Opérationnel et catégorique modèles de PCF : l'adressage des machines et la distribution semirings

Nicolas Munnich

UNIVERSITÉ SORBONNE PARIS NORD

INSTITUT GALILÉE

OPERATIONAL AND CATEGORICAL MODELS OF PCF:

ADDRESSING MACHINES AND DISTRIBUTING SEMIRINGS

THÈSE DE DOCTORAT/DOCTORAL THESIS

PRÉSENTÉE PAR/PRESENTED BY

NICOLAS MAX DAVID MÜNNICH

LABORATOIRE D'INFORMATIQUE PARIS NORD

POUR L'OBTENTION DU GRADE DE / FOR THE DEGREE OF

DOCTEUR EN INFORMATIQUE
DOCTOR OF COMPUTER SCIENCE

SOUTENUE LE 23 MAI 2024, DEVANT UN JURY COMPOSÉ DE
DEFENDED ON THE 23RD OF MAY, 2024, IN FRONT OF A JURY COMPOSED OF

GUY MCCUSKER	RAPPORTEUR
LAURENT REGNIER	RAPPORTEUR
RAPHAËLLE CRUBILLÉ	EXAMINATRICE
STEFANO GUERRINI	EXAMINATEUR
GIULIO MANZONETTO	DIRECTEUR DE THÈSE
FLAVIEN BREUVART	DIRECTEUR DE THÈSE

Abstract

Despite being introduced over 60 years ago, PCF remains of interest. Though the quest for a satisfactory fully abstract model of PCF was resolved around the turn of the millennium, new models of PCF still frequently appear in the literature, investigating unexplored avenues or using PCF as a lens or tool to investigate some other mathematical construct. In this thesis, we build upon our knowledge of models of PCF in two distinct ways: Constructing a brand new model, and building upon existing models.

Addressing Machines are a relatively new type of abstract machine taking inspiration from Turing Machines. These machines have been previously shown to model the full untyped λ-calculus. We build upon these machines to construct Extended Addressing Machines (EAMs) and endow them with a type system. We then show that these machines can be used to obtain a new and distinct fully abstract model of PCF: We show that the machines faithfully simulate PCF in such a way that a PCF term terminates in a numeral exactly when the corresponding Extended Addressing Machine terminates in the same numeral. Likewise, we show that every typed Extended Addressing Machine can be transformed into a PCF program with the same observational behaviour. From these two results, it follows that the model of PCF obtained by quotienting typable EAMs by a suitable logical relation is fully abstract.

There exist a plethora of sound categorical models of PCF, due to its close relationship with the λ-calculus. We consider two similar models (which are also models of Linear Logic) that are based on semirings: Weighted models, using semirings to quantify some internal value, and Multiplicity models, using semirings to linearly model functions (model the exponential !). We investigate the intersection between these two models by investigating the conditions under which two monads derived from specific semirings distribute. We discover that whether or not a semiring has an idempotent sum makes a large difference in its ability to distribute. Our investigation leads us to discover the notion of an unnatural distribution, which forms a monad on a Kleisli category. Finally, we present precise conditions under which a particular distribution can form between two semirings.

Resumé

Bien qu'elle ait été introduite il y a plus de 60 ans, PCF reste intéressante. Bien que la quête d'un modèle satisfaisant et totalement abstrait de PCF ait été résolue au tournant du millénaire, de nouveaux modèles de PCF apparaissent encore fréquemment dans la littérature, explorant des voies inexplorées ou utilisant PCF comme une lentille ou un outil pour étudier une autre construction mathématique. Dans cette thèse, nous nous appuyons sur notre connaissance des modèles de PCF de deux manières distinctes : En construisant un tout nouveau modèle, et en s'appuyant sur les modèles existants.

Les machines d'adressage sont un type relativement nouveau de machine abstraite qui s'inspire des machines de Turing. Il a déjà été démontré que ces machines peuvent modéliser l'ensemble du λ-calcul non typé. Nous nous appuyons sur ces machines pour construire des machines d'adressage étendues (EAM) et les doter d'un système de type. Nous montrons ensuite que ces machines peuvent être utilisées pour obtenir un nouveau modèle entièrement abstrait et distinct de PCF : Nous montrons que les machines simulent fidèlement PCF de telle sorte qu'un terme PCF se termine par un chiffre exactement lorsque la machine d'adressage étendue correspondante se termine par le même chiffre. De même, nous montrons que chaque machine d'adressage étendue typée peut être transformée en un programme PCF avec le même comportement d'observation. De ces deux résultats, il découle que le modèle de PCF obtenu en quotientant les EAM typables par une relation logique appropriée est totalement abstrait.

Il existe une pléthore de modèles catégoriels solides de PCF, en raison de sa relation étroite avec le λ-calcul. Nous considérons deux modèles similaires (qui sont aussi des modèles de logique linéaire) qui sont basés sur des sémirings : Les modèles pondérés, qui utilisent les sémirations pour quantifier une valeur interne, et les modèles de multiplicité, qui utilisent les sémirations pour modéliser linéairement des fonctions (modèle de l'exponentielle !). Nous étudions l'intersection entre ces deux modèles en examinant les conditions sous lesquelles deux monades dérivées de sémirings spécifiques se distribuent. Nous découvrons que le fait qu'un semi-anneau ait ou non une somme idempotente fait une grande différence dans sa capacité à distribuer. Notre étude nous conduit à découvrir la notion de distribution non naturelle, qui forme une monade sur une catégorie de Kleisli. Enfin, nous présentons des conditions précises sous lesquelles une distribution particulière peut se former entre deux semis.

Acknowledgements

I would first like to thank my supervisors, Giulio and Flavien, for all they have done for me over the years. I feel very blessed that I had the opportunity to have the two of you as my supervisors. You were both readily available for discussions, and I always felt that I learned something new or had a new direction to explore afterwards. I certainly would not have been able to navigate the administrative complexities or other issues without your help either, which you were always happy to give. I learned plenty of mathematical concepts from the two of you, but I also want to highlight the more esoteric things I learned from you:

From Giulio, I learned how to write papers, both for journals and conferences. Thanks to you, I feel confident in my ability to write about any interesting research results I obtain, even if they are not directly related to this field.

From Flavien, I learned to be more rigorous in the work that I do. Skipping over things that I viewed as "trivial" has always been a bad habit of mine, but I was never able to shake off this habit until working with you. I now feel far more confident in my ability to ensure that my work is correct, by insisting that no stone be left unturned.

Next, I would like to thank my reviewers, Guy McCusker and Laurent Regnier, for their careful reading and feedback of my thesis. I also wish to thank Guy in a separate capacity for inspiring me to enter this field to begin with. Our discussions were always interesting, and I was very happy that you had the time to act as a reviewer.

I also need to thank Stefano Guerrieri and Raphaëlle Crubillé for their role in the jury. On the day of my defense, Stefano was a calming presence, while Raphaëlle posed some questions that caused me to reflect on my work in a new way.

There are countless others that I need to thank as well, for various reasons: Thank you to all the members of LIPN, the LoVe team in particular. Thank you to all of the wonderful people I had the opportunity to meet at various schools and conferences. Thank you to all of my fellow PhD students, I wish I had gotten to know many of you better.

Finally, a great thanks to my mother and father, my brother Patrick, my girlfriend Victoria, and all of the other members of my family who never stopped supporting me and believing in me.

Contents

Notations and Symbols

Extended Addressing Machines

$\mathbb{A}$	The set of addresses	45
M	An extended addressing machine	48
$a_1 \cdot a_2$	Address application	49
$\mathcal{D}_\alpha$	The set of addresses belonging to machines typable with α	84
$\mathcal{D}_\alpha/_{\simeq_\alpha}$	The set of equivalence classes for the type α	84
$\#$	An address table map, usually assumed to be fixed	49
$\#^{-1}$	The bijective inverse function of the address table map	49
$\mathsf{M} \mathbin{@} T'$	The machine $\langle \mathsf{M}.\vec{R}, \mathsf{M}.P, \mathsf{M}.T @ T' \rangle$	49
Δ	A typing context for explicit addressing machines	55
$\Delta[i : \alpha]$	The typing context Δ with i given the type α	55
$\succ_{\mathsf{c}}$	Directional interconvertibility between extended addressing machines	69
$\equiv_\alpha \ (\equiv)$	Logical equivalence for EAMs, tapes (registers)	74
fix_n	A machine approximating the fixed point	75
$\leq_\alpha$	Approximation between typed machines with respect to the fixed point	75
$\mathcal{M}_{\mathbb{A}}^{\Omega}$	The set of machines with only approximations	75
$\mathscr{D}$	The model of PCF using EAMs shown to be fully abstract	84
$\mathsf{M}.P$	The program of the machine M	49
$\to_{\mathsf{c}} \ (\twoheadrightarrow_{\mathsf{c}})$	Reduction for extended addressing machines (multistep)	52
$\mathsf{M} \leftrightarrow_{\mathsf{c}} \mathsf{N}$	M and N share a common reduct	52
$\lvert \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N} \rvert$	The set of lengths of the reduction $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$	52
$\sqsubset$	Ordering between sets of lengths of reductions	52
$\mathsf{M} \twoheadrightarrow_{\mathsf{c}}^{n} \mathsf{N}$	n is a length of the reduction $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$	52
$\mathsf{M}.\vec{R}$	The list of registers of the machine M	49
$\mathsf{M}.R_i$	The i-th register of the machine M	49
$\mathcal{M}_{\mathbb{A}}$	The set of extended addressing machines over $\mathbb{A}$	49
$\simeq_\alpha$	Machine equivalence extended to addresses	84
$[a]_{\simeq_\alpha}$	The equivalence class of a modulo $\simeq_\alpha$	84
$\mathsf{M}.T$	The tape of the machine M	49
$\lvert M \rvert_{\vec{x}}$	The translation of the EPCF term M to a EAM with respect to $\vec{x}$	66

$(\!\|\mathsf{M}\|\!)_\alpha$ $((\!\|P,T\|\!)^\Delta_\alpha)$	The translation of a typable machine (typing auxiliaries) to an EPCF term	86
$\mathsf{M} : \alpha$	M is typable with α	55
$\Delta \Vdash^r (P,T) : \alpha$	The pair (P,T) is typable with α in the context Δ	55
$\vec{R} \models \Delta$	The context Δ can be generated from $\vec{R}$	55
$\varnothing$	Element representing an uninitialised register	46
$!R$	The value stored in the register R	46
$\vec{R}[R_i := a]$	The list of registers $\vec{R}$ with the value stored in R_i updated to be a	46
$a :: T$	The tape with head value a and tail T	46
$T \mathbin{@} T'$	The tape resulting from the concatenation of T and T'	46
$\mathbb{T}_\mathbb{A}$	The set of $\mathbb{A}$-valued tapes	46

Category Theory

$F \dashv G$	F and G are adjoint functors, with F as left adjoint	99
$\mathbb{C}(A,B)$	The class of arrows in $\mathbb{C}$ from A to B	95
$A \multimap B$	$\mathbb{C}(A,B)$ when considered as an object of $\mathbb{C}$	104
asoc	Associator natural isomorphism	103
$\mathbb{C}, \mathbb{D}$	Arbitrary categories	95
$\mathrm{Ob}(\mathbb{C})$	The class of objects of the category $\mathbb{C}$	95
$\mathbb{C}^{\mathsf{op}}$	The opposite category of $\mathbb{C}$	98
$\mathbf{Cmonoid}$	The category of commutative monoids and monoid maps	97
sym	Braiding natural isomorphism	103
$;$	Diagram order composition of arrows	96
$\epsilon^{(\mathrm{M})}_A$	The counit natural transformation of an adjunction (the comonad M)	99
$\mathbf{CPO}$	The category of complete partial orders and Scott-continuous functions	105
$[A,B]$	An arrow object in $\mathbf{CPO}$	106
id_A	The identity arrow of A	96
$\mathbb{C}_\mathrm{M}$	The Kleisli category of the monad M on $\mathbb{C}$	101
μ^M_A	The multiplication natural transformation of the monad M	100
$f : A \simeq B$	f is a natural isomorphism	100
$\mathbf{Poset}$	The category of posets and monotone maps	96
$\mathbb{C} \times \mathbb{D}$	The product category of $\mathbb{C}$ and $\mathbb{D}$	97
π_i	The i-th projection arrow out of a product	102

$\mathbf{Rel}$	The category of sets and relations	96
$\mathbf{Set}$	The category of sets and functions	96
$\otimes$	Tensor product	103
$\eta_A^{(\mathrm{M})}$	The unit natural transformation of an adjunction (the monad M)	99
unl	Left unitor natural isomorphism	103
1	Unit object	103
unr	Right unitor natural isomorphism	103

λ-Calculus & PCF

$=_\alpha$	α equivalence	12
λ	Denotes an abstraction/anonymous function	11
$\Downarrow$	A big-step reduction from a term to a value	19
Λ^0	The set of λ combinators	12
$C[P]$	A context C whose hole is "filled" with P	20
$C\Box$	A context C with a unique hole $\Box$	20
$\equiv_{app}$	Applicative equivalence between PCF terms	21
$\equiv_{obs}$	Observational equivalence between PCF terms	21
FV	Set of free variables of a given term	12
$[\![-]\!]$	An interpretation function	22
Λ	Set of λ-terms	11
Λ^{PCF}	The set of PCF terms	16
$\mathscr{M}$	A model of PCF	22
$\mathbb{T}$	The set of simple types for PCF – the ground type is set to be int	19
$\mathrm{Val}^{\mathsf{PCF}}$	The set of PCF values	18
$\rightarrow_\beta$	β reduction	13
$\twoheadrightarrow_\beta$	Multistep β reduction	13
$\rightarrow_{\mathsf{PCF}}$ $(\twoheadrightarrow_{\mathsf{PCF}})$	Weak head reduction for PCF (multistep)	17
$M[N/x]$	The capture free substitution of N for x in M	12
Γ	A typing context	14
$\mathbb{B}$	The set of base types for a type system	14
$\Gamma \vdash P : \alpha$	A typing judgement	14
$\mathbb{T}^{\mathbb{B}}$	The set of all simple types	14

Var	Set of variables from which terms can be constructed	11
$\vec{x}$	A list of variables $x_1, \ldots, x_n$	12

EPCF & Explicit Substitutions

$\Downarrow^{\mathsf{E}}$	Big step reduction for EPCF	33
$M^{\dagger}$	The PCF term formed by collapsing the EPCF term M	37
$\lfloor - \rfloor$	The size of the head of an EPCF term	37
$\mathcal{P}_{\mathsf{E}}$	The set of EPCF programs	29
$\lvert M \twoheadrightarrow_{\mathrm{wh}} N \rvert$	Length of the reduction $M \twoheadrightarrow_{\mathrm{wh}} N$	32
$\rightarrow_{\mathrm{wh}}$ $(\twoheadrightarrow_{\mathrm{wh}})$	Weak head reduction for EPCF (multistep)	30
$\mathsf{Val}^{\mathsf{E}}$	The set of EPCF values	29
$M\langle N/x \rangle$	An explicit substitution substituting N for x in M	25
TFV	Set of free variables of a given term with an additional tracker	29
Λ^{E}	The set of EPCF terms	28
M^{σ}	An EPCF term formed from a term M and a list of substitutions σ	28

Semirings & Semiring Monads

$\sum_{\sigma \in \mathcal{W}(A,\beta)} f(\sigma)$	Short for $\sum_{a \in \mathrm{S}} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} f(\sigma)$	124
arc	The arctic semiring using real numbers	111
bool	The boolean semiring	111
χ_A	The characteristic function of A with respect to a given semiring	109
$\frac{-\Vert -}{-}$	Coefficient forming an interaction triple with an exponential action	147
$\delta_{a,b}$	The Kronecker delta with respect to a given semiring	109
$(-)^{-}$	An exponential action between two semirings	119
$!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$	A noteworthy comonad on $\mathbf{Set}_{\bar{\mathrm{N}}}$	133
S	A semiring monad of $\mathcal{S}$	115
$\mathrm{S}^{\mathsf{fin}}$	The semiring monad of $\mathcal{S}$ using finite support	115
$\mathrm{S}^{\mathsf{inf}}$	The semiring monad of $\mathcal{S}$ using infinite support	115
$\mathrm{N_f}$	The natural number semiring monad using finite support	133
$\bar{\mathrm{N}}$	The natural number semiring monad with infinite sums	133
$\bar{\mathbb{N}}$	The natural numbers completed with an element representing infinity	111

$\mu_A^{\mathrm{S}-1}(\alpha)$	The set $\{\bar{\alpha} \mid \mu_A^{\mathrm{S}}(\alpha) = \bar{\alpha}\}$	117
$\prod_{a \lhd \alpha} f(a)$	Short for $\prod_{a \in A} (f(a))^{\alpha(a)}$	123
$\bar{\mathbb{R}}$	The real numbers completed with an element representing infinity	111
$\sqcup\!\sqcup(\alpha)$	Short for $\prod_{a \in A} \mathrm{fact}(\alpha(a))$	134
$\mathsf{supp}(\alpha)$	The support of α	108
trop	The tropical semiring using real numbers	111
$\mathcal{W}$	A set of witnesses	120

Set & Order Theory

bool	The set of boolean values $\{\mathtt{tt}, \mathtt{ff}\}$	15
dom	The domain of a function	14
$\emptyset$	The emptyset – the set that contains nothing	12
$\setminus$	Set difference operator	12
$(M, \cdot^M, 1^M)$	A monoid	96
$\mathbb{N}$	The set of natural numbers	13
$(X, \preceq)$	A partially ordered set (poset)	105
$\mathcal{P}$	Powerset	120
$A \times B$	Cartesian product	97
$\bigvee D$	The supremum of D	105

Specific Terms & Machines

$\mathbf{D}$	The term $\lambda x.xx$	12
$\mathbf{I}$	The term $\lambda x.x$	12
Ω	The term $(\lambda x.xx)(\lambda x.xx)$	12
$\mathbf{1}$	The term $\lambda xy.xy$	12
$\mathbf{F}$	The term $\lambda xy.y$	12
$\mathbf{T}$	The term $\lambda xy.x$	12
Add	A machine implementing addition of two numeral machines	55
Apply	A machine implementing application across tapes	63
Add_aux	An auxiliary machine used to define a machine implementing addition	51
I	A machine acting as an identity	51
Ifz	A machine branching on a conditional “if zero” argument	63

Pred	A machine implementing the predecessor	63
Pr	A machine implementing projection	63
Succ	A machine implementing the successor	63
Succ1	A machine finding the successor of an input	51
Succ2	A machine finding the successor of the successor of an input	51
n	The numeral machine representing the number n	49
Y	The machine used to represent the fixed point	49
$\underline{n}$	The PCF term $\mathbf{succ}^n(\mathbf{0})$	16
add	The PCF term $\mathbf{fix}(\lambda fxy.\mathbf{ifz}(x, y, f(\mathbf{pred}(x))(\mathbf{succ}(y))))$	18
succ2	The PCF term $(\lambda sn.s(sn))$succ1	18
is_one	The PCF term $\lambda x.\mathbf{ifz}(\mathbf{pred}(x), \underline{0}, \underline{1})$	18
Ω^{PCF}	The PCF term $\mathbf{fix}(\mathbf{I})$	18
succ	The PCF term $\lambda x.\mathbf{succ}(x)$	18

Introduction

The language Programming Computable Functions (PCF) was originally introduced by Gordon Plotkin in 1977 as an abstract programming language based on the Logic of Computable Functions (LCF) [Plo77]. It has stood the test of time as a language that is still highly of interest today, melding simplicity and desirable characteristics. It is commonly used as a lens to contextualise more abstract mathematical concepts to the field of computer science, and has spawned a number of useful tools and concepts via inspiration and analysis.

This work presents new models of PCF constructed from the intersections between various works from other authors. We will be taking a journey of sorts, discussing formal systems, explicit substitution, abstract machines, and category theory, with the overall goal of constructing models that are simultaneously meaningful with respect to PCF and have intrinsic value of their own.

Over the course of this journey we will be having short discussions on the origins, relevance, and work related to the subject at hand. This is at odds with the common approach found in papers of providing these discussions in the introduction, with the aim of giving you, the reader, a gentler and smoother reading experience.

Contributions and Layout

The content of this thesis can be divided into two parts: constructing a fully abstract model of PCF from a type of abstract machine known as Addressing Machines (AMs), and studying the intersection between two known models of Classical Linear Logic. As knowledge of Linear Logic is not necessary to understand the second part, and may in fact be a distraction to some readers, references to it outside of the introduction have been omitted. Instead, as both of the models in the second part are also models of PCF, the work is presented by using PCF as a lens. Each chapter begins by presenting some of the key notions used in said chapter, often through introducing some new preliminaries or concepts.

Chapter 1 is used as a "joint preliminary" between the two halves, introducing PCF through the λ-calculus. To readers familiar with PCF, the main interest of this chapter is in the fixing

of notations and conventions. This chapter was written with the aim of being approachable for readers unfamilar with the concepts presented, while being useful for referring back to when reading the remainder of the work.

A fully abstract model using Addressing Machines

The result that we aim to present is a fully abstract model of PCF using a variant of abstract machine called Extended Addressing Machines (EAMs). To arrive at this result, a lot of necessary buildup and introductions are presented first. Some of the work presented here was published previously in "Extended Addressing Machines for PCF, with Explicit Substitutions" [BMM23], and certainly some credit for the results of this section go to my co-authors on that paper.

Chapter 2 begins by introducing explicit substitutions in Section 2.1, discussing some of the history of the concept. Later in Section 2.2, we construct a variant of PCF that we call EPCF which makes use of explicit substitutions, albeit in a very unconventional way, with all explicit substitutions forced to be closed. Explicit substitutions can be treated as modelling the internal behaviour of abstract machines, and EPCF makes this more apparent than most – being designed to mirror EAMs as closely as possible is the source of their strange definition. The consequence of this is very obvious merely when looking at Definition 2.2.1 where EPCF is defined – specifically at the definition of free variables in EPCF terms. We take care to ensure that this variant of PCF behaves well – we define both a small-step and a big-step reduction, and then prove their equivalence in Proposition 2.2.9. We show that terms can be typed in a meaningful manner in Lemma 2.2.13. Finally, crucially, we prove that PCF and EPCF coincide in Theorem 2.2.20.

Chapter 3 begins with a discussion of abstract machines more generally, introducing the reader to some of the most notable examples of abstract machines. We have a brief discussion on the origins of Addressing Machines, but do not go into detail on their definition – to do so would involve a lot of unnecessary overlap with the definition of Extended Addressing Machines. Instead, when defining EAMs, we make explicit the (few) differences between these and Addressing Machines. The main components of Extended Addressing Machines are defined in Definitions 3.3.1, 3.3.2, and 3.3.5. We define their behaviour in Definition 3.3.11, and show that this behaviour is deterministic and consistent. Finally, in Definition 3.3.16, we introduce a type system along with an typing algorithm for these machines. We show that both of these behave well.

Chapter 4 contains the work linking PCF, EPCF, and EAMs together to construct the desired models. We begin by defining a translation from EPCF to Extended Addressing Machines. We show that this translation preserves typing in Theorem 4.1.6. Showing that the translation faithfully simulates behaviour is a bit more involved, but is proven after some additional definitions in Theorem 4.1.11. To construct a model from the translation, we define a equivalence relation

in Definition 4.2.1, equating machines which have the same meaning. We construct a model in Definition 4.2.16 by quotienting this relation – showing that two terms are equivalent whenever their corresponding machines are equivalent. To prove that this model is fully abstract, we also define a translation from Extended Addressing Machines to EPCF, and show that it is consistent regarding typing and equivalence in Proposition 4.2.22 and Theorem 4.2.23 respectively. This then neatly leads us to our conclusion.

