryński, J.I. Zucker and W.A. Howard.

[Tro98] A.S. Troelstra. Realizability. In S.R. Buss, editor, *Handbook of Proof Theory*, pages 407–473. North-Holland, 1998.

[vO00] J. v. Oosten. Realizability: An historical essay. To appear in Mathematical Structures in Computer Science. Available from `http://www.math.uu.nl/people/jvoosten/history.ps.gz`, October 2000.

[vOS00] J. v. Oosten and A. Simpson. Axioms and (counter)examples in synthetic domain theory. *Annals of Pure and Applied Logic*, 104(1-3):233–278, 2000.

# Curriculum Vitae

## Alexander Rohr

| | |
|---|---|
| | geboren am 21. November 1967 in Wiesbaden |
| 1987 | Allgemeine Hochschulreife,<br>Stefan-George-Gymnasium in Bingen/Rhein |
| 1989 – 1996 | Studium der Mathematik mit Wahlpflichtfach Informatik,<br>Technische Universität Darmstadt |
| | Schwerpunkte: topologische Algebra, dynamische Systeme,<br>Semantik von Programmiersprachen |
| 1996 | Hochschulabschluss als Diplom-Mathematiker, Diplomarbeit<br>„Chaotic Behaviour of General Semigroup Flows“ |
| 1996 – 2002 | Doktorand am Fachbereich Mathematik der TU Darmstadt<br>Arbeitsgruppe Logik und mathematische Grundlagen der<br>Informatik |
| | zeitweise gefördert durch ein Stipendium des Landes Hessen<br>zweitweise wissenschaftlicher Mitarbeiter |