Models constructed from semiring monads

The work presented here was done with the aim of finding new distributions between monads constructed from semirings. Such distributions would likely be models of classical Linear Logic, and as a consequence models of PCF, its variants, and similar formal systems. This work was done in collaboration with my supervisor Flavien Breuvart.

The first chapter of this half, Chapter 5, is dedicated to providing the categorical preliminaries necessary. Though many of the concepts presented are common knowledge in this field, the content is presented in a manner that makes it comprehensible to a reader with little to no knowledge of category theory. This allows it to also function as a reference for notations and concepts, much like Chapter 1. The later parts of the chapter contain more specialised background knowledge. In Section 5.1, we introduce the categories that model PCF piecewise, intuitively explaining how such categories are suitable for such models along the way. Section 5.2 is dedicated to the more specialist knowledge of semirings and semiring monads required to understand the following chapter – readers familiar with category theory are encouraged to skip Section 5.1, yet still read 5.2. Of particular importance are the notions of complete semiring, multiplicity semiring, and the definition of a monad constructed from a semiring – for many semirings, there are in fact two monads that can be constructed, though we often use them interchangeably.

Chapter 6 presents a subcase of the overarching problem – one where we restrict one of the semirings chosen to have an idempotent sum. Section 6.1 in particular is quite dry with a lot of definitions and proofs of lemmas – this is unfortunately unavoidable due to the technical nature of the content. Theorem 6.1.16 is the result, showing how some new distributions can be constructed. Later in Section 6.2, we provide more intutition and justification for the separation of the problem into an idempotent and non-idempotent half. In particular, Proposition 6.2.4 leads us to the conjecture that there cannot be a distribution between our types of semirings in the non-idempotent case.

Chapter 7 provides an alternative method of obtaining categories that have the potential to model PCF from two semirings without utilising a distribution. We define the concept of an "unnatural distributive law" in Definition 7.1.4 after providing some justification for the concept. We then prove in Theorem 7.1.5 that this causes a monad to arise. Finally, we provide all the

properties required to form a monad from either an unnatural or a standard distribution in Definition 7.2.1, and prove that this is the case in Theorem 7.2.8.

Chapter 1

Programming Computable Functions

Before defining Programming Computable Functions (PCF), we should first define and discuss the λ-calculus, to provide terminology and get a better idea of why PCF is interesting.

1.1 The λ-calculus

The untyped λ-calculus [CHU41] is a formal system with the key concept of having functions as primitive objects. The only notions present are those of application, the process of applying arguments to a function, and λ-abstraction, which is the process of creating a function from an expression. This formal system is Turing complete [Tur37], from which its interest in computer science arises. The untyped λ-calculus is also incredibly simple,[1] and as a result of the above properties is often used as the foundation of other formal systems.

1.1.1 λ-terms

We fix a countably infinite set Var of variables, the elements of which are typically referred to as $x, y, z, \ldots$ or $x_1, x_2, x_3, \ldots$. The set Λ of λ-terms is defined inductively by the following grammar [Bar84]:

$$M, N ::= x \mid \lambda x.M \mid MN$$

λ-terms of the form $\lambda x.M$ are referred to as *abstractions*, and λ-terms of the form MN are referred to as *applications*. We write $\lambda x_1.\lambda x_2 \ldots \lambda x_n.M$ as short for $\lambda x_1.\,(\lambda x_2.\,(\ldots(\lambda x_n.M)\ldots))$ and assume that application associates to the left, i.e.

$$M_1 M_2 M_3 \cdots M_n \quad \text{stands for} \quad (\cdots((M_1 M_2)\, M_3)\cdots)\, M_n$$

Abstraction is given priority over application, i.e. $\lambda x.MN = \lambda x.(MN)$.

[1] Whether it is the simplest Turing complete system depends on your interpretation of "simple" [Cur30, Jay19].

We will use the compact notation $\lambda x_1 x_2.M$ to denote $\lambda x_1.\lambda x_2.M$. When considering a list of variables $x_1, \ldots, x_n$, we refer to the entire list as $\vec{x}$.

Definition 1.1.1 (Free and bound variables)**.**
The set $\mathrm{FV}(M)$ *of* free variables *of a* λ*-term* M *are defined by induction:*

$$\mathrm{FV}(x) \;:=\; \{x\},\quad \mathrm{FV}(\lambda x.M) \;:=\; \mathrm{FV}(M)\setminus\{x\},\quad \mathrm{FV}(MN) \;:=\; \mathrm{FV}(M)\cup\mathrm{FV}(N)$$

A λ*-term* M *where* $\mathrm{FV}(M)=\emptyset$ *is said to be* closed*. Such* λ*-terms are also called* combinators*, and the set of all combinators is typically denoted* Λ^0.

When given an abstraction $\lambda x.M$, all free occurrences of x in M are said to be *bound* by the abstraction λx. A variable is thus *free* if it is not *bound* by any abstraction. We say that a variable y is *fresh* for M if it does not occur in M.

Example 1.1.2.
A few examples of well-known combinators which will be used throughout this work are the following:

$$\begin{array}{lcllcl} \mathbf{I} & := & \lambda x.x & \mathbf{1} & := & \lambda xy.xy \\ \mathbf{T} & := & \lambda xy.x & \mathbf{F} & := & \lambda xy.y \\ \mathbf{D} & := & \lambda x.xx & \Omega & := & \mathbf{DD} \end{array}$$

Some examples of λ*-terms with free variables are the following:*

$$xyz \qquad \lambda x.xy \qquad \lambda xy.z \qquad y\mathbf{I}$$

Definition 1.1.3 (Capture-free Substitution)**.**
Given λ*-terms* M *and* N*, the capture free substitution of* N *for* x *in* M*, written* $M[N/x]$*, is a* λ*-term defined by induction on the shape of* M*:*

$$x[N/x] \;=\; N,\qquad y[N/x] \;=\; y,\ \text{if } y\neq x,\qquad (\lambda x.M')\,[N/x] \;=\; \lambda x.M',$$

$$\begin{array}{rcl} (\lambda y.M')\,[N/x] & = & \lambda z.\,(M'[z/y][N/x])\,,\ \text{if } y\neq x, \\ & & \text{where } z\in \mathrm{Var}\setminus(\{y\}\cup\mathrm{FV}(M'N)), \\ (M_1M_2)\,[N/x] & = & (M_1[N/x])(M_2[N/x]) \end{array}$$

Definition 1.1.4 (α-equivalence)**.**
Terms which differ only in the naming of their bound variables are said to be α-equivalent*. This equivalence, denoted* $=_\alpha$*, is axiomatised as follows:*

$$\lambda x.M = \lambda y.\,(M[y/x])\,,\ \textit{where } y \textit{ is fresh for } M.$$

Two λ-terms are α-equivalent if and only if they are provably equal via the above rule. Essentially, the names we give to bound variables has no effect on the meaning of a term.

As a consequence, we can adopt the variable convention of giving all bound variables maximally distinguished names. In other words, when we write $M_1 = \lambda x.x$ and $M_2 = \lambda y.xy$, then the x occurring in M_1 is different from both the x and the y occurring in M_2.[2]

Definition 1.1.5 (β-reduction)**.**
The β-reduction *allows the application of arguments to a function to be "resolved". We define a basic reduction using capture free substitution*

$$(\lambda x.M)N \to_b M[N/x]$$

Using inference rules we then define β-reduction as follows:

$$\frac{M \to_b M'}{M \to_\beta M'} \qquad \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

We write $M \twoheadrightarrow_\beta N$ if there exists $n \in \mathbb{N}$ such that $M \to_\beta M_1 \to_\beta \cdots \to_\beta M_{n-1} \to_\beta N$. In other words, $\twoheadrightarrow_\beta$ is the transitive-reflexive closure of $\to_\beta$. Subterms of the form $(\lambda x.M) \cdot N$ are called redexes*.*

Regarding β-reduction, the untyped λ-calculus is known to be confluent – the order in which redexes are evaluated does not affect the final result. A λ-term is said to be in *normal form* if it cannot reduce via β-reduction. The non-deterministic nature of β-reduction results in multiple strategies that one can use to produce a deterministic reduction. Of particular note is head reduction:

Definition 1.1.6 (Head reduction)**.**

(1) A redex $(\lambda x.M_1)M_2$ *is said to be in* head position *in a term M if M has the following shape:*

$$\lambda y_1 \dots y_n.(\lambda x.M_1)M_2N_1 \cdots N_m, \text{ where } n, m \in \mathbb{N}$$

A β-reduction is a head reduction *if it reduces the redex in head position. A λ-term is in* head normal form *if it does not have a redex at head position.*

(2) A weak head reduction *is a head reduction which does not reduce under an abstraction. In other words, we can define weak head reductions $\to_{wh\beta} \subset \to_\beta$ as*

$$\frac{M \to_b M'}{M \to_{wh\beta} M'} \qquad \frac{M \to_{wh\beta} M'}{MN \to_{wh\beta} M'N}$$

A λ-term is in weak head normal form *if it cannot reduce via a weak head reduction step.*

[2]This variable convention is typically attributed to Henk Barendregt [Bar84].

1.1.2 The simply typed λ-calculus

Thus far we have discussed the untyped λ-calculus. The λ-calculus can be seen as an *abstract programming language*, and like many programming languages, it can be adorned with typing information. The simplest way of doing so results in what is known as the *simply typed λ-calculus* [BDS13, Part I.1].

Definition 1.1.7 (Simple types)**.**
Consider fixed a set of base types $\mathbb{B}$. The set of all simple types $\mathbb{T}^{\mathbb{B}}$ is defined as:

$$\alpha, \beta \quad ::= \quad \alpha \to \beta \mid o, \quad \text{where } o \in \mathbb{B}$$

In the above, α and β are called type variables *while o is a* base type*. The arrow operator associates to the right, in other words we write $\alpha_1 \to \cdots \to \alpha_n \to \beta$ for $\alpha_1 \to (\cdots(\alpha_n \to \beta)\cdots)$ ($= \vec{\alpha} \to \beta$, for short). If $n = 0$ then $\alpha_1 \to \cdots \to \alpha_n \to \beta = \beta$.*

Types are then assigned to λ-terms via *typing judgements*.

Definition 1.1.8 (Typing Contexts)**.**

(1) A typing context Γ *is a partial function from* Var *to $\mathbb{T}^{\mathbb{B}}$ such that the domain of Γ is finite.*

(2) Γ is often denoted as $x_1 : \alpha_1, \ldots, x_n : \alpha_n$, meaning that for all $i \leq n$, $\Gamma(x_i) = \alpha_i$. This notation is also used to define new typing contexts; $\Gamma, y : \beta$ is a typing context where

$$\begin{aligned} \mathrm{dom}(\Gamma, y : \beta) &= \mathrm{dom}(\Gamma) \cup \{y\} \\ (\Gamma, y : \beta)(x) &= \begin{cases} \beta & \text{if } x = y, \\ \Gamma(x) & \text{otherwise.} \end{cases} \end{aligned}$$

A typing context can also be understood as a finite set of assignments/pairs; a dictionary. Typing contexts are sometimes referred to as *environments*.

Definition 1.1.9 (Simple Typing Judgements)**.**
A typing judgement *for the simply typed λ-calculus consists of a λ-term M, a type α, and a typing context Γ, and are written $\Gamma \vdash M : \alpha$. The type inference rules for the simply typed λ-calculus are:*

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha} \qquad \frac{\Gamma, x : \beta \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \beta \to \alpha} \qquad \frac{\Gamma \vdash M : \beta \to \alpha \quad \Gamma \vdash N : \beta}{\Gamma \vdash MN : \alpha}$$

A typing derivation *is a finite tree built bottom-up in such a way that the root has shape $\Gamma \vdash M : \alpha$ and every node is an instance of one of the above rules. When writing $\Gamma \vdash M : \alpha$, we mean that this typing judgement is derivable.*

Many terms have multiple valid types which can be derived. For example, the term $\lambda x.x$ can be typed with $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta \to \gamma)$ or $(\alpha \to \beta) \to (\alpha \to \beta)$. The reason

for this is that if a type has at least one type variables, the type can be made more specific by substituting all occurrences of a type variable with any other type. For example, if a term is typable with $\alpha \to \mathrm{bool} \to \alpha$, all occurrences of α could be replaced with $\beta_1 \to \beta_2$, forming the type $(\beta_1 \to \beta_2) \to \mathrm{bool} \to (\beta_1 \to \beta_2)$. Typable terms always have a *most general type*, which is the least specific type assignable – the type which consists of a minimal amount of base types and type variables, and base types are replaced by type variables whenever possible. For $\lambda x.x$, the most general type is $\alpha \to \alpha$.

The simply typed λ-calculus is, unlike the untyped λ-calculus, *strongly normalising* – all terms reduce to a normal form regardless of the order in which one chooses redexes [BDS13, Part I.2]. Naturally, this means it cannot be Turing complete. This can be seen when attempting to type the terms in Example 1.1.2.

Example 1.1.10.
The following typing judgements for the examples found in Example 1.1.2 are derivable:

$$\begin{array}{llll}
\vdash & \mathbf{I} : \alpha \to \alpha & x : \alpha \to \beta \to \gamma, y : \alpha, z : \beta & \vdash \quad xyz : \gamma \\
\vdash & \mathbf{T} : \alpha \to \beta \to \alpha & z : \gamma & \vdash \quad \lambda xy.z : \alpha \to \beta \to \gamma \\
\vdash & \mathbf{1} : (\beta \to \alpha) \to \beta \to \alpha & y : \beta & \vdash \quad \lambda x.xy : (\beta \to \alpha) \to \alpha \\
\vdash & \mathbf{F} : \alpha \to \beta \to \beta & y : (\alpha \to \alpha) \to \beta & \vdash \quad y\mathbf{I} : \beta
\end{array}$$

$\mathbf{D}$ *and* Ω *are not typable. Attempting to construct a typing derivation for* $\mathbf{D}$ *results in the following tree:*

$$\dfrac{\dfrac{\overline{x : \beta \vdash x : \gamma \to \alpha} \quad \overline{x : \beta \vdash x : \gamma}}{x : \beta \vdash xx : \alpha}}{\vdash \lambda x.xx : \beta \to \alpha}$$

For the tree to "resolve", $\gamma \to \alpha$ *would have to be the same as* γ*, which is not possible due to the differing number of arrows.* Ω *is not typable as it features* $\mathbf{D}$ *as a subterm.*

A term is *weakly normalising* if there exists a particular order of reductions allowing it to reach a normal form. An example of such an object in the untyped λ-calculus would be $(\lambda x.y)\Omega$. Reduction strategies which do not reduce arguments or under an abstraction are called *weak head reduction* strategies, as they reduce to a weak head normal form where possible and then do not reduce further.

1.2 Programming Computable Functions

The Logic of Computable Functions was proposed by Dana Scott in 1969[3] with the goal of having a logical calculus for computable functions. This work introduced the type system which inspired PCF. At the time, PCF was quite distinct from languages in common use. Logic of Computable Functions inspired the language ML, which PCF could be seen as a simplification of, and in turn inspired modern functional languages such as Haskell and OCaml. PCF is the foundation upon which typed functional languages are built.

The simplest approach to working with PCF is to consider it in terms of its constituent parts. Essentially, PCF is a language obtained from the simply typed λ-calculus upon adding features to regain Turing completeness. It could (debatably) be called the simplest typed Turing complete declarative programming language.

PCF can be informally described as the simply typed λ-calculus with numerical constants, arithmetic operators, conditional branching, and a fixed point operator.[4]

Definition 1.2.1 (PCF terms)**.**
The set Λ^{PCF} *of* PCF terms *is defined by induction as follows:*

$$P, Q, Q' ::= x \mid \lambda x.P \mid PQ \mid \mathbf{fix}\, P \mid \mathbf{0} \mid \mathbf{pred}\, P \mid \mathbf{succ}\, P \mid \mathbf{ifz}(P, Q, Q') \qquad (\Lambda^{\mathsf{PCF}})$$

where $\lambda x.P$ *represents the abstraction,* PQ *the application,* $\mathbf{fix}$ *the fixed point operator,* $\mathbf{0}$ *the constant zero,* $\mathbf{pred}$ *and* $\mathbf{succ}$ *the predecessor and successor (respectively), and* $\mathbf{ifz}$ *the conditional test on zero (namely, 'is zero?-then-else').*

As a matter of notation, we write $\underline{n}$ for $\mathbf{succ}^{\,n}(\mathbf{0})$, repeating $\mathbf{succ}$ n-many times on $\mathbf{0}$.

PCF inherits terminology, notations, and conventions – but not definitions – from the λ-calculus. For example, the concept of free variables in PCF remains the same as in λ-calculus, but the definition changes to account for additional cases.

Definition 1.2.2 (PCF Free Variables)**.**
The set $\mathrm{FV}(P)$ *of* free variables of $P \in \Lambda^{\mathsf{PCF}}$ *is defined as*

$$\begin{aligned}
\mathrm{FV}(x) &:= \{x\}, & \mathrm{FV}(\mathbf{0}) &:= \emptyset \\
\mathrm{FV}(\lambda x.P) &:= \mathrm{FV}(P) \setminus \{x\}, & \mathrm{FV}(PQ) &:= \mathrm{FV}(P) \cup \mathrm{FV}(Q) \\
\mathrm{FV}(\mathbf{pred}\, P) &:= \mathrm{FV}(P) & \mathrm{FV}(\mathbf{succ}\, P) &:= \mathrm{FV}(P) \\
\mathrm{FV}(\mathbf{ifz}(P, Q_1, Q_2)) &:= \mathrm{FV}(P) \cup \mathrm{FV}(Q_1) \cup \mathrm{FV}(Q_2) & \mathrm{FV}(\mathbf{fix}\, P) &:= \mathrm{FV}(P)
\end{aligned}$$

We will refer to PCF terms which are closed as PCF *programs*.

[3] Though the work remained unpublished until 1993.
[4] The simply typed λ-calculus together with a fixed point operator is not Turing complete [Sta02].

Definition 1.2.3 (Capture Free Substitution for PCF)**.**
Given PCF *terms P and Q, the capture free substitution of Q for x in P, written $P[Q/x]$, is a* PCF *term defined by induction on the shape of P:*

$$\begin{array}{rclcrcl}
x[P'/x] & = & P', & & (\mathbf{pred}\, P')[Q/x] & = & \mathbf{pred}\,(P'[Q/x]),\\
y[P'/x] & = & y, \text{ if } y \neq x, & & (\mathbf{succ}\, P')[Q/x] & = & \mathbf{succ}\,(P'[Q/x]),\\
(\mathbf{fix}\, P')[Q/x] & = & \mathbf{fix}\,(P'[Q/x]), & & (\lambda x.P')\,[Q/x] & = & \lambda x.P',\\
 & & & & (P_1P_2)\,[Q/x] & = & (P_1[Q/x])\,(P_2[Q/x])\,,\\
 & & & & (\mathbf{ifz}(P_1,P_2,P_3))[Q/x] & = & \mathbf{ifz}(P_1[Q/x],P_2[Q/x],P_3[Q/x]),\\
 & & & & (\lambda y.P')\,[Q/x] & = & \lambda z.\,(P'[z/y][Q/x])\,, \text{ if } y \neq x,\\
 & & & & & & \text{where } z \in \mathrm{Var} \setminus (\{y\} \cup \mathrm{FV}(P'Q)).
\end{array}$$

The α-equivalence is defined as in the λ-calculus. Hereafter, PCF terms are considered up to α-conversion.

As an abstract programming language, PCF is generally considered with deterministic reduction strategies. As with the λ-calculus, there exist multiple strategies to choose from. The most prominent of these are *call-by-name* and *call-by-value*. Here we will focus on a variant of call-by-name PCF, which is usually referred to simply as PCF. We introduce the *small-step operational semantics* of PCF.

Definition 1.2.4 (Small Step Operational Semantics of PCF)**.**
The basic reduction rules of PCF *are the following:*

$$\begin{array}{rclcrcl}
(\lambda x.P)Q & \to_p & P[Q/x], & & \mathbf{fix}\,(P) & \to_p & P(\mathbf{fix}\,(P)),\\
\mathbf{pred}\,(\mathbf{succ}\,(\underline{n})) & \to_p & \underline{n}, & & \mathbf{pred}\,(\mathbf{0}) & \to_p & \mathbf{0},\\
\mathbf{ifz}(\mathbf{0},P_1,P_2) & \to_p & P_1, & & \mathbf{ifz}(\underline{n+1},P_1,P_2) & \to_p & P_2,
\end{array}$$

The weak head reduction $\to_{\mathsf{PCF}} \subseteq \Lambda^{\mathsf{PCF}} \times \Lambda^{\mathsf{PCF}}$ *is defined as follows:*

$$\frac{P \to_p P'}{P \to_{\mathsf{PCF}} P'} \quad \frac{P \to_{\mathsf{PCF}} P'}{PQ \to_{\mathsf{PCF}} P'Q} \quad \frac{P \to_{\mathsf{PCF}} P'}{\mathbf{ifz}(P,Q_1,Q_2) \to_{\mathsf{PCF}} \mathbf{ifz}(P',Q_1,Q_2)}$$

$$\frac{P \to_{\mathsf{PCF}} P'}{\mathbf{pred}\, P \to_{\mathsf{PCF}} \mathbf{pred}\, P'} \quad \frac{P \to_{\mathsf{PCF}} P'}{\mathbf{succ}\, P \to_{\mathsf{PCF}} \mathbf{succ}\, P'}$$

The *multistep reduction* $\twoheadrightarrow_{\mathsf{PCF}}$ is defined as the transitive-reflexive closure of $\to_{\mathsf{PCF}}$.

Remark 1.2.5.
There are some call-by-name elements present in the handling of the numerals. This is a technical choice required for later parts of this work – exploring other reduction strategies including "pure" call-by-name is left for future work.

Example 1.2.6.
All of the λ-terms listed in Example 1.1.2 are also PCF *terms, as $\Lambda \subset \Lambda^{\mathsf{PCF}}$. Here are some examples of terms which are* PCF *terms but not λ-terms.*

(1) For all $n \in \mathbb{N}$, $\underline{n}$ is a PCF *term representing a number.*

(2) $succ1 = \lambda x.\mathbf{succ}\, x$, representing the successor function.

(3) $succ2 = (\lambda sn.s(sn))\mathtt{succ1}$, a double successor function

(4) $is\_one = \lambda x.\mathbf{ifz}(\mathbf{pred}\, x, \underline{0}, \underline{1})$, a function which outputs 0 or 1 depending on whether an input is greater than 1.

(5) $add = \mathbf{fix}\,(\lambda fxy.\mathbf{ifz}(x, y, f(\mathbf{pred}\, x)(\mathbf{succ}\, y)))$, a function which computes addition.

(6) $\Omega^{\mathsf{PCF}} = \mathbf{fix}\,(\mathbf{I})$, the looping program using $\mathbf{fix}$.

Example 1.2.7.
As an example of a PCF *reduction, we present the reduction of* $\mathtt{add}\ \underline{1}\ \underline{2}$*:*

$$\begin{aligned}
\mathtt{add}\underline{1}\underline{2} &\to_{\mathsf{PCF}} (\lambda fxy.\mathbf{ifz}(x, y, f(\mathbf{pred}\, x)(\mathbf{succ}\, y)))\mathtt{add}\ \underline{1}\ \underline{2}\\
&\to_{\mathsf{PCF}} (\lambda xy.\mathbf{ifz}(x, y, \mathtt{add}(\mathbf{pred}\, x)(\mathbf{succ}\, y)))\underline{1}\ \underline{2}\\
&\to_{\mathsf{PCF}} (\lambda y.\mathbf{ifz}(\underline{1}, y, \mathtt{add}(\mathbf{pred}\, \underline{1})(\mathbf{succ}\, y)))\underline{2}\\
&\to_{\mathsf{PCF}} \mathbf{ifz}(\underline{1}, \underline{2}, \mathtt{add}(\mathbf{pred}\, \underline{1})(\mathbf{succ}\, \underline{2}))\\
&\to_{\mathsf{PCF}} \mathtt{add}(\mathbf{pred}\, \underline{1})(\mathbf{succ}\, \underline{2})\\
&\to_{\mathsf{PCF}} (\lambda fxy.\mathbf{ifz}(x, y, f(\mathbf{pred}\, x)(\mathbf{succ}\, y)))\mathtt{add}(\mathbf{pred}\, \underline{1})(\mathbf{succ}\, \underline{2})\\
&\to_{\mathsf{PCF}} (\lambda xy.\mathbf{ifz}(x, y, \mathtt{add}(\mathbf{pred}\, x)(\mathbf{succ}\, y)))(\mathbf{pred}\, \underline{1})(\mathbf{succ}\, \underline{2})\\
&\to_{\mathsf{PCF}} (\lambda y.\mathbf{ifz}((\mathbf{pred}\, \underline{1}), y, \mathtt{add}(\mathbf{pred}\, (\mathbf{pred}\, \underline{1}))(\mathbf{succ}\, y)))(\mathbf{succ}\, \underline{2})\\
&\to_{\mathsf{PCF}} \mathbf{ifz}((\mathbf{pred}\, \underline{1}), (\mathbf{succ}\, \underline{2}), \mathtt{add}(\mathbf{pred}\, (\mathbf{pred}\, \underline{1}))(\mathbf{succ}\, (\mathbf{succ}\, \underline{2})))\\
&\to_{\mathsf{PCF}} \mathbf{ifz}(\underline{0}, (\mathbf{succ}\, \underline{2}), \mathtt{add}(\mathbf{pred}\, (\mathbf{pred}\, \underline{1}))(\mathbf{succ}\, (\mathbf{succ}\, \underline{2})))\\
&\to_{\mathsf{PCF}} \mathbf{succ}\, \underline{2} = \underline{3}
\end{aligned}$$

This deterministic reduction strategy is weak as it does not reduce under abstractions.

Proposition 1.2.8 (PCF Values)**.**
A PCF value *is either a numeral or an abstraction. The* set of PCF values *is defined by:*

$$\mathrm{Val}^{\mathsf{PCF}} = \{\underline{n} \mid n \in \mathbb{N}\} \cup \{\lambda x.P \mid P \in \Lambda^{\mathsf{PCF}}\}$$

We will use U, occasionally with indicies, as the meta-variable for PCF *values.*

Definition 1.2.9 (Big Step Operational Semantics of PCF)**.**
The big step reduction $\Downarrow \subseteq \Lambda^{\mathsf{PCF}} \times \Lambda^{\mathsf{PCF}}$ *maps a* PCF *term to a* PCF *value:*

$$\frac{U \in \mathrm{Val}^{\mathsf{PCF}}}{U \Downarrow U}\ (\mathrm{val}) \qquad \frac{P \Downarrow \mathbf{0}}{\mathbf{pred}\, P \Downarrow \mathbf{0}}\ (\mathrm{pr}_0) \qquad \frac{P \Downarrow \underline{n+1}}{\mathbf{pred}\, P \Downarrow \underline{n}}\ (\mathrm{pr})$$

$$\frac{P \Downarrow \mathbf{0} \quad Q \Downarrow U_1}{\mathbf{ifz}(P, Q, Q') \Downarrow U_1}\ (\mathrm{ifz}_0) \qquad \frac{P \Downarrow \underline{n+1} \quad Q' \Downarrow U_2}{\mathbf{ifz}(P, Q, Q') \Downarrow U_2}\ (\mathrm{ifz}_{>0}) \qquad \frac{P \Downarrow \underline{n}}{\mathbf{succ}\, P \Downarrow \underline{n+1}}\ (\mathrm{sc})$$

$$\frac{P(\mathbf{fix}\, P) \Downarrow U}{\mathbf{fix}\, P \Downarrow U}\ (\mathrm{fix}) \qquad \frac{P \Downarrow \lambda x.P' \quad P'[Q/x] \Downarrow U}{PQ \Downarrow U}\ (\beta_{\mathrm{val}})$$

It is well-known that the big-step and small-step semantics of PCF are equivalent on PCF programs, i.e for $M \in \Lambda^{\mathsf{PCF}}$ and $N \in \mathrm{Val}^{\mathsf{PCF}}$, $M \twoheadrightarrow_{\mathsf{PCF}} N$ if and only if $M \Downarrow N$. For a proof of this equivalence, see e.g. [Mit96, Chapter 2].

Definition 1.2.10 (Types for PCF)**.**
The set $\mathbb{T}$ *of* (simple) types *over a* ground type int *is defined by:*

$$\alpha, \beta \ ::= \ \mathsf{int} \mid \alpha \to \beta \tag{$\mathbb{T}$}$$

Again, PCF inherits terminology, notations, and conventions but not definitions from the simply typed λ-calculus regarding typing.

Definition 1.2.11 (Typing PCF terms)**.**
The following are the type derivation rules for PCF.

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathsf{int}}\ (0) \qquad \frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\mathrm{ax})$$

$$\frac{\Gamma \vdash P : \alpha \to \alpha}{\Gamma \vdash \mathbf{fix}\, P : \alpha}\ (\mathrm{Y}) \qquad \frac{\Gamma, x : \alpha \vdash P : \beta}{\Gamma \vdash \lambda x.P : \alpha \to \beta}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash P : \mathsf{int}}{\Gamma \vdash \mathbf{pred}\, P : \mathsf{int}}\ (-) \qquad \frac{\Gamma \vdash P : \mathsf{int}}{\Gamma \vdash \mathbf{succ}\, P : \mathsf{int}}\ (+)$$

$$\frac{\Gamma \vdash P : \alpha \to \beta \quad \Gamma \vdash Q : \alpha}{\Gamma \vdash P \cdot Q : \beta}\ (\to_{\mathrm{E}}) \qquad \frac{\Gamma \vdash P : \mathsf{int} \quad \Gamma \vdash Q : \alpha \quad \Gamma \vdash Q' : \alpha}{\Gamma \vdash \mathbf{ifz}(P, Q, Q') : \alpha}\ (\mathrm{ifz})$$

Remark 1.2.12.
All the normal forms of typed PCF *programs are* PCF *values.*

Example 1.2.13.
The type derivation rules for PCF *is a superset of the rules for the simply typed* λ*-calculus, so the examples found to be typable in Example 1.1.10 remain typable with the same types. The additional terms named in Example 1.2.6 are typable with the following judgements.*

(1) $\vdash \underline{n} : \mathsf{int}$.

(2) $\vdash \mathtt{succ1} : \mathsf{int} \to \mathsf{int}$.

(3) $\vdash \mathtt{succ2} : \mathsf{int} \to \mathsf{int}$.

(4) $\vdash \mathtt{is\_one} : \mathsf{int} \to \mathsf{int}$.

(5) $\vdash \mathtt{add} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$.

(6) $\vdash \Omega^{\mathsf{PCF}} : \alpha$, *for all* $\alpha \in \mathbb{T}$. *In particular, we have* $\vdash \Omega^{\mathsf{PCF}} : \mathsf{int}$.

Clearly, a λ*-term is typable if and only if it is typable as a* PCF *term.*

1.3 Modelling PCF

1.3.1 Observational Equivalence

Determining whether two programs are equivalent is fundamental to computer science. For example, ensuring that optimisations performed by a compiler do not affect the overall outcome. In formal systems based on the λ-calculus, we usually consider two terms equivalent when they are *observationally equivalent* [Mor69]. The idea of observational equivalence is that two terms have the same "meaning" if they can be substituted for one another without changing an observable outcome, for some set of observable outcomes.

To define observational equivalence formally, we must first introduce the idea of a context. Essentially, a *context* is a term with exactly one "hole" occurring as a subterm. We define contexts for PCF.

Definition 1.3.1 (PCF Contexts)**.**

(1) PCF *contexts* C *with a hole* $\Box$ *are defined inductively with the following grammar:*

$$\begin{aligned} C\Box \quad ::= \quad & \Box \mid \lambda x.C\Box \mid P \cdot C\Box \mid C\Box \cdot Q \mid \mathbf{pred}\, C\Box \mid \mathbf{succ}\, C\Box \\ & \mid \mathbf{ifz}(C\Box, P, Q) \mid \mathbf{ifz}(P, C\Box, Q) \mid \mathbf{ifz}(P, Q, C\Box) \mid \mathbf{fix}\, C\Box \end{aligned}$$

(2) We write $C[P]$ *for the* PCF *term formed by substituting* P *for the hole* $\Box$ *in* $C\Box$.

(3) When substituting P *for* $\Box$ *in* $C\Box$*, we do not perform capture-free substitution. For example, if* $C\Box = \lambda x.\Box$ *and* $P = x$*, then* $C[P] = \lambda x.x$.

(4) Given a PCF *context* $C\Box$*, a typing context* Γ*, and types* $\alpha, \beta \in \mathbb{T}$*, we write* $C\Box : (\Gamma, \alpha) \to \beta$ *if for all* P *such that* $\Gamma \vdash P : \alpha$, $\vdash C[P] : \beta$.

For PCF, observational equivalence is defined with the set of observable outcomes being $\mathbb{N}$. We observe termination at ground type as we do not reduce under an abstraction. For example, we would consider $\lambda x.\underline{0}$ and $\lambda x.((\lambda y.y) \cdot \underline{0})$ to have the same meaning, as once any argument is applied they would both reduce to $\underline{0}$.

Definition 1.3.2 (Observational Equivalence)**.**
Let P, P' be two PCF *terms such that for a given typing context Γ and type α, $\Gamma \vdash P : \alpha$ and $\Gamma \vdash P' : \alpha$. We say that P and P' are* observationally equivalent*, written $P \equiv_{obs} P'$, if the following holds:*

For all contexts $C\square : (\Gamma, \alpha) \to \mathsf{int}$, $C[P] \Downarrow \underline{n}$ if and only if $C[P'] \Downarrow \underline{n}$.

Working with observational equivalence directly can be a bit difficult, but for the specific case of PCF there exists the notion of *applicative equivalence* which can (mostly) be used instead.

Definition 1.3.3 (Applicative Equivalence)**.**
Given two PCF *programs P, P' such that $\vdash P : \alpha_1 \to \cdots \to \alpha_k \to \mathsf{int}$ and $\vdash P' : \alpha_1 \to \cdots \to \alpha_k \to \mathsf{int}$, for $k \in \mathbb{N}, \alpha_1, \ldots, \alpha_k \in \mathbb{T}$, the following are one and the same.*

(1) P and P' are applicatively equivalent, written $P \equiv_{app} P'$;

(2) For all PCF *programs $\vdash Q_1 : \alpha_1, \ldots, \vdash Q_k : \alpha_k$, $P \cdot Q_1 \cdots Q_k \Downarrow \underline{n}$ if and only if $P' \cdot Q_1 \cdots Q_k \Downarrow \underline{n}$.*

Note that applicative equivalence is only defined on PCF programs, while observational equivalence is defined on all PCF terms.

Proposition 1.3.4 ([Mil77], [Ong95])**.**
For all PCF *programs P, P' of type α, we have:*

$$P \equiv_{obs} P' \Leftrightarrow P \equiv_{app} P'$$

1.3.2 Denotational Semantics of PCF

When defining an abstract programming language, it has become the norm to consider both the operational and denotational semantics for programs in the language. While operational semantics describe how terms in a language are executed, such as the previously defined small-step and big-step reductions, denotational semantics aim to describe the "meaning" of a term. This involves an assignment from programs to some values of a mathematical structure. Identifying a denotational semantics for a programming language is also referred to as constructing a *model* of said programming.

The interest in constructing models of a language lies in the ability to use mathematical "tools" from whichever domain the model was constructed in. For example, it is well known that any cartesian closed category[5] can be used to model the simply typed λ-calculus. Regarding models of the untyped λ-calculus, refer to Giulio Manzonetto's PhD thesis [Man08]. We will define models of PCF.

Definition 1.3.5 (Models of PCF [Mil77])**.**
A model of PCF *is a triple* $M = \langle (\mathrm{M}_\alpha)_{\alpha\in\mathbb{T}}, (\ \cdot^{\alpha,\beta})_{\alpha,\beta\in\mathbb{T}}, [\![-]\!]\rangle$ *where:*

- $(\mathrm{M}_\alpha)_{\alpha\in\mathbb{T}}$ *is a type-indexed collection of sets;*
- $(\ \cdot^{\alpha,\beta}) : \mathrm{M}_{\alpha\to\beta} \times \mathrm{M}_\alpha \to \mathrm{M}_\beta$ *is a well-typed operation called* application*;*
- $[\![-]\!]$ *is an* interpretation function *mapping a derivation of* $x_1 : \beta_1, \ldots, x_n : \beta_n \vdash P : \alpha$ *to an element* $[\![x_1 : \beta_1, \ldots, x_n : \beta_n \vdash P : \alpha]\!] \in \mathrm{M}_{\beta_1\to\cdots\to\beta_n\to\alpha}$.

Due to syntax-directedness of PCF *type system, we can simply write* $[\![P]\!]^{\Gamma,\alpha}$ *for* $[\![\Gamma \vdash P : \alpha]\!]$.

For the sake of readability, we write $a \cdot b$ for $a \cdot^{\alpha,\beta} b$ when α, β are clear from the context.

When constructing a model, we require a certain level of "correctness". At the very minimum, we would like the model to be *consistent* in the sense that it does not equate all programs, and that it be *sound* in that it equates inter-convertible terms. Ideally, we would like to find a model which equates all terms displaying the same computational behaviour.

Definition 1.3.6 (Properties of a Model)**.**
Let $\mathcal{L}$ *be an arbitrary language, and let* $\equiv_{\mathcal{L}}$ *be a an equivalence between terms in* $\mathcal{L}$*. Let* $\mathscr{M}$ *be a model of* $\mathcal{L}$*, whose interpretation function is* $[\![-]\!]$.

(1) The model is said to be adequate *if for all* $A, B \in \mathcal{L}$*,*

$$A \equiv_{\mathcal{L}} B \quad \Leftarrow \quad [\![A]\!] = [\![B]\!]$$

(2) The model is said to be complete *if for all* $A, B \in \mathcal{L}$*,*

$$A \equiv_{\mathcal{L}} B \quad \Rightarrow \quad [\![A]\!] = [\![B]\!]$$

(3) A model which is both adequate and complete is said to be fully abstract*.*

Full abstraction is the strongest notion of correctness between a language and a model. The quest to find a fully abstract model of PCF was put forward in the same paper in which it was

[5] Cartesian closed categorys are defined in a later chapter.

introduced [Plo77]. It also showed that the classical approach [Sco70], while producing a model that is adequate, was not complete. Robin Milner later found a solution by constructing a fully abstract model based on continuous functions [Mil77]. However, his model is syntactic in nature, essentially constructed directly from the operational semantics, and is thus generally seen as unsatisfactory as a result [Ong95]. His results are still of value though, as he also shows that all fully abstract models of PCF are isomorphic to one another – though this intuitively makes sense, it is not the case for all language [Bre15].

Over the years, many attempts at finding a more satisfactory fully abstract model were made, and around the turn of the century it was one of the longest standing problems in the semantics of programming languages [AMJ94, Cur07]. This was at least partially due to some disagreement about what constitutes a "satisfactory" model. At the very least, a decoupling of the model from the syntax and operational semantics of PCF is desired [OR95, AMJ94]. Some have also argued that the model should be represented as a ($\mathbf{CPO}$-enriched) cartesian closed category[6] [AMJ94, JS93]. The fully abstract models which one might discuss today can be (roughly) separated into three groups:

- The classic variant based on continuous functions by Milner [Mil77];
- Models based on game semantics. [AMJ94, Nic94, HO00]
- Every other model, such as those based on realisability techniques [MRS99] or on Kripke logical relations [OR95].

Today, a common consensus has been reached that the game semantics models are indeed "satisfactory" [Cur07]. This is partially as they satisfy the demands mentioned earlier, and partially due to a result from Loader [Loa01] that the observational equivalence of (the strongly normalising fragment of) PCF is undecidable, limiting how effectively a fully abstract model can be presented.

One common "flaw" of the game semantics and operational models is their reliance on a "quotient" to construct their model. Essentially, there are PCF terms which are observationally equivalent, yet distinct in the structure upon which the model is based. Only after a semantic quotienting action, stating that two objects in the model are equal based on some other property in the source structure, is full abstraction obtained. There do exist models which do not make use of such a quotient, though [OR95, CLM10].

Regardless of the perceived intrinsic value of new fully abstract models of PCF for better understanding of PCF, all such models being isomorphic means that new models can still have value by furthering our understanding of the tools and systems used to construct such models in

[6] This type of model is defined in a later chapter.

the past. Indeed, in [Loa01], Loader argues that it is precisely these techniques resulting from the study of PCF that are important, moreso than the results obtained from PCF itself.

Chapter 2

Explicit Substitutions

2.1 Explicit Substitutions in the λ-Calculus

Substitution is an integral part of the λ-calculus and its derivatives. Presented as it has been in Definition 1.1.3, it appears to be very simple, but this presentation hides its unusual aspects. When reading a substitution $M[N/x]$, it is implied that N replaces all occurrences of x in M simultaneously regardless of the shape of M or the number of occurrences of x.

Implementations of substitution (usually) take a different approach. Depending on the number of occurrences of x and the size of N, an immediate resolution of the substitution could result in a "size explosion". The substitution is instead typically delayed and stored alongside the term – N is then substituted for x whenever x is used during the evaluation.

2.1.1 The $\lambda\sigma$-Calculus

To reconcile this difference between presentation and implementation, Abadi et. al. introduced the notion of *explicit substitutions* [ACCL91]. The general idea is to introduce substitutions as a part of the syntax rather than as a meta-notation. These substitutions are then stored alongside the term as it reduces, with elements being substituted only when necessary. This results in a calculus which not only corresponds more closely to how implementations would commonly be designed, but acts in a manner quite similar to common abstract machines.

Definition 2.1.1 (The $\lambda\sigma$-calculus [ACCL91])**.**

(1) Define the set Λ^σ of $\lambda\sigma$ terms by extending the grammar of the λ-calculus with a case

representing an explicit substitution as follows[1]*:*

$$M, N \quad ::= \quad x \mid MN \mid \lambda x.M \mid M\langle N/x\rangle$$

(2) We give substitution priority over abstraction and application and assume that explicit substitution associates to the left, i.e.

$$\begin{aligned} \lambda x.M\langle N/y\rangle &= \lambda x.(M\langle N/y\rangle) \\ M_1M_2\langle N/y\rangle &= M_1(M_2\langle N/y\rangle) \\ M\langle N_1/x_1\rangle\cdots\langle N_n/x_n\rangle &= (\cdots((M\langle N_1/x_1\rangle)\langle N_2/x_2\rangle)\cdots)\langle N_n/x_n\rangle \end{aligned}$$

(3) The set $\mathrm{FV}(M)$ *of* free variables *of a* $\lambda\sigma$ *term is defined as in the* λ*-calculus, with the addition of the below case.*

$$\mathrm{FV}(M\langle N/x\rangle) := (\mathrm{FV}(M) \setminus \{x\}) \cup \mathrm{FV}(N)$$

(4) Capture free substitution *is defined by extending the capture free substitution of the* λ*-calculus with a case for the explicit substitution.*

$$(M\langle N/x\rangle)[L/y] = \begin{cases} M\langle N/x\rangle & \text{if } x = y, \\ M[L/y]\langle N[L/y]/x\rangle & \text{otherwise.} \end{cases}$$

(5) α-equivalence *and the* variable convention *are defined as in the* λ*-calculus, making use of the capture free substitution case defined above.*

(6) Reduction *is defined through a collection of reduction rules.*

$$\begin{aligned} (\lambda x.M)N &\to_\sigma M\langle N/x\rangle && && (\beta) \\ x\langle M/x\rangle &\to_\sigma M && && \text{(Var1)} \\ y\langle M/x\rangle &\to_\sigma y, && \text{where } x \neq y && \text{(Var2)} \\ (MN)\langle L/x\rangle &\to_\sigma (M\langle L/x\rangle)(N\langle L/x\rangle) && && \text{(App)} \\ (\lambda x.M)\langle L/x\rangle &\to_\sigma \lambda x.M && && \text{(Abs1)} \\ (\lambda y.M)\langle L/x\rangle &\to_\sigma \lambda z.(M\langle z/y\rangle\langle L/x\rangle), && \text{where } x \neq y, z \notin \mathrm{FV}(ML) && \text{(Abs2)} \end{aligned}$$

We finish the definition of the $\to_\sigma$ reduction *with the addition of the following inference*

[1]The $\lambda\sigma$-calculus was originally presented primarily using de Bruijn notation [Bru72]. This presentation uses traditional variables instead and is slightly different from the presentations shown in the paper.

rules:

$$\frac{M \to_\sigma M'}{\lambda x.M \to_\sigma \lambda x.M'} \qquad \frac{M \to_\sigma M'}{MN \to_\sigma M'N} \qquad \frac{N \to_\sigma N'}{MN \to_\sigma MN'}$$

$$\frac{M \to_\sigma M'}{M\langle N/x\rangle \to_\sigma M'\langle N/x\rangle} \qquad \frac{N \to_\sigma N'}{M\langle N/x\rangle \to_\sigma M\langle N'/x\rangle}$$

This definition can be extended with further $\to_\sigma$ reduction rules, eliminating unnecessary substitutions. Of particular interest is a rule which would allow substitutions to be reordered. With the current definition it is not possible to evaluate substitutions independently from one another – terms with multiple consecutive substitutions such as $t\langle x/y\rangle\langle u/t\rangle$ necessitate the evaluation of $\langle x/y\rangle$ before $\langle u/t\rangle$. This rule would be

$$M\langle N/x\rangle\langle L/y\rangle \to_\sigma M\langle L/y\rangle\langle N\langle L/y\rangle/x\rangle, \text{where } x \notin \mathrm{FV}(L) \text{ by } \alpha\text{-equivalence.}$$

The introduction of this rule also resolves a mismatch between the reductions of the $\lambda\sigma$-calculus and its capture free substitutions. However, a degeneracy arises – as the reduct has the shape of the redex, the new rule loops by itself. The addition of side conditions results in other pitfalls, as shown by Melliès [Mel95]. This means that many terms which would be strongly normalising in the untyped λ-calculus are now only weakly normalising.

2.1.2 The Linear Substitution Calculus

Over the years, a number of different calculi based on explicit substitutions were introduced to improve the rewriting issues, or provide the calculus with nicer properties regarding confluence and normalisation [LM99, CHL96, BBLRD95, DCK97]. Finally, in recent years a new calculus was introduced which resolves Melliès' degeneracy and has all the properties (confluence, full composition, preservation of strong normalisation, ...) one would desire of a calculus with explicit substitutions – the *Linear Substitution Calculus (LSC)* [Acc18, AK10].

Definition 2.1.2 (Linear Substitution Calculus)**.**

(1) The terms and contexts of the Linear Substitution Calculus are defined as follows:

$$\begin{array}{rrcl} \textit{LSC Terms} & M, N & ::= & x \mid \lambda x.M \mid MN \mid M\langle N/x\rangle \\ \textit{Contexts} & L\Box & ::= & \Box \mid \lambda x.L\Box \mid L\Box M \mid ML\Box \\ & & & \mid L\Box\langle M/x\rangle \mid M\langle L\Box/x\rangle \\ \textit{Substitution Contexts} & S\Box & ::= & \Box \mid S\Box\langle M/x\rangle \end{array}$$

(2) There are three rewriting rules present in the Linear Substitution Calculus.
The α-equivalence is used when necessary to avoid variable capture occurring.

$$\begin{array}{rrcl} \textit{Distant B} & S[\lambda x.M]N & \to_{\mathsf{dB}} & S[M\langle N/x\rangle] \\ \textit{Linear Substitution} & L[x]\langle M/x\rangle & \to_{\mathsf{ls}} & L[M]\langle M/x\rangle \\ \textit{Garbage Collection} & M\langle N/x\rangle & \to_{\mathsf{gc}} & M, \text{ where } x \notin \mathrm{FV}(M) \end{array}$$

These reductions are closed under LSC contexts. For $\mathsf{a} \in \{\mathsf{dB}, \mathsf{ls}, \mathsf{gc}\}$*,*

$$\frac{M \to_{\mathsf{a}} M'}{L[M] \to_{\mathsf{a}} L[M']}$$

Essentially, reduction of an explicit substitution $M\langle N/x\rangle$ occurs by substituting N for one occurrence of x in M with each reduction step, regardless of where x appears in M. This approach makes explicit the duplication of arguments native to the λ-calculus. It also separates the concepts of searching for free occurrences of x from reducing the term.

This does come with a downside, however – the search for and substitution of x at any depth does not come naturally from within the language. This is very different from how substitutions are handled in abstract machines with local environments – unlike classic explicit substitutions, which "push" explicit substitutions through their terms much like how such abstract machines function.

2.2 Explicit Substitutions for PCF

2.2.1 Extending PCF with Explicit Substitutions

Unlike the λ-calculus, when considering explicit substitutions for PCF one can simply sidestep most of the issues. PCF is generally considered with a deterministic reduction strategy, so one does not need to concern themselves with confluence, and preservation of strong normalisation boils down to not diverging from standard PCF. To simplify working with explicit substitutions, we can also (ab)use the type system to force our explicit substitutions to be closed. We shall call PCF endowed with closed explicit substitutions EPCF. These explicit substitutions will in fact be immutable, and as a consequence we need to use a non-standard definition of the set of free variables of a term – intuitively, we take the standard definition, and adjust it so that free variables inside explicit substitutions can no longer be captured. We will define the set first, and provide a more detailed explanation later.

Definition 2.2.1 (EPCF)**.**

(1) The set Λ^{E} *of* EPCF terms *is defined by the grammar (for* $x \in \mathrm{Var}$*):*

$$\begin{array}{rcl} L, M, N & ::= & x \mid MN \mid \lambda x.M \mid \mathbf{fix}\, M \mid M\langle N/x\rangle \mid \\ & & \mid \mathbf{0} \mid \mathbf{pred}\, M \mid \mathbf{succ}\, M \mid \mathbf{ifz}(L, M, N) \end{array} \qquad (\Lambda^{\mathsf{E}})$$

(2) When considering a (possibly empty) list of explicit substitutions

$$\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$$

we assume that the variables $\vec{x}$ *in* σ *are pairwise distinguished. Given* σ *as above, we let* $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$ *be the domain of* σ *and write* $\sigma(x_i) = N_i$*, for* $x_i \in \mathrm{dom}(\sigma)$*. Note that, unlike in previous systems, the order of the explicit substitutions does not matter.*

(3) Given $M \in \Lambda^{\mathsf{E}}$*, we write* M^σ *for* $M\langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$*.*

(4) The set $\mathrm{Val}^{\mathsf{E}}$ *of* EPCF values *contains abstractions under substitutions and numerals, i.e.*

$$\mathrm{Val}^{\mathsf{E}} = \{\underline{n} \mid n \in \mathbb{N}\} \cup \{(\lambda x.M)^\sigma \mid M \in \Lambda^{\mathsf{E}}\}$$

(5) Given $M \in \Lambda^{\mathsf{E}}$*, we define* $\mathrm{TFV}(M) \subset \{0,1\} \times \mathrm{Var}$ *the set of all free variables of* M *with an additional "tracker" by induction on* M*. The tracker is necessary to differentiate free variables which occur in* EPCF *substitutions, as these free variables cannot later be bound by an abstraction or substitution. We define two functions* exclude *and* forget*:*

$$\begin{array}{rcl} \mathsf{exclude}\left(\left\{\begin{array}{l}(0,x_1),\ldots,(0,x_n),\\(1,y_1),\ldots,(1,y_m)\end{array}\right\}\right) & = & \{(1,x_1),\ldots,(1,x_n),(1,y_1),\ldots,(1,y_m)\} \\ \mathsf{forget}\left(\left\{\begin{array}{l}(0,x_1),\ldots,(0,x_n),\\(1,y_1),\ldots,(1,y_m)\end{array}\right\}\right) & = & \{x_1,\ldots,x_n,y_1,\ldots,y_m\} \end{array}$$

$$\begin{array}{rcl} \mathrm{TFV}(x) & := & \{(0,x)\}, \\ \mathrm{TFV}(\mathbf{0}) & := & \emptyset, \\ \mathrm{TFV}(\lambda x.P) & := & \mathrm{TFV}(P) \setminus \{(0,x)\}, \\ \mathrm{TFV}(PQ) & := & \mathrm{TFV}(P) \cup \mathrm{TFV}(Q), \\ \mathrm{TFV}(\mathbf{pred}\, P) & := & \mathrm{TFV}(P) \\ \mathrm{TFV}(\mathbf{succ}\, P) & := & \mathrm{TFV}(P), \\ \mathrm{TFV}(\mathbf{ifz}(P,Q_1,Q_2)) & := & \mathrm{TFV}(P) \cup \mathrm{TFV}(Q_1) \cup \mathrm{TFV}(Q_2) \\ \mathrm{TFV}(\mathbf{fix}\, P) & := & \mathrm{TFV}(P), \\ \mathrm{TFV}(M\langle N/x\rangle) & := & (\mathrm{TFV}(M) \setminus \{(0,x)\}) \cup \mathsf{exclude}(\mathrm{TFV}(N)). \end{array}$$

The set $\mathrm{FV}(M)$ *is defined with respect to* $\mathrm{TFV}(M)$*.*

$$\mathrm{FV}(M) = \mathsf{forget}(\mathrm{TFV}(M))$$

We define α-equivalence as usual, with the introduction of an additional case.

$$M\langle N/x\rangle =_\alpha M[y/x]\langle N/y\rangle, \text{ where } y \text{ is fresh for } M \text{ and } N$$

(6) Capture free substitution *as defined for* PCF *is extended with an additional case for the explicit substitution, namely:*

$$(M\langle N/x\rangle)[L/y] = (M[z/x][L/y])\langle N/z\rangle, \text{ where } y \text{ is fresh for } M, N, \text{ and } L$$

(7) An EPCF *term M is called* an EPCF program*, written $M \in \mathcal{P}_{\mathsf{E}}$, if it is closed (i.e. $\mathrm{FV}(M) = \emptyset$).*

Notice that, in an EPCF program, all subterms of the form $M\langle N/x\rangle$ must have $N \in \mathcal{P}_{\mathsf{E}}$. Clearly $\Lambda^{\mathsf{PCF}} \subsetneq \Lambda^{\mathsf{E}}$, moreover all PCF programs belong to $\mathcal{P}_{\mathsf{E}}$. The immutability of the explicit substitutions causes the free variables found within an explicit substitution to essentially be "permanently" free – they cannot be captured by an external abstraction. The non-standard definition of free variables arises from this unusual property. The immutability property is certainly not intuitive, and the resulting definitions can only hold for weak head reduction strategies – reduction under abstraction would otherwise lead to the "creation" of free variables. We will go into further detail on this after defining the operational semantics of EPCF.

Example 2.2.2.
All PCF *terms introduced previously are also* EPCF *terms. Some examples of* EPCF *programs that are not* PCF *terms are $x\langle\underline{1}/x\rangle$ and $\lambda x.(\mathbf{ifz}(x, y, z)\langle\underline{1}/y\rangle\langle\underline{2}/z\rangle)$.*

We endow EPCF with a small-step call-by-name operational semantics capturing weak head reduction.

Definition 2.2.3.

(1) The computation reduction $\to_{\mathsf{cr}}$ *on* EPCF *terms is defined as:*

$$\begin{array}{rcl@{\qquad}rcl}
(\lambda x.M)^\sigma N & \to_{\mathsf{cr}} & M^\sigma\langle N/x\rangle, & \mathbf{pred}\,\mathbf{0} & \to_{\mathsf{cr}} & \mathbf{0}, \\
\mathbf{ifz}(\mathbf{0}, M, N) & \to_{\mathsf{cr}} & M, & \mathbf{pred}\,(\mathbf{succ}\,(\underline{n})) & \to_{\mathsf{cr}} & \underline{n}, \\
\mathbf{ifz}(\underline{n+1}, M, N) & \to_{\mathsf{cr}} & N, & \mathbf{fix}\,(M) & \to_{\mathsf{cr}} & M(\mathbf{fix}\,(M)).
\end{array}$$

(2) The percolation reduction $\to_{\mathsf{pr}}$ *on* EPCF *terms is defined as (where σ is non-empty):*

$$\begin{array}{rcl@{\qquad}rcl}
x^\sigma & \to_{\mathsf{pr}} & N, \text{ if } \sigma(x) = N, & \mathbf{0}^\sigma & \to_{\mathsf{pr}} & \mathbf{0}, \\
y^\sigma & \to_{\mathsf{pr}} & y, \text{ if } y \notin \mathrm{dom}(\sigma), & (MN)^\sigma & \to_{\mathsf{pr}} & M^\sigma N^\sigma, \\
(\mathbf{pred}\,(M))^\sigma & \to_{\mathsf{pr}} & \mathbf{pred}\,(M^\sigma), & (\mathbf{succ}\,(M))^\sigma & \to_{\mathsf{pr}} & \mathbf{succ}\,(M^\sigma), \\
(\mathbf{ifz}(L, M, N))^\sigma & \to_{\mathsf{pr}} & \mathbf{ifz}(L^\sigma, M^\sigma, N^\sigma), & (\mathbf{fix}\,(M))^\sigma & \to_{\mathsf{pr}} & \mathbf{fix}\,(M^\sigma).
\end{array}$$

(3) The reductions $\to_{\mathsf{cr}}$ and $\to_{\mathsf{pr}}$ are closed under head contexts, namely (for $a \in \{\to_{\mathsf{cr}}, \to_{\mathsf{pr}}\}$):

$$\frac{M \to_a M'}{MN \to_a M'N} \qquad \frac{M \to_a M'}{\mathbf{pred}\, M \to_a \mathbf{pred}\, M'}$$

$$\frac{M \to_a M'}{\mathbf{ifz}(M, N_1, N_2) \to_a \mathbf{ifz}(M', N_1, N_2)} \qquad \frac{M \to_a M'}{\mathbf{succ}\, M \to_a \mathbf{succ}\, M'}$$

(4) The (one step) weak head (w.h.) reduction $\to_{\mathtt{wh}}$ *is defined as the union of $\to_{\mathsf{cr}}$ and $\to_{\mathsf{pr}}$.*

(5) As it is customary, we denote by $\twoheadrightarrow_{\mathtt{wh}}$ the transitive-reflexive closure of $\to_{\mathtt{wh}}$.

(6) We define $\leftrightarrow_{\mathtt{wh}}$ as the symmetric, transitive and reflexive closure of $\to_{\mathtt{wh}}$.

EPCF is a language primarily designed with EPCF programs in mind. Making explicit substitutions immutable simplifies them significantly, while losing some of their flexibility. This does result in some interactions which appear strange at first glance, such as $\lambda x.(y\langle x/y\rangle)$ not being equivalent to $\lambda x.x$. An example using reduction would be the following:

$$(\lambda x.(y\langle x/y\rangle))\mathbf{0} \to_{\mathtt{wh}} y\langle x/y\rangle\langle \mathbf{0}/x\rangle \to_{\mathtt{wh}} x$$

These interactions seem a bit less unexpected when one considers that

$$\begin{array}{rcl} \lambda x.(y\langle x/y\rangle) & =_\alpha & \lambda z.(y\langle x/y\rangle) \\ (\lambda x.(y\langle x/y\rangle))\mathbf{0} & =_\alpha & (\lambda z.(y\langle x/y\rangle))\mathbf{0} \end{array}$$

Essentially, when given a term $M\langle N/x\rangle$, the free variables of N are stuck in a sort of "limbo". Despite being free variables, they cannot be captured by any abstraction or substitution. Thus the only valid reduction strategies are weak head reduction, as reducing under an abstraction would instantly lead to loss of confluence and require a change in the definition of α-equivalence and $\mathrm{FV}(M)$ (using $\to_{\mathsf{wrong}}$ for this hypothetical reduction strategy):

$$(\lambda x.(\lambda y.y)x)\underline{0} \to_{\mathsf{wrong}} (\lambda x.y\langle y/x\rangle)\underline{0} \to_{\mathtt{wh}} y\langle y/x\rangle\langle \underline{0}/x\rangle \to_{\mathtt{wh}} x$$
$$(\lambda x.(\lambda y.y)x)\underline{0} \to_{\mathsf{wrong}} (\lambda x.y\langle y/x\rangle)\underline{0} \to_{\mathsf{wrong}} (\lambda x.x)\underline{0} \to_{\mathtt{wh}} x\langle \underline{0}/x\rangle \to_{\mathtt{wh}} 0$$
$$(\lambda x.(\lambda y.y)x)\underline{0} \to_{\mathsf{wrong}} (\lambda x.y\langle y/x\rangle)\underline{0} =_\alpha (\lambda z.y\langle y/x\rangle)\underline{0} \to_{\mathsf{wrong}} (\lambda z.x)\underline{0} \to_{\mathtt{wh}} x\langle \underline{0}/z\rangle \to_{\mathtt{wh}} x$$

We will later rely on the type system to forbid free variables in explicit substitutions.

This strange inherent property is why EPCF cannot claim to be a completely satisfactory presentation of PCF with explicit substitutions – for a satisfactory system, one would expect to turn to the Linear Substitution Calculus for inspiration. However, when one is only interested in observational equivalence, EPCF is perfectly sufficient.

Lemma 2.2.4.
Given $M, N \in \Lambda^{\mathsf{E}}$ such that $M \to_{\mathtt{wh}} N$, $\mathrm{FV}(N) \subseteq \mathrm{FV}(M)$.

Proof. By induction on a derivation of $M \to_{\mathtt{wh}} N$.

- $M = (\lambda x.M')^{\sigma} N'$: Let $\sigma = \langle L_1/y_1\rangle \cdots \langle L_n/y_n\rangle$. Then we have

$$\begin{aligned} \mathrm{FV}((\lambda x.M')^{\sigma} N') &= \mathsf{forget}\left[\begin{array}{l}(\mathrm{TFV}(M') \setminus \{(0,x),(0,y_1),\ldots,(0,y_n)\}) \\ \cup \mathrm{TFV}(N') \cup \mathsf{exclude}(\mathrm{TFV}(L_1 \cdots L_n))\end{array}\right] \\ &= \mathsf{forget}\left[\begin{array}{l}(\mathrm{TFV}(M') \setminus \{(0,x),(0,y_1),\ldots,(0,y_n)\}) \\ \cup \mathsf{exclude}(\mathrm{TFV}(N' L_1 \cdots L_n))\end{array}\right] \\ &= \mathrm{FV}((M')^{\sigma}\langle N'/x\rangle) = \mathrm{FV}(N) \end{aligned}$$

- All other cases not closed under head contexts are trivial. As an example of a case where $\mathrm{FV}(N) \subsetneq \mathrm{FV}(M)$, see $(\underline{0})^{\sigma} \to_{\mathtt{wh}} \underline{0}$.

- $M = M_1 M_2$ and $N = N_1 M_2$. We have $M_1 \to_{\mathtt{wh}} N_1$, so by IH we have $\mathrm{FV}(N_1) \subseteq \mathrm{FV}(M_1)$ and conclude as $\mathrm{FV}(N) = \mathrm{FV}(N_1) \cup \mathrm{FV}(M_2) \subseteq \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) = \mathrm{FV}(M)$.

- All other head context cases are analogous. □

Corollary 2.2.5.
The set $\mathcal{P}_{\mathsf{E}}$ is closed under $\to_{\mathtt{wh}}$, hence under $\twoheadrightarrow_{\mathtt{wh}}$ as well.

Remark 2.2.6.

(1) The w.h. reduction is deterministic: $N_1 \,{}_{\mathtt{wh}}\!\leftarrow M \to_{\mathtt{wh}} N_2$ implies $N_1 = N_2$.

(2) Immutable explicit substitution, while valid for systems with a deterministic weak head reduction, would result in the loss of confluence for the λ-calculus – one needs only consider $((\lambda x.x)y)\langle z/y\rangle$ for an example.

As a consequence, we can safely write $|M \twoheadrightarrow_{\mathtt{wh}} N|$ to denote the length n of the unique reduction sequence $M = M_1 \to_{\mathtt{wh}} M_2 \to_{\mathtt{wh}} \cdots \to_{\mathtt{wh}} M_n = N$.

Lemma 2.2.7.
The following hold for EPCF *terms which reduce to values.*

1. *If $MN \twoheadrightarrow_{\mathtt{wh}} V$, then there exist M', σ such that $M \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M')^{\sigma}$;*
2. *If $\mathbf{pred}\, M \twoheadrightarrow_{\mathtt{wh}} V$, then $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$;*
3. *If $\mathbf{succ}\, M \twoheadrightarrow_{\mathtt{wh}} V$, then $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$;*
4. *If $\mathbf{ifz}(M, N_1, N_2) \twoheadrightarrow_{\mathtt{wh}} V$, then $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, for some $\underline{n} \in \mathbb{N}$;*

Proof. We will prove the statement for $MN \twoheadrightarrow_{\mathtt{wh}} V$ by induction on the length k of the reduction.

Case $k = 0$: Vacuous, as MN is not a value.

Case $k > 0$: There are two subcases.

Subcase $M \in \{(\lambda x.L)^\sigma \mid L \in \Lambda^{\mathsf{E}}\}$: M has the form desired by the statement, so we are done.

Subcase $M \notin \{(\lambda x.L)^\sigma \mid L \in \Lambda^{\mathsf{E}}\}$: Then $MN \to_{\mathtt{wh}} M'N$ and so $M \to_{\mathtt{wh}} M'$. We apply the IH on $M'N \twoheadrightarrow_{\mathtt{wh}} V$ and conclude.

All other statements are analogous. □

Definition 2.2.8.
From $\to_{\mathtt{wh}}$ and mirroring the big-step reduction of PCF*, we can derive a* big-step reduction $\Downarrow^{\mathsf{E}} \subseteq \Lambda^{\mathsf{EPCF}} \times \mathrm{Val}^{\mathsf{E}}$ *relating an* EPCF *term with an* EPCF *value:*

$$\frac{\underline{n} \in \mathbb{N}}{(\underline{n})^\sigma \Downarrow^{\mathsf{E}} \underline{n}}\ (\mathrm{nat}^{\mathsf{E}}) \qquad \frac{}{(\lambda x.M)^\sigma \Downarrow^{\mathsf{E}} (\lambda x.M)^\sigma}\ (\lambda^{\mathsf{E}}) \qquad \frac{\sigma(x) = N \quad N \Downarrow^{\mathsf{E}} V}{x^\sigma \Downarrow^{\mathsf{E}} V}\ (\mathrm{var}^{\mathsf{E}})$$

$$\frac{M^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}}{(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}}\ (\mathrm{pr}_0{}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} \underline{n+1}}{(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n}}\ (\mathrm{pr}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} \underline{n}}{(\mathbf{succ}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n+1}}\ (\mathrm{sc}^{\mathsf{E}})$$

$$\frac{L^\sigma \Downarrow^{\mathsf{E}} \mathbf{0} \quad M^\sigma \Downarrow^{\mathsf{E}} V_1}{(\mathbf{ifz}(L, M, N))^\sigma \Downarrow^{\mathsf{E}} V_1}\ (\mathrm{ifz}_0{}^{\mathsf{E}}) \qquad \frac{L^\sigma \Downarrow^{\mathsf{E}} \underline{n+1} \quad N^\sigma \Downarrow^{\mathsf{E}} V_2}{(\mathbf{ifz}(L, M, N))^\sigma \Downarrow^{\mathsf{E}} V_2}\ (\mathrm{ifz}_{>0}{}^{\mathsf{E}})$$

$$\frac{M^\sigma \mathbf{fix}\,(M^\sigma) \Downarrow^{\mathsf{E}} V}{(\mathbf{fix}\, M)^\sigma \Downarrow^{\mathsf{E}} V}\ (\mathrm{fix}^{\mathsf{E}}) \qquad \frac{M^\sigma \Downarrow^{\mathsf{E}} (\lambda x.M')^{\sigma'} \quad (M')^{\sigma'}\langle N^\sigma/x\rangle \Downarrow^{\mathsf{E}} V}{(MN)^\sigma \Downarrow^{\mathsf{E}} V}\ (\beta_v^{\mathsf{E}})$$

Proposition 2.2.9.
Given an EPCF *program M and an* EPCF *value V, we have*

$$M \Downarrow^{\mathsf{E}} V \iff M \twoheadrightarrow_{\mathtt{wh}} V.$$

Proof. We prove both halves independently.

($\Rightarrow$) We refer to the statement $M \Downarrow^{\mathsf{E}} V \Rightarrow M \twoheadrightarrow_{\mathtt{wh}} V$ as IH and proceed by induction on the height of a derivation of $M \Downarrow^{\mathsf{E}} V$:

- $(\underline{n})^\sigma \Downarrow^{\mathsf{E}} \underline{n}$: Base case, by percolation $(\underline{n})^\sigma \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ holds.
- $(\lambda x.M)^\sigma \Downarrow^{\mathsf{E}} (\lambda x.M)^\sigma$: Base case, as above.
- $x^\sigma \Downarrow^{\mathsf{E}} V$: As $\sigma(x) = N$, we have $x^\sigma \to_{\mathtt{wh}} N$. Conclude by applying the IH to $N \Downarrow^{\mathsf{E}} V$.

- $(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}$: We have $(\mathbf{pred}\, M)^\sigma \to_{\mathtt{wh}} \mathbf{pred}\, (M^\sigma)$. By IH we get $M^\sigma \twoheadrightarrow_{\mathtt{c}} \mathbf{0}$, so we can conclude as $\mathbf{pred}\, \mathbf{0} \to_{\mathtt{wh}} \mathbf{0}$.

- $(\mathbf{pred}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n}$: proceed as above, using $\mathbf{pred}\, (\mathbf{succ}\, (\underline{n})) \to_{\mathtt{wh}} \underline{n}$.

- $(\mathbf{succ}\, M)^\sigma \Downarrow^{\mathsf{E}} \underline{n+1}$: $(\mathbf{succ}\, M)^\sigma \to_{\mathtt{wh}} \mathbf{succ}\, (M^\sigma)$. By IH, $M^\sigma \twoheadrightarrow_{\mathtt{wh}} \underline{n}$, so we conclude, remembering that $\underline{n} = \mathbf{succ}^{\,n}(\mathbf{0})$.

- $(\mathbf{ifz}(L, M, N))^\sigma \Downarrow^{\mathsf{E}} V_1$: $(\mathbf{ifz}(L, M, N))^\sigma \to_{\mathtt{wh}} \mathbf{ifz}(L^\sigma, M^\sigma, N^\sigma)$. By IH we have $L^\sigma \twoheadrightarrow_{\mathtt{wh}} \underline{0}$, so $\mathbf{ifz}(L^\sigma, M^\sigma, N^\sigma) \twoheadrightarrow_{\mathtt{wh}} M^\sigma$. Conclude, as by IH $M^\sigma \twoheadrightarrow_{\mathtt{wh}} V_1$.

- $(\mathbf{ifz}(L, M, N))^\sigma \Downarrow^{\mathsf{E}} V_2$: Proceed as above.

- $(\mathbf{fix}\, M)^\sigma \Downarrow^{\mathsf{E}} V$: We have $(\mathbf{fix}\, M)^\sigma \to_{\mathtt{wh}} \mathbf{fix}\, (M^\sigma) \to_{\mathtt{wh}} M^\sigma \mathbf{fix}\, (M^\sigma)$. Conclude by IH.

- $(MN)^\sigma \Downarrow^{\mathsf{E}} V$: We have $(MN)^\sigma \to_{\mathtt{wh}} M^\sigma N^\sigma$. By IH, we have $M^\sigma \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M')^{\sigma'}$. As $(\lambda x.M')^{\sigma'} N^\sigma \to_{\mathtt{wh}} (M')^{\sigma'}\langle N^\sigma/x\rangle$, we conclude using the IH.

($\Leftarrow$) We refer to the statement $M \Downarrow^{\mathsf{E}} V \Leftarrow M \twoheadrightarrow_{\mathtt{wh}} V$ as IH and proceed by induction on the length $k = |M \twoheadrightarrow_{\mathtt{wh}} V|$.

Case $k = 0$. Then $M = V$. Apply either $(\mathrm{nat}^{\mathsf{E}})$ or (λ^{E}), depending on the shape of V.

Case $k > 0$. Then $M \to_{\mathtt{wh}} N \twoheadrightarrow_{\mathtt{wh}} V$, for some $N \in \Lambda^{\mathsf{E}}$ with $|N \twoheadrightarrow_{\mathtt{wh}} V| = k - 1 < k$. We proceed by induction on the shape of M.

- Subcase $M = x$ does not apply, as x does not reduce to a value.

- Subcases $M = \underline{0}$ and $M = \lambda x.M$ are already values and do not reduce.

- Subcase $M = M_1 M_2$: Since $M \twoheadrightarrow_{\mathtt{wh}} V$, we use Lemma 2.2.7 to obtain

$$M_1 M_2 \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M_1')^\sigma M_2 \to_{\mathtt{wh}} M_1'^\sigma[x/M_2] \twoheadrightarrow_{\mathtt{wh}} V$$

  with $M_1 \twoheadrightarrow_{\mathtt{wh}} (\lambda x.M_1')^\sigma$ and $M_1'^\sigma[x/L] \twoheadrightarrow_{\mathtt{wh}} V$ shorter than k. Using the IH, we obtain $M_1 \Downarrow^{\mathsf{E}} (\lambda x.M_1')^\sigma$ and $M_1'^\sigma[x/L] \Downarrow^{\mathsf{E}} V$, so we conclude by (β_v^{E}).

- Subcase $M = (M')^\sigma$. There are 9 (sub)subcases.

  - $M' = (\lambda x.M'')$ is already a value, and does not reduce.
  - $M' = x$ and $\sigma(x) = N$: By IH we obtain $N \Downarrow^{\mathsf{E}} V$. We apply $(\mathrm{var}^{\mathsf{E}})$ to infer $x^\sigma \Downarrow^{\mathsf{E}} V$.
  - $M' = y$, where $y \notin \mathrm{dom}(\sigma)$, does not apply as it is neither a value nor does it reduce.
  - $M' = \mathbf{0}$: We apply $(\mathrm{nat}^{\mathsf{E}})$ to infer $(\mathbf{0})^\sigma \Downarrow^{\mathsf{E}} \mathbf{0}$.

  - $M' = \mathbf{pred}\, L$ and $N = \mathbf{pred}\,(L^\sigma)$: $(\mathbf{pred}\, L)^\sigma \to_{\mathsf{pr}} \mathbf{pred}\,(L^\sigma)$, at which point our reducing term is no longer at head position. We proceed in the same manner as in the subcase where $M = \mathbf{pred}\, M'$ to conclude that $\mathbf{pred}\,(L^\sigma) \Downarrow^{\mathsf{E}} \underline{n}$, from which we can infer $(\mathbf{pred}\, L)^\sigma \Downarrow^{\mathsf{E}} \underline{n}$.
  - Proceed as in the above case for the remaining four cases.

- Subcase $M = \mathbf{pred}\, M'$. We use Lemma 2.2.7 to obtain

$$\mathbf{pred}\, M' \twoheadrightarrow_{\mathtt{wh}} \mathbf{pred}\, \underline{n} \to_{\mathtt{wh}} \begin{cases} \underline{0} & \text{if } \underline{n} = \underline{0}, \\ \underline{n-1} & \text{otherwise.} \end{cases}$$

  with $|M' \twoheadrightarrow_{\mathtt{wh}} \underline{n}|$ shorter than k. By IH we get $M' \Downarrow^{\mathsf{E}} \underline{n}$, so we conclude by applying either $(\mathrm{pr}^{\mathsf{E}})$ or $(\mathrm{pr}_0{}^{\mathsf{E}})$.

- Subcase $M = \mathbf{succ}\, M'$. Analogous to the above.

- Subcase $M = \mathbf{ifz}(M', N_1, N_2)$. We use Lemma 2.2.7 to obtain

$$\mathbf{ifz}(L, N_1, N_2) \twoheadrightarrow_{\mathtt{wh}} \mathbf{ifz}(\underline{n}, N_1, N_2) \to_{\mathtt{wh}} \begin{cases} N_1 & \text{if } n = 0, \\ N_2 & \text{otherwise.} \end{cases}$$

  with $|M' \twoheadrightarrow_{\mathtt{wh}} \underline{n}|$ shorter than k. We then have the following, depending on the value of $\underline{n}$:

  - $(\underline{n} = \underline{0})$ By IH we get $M' \Downarrow^{\mathsf{E}} \underline{0}$ and $N_1 \Downarrow^{\mathsf{E}} V_1$, so we conclude by applying $(\mathrm{ifz}_0{}^{\mathsf{E}})$.
  - $(\underline{n} > \underline{0})$ By IH we get $M' \Downarrow^{\mathsf{E}} \underline{n}$ and $N_2 \Downarrow^{\mathsf{E}} V_2$, so we conclude by applying $(\mathrm{ifz}_{>0}{}^{\mathsf{E}})$.

- Subcase $M = \mathbf{fix}\, M'$. Then $N = M'(\mathbf{fix}\, M')$. By IH we obtain $M(\mathbf{fix}\, M) \Downarrow^{\mathsf{E}} V$. We apply $(\mathrm{fix}^{\mathsf{E}})$ to infer $\mathbf{fix}\, M \Downarrow^{\mathsf{E}} V$. □

EPCF terms can be typed in a manner very similar to PCF terms.

Definition 2.2.10.
An EPCF typing judgement *is, like in* PCF*, a triple of* $\Gamma \vdash M : \alpha$*. The rules for typing an* EPCF *term are as follows:*

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathsf{int}}\ (0) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ (\to_{\mathrm{E}}) \qquad \frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\mathrm{ax})$$

$$\frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{succ}\, M : \mathsf{int}}\ (+) \qquad \frac{\Gamma, x : \beta \vdash M : \alpha \quad \vdash N : \beta}{\Gamma \vdash M\langle N/x\rangle : \alpha}\ (\sigma) \qquad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (\to_{\mathrm{I}})$$

$$\frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathbf{pred}\, M : \mathsf{int}}\ (-) \qquad \frac{\Gamma \vdash L : \mathsf{int} \quad \Gamma \vdash M : \alpha \quad \Gamma \vdash N : \alpha}{\Gamma \vdash \mathbf{ifz}(L, M, N) : \alpha}\ (\mathrm{ifz}) \qquad \frac{\Gamma \vdash M : \alpha \to \alpha}{\Gamma \vdash \mathbf{fix}\, M : \alpha}\ (\mathrm{Y})$$

The only change compared to PCF typing rules is the introduction of the (σ) rule. In this rule, we rely on the fact that in a program of shape $M\langle N/x\rangle$, the subterm N must be closed and

remove the typing environment in $\vdash N : \beta$. This is not standard when considering more general notions of explicit substitutions, but simplifies our definitions later.

Remark 2.2.11.
As in PCF*, all normal forms of typed* EPCF *programs are* EPCF *values.*

Example 2.2.12.
The running examples can be typed without the use of the (σ) *rule, as they are* PCF *terms. For an* EPCF *exclusive term, as an example, we type* $x\langle \mathbf{succ}\,\mathbf{0}/x\rangle$*:*

$$\dfrac{\dfrac{}{x : \mathsf{int} \vdash x : \mathsf{int}}\,(\mathrm{ax}) \quad \dfrac{\dfrac{}{\vdash \mathbf{0} : \mathsf{int}}\,(0)}{\vdash \mathbf{succ}\,\mathbf{0} : \mathsf{int}}\,(+)}{\vdash x\langle \mathbf{succ}\,\mathbf{0}/x\rangle : \mathsf{int}}\,(\sigma)$$

The following lemma summarizes the main properties of the type assignment system.

Lemma 2.2.13.
Let $M \in \Lambda^{\mathsf{E}}$*,* $\alpha, \beta \in \mathbb{T}$ *and* Γ *be a typing environment.*

(1) (Syntax directedness) Every derivable judgement $\Gamma \vdash M : \alpha$ *admits a unique derivation, up to the most general types chosen during applications of* $(\to_{\mathrm{E}})$ *and* (σ)*.*

(2) (Weakening) $\Gamma, x : \beta \vdash M : \alpha$ *with* $x \notin \mathrm{FV}(M)$ *holds if and only if* $\Gamma \vdash M : \alpha$ *does. Thus, an* EPCF *program* M *is typable if it is typable in the empty environment.*

(3) (Subject reduction) For all $M \in \mathcal{P}_{\mathsf{E}}$*,* $\vdash M : \alpha$ *and* $M \to_{\mathtt{wh}} M'$ *entail* $\vdash M' : \alpha$*.*

Proof. (1) (Syntax directedness) Trivial proof by inspection – there exists only one case for each shape of the term, $(\to_{\mathrm{E}})$ and (σ) are the only cases where a new type variable is introduced, and we force said type variable to be the most general one possible, so there is only one choice.

(2) (Weakening) An easy proof by induction on the shape of M proves this in both directions – the statement holds for $\Gamma, x : \beta \vdash \mathbf{0} : \mathsf{int}$ and $\Gamma, x : \beta \vdash y : \alpha$ (as $y \neq x$ due to $x \notin \mathrm{FV}(M)$), and all other cases are derived from those two. We will prove the case for $M = \lambda x.M'$ as an example.

($\Leftarrow$) We have $\Gamma \vdash \lambda x.M' : \beta \to \alpha$, from which we derive $\Gamma, x : \beta \vdash M' : \alpha$. By IH, we have $\Gamma, y : \gamma, x : \beta \vdash M' : \alpha$ such that $y \notin \mathrm{FV}(M')$, from which we can infer $\Gamma, y : \gamma \vdash \lambda x.M' : \beta \to \alpha$.

($\Rightarrow$) We have $\Gamma, y : \gamma \vdash \lambda x.M' : \beta \to \alpha$ such that $y \notin \mathrm{FV}(M)'$, from which we derive $\Gamma, y : \gamma, x : \beta \vdash M' : \alpha$. By IH, we have $\Gamma, x : \beta \vdash M' : \alpha$, from which we can infer $\Gamma \vdash \lambda x.M' : \beta \to \alpha$. Note that if $y = x$, then from $\Gamma, x : \gamma \vdash \lambda x.M' : \beta \to \alpha$ we

directly derive $\Gamma, x : \beta \vdash M' : \alpha$ without the use of the IH and conclude in the same manner.

(3) (Subject reduction) We will prove this by induction on a type derivation of $\vdash M : \alpha$. There are only two interesting cases, namely $M = (\lambda x.N_1)^\sigma N_2$ and $M = M'\langle N/x\rangle$, as all other cases are identical to PCF where subject reduction holds.

- Case $M = (\lambda x.N_1)^\sigma N_2$: Then $(\lambda x.N_1)^\sigma N_2 \to_{\mathtt{wh}} N_1^\sigma\langle N_2/x\rangle$. From the judgement $\vdash (\lambda x.N_1)^\sigma N_2 : \alpha$ we obtain the judgement $y_1 : \gamma_1, \ldots, y_n : \gamma_n, x : \beta \vdash N_1 : \alpha$ alongside the judgements $\vdash N_2 : \beta$, $\vdash \sigma(y_1) : \gamma_1, \ldots, \vdash \sigma(y_n) : \gamma_n$ via a single application of the $(\to_{\mathrm{E}})$ rule followed by repeated applications of the (σ) rule. Using all of these judgements we can then derive the judgement $\vdash N_1\langle N_2/x\rangle : \alpha$, through $n+1$ applications of the (σ) rule.
- Case $M = M'\langle N/x\rangle$: There are 8 subcases.
  - Subcase $M = x^\sigma$, where $\sigma(x) = N$: Using (σ) we obtain the judgement $\vdash N : \alpha$.
  - Subcase $M = y^\sigma$, where $y \notin \mathrm{dom}(\sigma)$: Does not apply, as this term is not closed.
  - Subcase $M = \mathbf{0}^\sigma$: Trivial, as this has the type int and $\mathbf{0}$ has the type int.
  - Subcase $M = (\mathbf{pred}\, N)^\sigma$: We obtain $y_1 : \gamma_1, \ldots, y_n : \gamma_n \vdash N : \mathsf{int}$ alongside the judgements $\vdash \sigma(y_1) : \gamma_1, \ldots, \vdash \sigma(y_n) : \gamma_n$. We can reassemble these to form the judgement $\vdash N^\sigma : \mathsf{int}$, from which we derive $\vdash \mathbf{pred}\,(N^\sigma) : \mathsf{int}$.
  - All other subcases proceed identically to $(\mathbf{pred}\, N)^\sigma$. □

2.2.2 PCF and EPCF

We now move on to proving that PCF and EPCF operational semantics coincide on closed terms of type int. For this purpose, we first introduce the *collapse* $M^\dagger$ of an EPCF term M defined by performing all of the internal explicit substitutions. While we define the general case, note that inconsistencies can occur when M is not a program.

Definition 2.2.14.

(1) Given an EPCF *term* M*, define a* PCF *term* $M^\dagger \in \Lambda^{\mathsf{PCF}}$ *as follows:*

$$
\begin{array}{rclcrcl}
x^\dagger & = & x & & \mathbf{0}^\dagger & = & \mathbf{0} \\
(MN)^\dagger & = & M^\dagger N^\dagger & & (\mathbf{pred}\, M)^\dagger & = & \mathbf{pred}\, M^\dagger \\
(\lambda x.M)^\dagger & = & \lambda x.M^\dagger & & (\mathbf{succ}\, M)^\dagger & = & \mathbf{succ}\, M^\dagger \\
(\mathbf{fix}\, M)^\dagger & = & \mathbf{fix}\,(M^\dagger) & & (\mathbf{ifz}(L, M, N))^\dagger & = & \mathbf{ifz}(L^\dagger, M^\dagger, N^\dagger) \\
(M\langle N/x\rangle)^\dagger & = & M^\dagger[N^\dagger/x] & & & &
\end{array}
$$

(2) The head size $\lfloor - \rfloor : \Lambda^{\mathsf{E}} \to \mathbb{N}$ *of an* EPCF *term is defined as follows:*

$$\begin{array}{rclrcl} \lfloor x \rfloor = \lfloor \mathbf{0} \rfloor & = & 1 & \lfloor M\langle N/x\rangle \rfloor & = & \lfloor M \rfloor \cdot (\lfloor N \rfloor + 1) \\ \lfloor MN \rfloor & = & \lfloor M \rfloor + 1 & \lfloor \lambda x.M \rfloor & = & \lfloor M \rfloor + 1 \\ \lfloor \mathbf{pred}\, M \rfloor & = & \lfloor M \rfloor + 1 & \lfloor \mathbf{fix}\, M \rfloor & = & \lfloor M \rfloor + 1 \\ \lfloor \mathbf{succ}\, M \rfloor & = & \lfloor M \rfloor + 1 & \lfloor \mathbf{ifz}(L, M, N) \rfloor & = & \lfloor L \rfloor + 1 \end{array}$$

(3) The map $\lfloor - \rfloor$ *is extended to explicit substitutions* $\sigma = \langle N_1/x_1\rangle \cdots \langle N_n/x_n\rangle$*, by setting*

$$\lfloor \sigma \rfloor = \prod_{i=1}^{n} (\lfloor N_i \rfloor + 1).$$

Proposition 2.2.15.

(1) If $M \in \mathcal{P}_{\mathsf{E}}$ *then* $M^\dagger$ *is a* PCF *program.*

(2) If $P \in \Lambda^{\mathsf{PCF}}$ *then* $P^\dagger = P$*.*

Proof.

(1) A trivial proof by structural induction on M.

(2) A trivial proof by structural induction on P. □

Proposition 2.2.16.

(1) Let $M, N \in \mathcal{P}_{\mathsf{E}}$ *be such that* $M \to_{\mathsf{cr}} N$*. Then* $M^\dagger \to_{\mathsf{PCF}} N^\dagger$*.*

(2) Let $M, N \in \mathcal{P}_{\mathsf{E}}$ *be such that* $M \to_{\mathsf{pr}} N$*. Then* $M^\dagger = N^\dagger$*.*

Proof.

(1) By induction on a derivation of $M \to_{\mathsf{cr}} N$.

- Base cases.
  - Case $M = (\lambda x.L_1)^\sigma L_2$ and $N = L_1^\sigma\langle L_2/x\rangle$ where, say, $\sigma = \langle \vec{Y}/\vec{y}\rangle$ with $x \notin \vec{y}$. Then, we have:

$$\begin{array}{rcll} M^\dagger & = & (\lambda x.L_1^\dagger)[\vec{Y}^\dagger/\vec{y}]L_2^\dagger & \\ & = & (\lambda x.L_1^\dagger[\vec{Y}^\dagger/\vec{y}])L_2^\dagger & \to_{\mathsf{PCF}} \quad L_1^\dagger[\vec{Y}^\dagger/\vec{y}][L_2^\dagger/x] \\ & & & = \quad (L_1^\sigma\langle L_2/x\rangle)^\dagger \\ & & & = \quad N^\dagger \end{array}$$

  - Case $M = \mathbf{ifz}(\underline{0}, N, L)$. Then $L^\dagger = \mathbf{ifz}(\underline{0}, N^\dagger, L^\dagger) \to_{\mathsf{PCF}} N^\dagger$.
  - Case $M = \mathbf{ifz}(\underline{n+1}, L, N)$. Then $L^\dagger = \mathbf{ifz}(\underline{n+1}, L^\dagger, N^\dagger) \to_{\mathsf{PCF}} N^\dagger$.

  - All other base cases hold trivially.

- Case $M = M_1 M_2$ and $N = N_1 M_2$ with $M_1 \to_{\mathsf{cr}} N_1$. By induction hypothesis, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Therefore $M^\dagger = M_1^\dagger M_2^\dagger \to_{\mathsf{PCF}} N_1^\dagger M_2^\dagger = (N_1 M_2)^\dagger = N^\dagger$.
- Case $M = \mathbf{ifz}(M_1, L_1, L_2)$ and $N = \mathbf{ifz}(N_1, L_1, L_2)$ with $M_1 \to_{\mathtt{wh}} N_1$. By induction hypothesis, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Thus we obtain

$$M^\dagger = \mathbf{ifz}(M_1^\dagger, L_1^\dagger, L_2^\dagger) \to_{\mathsf{PCF}} \mathbf{ifz}(N_1^\dagger, L_1^\dagger, L_2^\dagger) = (\mathbf{ifz}(N_1, L_1, L_2))^\dagger = N^\dagger$$

- Case $M = \mathbf{pred}\, M_1$ and $N = \mathbf{pred}\, N_1$ with $M_1 \to_{\mathsf{cr}} N_1$. By applying the IH, we have $M_1^\dagger \to_{\mathsf{PCF}} N_1^\dagger$. Thus $M^\dagger = \mathbf{pred}\,(M_1^\dagger) \to_{\mathsf{PCF}} \mathbf{pred}\,(N_1^\dagger) = (\mathbf{pred}\, N_1)^\dagger = N^\dagger$.
- Case $M = \mathbf{succ}\, M_1$ and $N = \mathbf{succ}\, N_1$ with $M_1 \to_{\mathsf{cr}} N_1$. Analogous.

(2) By induction on a derivation of $M \to_{\mathsf{pr}} N$.

- Base cases.
  - Case $M = (x^{\sigma_1}\langle N/x\rangle)^{\sigma_2}$. Since $N \in \mathcal{P}_{\mathsf{E}}$, we have $\mathrm{FV}(N) = \emptyset$. Thus we arrive at $M^\dagger = x[N^\dagger/x] = N^\dagger$.
  - Case $M = y^\sigma$, where $y \notin \mathrm{dom}(\sigma)$ does not apply, as then $M \notin \mathcal{P}_{\mathsf{E}}$.
  - All other base cases hold trivially.
- Case $M = M_1 M_2$ and $N = N_1 M_2$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Therefore $M^\dagger = M_1^\dagger M_2^\dagger = N_1^\dagger M_2^\dagger = (N_1 M_2)^\dagger = N^\dagger$.
- Case $M = \mathbf{ifz}(M_1, L_1, L_2)$ and $N = \mathbf{ifz}(N_1, L_1, L_2)$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Thus we have

$$M^\dagger = \mathbf{ifz}(M_1^\dagger, L_1^\dagger, L_2^\dagger) = \mathbf{ifz}(N_1^\dagger, L_1^\dagger, L_2^\dagger) = (\mathbf{ifz}(N_1, L_1, L_2))^\dagger = N^\dagger$$

- Case $M = \mathbf{pred}\, M_1$ and $N = \mathbf{pred}\, N_1$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $M_1^\dagger = N_1^\dagger$. Thus $M^\dagger = \mathbf{pred}\,(M_1^\dagger) = \mathbf{pred}\,(N_1^\dagger) = (\mathbf{pred}\, N_1)^\dagger = N^\dagger$.
- Case $M = \mathbf{succ}\, M_1$ and $N = \mathbf{succ}\, N_1$ with $M_1 \to_{\mathsf{pr}} N_1$. Analogous. □

Corollary 2.2.17.
Let $M \in \mathcal{P}_{\mathsf{E}}$ be such that $M \twoheadrightarrow_{\mathtt{wh}} \underline{n}$. Then $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$.

Proof. By induction on the length of the reduction sequence $\ell = |M \twoheadrightarrow_{\mathtt{wh}} \underline{n}|$.

Case $\ell = 0$. Then $M = \underline{n} = M^\dagger$, so this case follows by reflexivity of $\twoheadrightarrow_{\mathsf{PCF}}$.

Case $\ell > 0$. Then there exists $N \in \mathcal{P}_{\mathsf{E}}$ such that $M \to_{\mathtt{wh}} N \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ where $|N \twoheadrightarrow_{\mathtt{wh}} \underline{n}| < \ell$. By Proposition 2.2.16, we have $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} N^\dagger$. By induction hypothesis, we obtain $N^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. By transitivity, we conclude $M^\dagger \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. □

Lemma 2.2.18.
The percolation reduction $\to_{\mathsf{pr}}$ on EPCF *terms is strongly normalising. More precisely:*

$$M \to_{\mathsf{pr}} N \;\Rightarrow\; \lfloor M \rfloor > \lfloor N \rfloor$$

Proof. By induction on a derivation of $M \to_{\mathsf{pr}} N$. In the following we consider a non-empty list of explicit substitutions $\sigma = \langle N_1/x_1 \rangle \cdots \langle N_n/x_n \rangle$, i.e. $n > 0$.

It follows that $\lfloor \sigma \rfloor > 1$.

- Base cases.
  - Case $M = x_i^\sigma$ and $N = \sigma(x_i) = N_i$. Then $\lfloor x_i^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > \lfloor N_i \rfloor = \lfloor N \rfloor$.
  - Case $M = \mathbf{0}^\sigma$ and $N = \mathbf{0}$. Then $\lfloor \mathbf{0}^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > 1 = \lfloor \mathbf{0} \rfloor$.
  - Case $M = y^\sigma$, with $y \notin \mathrm{dom}(\sigma)$, and $N = y$. Then $\lfloor y^\sigma \rfloor = 1 \cdot \lfloor \sigma \rfloor > 1 = \lfloor y \rfloor$.
  - Case $M = (M_1 M_2)^\sigma$. Then we have
    $\lfloor (M_1 M_2)^\sigma \rfloor = (\lfloor M_1 \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M_1 \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor M_1^\sigma M_2^\sigma \rfloor$
  - Case $M = (\mathbf{pred}\, M')^\sigma$. Then we have
    $\lfloor (\mathbf{pred}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{pred}\, (M')^\sigma \rfloor$
  - Case $M = (\mathbf{succ}\, M')^\sigma$. Then we have
    $\lfloor (\mathbf{succ}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{succ}\, (M')^\sigma \rfloor$
  - Case $M = (\mathbf{ifz}(M_1, M_2, M_3))^\sigma$. Then we have
    $\lfloor (\mathbf{ifz}(M_1, M_2, M_3))^\sigma \rfloor = (\lfloor M_1 \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M_1 \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{ifz}(M_1^\sigma, M_2^\sigma, M_3^\sigma) \rfloor$
  - Case $M = (\mathbf{fix}\, M')^\sigma$. Then we have
    $\lfloor (\mathbf{fix}\, M')^\sigma \rfloor = (\lfloor M' \rfloor + 1) \cdot \lfloor \sigma \rfloor > \lfloor M' \rfloor \cdot \lfloor \sigma \rfloor + 1 = \lfloor \mathbf{fix}\, (M')^\sigma \rfloor$
- Case $M = M_1 M_2$ and $N = N_1 M_2$ with $M_1 \to_{\mathsf{pr}} N_1$. By induction hypothesis, we have $\lfloor M_1 \rfloor > \lfloor N_1 \rfloor$. Therefore $\lfloor M_1 \rfloor + 1 > \lfloor N_1 \rfloor + 1$.

The remaining cases follow analogously from the induction hypothesis. □

Proposition 2.2.19.
Let $(M_n)_{n \in \mathbb{N}}$ be an infinite sequence of EPCF *programs such that $M_n \to_{\mathtt{wh}} M_{n+1}$. Then for all $i \in \mathbb{N}$ there exists an index $j > i$ such that $M_i \twoheadrightarrow_{\mathtt{wh}} M_j$ and $M_i^\dagger \to_{\mathsf{PCF}} M_j^\dagger$.*

Proof. Consider an arbitrary M_i. As a consequence of Lemma 2.2.18, $M_i \twoheadrightarrow_{\mathsf{pr}} M_k$ for some $k \geq i$ such that M_k is in $\to_{\mathsf{pr}}$-normal form. Therefore $M_k \to_{\mathtt{wh}} M_{k+1}$ must be a computation step, i.e. $M_k \to_{\mathsf{cr}} M_{k+1}$. By Proposition 2.2.16, $M_i^\dagger = M_k^\dagger \to_{\mathsf{PCF}} M_{k+1}^\dagger$. Conclude by taking $k = k + 1 > i$. □

Theorem 2.2.20.
For a PCF *program* P *having type* int*, we have:*

$$P \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$$

Proof. $(\Leftarrow)$ Assume $P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$. Since P is a PCF program it also belongs to $\mathcal{P}_{\mathsf{E}}$. By Corollary 2.2.17 we have $P^{\dagger} \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. Conclude since, by Proposition 2.2.15((2)), $P^{\dagger} = P$.

$(\Rightarrow)$ We prove the contrapositive: for all $n \in \mathbb{N}$, $P \not\twoheadrightarrow_{\mathsf{PCF}} \underline{n} \Leftarrow P \not\twoheadrightarrow_{\mathtt{wh}} \underline{n}$. From $(\Leftarrow)$ we cannot have $P \twoheadrightarrow_{\mathtt{wh}} \underline{m}$ and $P \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$ for $m \neq n$. By Subject Reduction (Lemma 2.2.13((3))) and Remark 2.2.11, as numerals are the only EPCF programs having type int which do not reduce, P must have an infinite $\rightarrow_{\mathtt{wh}}$ reduction path. By Proposition 2.2.19, $P^{\dagger}$ must have an infinite $\rightarrow_{\mathsf{PCF}}$ reduction path. Conclude since, by Proposition 2.2.15((2)), $P^{\dagger} = P$. □

Chapter 3

Addressing Machines

3.1 Abstract Machines

An *abstract machine* is a theoretical model of how a computer system functions. Abstract machines differ from (abstract) programming languages in that they not only specify the program to be executed, but also how it is executed step by step. Their "abstract" nature comes from omitting many details of how the computation would proceed on actual hardware. Turing Machines are the first and most fundamental abstract machines in literature [Tur37]. With them and physical computers as inspiration, a large number of abstract machines have been presented in literature [DHS00, FW87, Cre91, Lan07, Ler90, GMR89, Can01].

Abstract machines are typically used for one of two purposes:

- As a tool to study the computational complexity of algorithms, or make more general statements about computability;
- To demonstrate how terms in a language could be evaluated – abstract machines used as an intermediate step or source/target by a compiler or interpreter is an application of this use.

Along the second line of thought, one could separate abstract machines by which type of programming language they are most suitable to emulate [DHS00]. We will discuss three examples of abstract machines in literature which have a strong link to functional languages.

3.1.1 The SECD Machine

The SECD Machine was described by Peter J. Landin in "The Mechanical Evaluation of Expressions" [Lan64]. The letters stand for **S**tack, **E**nvironment, **C**ontrol, **D**ump. This machine was

the first abstract machine designed to interpret λ-terms, and in doing so introduced a number of concepts which we now take for granted when discussing functional languages – among others closures, thunks, circular definition, partial evaluation, call-by-need, and the more general use of the λ-calculus as a meta-language for writing programs [Dan05]. We will informally present the SECD machine with a focus on its use of closures.

The SECD machine consists of four parts:

- A **S**tack of intermediate values;
- An **E**nvironment, which is a dictionary (set of pairings) from variable names to values;
- A **C**ontrol list of expressions to be evaluated;
- A **D**ump stack, which contains triples of all three of the above and is used to "save" the current state of the machine, to allow it to evaluate a different expression.

Landin specifies that the **E**nvironment and the **D**ump would be unnecessary if λ-expressions (first class functions) were prohibited. The environment is necessary to define closures.

A *closure* is triple of (expression, **E**nvironment, bound variables) which represents a function. When a λ-expression is to be constructed, a closure is created encapsulating the local variables of the expression, the variables(s) bound by the λ-expression, and the instructions of the expression. The **E**nvironment is necessary to allow the capture of local variable assignments.

When an argument is applied to a closure, the **D**ump stack comes into effect. The machine's current state is saved to **D**ump, minus the closure, argument, and application instruction. Then the machine is free to calculate the outcome of the application, without any potential conflicts occurring in the environment. Another issue that is solved via the use of the **D**ump is that the value returned from the function may be an application, which then first needs to be evaluated before it can be returned.

First class functions through the use of closures automatically result in permitting thunks, partial evaluation, and call-by-need (the SECD machine is natively call-by-value). We also mentioned *circular definition* – a self-referential definition, such as $x = (1, x, x)$ which produces a triple of infinite depth $(1, (1, (\dots), (\dots)), (1, (\dots), (\dots)))$. The SECD machine allows such definitions by rearranging the definition to first form an abstraction over the self-reference $\lambda x.(1, x, x)$ and then taking the fixed point of the resulting abstraction.

3.1.2 KAM

The Krivine Abstract Machine (KAM) was first described by Jean-Louis Krivine sometime in the 1980's, though a dedicated paper was not published until 2007 [Kri07].[1] The purpose of the machine, according to Krivine, was to execute programs obtained via the translation of mathematical proofs into the λ-calculus through the Curry-Howard correspondence [How80]. Its more general interest is that the Krivine Abstract Machine is a very simple machine which directly executes untyped λ-terms with a call-by-name evaluation strategy. In doing so, it takes inspiration from the SECD machine in that it also makes use of closures to handle first order functions. The Krivine Abstract Machine uses the λ-calculus in *de Bruijn notation* [Bru72] as its instruction language – we will skip over the details of this notation and give a description of the Krivine Abstract Machine more informally.

The Krivine Abstract Machine consists of three parts:

- A term area, where an untyped λ-term to be evaluated is placed;
- A stack to collect arguments, referred to simply as the stack;
- A list of variable assignments, referred to simply as the environment.

Both the stack and the environment collect exclusively closures – here, a closure is a pairing of a λ-term with an environment. The Krivine Abstract Machine has (vaguely) three execution rules, one for each potential shape of a λ-term:

- If the term is an application MN, then the pairing of N with the current environment is pushed to the stack and we continue evaluating M.
- If the term is an abstraction $\lambda x_1.M$, then we first check the length of continuous abstractions in the term, i.e. $M = \lambda x_2. \ldots \lambda x_n.N$. We then attempt to pop n arguments from the stack and append these arguments to the current environment, before then evaluating N.
- If the term is a variable, then we identify which closure in our environment list corresponds to said variable[2] and set the current term and environment to those found within said closure.

Compared to the SECD machine, the Krivine Abstract Machine is a lot simpler. This is partially due to its instruction language being a simpler, but also partially due to its "all objects are closures" approach.

[1] As one can imagine, the difference in time between the first description and the first paper resulted in many different presentations and slight variations on what is essentially the same machine. We will focus on the one described in the paper.

[2] De Bruijn notation makes matching variables to environment closures easy, as variables are represented by numbers and so the correct closure is identified via the index.

3.1.3 MAM

The Milner Abstract Machine (MAM) is a far more recent machine than either of the two, introduced by Accattoli et. al. in "Distilling Abstract Machines" [ABM14]. It is one of a number of machines introduced in said paper as variants of existing machines, with the Milner Abstract Machine as a variant of the Krivine Abstract Machine. These variants were introduced by investigating the link between abstract machines and the Linear Substitution Calculus – to quote from the paper, "Traditionally, calculi with explicit substitutions simulate machines. The Linear Substitution Calculus, instead, distills them.". What is meant by distilling abstract machines is the separation of the search for redexes from the action of substitution – essentially, the Linear Substitution Calculus has the same relationship with common abstract machines as it does with explicit substitutions.

The Milner Abstract Machine, then, is one of several abstract machines designed by taking inspiration from the Linear Substitution Calculus. The Milner Abstract Machine takes a different approach to first order functions than either of the previously named machines – rather than closures, all variables are global variables. Variables which were previously local are then distinguished from one another via an explicit α-equivalence operation. The inspiration here is from the "β-reduction at a distance" part of the Linear Substitution Calculus. Without going into the details, one can roughly picture the functionality of the Milner Abstract Machine by replacing the deletion portion of the variable reduction from the Krivine Abstract Machine with a renaming step instead, and removing environments from closures.

3.2 Addressing Machines

Now that we have introduced some background on abstract machines, we can move on to the machines of interest for this thesis. Addressing Machine were introduced by Della Penna et. al. in "Addressing Machines as Models of λ-Calculus" [DIM22].[3] Their purpose was the construction of the first model of the λ-calculus based on abstract machines – though Turing machines and the λ-calculus are equivalent, constructing a model of the λ-calculus using Turing machines is extremely convoluted, and we know of no successful attempts.

Addressing Machines differ from other abstract machines in a number of ways. When one discusses Addressing Machines, one is really referring to a "network" of Addressing Machines. Most abstract machines can be thought of as a single object, which takes in an expression in a language and then executes said expression. Addressing Machines, on the other hand, rely on a "communication" process between them in order to perform most computations.

[3] A preliminary version of Addressing Machines appeared in Della Penna's MSc thesis [Del97].

Intuitively, an addressing machine consists of three parts:

- A fixed number of registers, which can each contain a single address;
- A list of (imperative) program instructions;
- An input tape, which is a list of addresses that functions as a queue.

Each addressing machine has an address. Addressing Machines can read addresses from their tape, store addresses in registers, pass addresses as arguments to other AMs, and transfer the computation to another Addressing Machine. The internal design takes inspiration from Turing machines and register machines (such as the RASP machine [ER64]), while the communication process takes inspiration from the behaviour of λ-terms. This communication process differs significantly from that of systems focused on the analysis of communication, such as the π-calculus [Mil99], as AMs are not designed for this purpose.

The machines discussed previously in Section 3.1 were chosen to highlight one particular point – if one wishes to design an abstract machine with first class functions, then the method which allows for said first class functions is the primary source of complexity for the machine. Addressing Machines take a novel approach to this problem in that the problem is, in a way, ignored. Addressing Machines themselves can represent functions in a way that reminds one of closures, and their addresses can be manipulated as a data type. They take the opposite approach to the Milner Abstract Machine – all variables are local, rather than global.

3.3 Extended Addressing Machines

We will be extending the Addressing Machines from [DIM22] with instructions for performing arithmetic operations and conditional testing. These Addressing Machines will be called Extended Addressing Machine.

To enable the arithmetic operations, natural numbers are represented by particular EAMs playing the role of numerals.

3.3.1 Main Definitions

We consider fixed a countably infinite set $\mathbb{A}$ of *addresses* together with a distinguished countable subset $\mathbb{X} \subset \mathbb{A}$, such that $\mathbb{A} - \mathbb{X}$ remains infinite.[4] Intuitively, $\mathbb{X}$ is the set of addresses that

[4] Addressing Machines do not require the presence of a distinguished countable subset, it is only necessary for EAMs to have machines representing the natural numbers.

we reserve for the numerals, therefore hereafter we work under the hypothesis that $\mathbb{X} = \mathbb{N}$, an assumption that we can make without loss of generality.

Definition 3.3.1 (Addresses and Registers)**.**

(1) Let $\varnothing \notin \mathbb{A}$ be a "null" constant representing an uninitialised register. Set $\mathbb{A}_\varnothing = \mathbb{A} \cup \{\varnothing\}$.

(2) An $\mathbb{A}$-valued tape *T is a finite ordered list of addresses $T = [a_1, \ldots, a_n]$ with $a_i \in \mathbb{A}$ for all $i\,(1 \leq i \leq n)$. When $\mathbb{A}$ is clear from the context, we simply call T a tape. We denote by $\mathcal{T}_\mathbb{A}$ the set of all $\mathbb{A}$-valued tapes.*

(3) Let $a \in \mathbb{A}$ and $T, T' \in \mathcal{T}_\mathbb{A}$. We denote by $a :: T$ the tape having a as first element and T as tail. We write $T \mathbin{@} T'$ for the concatenation of T and T', which is an $\mathbb{A}$-valued tape itself.

(4) Given an index $i \geq 0$, an $\mathbb{A}_\varnothing$-valued register *R_i is a memory-cell capable of storing either $\varnothing$ or an address $a \in \mathbb{A}$. We write $!R_i$ to represent the value stored in the register R_i. (The notation $!R_i$ is borrowed from ML, where $!$ represents an explicit dereferencing operator.)*

(5) Given $\mathbb{A}_\varnothing$-valued registers $R_0, \ldots, R_r$ for $r \geq 0$, an address $a \in \mathbb{A}$ and an index $i \geq 0$, we write $\vec{R}[R_i := a]$ for the list of registers $\vec{R}$ where the value of R_i has been updated by setting $!R_i = a$. Notice that, whenever $i > r$, we assume that the contents of $\vec{R}$ remains unchanged, i.e. $\vec{R}[R_i := a] = \vec{R}$.

Intuitively, the contents of the registers $R_0, \ldots, R_r$ constitutes the *state* of a machine, while the tape correspond to the list of its inputs. Every address refers to a particular machine in a particular, fixed state. One could understand an address as the encoding of a machine. When reducing a machine, its address thus changes, with the previous address still referring to the machine in its previous state.

Addressing Machines are endowed with three instructions (i, j, k, l range over indices of registers):

(1) `Load` i : If the tape is non-empty, 'pops' the address a from the input tape $a :: T$ and stores a in the register R_i. If R_i does not exist then a is discarded. If the tape is empty then the machine halts its execution.

(2) $k \leftarrow \texttt{App}(i, j)$: Let a_1 and a_2 be the addresses stored in R_i and R_j respectively. This instruction applies a_1 to a_2 by extending the tape of the machine of address a_1 with the address a_2. The address of the resulting machine (it has an address distinct from a_1, as it is no longer the same machine) is then stored in R_k. The address produced by the application is not calculated internally, but rather obtained calling an external *application map* denoted $a_1 \cdot a_2$.

(3) $\texttt{Call}\ i$: The computation is transferred to the machine having as address the value stored in R_i, whose tape is extended with the remainder of the current machine's tape.

As a general principle, writing on a non-existing register does not cause issues as the value is simply discarded—this is in fact the way one can erase an argument. Attempts to read an uninitialized register can be avoided statically (see Lemma 3.3.4). Notice that, as the execution of an instruction changes the state of a machine, the execution of an instruction will also change the address which points to a machine.

To obtain the instruction set for EAMs, we enrich the above set of instructions with arithmetic operations mimicking the ones present in PCF:

(4) $l \leftarrow \texttt{Test}(i,\, j,\, k)$: implements the "*is zero?*" test on $!R_i$. Assuming that the value of R_i is an address $n \in \mathbb{N}$, the instruction stores in R_l the value of R_j or R_k, depending on whether $n = 0$.

(5) $j \leftarrow \texttt{Pred}(i)$: if $!R_i \in \mathbb{N}$, the value of R_j becomes $!R_i \ominus 1 = \max(!R_i - 1, 0)$.

(6) $j \leftarrow \texttt{Succ}(i)$: if $!R_i \in \mathbb{N}$, then the value of R_j becomes $!R_i + 1$.

Notice that the instructions above need R_i to contain a natural number to perform the corresponding operation. However, they are also supposed to work on addresses of machines that compute a numeral. For this reason, the machine whose address is stored in R_i must first be executed, and only if the computation terminates with a numeral is the arithmetic operation performed. If the computation terminates in an address not representing a numeral, then the machine halts. We will see that these terminations can be avoided using a type inference algorithm (see Proposition 3.3.21, below).

Definition 3.3.2 (Programs)**.**

(1) A program P *is a finite list of instructions generated by the following grammar, where* ε *represents the empty string and* i, j, k, l *are indices of registers:*

$$\begin{aligned}
\texttt{P} &::= \texttt{Load}\ i;\ \texttt{P} \mid \texttt{A} \\
\texttt{A} &::= k \leftarrow \texttt{App}(i,\, j);\ \texttt{A} \mid l \leftarrow \texttt{Test}(i,\, j,\, k);\ \texttt{A} \mid j \leftarrow \texttt{Pred}(i);\ \texttt{A} \mid j \leftarrow \texttt{Succ}(i);\ \texttt{A} \mid \texttt{C} \\
\texttt{C} &::= \texttt{Call}\ i \mid \varepsilon
\end{aligned}$$

Thus, a program starts with a list of Load*'s, continues with a list of* App*,* Test*,* Pred*,* Succ*, and possibly ends with a* Call*. Each of these lists may be empty, in particular the empty program* ε *can be generated.*

(2) In a program, we write $\texttt{Load}\ (i_1, \ldots, i_n)$ *as an abbreviation for the instruction sequence* $\texttt{Load}\ i_1;\ \cdots\ ;\ \texttt{Load}\ i_n$*. For a given instruction* ins*, we write* $\texttt{ins}^a$ *for the repetition of* ins a*-many times.*

(3) Let P be a program, $r \geq 0$, and $\mathcal{I} \subseteq \{0, \ldots, r-1\}$ be a set of indices corresponding to the indices of initialized registers. Define the relation $\mathcal{I} \models^r P$, whose intent is to specify that P does not read uninitialized registers, as the least relation closed under the rules:

$$\frac{}{\mathcal{I} \models^r \varepsilon} \qquad \frac{i \in \mathcal{I}}{\mathcal{I} \models^r \texttt{Call}\ i} \qquad \frac{\mathcal{I} \cup \{j\} \models^r \texttt{A} \quad i \in \mathcal{I} \quad j < r}{\mathcal{I} \models^r j \leftarrow \texttt{Pred}(i);\ \texttt{A}}$$

$$\frac{\mathcal{I} \cup \{i\} \models^r \texttt{P} \quad i < r}{\mathcal{I} \models^r \texttt{Load}\ i;\ \texttt{P}} \qquad \frac{\mathcal{I} \models^r \texttt{P} \quad i \geq r}{\mathcal{I} \models^r \texttt{Load}\ i;\ \texttt{P}} \qquad \frac{\mathcal{I} \cup \{j\} \models^r \texttt{A} \quad i \in \mathcal{I} \quad j < r}{\mathcal{I} \models^r j \leftarrow \texttt{Succ}(i);\ \texttt{A}}$$

$$\frac{\mathcal{I} \cup \{k\} \models^r \texttt{A} \quad i, j \in \mathcal{I} \quad k < r}{\mathcal{I} \models^r k \leftarrow \texttt{App}(i,\ j);\ \texttt{A}} \qquad \frac{\mathcal{I} \cup \{l\} \models^r \texttt{A} \quad i, j, k \in \mathcal{I} \quad l < r}{\mathcal{I} \models^r l \leftarrow \texttt{Test}(i,\ j,\ k);\ \texttt{A}}$$

(4) A program P is valid with respect to $R_0, \ldots, R_{r-1}$ *if $\mathcal{R} \models^r P$ holds for*

$$\mathcal{R} = \{i \mid R_i \neq \varnothing \ \wedge\ 0 \leq i < r\}$$

Example 3.3.3.
For each of these programs, we specify its validity with respect to $R_0 = 7, R_1 = a, R_2 = \varnothing$ (i.e., $r = 3$).

$P_1 = 2 \leftarrow \texttt{Pred}(0);\ \texttt{Call}\ 2$ *(valid)*
$P_2 = \texttt{Load}\ (2, 8);\ 0 \leftarrow \texttt{Test}(0,\ 1,\ 2);\ \texttt{Call}\ 0$ *(valid, discarding an argument)*
$P_3 = \texttt{Load}\ (0, 2, 8);\ \texttt{Call}\ 8$ *(calling the non-existent register R_8, thus not valid)*
$P_4 = 0 \leftarrow \texttt{Succ}(2); \texttt{Call}\ 1$ *(reading from the uninitialized register R_2, thus not valid)*
$P_5 = 8 \leftarrow \texttt{Pred}(0); \texttt{Call}\ 0$ *(storing in non-existent register R_8, thus not valid)*

Lemma 3.3.4 (Program Validity)**.**
Given $\mathbb{A}_\varnothing$-valued registers $\vec{R}$ and a program P it is decidable whether P is valid w.r.t. $\vec{R}$.

Proof from [DIM22]. Decidability follows from the syntax directedness of Definition 3.3.2((3)), and the preservation of the invariant $\mathcal{I} \subseteq \{0, \ldots, r-1\}$, since $\mathcal{I}$ is only extended with $k < r$. The grammar in Definition 3.3.2(1) is right-linear, so it is decidable whether P is a production. Also, $r \in \mathbb{N}$ and therefore the set $\mathcal{R}$ in Definition 3.3.2(4) is finite. Since P is also finite, the set $\mathcal{R}$ remains finite during the execution of $\mathcal{R} \models^r P$. Decidability follows from these properties, together with the fact that the first instruction of P uniquely determines which rule from Definition 3.3.2(3) should be applied (and these rules are exhaustive). □

Hereafter, we will focus on EAMs having valid programs.

Definition 3.3.5 (Extended Addressing Machines)**.**

(1) An extended addressing machine *(*Extended Addressing Machine*) M over $\mathbb{A}$ (having $r+1$ registers) is given by a tuple $\mathsf{M} = \langle R_0, \ldots, R_r, P, T \rangle$ where*

- $\vec{R}$ *are* $\mathbb{A}$*-valued registers,*
- P *is a program valid w.r.t.* $\vec{R}$*, and*
- $T \in \mathcal{T}_{\mathbb{A}}$ *is an (input) tape.*

(2) We denote by $\mathcal{M}_{\mathbb{A}}$ *the set of all Extended Addressing Machines over* $\mathbb{A}$*.*

(3) Given an Extended Addressing Machine M*, we write* $\mathsf{M}.\vec{R}$ *for the list of its registers,* $\mathsf{M}.R_i$ *for its* i*-th register,* $\mathsf{M}.P$ *for the associated program and* $\mathsf{M}.T$ *for its input tape.*

(4) Given $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ *and* $T' \in \mathcal{T}_{\mathbb{A}}$*, we write* $\mathsf{M} \mathbin{@} T'$ *for the machine*

$$\langle \mathsf{M}.\vec{R}, \mathsf{M}.P, \mathsf{M}.T \mathbin{@} T' \rangle.$$

Definition 3.3.6 (Special Machines)**.**

(1) For $n \in \mathbb{N}$*, the* n-th numeral machine *is defined as* $\mathsf{n} = \langle R_0, \varepsilon, [] \rangle$*, with* $!R_0 = n$*.*

(2) For every $a \in \mathbb{A}$*, define the following extended addressing machine:*

$$\mathsf{Y}^a = \langle R_0 = \varnothing, R_1 = \varnothing, \texttt{Load}\ (0,1); 0 \leftarrow \texttt{App}(0,\ 1); 1 \leftarrow \texttt{App}(1,\ 0); \texttt{Call}\ 1, [a] \rangle.$$

We now enter into the details of the addressing mechanism which constitutes the core of this formalism.

Definition 3.3.7 (Address Table Map)**.**
Recall that $\mathbb{N}$ *stands for an infinite subset of* $\mathbb{A}$*, here identified with the set of natural numbers, and that* Y^a *has been introduced in Definition 3.3.6(2).*

(1) Since $\mathcal{M}_{\mathbb{A}}$ *is countable, we can fix a bijective function* $\# : \mathcal{M}_{\mathbb{A}} \to \mathbb{A}$ *which satisfies the following conditions:*

(a) (Numerals) $\forall n \in \mathbb{N} \,.\, \#\mathsf{n} = n$*, where* n *is the* n*-th numeral machine;*

(b) (Fixed point machine) $\exists a \in \mathbb{A} - \mathbb{N} \,.\, \#(\mathsf{Y}^a) = a$*.*

We call this function an address table map

(2) We write Y *for the Extended Addressing Machine* Y^a *satisfying* $\#(\mathsf{Y}^a) = a$ *(which must exist by 3.3.7.(1)(1)b).*

(3) Given $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$*, we call the element* $\#\mathsf{M}$ *the* address of the EAM M*. Conversely, the Extended Addressing Machine having as address* $a \in \mathbb{A}$ *is denoted by* $\#^{-1}(a)$*. In other words,* $\#^{-1}(a) = \mathsf{M} \iff \#\mathsf{M} = a$*.*

(4) Define the application map $(\cdot) : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$ *by setting*

$$a \cdot b = \#(\#^{-1}(a) \mathbin{@} [b]\,).$$

I.e., the application *of a to b is the unique address c of the Extended Addressing Machine obtained by appending b at the end of the input tape of the Extended Addressing Machine $\#^{-1}(a)$.*

In general, there are uncountably many possible address table maps of arbitrary computational complexity. A natural example of such maps is given by *Gödelization*, which can be performed effectively. The framework is however more general and allows to consider non-recursively enumeral sets of addresses like the complement K^c of the halting set

$$K = \{(i, x) \mid \text{the program } i \text{ terminates when run on input } x\}$$

and a non-computable function $\#{:}\mathcal{M}_{K^c} \to K^c$ as a map.

A practical application of alternate address table maps would be the encoding of some additional quantitative information about Addressing Machines, such as a cost value for calling the machine.

In an implementation of EAMs the address table map should be computable—one can choose a fresh address from $\mathbb{A}$ whenever a new machine is constructed, save the correspondence in some table and retrieve it in constant time.

Notation 3.3.8 (Numbers)**.**
A quick overview of the different numeric notations:

- n *is used to refer to the number* $n \in \mathbb{N} \subset \mathbb{A}$
- $\underline{n}$ *is used refer to the* PCF *term* $\mathbf{succ}^n(\mathbf{0})$. *Note that* $\mathbf{0} = \underline{0}$.
- n *is used to refer to the corresponding numeral machine. Note that* $\#\mathsf{n} = n$, *and* $\#^{-1}(n) = \mathsf{n}$

We will use both $\#^{-1}(n)$ and n, depending on whether there are clarity issues with writing n. For example, we tend to write n to refer to the n-numeral machine, but $\#^{-1}(n+1)$ for the $(n+1)$-numeral machine.

Remark 3.3.9 (Address Table Map Peculiarities)**.**

(1) $\mathcal{M}_{\mathbb{A}}$ is countably infinite.

(2) Since $\mathcal{M}_{\mathbb{A}}$ and $\mathbb{A}$ are both countable, the existence of $2^{\aleph_0}$ address table maps follows from cardinality reasons. This includes effective maps as well as non-computable ones.

(3) Depending on the chosen address table map, there might exist infinite (static) chains of EAMs, e.g., EAMs $(\mathsf{M}_n)_{n\in\mathbb{N}}$ *satisfying* $\mathsf{M}_n = \langle R_0, \varepsilon, [] \rangle$ *with* $!R_0 = \#\mathsf{M}_{n+1}$.

(4) Depending on the chosen address table map, there could exist EAMs with varying degrees of circular references – for EAMs we guarantee the presence of the numeral machines and the fixed point machine which have access to their own address. There could be more, but there could also be "loops" of references of various sizes, i.e. M *having access to the address of* N *while simultaneously* N *having access to the address of* M.

(5) Regardless of the chosen address table map, all finite "trees" of machines are defined.

The results presented in this work are independent from the choice of the address table map, subject to the constraints mentioned previously.

Example 3.3.10.
The following are examples of EAMs (whose registers are assumed uninitialized where unspecified, i.e. $\vec{R} = \vec{\varnothing}$*).*

(1) $\mathsf{I} := \langle R_0 = \varnothing, \texttt{Load}\ 0; \texttt{Call}\ 0, [] \rangle$,

(2) For some $a \in \mathbb{A}$, $\langle R_0 = a, R_1 = \varnothing, 0 \leftarrow \texttt{App}(1,\ 0); \texttt{Call}\ 1, [] \rangle$

(3) $\mathsf{Succ1} := \langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \rangle$.

(4) $\mathsf{Succ2} := \langle R_0, R_1, \texttt{Load}\ 0; \texttt{Load}\ 1; 1 \leftarrow \texttt{App}(0,\ 1); 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [\#\mathsf{Succ1}] \rangle$.

(5) $\mathsf{Add\_aux} := \langle \vec{R}, P, [] \rangle$ *with* $\mathsf{Add\_aux}.r = 5$ *and* $P = \texttt{Load}\ (0, 1, 2); 3 \leftarrow \texttt{Pred}(1); 4 \leftarrow \texttt{Succ}(2); 0 \leftarrow \texttt{App}(0,\ 3); 0 \leftarrow \texttt{App}(0,\ 4); 0 \leftarrow \texttt{Test}(1,\ 2,\ 0); \texttt{Call}\ 0$.

3.3.2 Operational semantics

The operational semantics of Extended Addressing Machines is given through a small-step rewriting system. The reduction strategy is deterministic, since the only applicable rule at every step is univocally determined by the first instruction of the internal program, the contents of the registers and the head of the tape.

Definition 3.3.11 (Reduction of EAMs)**.**

(1) Define a reduction strategy $\to_{\mathsf{c}}$ *on EAMs, representing one step of computation, as the least relation* $\to_{\mathsf{c}} \subseteq \mathcal{M}_{\mathbb{A}} \times \mathcal{M}_{\mathbb{A}}$ *closed under the following rules:*

Unconditional rewriting rules

$$\begin{array}{rcl}
\langle \vec{R}, \mathtt{Call}\ i, T\rangle & \to_{\mathsf{c}} & \#^{-1}(!R_i) \mathbin{@} T \\
\langle \vec{R}, \mathtt{Load}\ i; P, a :: T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_i := a], P, T\rangle \\
\langle \vec{R}, k \leftarrow \mathtt{App}(i,\ j); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_k := !R_i \cdot !R_j], P, T\rangle
\end{array}$$

Under the assumption that $!R_i \in \mathbb{N}$**.**

$$\begin{array}{rcl}
\langle \vec{R}, j \leftarrow \mathtt{Pred}(i); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_j := !R_i \ominus 1], P, T\rangle, \\
 & & \textit{where } n \ominus 1 := \max\{n-1, 0\} \\
\langle \vec{R}, j \leftarrow \mathtt{Succ}(i); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_j := !R_i + 1], P, T\rangle \\
\langle \vec{R}, l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T\rangle & \to_{\mathsf{c}} & \begin{cases} \langle \vec{R}[R_l := !R_j], P, T\rangle, & \textit{if } !R_i = 0, \\ \langle \vec{R}[R_l := !R_k], P, T\rangle, & \textit{otherwise}. \end{cases}
\end{array}$$

Under the assumption that $\#^{-1}(!R_i) \to_{\mathsf{c}} \mathsf{M}$**.**

$$\begin{array}{rcl}
\langle \vec{R}, j \leftarrow \mathtt{Pred}(i); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_i := \#\mathsf{M}], j \leftarrow \mathtt{Pred}(i); P, T\rangle \\
\langle \vec{R}, j \leftarrow \mathtt{Succ}(i); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_i := \#\mathsf{M}], j \leftarrow \mathtt{Succ}(i); P, T\rangle \\
\langle \vec{R}, l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T\rangle & \to_{\mathsf{c}} & \langle \vec{R}[R_i := \#\mathsf{M}], l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T\rangle
\end{array}$$

(2) The multistep reduction $\twoheadrightarrow_{\mathsf{c}}$ *is defined as the transitive-reflexive closure of* $\to_{\mathsf{c}}$*.*

(3) The conversion relation $\leftrightarrow_{\mathsf{c}}$ *is the transitive-reflexive-symmetric closure of* $\to_{\mathsf{c}}$*.*

(4) Given $\mathsf{M}, \mathsf{N}, \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$*, we write* $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ *for the set of lengths of reduction paths from* M *to* N*. We write* $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}| \sqsubset |\mathsf{M}' \twoheadrightarrow_{\mathsf{c}} \mathsf{N}'|$ *to mean that there exists* $i \in |\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ *and* $j \in |\mathsf{M}' \twoheadrightarrow_{\mathsf{c}} \mathsf{N}'|$ *such that* $i > j$*.*

(5) For $n \geq 0$*, we write* $\mathsf{M} \twoheadrightarrow_{\mathsf{c}}^{n} \mathsf{N}$ *whenever* $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ *and* $n \in |\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ *hold.*

As a matter of terminology, we say that an Extended Addressing Machine M:

- is *stuck* if its program has shape $\mathsf{M}.P = \mathtt{Load}\ i; P$ but $\mathsf{M}.T = []$;

- *is in final state* if M is not in an error state, but cannot reduce further, i.e. $\mathsf{M} \not\to_{\mathsf{c}}$;
- *is in an error state* if its program has the shape $\mathsf{M}.P = \mathtt{ins}; P'$ for some particular instruction $\mathtt{ins} \in \{j \leftarrow \mathtt{Pred}(i), j \leftarrow \mathtt{Succ}(i), l \leftarrow \mathtt{Test}(i, j, k)\}$, but $!R_i \notin \mathbb{N}$ and $\#^{-1}(!R_i)$ cannot reduce further.
- *reaches a final state* (resp. *raises an error*) if $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$ for some M' in final (resp. error) state; M *does not terminate*, otherwise.

Given an Extended Addressing Machine M, the first instruction of its program, together with the contents of its registers and tape, univocally determine which rule is applicable (if any). When M tries to perform an arithmetic operation in one of its registers, say $R_i = a$, it needs to wait until the Extended Addressing Machine $\#^{-1}(a)$ has been reduced to a final state. If it does then the success of the operation depends on whether the result is a numeral, otherwise M is in an error state.

Allowing some machines to reduce inside the registers of other machines seems a bit strange at a glance. It is a technical choice, albeit one that is not too surprising – this functionality bears some similarities to the **D**ump of the SECD machine. Some limited method of "returning" from a function call appears to be a necessity for arithmetic and branching to behave well.

Lemma 3.3.12 (Reduction Consistency)**.**

(1) The strategy $\to_{\mathsf{c}}$ *is deterministic:* $\mathsf{N} \,{}_{\mathsf{c}}\!\leftarrow \mathsf{M} \to_{\mathsf{c}} \mathsf{N}'$ *implies* $\mathsf{N} = \mathsf{N}'$.

(2) The reduction $\to_{\mathsf{c}}$ *is Church-Rosser:* $\mathsf{M} \leftrightarrow_{\mathsf{c}} \mathsf{N} \Leftrightarrow \exists \mathsf{Z} \in \mathcal{M}_{\mathbb{A}} \,.\, \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z} \,{}_{\mathsf{c}}\!\twoheadleftarrow \mathsf{N}$.

(3) If $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$*, then* $\mathsf{M} @ [\#\mathsf{N}] \twoheadrightarrow_{\mathsf{c}} \mathsf{M}' @ [\#\mathsf{N}]$.

(4) If M *is in a final state and is not stuck, then* $\mathsf{M}.P = \epsilon$.

Proof.

(1) Trivial: There is at most one applicable reduction rule for each state.

(2) Trivial as a consequence of (1).

(3) By induction on the length of $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$. We will use k as a variable referring to the number of steps in the reduction.

Case $k = 0$ is trivial.

Case $k > 0$: Then $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}_1 \twoheadrightarrow_{\mathsf{c}} \mathsf{M}'$ for some M_1. We will be showing that $\mathsf{M} @ [\#\mathsf{N}] \to_{\mathsf{c}} \mathsf{M}_1 @ [\#\mathsf{N}] \twoheadrightarrow_{\mathsf{c}} \mathsf{M}' @ [\#\mathsf{N}]$ by cases on the first instruction of $\mathsf{M}.P$.

Subcase $\mathsf{M}.P = \mathtt{Call}\ i$. We have $\mathsf{M}_1 = \#^{-1}(!\mathsf{M}.R_i) @ \mathsf{M}.T$. From the rewriting rules we obtain $\left\langle \mathsf{M}.\vec{R}, \mathtt{Call}\ i, \mathsf{M}.T @ [\#\mathsf{N}] \right\rangle \to_{\mathsf{c}} \#^{-1}(!\mathsf{M}.R_i) @ \mathsf{M}.T @ [\#\mathsf{N}]$ and conclude.

Subcase $\mathsf{M}.P = \texttt{Load}\ i; P'$. As M is not in a final state, $\mathsf{M}.T = a :: T'$ for some $a \in \mathbb{A}$ and $T' \in \mathcal{T}_{\mathbb{A}}$. We have $\mathsf{M}_1 = \left\langle \mathsf{M}.\vec{R}[R_i := a], P', T' \right\rangle$. From the rewriting rules we obtain $\left\langle \mathsf{M}.\vec{R}, \texttt{Load}\ i; P', a :: T' \mathbin{@} [\#\mathsf{N}] \right\rangle \to_{\mathsf{c}} \left\langle \mathsf{M}.\vec{R}[R_i := a], P', T' \mathbin{@} [\#\mathsf{N}] \right\rangle$ and conclude.

All other subcases are trivial as they do not affect the tape.

(4) An easy proof by contradiction on the first instruction of $\mathsf{M}.P$, relying on program validity. □

Corollary 3.3.13 (Multiple Reduction Lengths)**.**
Given M, N *such that* $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$, $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ *contains more than one element if and only if there is a reduction path* $\mathsf{N} \to_{\mathsf{c}} \mathsf{N}_1 \to_{\mathsf{c}} \mathsf{N}_2 \to_{\mathsf{c}} \cdots \to_{\mathsf{c}} \mathsf{N}$*, i.e. the machine loops. When this is the case,* $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ *is infinite.*

Proof. This is a direct consequence of Lemma 3.3.12(1). □

Note that if we only have singletons then $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}|$ contains a single natural number, so $\sqsubset$ between singleton sets is a strict total order.

Example 3.3.14.
See Example 3.3.10 for the definition of I, Succ1, Succ2, Add_aux.

(1) For some $a \in \mathbb{A}$, $\mathsf{I} \mathbin{@} [a] \to_{\mathsf{c}} \langle R_0 = a, \texttt{Call}\ 0, [] \rangle \to_{\mathsf{c}} \#^{-1}(a)$

(2)
$$\begin{aligned}
\mathsf{Succ1} \mathbin{@} [0] &= \left\langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [0] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = 0, 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = 1, \texttt{Call}\ 0, [] \right\rangle \\
&\to_{\mathsf{c}} \#^{-1}(1)
\end{aligned}$$

(3)
$$\begin{aligned}
\mathsf{Succ2} \mathbin{@} [1] &= \left\langle \begin{array}{l} R_0, R_1, \texttt{Load}\ 0; \texttt{Load}\ 1; 1 \leftarrow \texttt{App}(0,\ 1); \\ \qquad 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [\#\mathsf{Succ1}, 1] \end{array} \right\rangle \\
&\to_{\mathsf{c}} \left\langle \begin{array}{l} R_0 = \#\mathsf{Succ1}, R_1, \texttt{Load}\ 1; 1 \leftarrow \texttt{App}(0,\ 1); \\ \qquad\qquad 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [1] \end{array} \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = \#\mathsf{Succ1}, R_1 = 1, 1 \leftarrow \texttt{App}(0,\ 1); 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = \#\mathsf{Succ1}, R_1 = \#\mathsf{Succ1} \cdot 1, 1 \leftarrow \texttt{App}(0,\ 1); \texttt{Call}\ 1, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = \#\mathsf{Succ1}, R_1 = \#\mathsf{Succ1} \cdot (\#\mathsf{Succ1} \cdot 1), \texttt{Call}\ 1, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [\#\mathsf{Succ1} \cdot 1] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = \#\mathsf{Succ1} \cdot 1, 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = 2, 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \right\rangle \\
&\to_{\mathsf{c}} \left\langle R_0 = 3, \texttt{Call}\ 0, [] \right\rangle \\
&\to_{\mathsf{c}} \#^{-1}(3)
\end{aligned}$$

(4) Define $\mathsf{Add} = \mathsf{Y} \mathbin{@} [\#\mathsf{Add\_aux}]$*, an Extended Addressing Machine performing the addition. We show:*

$$
\begin{aligned}
\mathsf{Add} \mathbin{@} [1,3] \quad &\to_{\mathsf{c}} \left\langle \begin{array}{r} (R_0 = \#\mathsf{Y}, R_1 = \#\mathsf{Add\_aux}), 0 \leftarrow \mathtt{App}(0,\,1); \\ 1 \leftarrow \mathtt{App}(1,\,0); \mathtt{Call}\ 1, [1,3] \end{array} \right\rangle \\
&\to\!\!\to_{\mathsf{c}} \left\langle \begin{array}{r} \vec{R}, \mathtt{Load}\ (0,1,2); 3 \leftarrow \mathtt{Pred}(1); 4 \leftarrow \mathtt{Succ}(2); 0 \leftarrow \mathtt{App}(0,\,3); \\ 0 \leftarrow \mathtt{App}(0,\,4); 0 \leftarrow \mathtt{Test}(1,\,2,\,0); \mathtt{Call}\ 0, [\#\mathsf{Add}, 1, 3] \end{array} \right\rangle \\
&\to\!\!\to_{\mathsf{c}} \left\langle \begin{array}{r} R_0 = \#\mathsf{Add}, R_1 = 1, R_2 = 3, R_3, R_4, 3 \leftarrow \mathtt{Pred}(1); 4 \leftarrow \mathtt{Succ}(2); \\ 0 \leftarrow \mathtt{App}(0,\,3); 0 \leftarrow \mathtt{App}(0,\,4); 0 \leftarrow \mathtt{Test}(1,\,2,\,0); \mathtt{Call}\ 0, [] \end{array} \right\rangle \\
&\to\!\!\to_{\mathsf{c}} \left\langle \begin{array}{r} R_0 = \#(\mathsf{Add} \mathbin{@} [0,4]\,), R_1 = 1, R_2 = 3, \\ R_3 = 0, R_4 = 4, 0 \leftarrow \mathtt{Test}(1,\,2,\,0); \mathtt{Call}\ 0, [] \end{array} \right\rangle \\
&\to\!\!\to_{\mathsf{c}} \left\langle \begin{array}{r} R_0 = \#(\mathsf{Add} \mathbin{@} [0,5]\,), R_1 = 0, R_2 = 4, \\ R_3 = 0, R_4 = 5, 0 \leftarrow \mathtt{Test}(1,\,2,\,0); \mathtt{Call}\ 0, [] \end{array} \right\rangle \\
&\to\!\!\to_{\mathsf{c}} \#^{-1}(4)
\end{aligned}
$$

3.3.3 Typing Extended Addressing Machines

We now show that certain EAMs can be typed, and that typable machines do not raise an error. EAMs share the same set of types $\mathbb{T}$ as PCF and EPCF.

Definition 3.3.15 (Extended Addressing Machine Typing Contexts)**.**

(1) A typing context Δ *is a finite set of associations between indices and types, represented as a list* $i_1 : \alpha_1, \ldots, i_r : \alpha_r$*. The indices* $i_1, \ldots, i_r$ *are not necessarily consecutive.*

(2) We denote by $\Delta[i : \alpha]$ *the typing context* Δ *where the type associated with* i *becomes* α*. Note that* $\mathrm{dom}(\Delta[i : \alpha]) = \mathrm{dom}(\Delta) \cup \{i\}$*. If* i *is not present in* Δ*, then* $\Delta[i : \alpha] = \Delta, i : \alpha$*.*

Intuitively, registers act as "internal free variables" for an Addressing Machine. Extended Addressing Machine typing contexts are used to assign types to registers based on the addresses stored in them. The types assigned to registers must be updated during the analysis of a machine's program, to reflect changes which may occur during the execution of the machine.

Definition 3.3.16 (Extended Addressing Machine Typing Judgements)**.**
Let Δ *be a typing environment,* $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$*,* $r \geq 0$*,* $R_0, \ldots, R_r$ *be registers,* P *be a program,* $T \in \mathcal{T}_{\mathbb{A}}$ *and* $\alpha \in \mathbb{T}$*. We define the typing judgements*

$$\mathsf{M} : \alpha \qquad\qquad \Delta \Vdash^{r} (P, T) : \alpha \qquad\qquad R_0, \ldots, R_r \models \Delta$$

by mutual induction as the least relations closed under the rules in Figure 3.1, where the rules (nat) *and* (fix) *are given priority.*

$$\frac{}{\mathsf{n} : \mathsf{int}}\ (\mathrm{nat}) \qquad \frac{}{\mathsf{Y} : (\alpha \to \alpha) \to \alpha}\ (\mathrm{fix}) \qquad \frac{R_0, \ldots, R_r \models \Delta \quad \Delta \Vdash^r (P, T) : \alpha}{\langle R_0, \ldots, R_r, P, T\rangle : \alpha}\ (\vec{R})$$

$$\frac{R_0, \ldots, R_{r-1} \models \Delta \quad !R_r = \varnothing}{R_0, \ldots, R_r \models \Delta}\ (R_\varnothing) \qquad \frac{R_0, \ldots, R_{r-1} \models \Delta \quad \#^{-1}(!R_r) : \alpha}{R_0, \ldots, R_r \models \Delta, r : \alpha}\ (R_\mathbb{T})$$

$$\frac{\Delta[i : \beta] \Vdash^r (P, []) : \alpha}{\Delta \Vdash^r (\mathtt{Load}\ i; P, []) : \beta \to \alpha}\ (\mathrm{load}_\varnothing) \qquad \frac{\Delta[i : \beta] \Vdash^r (P, T) : \alpha \quad \#^{-1}(a) : \beta}{\Delta \Vdash^r (\mathtt{Load}\ i; P, a :: T) : \alpha}\ (\mathrm{load}_\mathbb{T})$$

$$\frac{(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int} \Vdash^r (j \leftarrow \mathtt{Pred}(i); P, T) : \alpha}\ (\mathrm{pred}) \qquad \frac{(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int} \Vdash^r (j \leftarrow \mathtt{Succ}(i); P, T) : \alpha}\ (\mathrm{succ})$$

$$\frac{(\Delta, i : \mathsf{int}, j : \beta, k : \beta)[l : \beta] \Vdash^r (P, T) : \alpha}{\Delta, i : \mathsf{int}, j : \beta, k : \beta \Vdash^r (l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T) : \alpha}\ (\mathrm{test})$$

$$\frac{(\Delta, i : \alpha \to \beta, j : \alpha)[k : \beta] \Vdash^r (P, T) : \delta}{\Delta, i : \alpha \to \beta, j : \alpha \Vdash^r (k \leftarrow \mathtt{App}(i,\ j); P, T) : \delta}\ (\mathrm{app})$$

$$\frac{\mathsf{M}_1 : \alpha_1 \quad \cdots \quad \mathsf{M}_n : \alpha_n}{\Delta, i : \alpha_1 \to \cdots \to \alpha_n \to \alpha \Vdash^r (\mathtt{Call}\ i, [\#\mathsf{M}_1, \ldots, \#\mathsf{M}_n]) : \alpha}\ (\mathrm{call})$$

Figure 3.1 EPCF typing judgement rules

A machine M is called *typable* if the judgement $\mathsf{M} : \alpha$ is derivable for some $\alpha \in \mathbb{T}$.

The process of typing an Extended Addressing Machine is best thought of as an algorithm where one reasons bottom-up. To give a machine M a type α, one needs to derive the judgement $\mathsf{M} : \alpha$, which proceeds roughly as follows:

- The machines n and Y are recognizable from their addresses, and one immediately applies rule (nat) or (fix) respectively when encountered.
- For all other cases:
  - One begins by giving all the registers a type using the rule $(\vec{R})$, applying $(R_\mathbb{T})$ or $(R_\varnothing)$ for each register. During the process of doing so, one recursively types all machines whose addresses are present in the registers.
  - Once all registers have been given an initial type, one needs to derive a judgement of the form $i_1 : \beta_{i_1}, \ldots, i_n : \beta_{i_n} \Vdash^r (P, T) : \alpha$, where P and T are the program and the input tape of the original machine respectively. This is done by verifying the coherence of the instructions in the program with the types of the registers and of the values in the input tape.

Remark 3.3.17.

(1) Extended Addressing Machine typing judgements are deterministic only with the additional priority given to the (nat) *and* (fix) *rules.*

(2) With the additional priority given to the (nat) *and* (fix) *rules, each machine has a unique type derivation for each type that it is typable with.*

(3) Aside from the numerals, all typable machines have their program ending with a $\texttt{Call}\ i$.

(4) Typable machines in a final state are numerals or stuck.

(5) For all $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ *and* $\alpha \in \mathbb{T}$*, we have* $\mathsf{M} : \alpha$ *if and only if there exists* $a \in \mathbb{A}$ *such that both* $\#^{-1}(a) : \alpha$ *and* $\#\mathsf{M} = a$ *hold.*

(6) If $\#\mathsf{M} \notin \mathbb{N} \cup \{\#\mathsf{Y}\}$*, then* $\mathsf{M} : \alpha \iff \exists \Delta \,.\, [\Delta \models \mathsf{M}.\vec{R} \ \wedge\ \Delta \Vdash^r (\mathsf{M}.P, \mathsf{M}.T) : \alpha]$

(7) The superscript $r \geq 0$ *in* $\Vdash^r$ *keeps track of the initial amount of registers, i.e.* $\vec{i} \in \{0, \ldots, r\}$.

(8) The algorithm does not differentiate between addresses in the tape which are stored in registers and those which are discarded – the machines found at addresses whose only fate is to be discarded must still be typable for the overall machine to be typable.

(9) While in many typing systems terms which are not typable typically feature unusual behaviour, there exist a number of untypable EAMs that are well behaved. In particular, the empty machine $\left\langle R_0, \epsilon, [] \right\rangle$ *is not typable.*

For a machine to be typable one needs to be able to construct a finite typing derivation tree, thus by default any machines which feature circular references or infinite chains as mentioned in Remark 3.3.9 are not typable. Thus the (nat) and (fix) rules are necessary to allow those machines to be typable. It is necessary for the (nat) to have priority to allow the numerals to be typable as their program does not have any instructions. The (fix) rule having a higher priority does not modify the set of typable machines, but guarantees the syntax-directedness of the system.

As a final consideration, notice that the rules presented in Definition 3.3.16 can only be considered as an algorithm when the address table map is effectively given. Otherwise, the algorithm would depend on an oracle deciding $a = \#\mathsf{M}$.

Example 3.3.18.
The following are some examples of derivable typing judgements.

(1) I *can be typed with* $\alpha \to \alpha$ *for any* $\alpha \in \mathbb{T}$*:*

$$\dfrac{\dfrac{!R_0 = \varnothing}{R_0 \models} \quad \dfrac{\dfrac{}{0 : \mathsf{int} \Vdash^1 \langle \texttt{Call}\ 0, [] \rangle : \alpha}}{\Vdash^1 \langle \texttt{Load}\ 0; \texttt{Call}\ 0, [] \rangle : \alpha \to \alpha}}{\langle R_0 = \varnothing, \texttt{Load}\ 0; \texttt{Call}\ 0, [] \rangle : \alpha \to \alpha}$$

(2) $\mathsf{Succ1}$ *can be typed with* $\mathsf{int} \to \mathsf{int}$*:*

$$\dfrac{\dfrac{!R_0 = \varnothing}{R_0 \models} \quad \dfrac{\dfrac{\dfrac{}{0 : \mathsf{int} \Vdash^1 \langle \texttt{Call}\ 0, [] \rangle : \mathsf{int}}}{0 : \mathsf{int} \Vdash^1 \langle 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \rangle : \mathsf{int}}}{\Vdash^1 \langle \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \rangle : \mathsf{int} \to \mathsf{int}}}{\langle R_0 = \varnothing, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [] \rangle : \mathsf{int} \to \mathsf{int}}$$

(3) $\mathsf{Succ2} : \mathsf{int} \to \mathsf{int}$ *and* $\mathsf{Add} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$ *are also derivable typing judgements, but the full trees are omitted due to being unreasonably large.*

Lemma 3.3.19 (Typing Decidability)**.**
Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$, $\alpha \in \mathbb{T}$. *Assume that* $\# : \mathsf{M} \to \mathbb{A}$ *is effectively given.*

(1) If $\mathsf{M} = \langle \vec{R} = \varnothing, P, [] \rangle$ *then the typing algorithm is capable of deciding whether* $\mathsf{M} : \alpha$ *holds, for some* $\alpha \in \mathbb{T}$.

(2) In general, the typing algorithm semi-decides whether $\mathsf{M} : \alpha$ *holds.*

Proof.

(1) By the rules $(\vec{R})$ and $(R_\varnothing)$, $\mathsf{M} : \alpha$ holds if and only if $\Vdash^r (\mathsf{M}.P, []) : \alpha$ does. We prove the more general statement "Given a typing environment Δ, a program P, $r \in \mathbb{N}, \alpha \in \mathbb{T}$, the typing algorithm is capable of deciding whether $\Delta \Vdash^r (P, []) : \alpha$ holds."

We will proceed by induction on the length of P, using k as a meta-variable representing the length. We will collect a list of constraints C on the types while performing the induction, to conclude with at the end (We will reuse the list notation used for the tape when discussing C). We also introduce an infinite set of type variables $\mathrm{Var}^{\mathbb{T}}$. We will begin by setting $C := [(\alpha, \delta)]$, where δ is fresh, and then commence the induction.

Case $k = 0$: There is no case for the empty program, hence $\Delta \Vdash^r (\epsilon, []) : \alpha$ does not hold.

Case $k = 1$: There is only one subcase of note: $P = \texttt{Call}\ i$. Let β be a fresh type variable. Let $\Delta = \Delta', i : \beta$. As the tape is empty, and we know that $\Delta', i : \delta \Vdash^r (\texttt{Call}\ i, []) : \delta$ holds using rule (call), we set $C := (\beta, \delta) :: C$ and conclude that any type substitution which respects the constraints found in C produces a correct typing. For all other subcases we immediately conclude that $\Delta \Vdash^r (P, []) : \alpha$ does not hold by Remark 3.3.17(3).

Case $k > 1$: We have five subcases:

- $P = \texttt{Load}\ i; P'$: Let β, γ be fresh type variables. We apply the induction hypothesis to identify whether or not $\Delta[i : \beta] \Vdash^r (P', []) : \gamma$ holds and set $C := (\beta \to \gamma, \delta) :: C$. By $(\mathrm{load}_\emptyset)$, $\Delta \Vdash^r (\texttt{Load}\ i; P', []) : \beta \to \gamma$, so we conclude that any type substitution which respects the constraints found in C produces a correct typing.
- $P = k \leftarrow \texttt{App}(i,\ j); P'$: Let $\beta_1, \beta_2, \beta_3$ be fresh type variables. Let $\Delta = \Delta', i : \beta_1, j : \beta_2$. We check whether $(\Delta', i : \beta_2 \to \beta_3, j : \beta_2)[k : \beta_3] \Vdash^r (P', []) : \delta$ holds by applying the IH, and we set $C = (\beta_1, \beta_2 \to \beta_3) :: C$. Then, by (app), any type substitution which respects the constraints found in C produces a correct typing.
- $P = j \leftarrow \texttt{Pred}(i); P'$: Let $\Delta = \Delta', i : \beta$ where β is a fresh type variable. We apply the IH to check whether $(\Delta', i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P', []) : \delta$ holds and we set $C = (\beta, \mathsf{int}) :: C$. Then, by (pred), any type substitution which respects the constraints found in C produces a correct typing.

- $P = j \leftarrow \mathtt{Succ}(i); P'$: Analogous.
- $P = l \leftarrow \mathtt{Test}(i, j, k); P'$: Let $\Delta = \Delta', i : \beta_1, j : \beta_2, k : \beta_3$, where $\beta_1, \beta_2, \beta_3$ are fresh type variables. We check whether $(\Delta', i : \mathsf{int}, j : \beta_2, k : \beta_2)[l : \beta_2] \Vdash^r (P', []) : \alpha$ holds by applying the IH, and we set $C = (\beta_1, \mathsf{int}) :: (\beta_2, \beta_3) :: C$. Then, by (test), any type substitution which respects the constraints found in C produces a correct typing.

To check whether all of the constraints are satisfied we make use of the unification algorithm developed by Robinson [Rob65]. [5][6]

Proposition 3.3.20 (Robinson)**.**
There is an algorithm U which, given a pair of simple types, either returns a substitution S or fails. This algorithm exhibits the following properties:

- *If $U(\alpha_1, \alpha_2)$ returns S, then S unifies α_1 and α_2, i.e. $S(\alpha_1) = \alpha_2$.*
- *If V unifies α_1 and α_2, then $U(\alpha_1, \alpha_2)$ returns some S and there is another substitution R such that $V = RS$.*

Moreover, S involves only variables in α_1 and α_2.

When we write $S(C)$ (a substitution applied to a list of constraints), we take this to mean the list resulting from applying S elementwise to every element found in the list, i.e. if the list $C = [(\alpha_1, \alpha_2), (\beta_1, \beta_2)]$, then $S(C) = [(S(\alpha_1), S(\alpha_2)), (S(\beta_1), S(\beta_2))]$. We use the algorithm to define the function $\mathrm{maybe\_satisfied}$ which takes in a list of constraints and returns $\mathrm{success}$ or fail depending on whether the list of constraints can be satisfied or not.

$$\mathrm{maybe\_satisfied}(C) = \begin{cases} \mathrm{success} & \text{if } C = [], \\ \mathrm{success} & \text{if } \begin{pmatrix} C = (\alpha_1, \alpha_2) :: C', \\ U(\alpha_1, \alpha_2) = S, \\ S(C') = \mathrm{success} \end{pmatrix}, \\ \mathrm{fail} & \text{otherwise.} \end{cases}$$

(2) In the rules $(R_{\mathbb{T}})$ and $(\mathrm{load}_{\mathbb{T}})$, one needs to show that a type for the premises exists. As the set of types is countable, and effectively given, one can easily design an algorithm constructing a derivation tree (by dovetailing). However, the algorithm cannot terminate when executed on the machine M_0 defined in Remark 3.3.9(3) because it would require an infinite derivation tree.

□

[5]The general idea is to use the standard Hindley-Milner algorithm for type inference [DM82], adjusted for the presentation given.

[6]The algorithm was developed for use in Robinson's work on first order logic, but its syntactic nature makes it widely applicable.

Proposition 3.3.21 (Typing Properties)**.**
Let $\mathsf{M}, \mathsf{M}', \mathsf{N}, \in \mathcal{M}_{\mathbb{A}}$ *and* $\alpha, \beta \in \mathbb{T}$.

(1) If $\mathsf{M} : \beta \to \alpha$ *and* $\mathsf{N} : \beta$ *then* $\mathsf{M} \mathbin{@} [\#\mathsf{N}] : \alpha$.

(2) If $\mathsf{M} : \alpha$ *and* $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$ *then* $\mathsf{N} : \alpha$.

(3) If $\mathsf{M} : \mathsf{int}$ *then either* M *does not terminate or* $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$*, for some* $n \geq 0$.

(4) If $\mathsf{M} : \alpha$ *then* M *does not raise an error.*

Proof.

(1) Simultaneously, one proves that if both $\Delta \Vdash^r (P, T) : \beta \to \alpha$ and $\mathsf{N} : \beta$ hold, then so does $\Delta \Vdash^r (P, T \mathbin{@} [\#\mathsf{N}]) : \alpha$. We proceed by induction on a derivation of $\mathsf{M} : \beta \to \alpha$ (resp. $\Delta \Vdash^r (P, T) : \beta \to \alpha$).

Case (nat) is vacuous.

Case (fix). By definition of Y (see Definition 3.3.7(2)), we have:

$$\mathsf{Y} \mathbin{@} [\#\mathsf{N}] = \Big\langle R_0 = \varnothing, R_1 = \varnothing, \mathtt{Load}\ 0; 1 \leftarrow \mathtt{App}(1,\ 0); 0 \leftarrow \mathtt{App}(0,\ 1); \mathtt{Call}\ 0, [\#\mathsf{Y}, \#\mathsf{N}] \Big\rangle$$

Notice that, in this case, $\beta = \alpha \to \alpha$. We derive (omitting the $(R_\varnothing)$ rule usages) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 : \alpha, 1 : \alpha \Vdash^2 (\mathtt{Call}\ 1, []) : \alpha}}{0 : \alpha, 1 : \beta \Vdash^2 (1 \leftarrow \mathtt{App}(1,\ 0); \ldots, []) : \alpha}}{0 : \beta \to \alpha, 1 : \beta \Vdash^2 (0 \leftarrow \mathtt{App}(0,\ 1); \ldots, []) : \alpha} \quad \vdash \mathsf{N} : \beta}{0 : \beta \to \alpha \Vdash^2 \langle \mathtt{Load}\ 1; \ldots, [\#\mathsf{N}] \rangle : \alpha} \qquad \dfrac{}{\mathsf{Y} : \beta \to \alpha}}{\dfrac{\Vdash^2 \langle \mathsf{Y}.P, [\#\mathsf{Y}, \#\mathsf{N}] \rangle : \alpha}{\mathsf{Y} \mathbin{@} [\#\mathsf{N}] : \alpha}}$$

Case $\mathrm{load}_\varnothing$. Then $P = \mathtt{Load}\ i; P'$, $T = []$ and $\Delta[i : \beta] \Vdash^r (P', []) : \alpha$. By assumption $\mathsf{N} : \beta$, so we conclude $\Delta \Vdash^r (\mathtt{Load}\ i; P', [\#\mathsf{N}]) : \alpha$ by applying $\mathrm{load}_{\mathbb{T}}$.

All other cases derive straightforwardly from the IH.

(2) The cases $\mathsf{M} = \mathsf{Y}$ or $\mathsf{M} = \mathsf{n}$ for some $n \in \mathbb{N}$ are vacuous, as these machines are in final state. Otherwise, by Remark 3.3.17(6), $\Delta \Vdash^r (\mathsf{M}.P, \mathsf{M}.T) : \alpha$ for some $\Delta \models \mathsf{M}.\vec{R}$, with r referring to the number of registers of M. By cases on the shape of $\mathsf{M}.P$.

Case $P = \mathtt{Load}\ i; P'$. Then $\mathsf{M}.T = a :: T'$ otherwise M would be in final state, and $\mathsf{N} = \langle \vec{R}[R_i := a], P', T' \rangle$. From $(\mathrm{Load}_{\mathbb{T}})$ we get $\Delta[i : \beta] \Vdash^r (P', T') : \alpha$ for some $\beta \in \mathbb{T}$ satisfying $\#^{-1}(a) : \beta$. As $\vec{R} \models \Delta$ we derive $\vec{R}[R_i := a] \models \Delta[i : \beta]$, so as $N = \Big\langle \vec{R}[R_i := a], P', T' \Big\rangle$, by Remark 3.3.17(6), $N : \alpha$.

Case $P = \texttt{Call}\ i$. Then $i : \alpha_1 \to \cdots \to \alpha_n \to \alpha$, $T = [\#\mathsf{M}_1, \ldots, \#\mathsf{M}_n]$ and $\mathsf{M}_j : \alpha_j$, for all $j \leq n$. In this case, $\mathsf{N} = \#^{-1}(!(\mathsf{M}.R_i)) \mathbin{@} T$ with $\#^{-1}(!(\mathsf{M}.R_i)) : \alpha_1 \to \cdots \to \alpha_n \to \alpha$, so we conclude by (1).

Case $P = k \leftarrow \texttt{App}(i,\ j); P'$. We have

$$\Delta, i : \beta \to \gamma, j : \beta \models \mathsf{M}.\vec{R},\ (\Delta, i : \beta \to \gamma, j : \beta)[k : \gamma] \Vdash^r (P', \mathsf{M}.T) : \alpha$$
$$\mathsf{N} = \left\langle \mathsf{M}.\vec{R}[R_k :=\! !\mathsf{M}.R_i \cdot !\mathsf{M}.R_j], \mathsf{M}.P, \mathsf{M}.T \right\rangle$$

By (1), $\#!\mathsf{M}.R_i \mathbin{@} [!\mathsf{M}.R_j] : \gamma$, so we have

$$(\Delta, i : \beta \to \gamma, j : \beta)[k : \gamma] \models \mathsf{M}.\vec{R}[R_k :=\! !\mathsf{M}.R_i \cdot !\mathsf{M}.R_j]$$

We conclude by Remark 3.3.17(6).

Case $P = j \leftarrow \texttt{Pred}(i); P'$. We have $(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P', \mathsf{M}.T) : \alpha$ and we have $\Delta, i : \mathsf{int} \models \mathsf{M}.\vec{R}$. There are two subcases:

- $!R_i \in \mathbb{N}$: Then $\mathsf{N} = \left\langle \mathsf{M}.\vec{R}[R_j :=\! !R_i \ominus 1], P', \mathsf{M}.T \right\rangle$. We can now conclude by Remark 3.3.17(6), as $(\Delta, i : \mathsf{int})[j : \mathsf{int}] \models \mathsf{M}.\vec{R}[R_j :=\! !R_i \ominus 1]$.
- $!R_i \notin \mathbb{N}$: Then there must exist a L such that $\mathsf{N} = \left\langle \mathsf{M}.\vec{R}[R_j := \mathsf{L}], \mathsf{M}.P, \mathsf{M}.T \right\rangle$ and $\#!R_i \to_{\mathsf{c}} \mathsf{L}$, so we use the induction hypothesis to infer that $\mathsf{L} : \mathsf{int}$ and thus we have $(\Delta, i : \mathsf{int})[j : \mathsf{int}] \models \mathsf{M}.\vec{R}[R_j := \mathsf{L}]$. We conclude by Remark 3.3.17(6).

Case $P = j \leftarrow \texttt{Succ}(i); P'$. Analogous.

Case $P = l \leftarrow \texttt{Test}(i,\ j,\ k); P'$. Analogous.

(3) Assume that $\mathsf{M} : \mathsf{int}$ and $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}$ for some N in final state. By (2), we obtain that $\mathsf{N} : \mathsf{int}$ holds. By Remark 3.3.17(4), numerals are the only machines in final state typable with int, thus $\mathsf{N} = \mathsf{n}$.

(4) The three reduction cases where a machine can raise an error are ruled out by the typing rules (pred), (succ) and (test), respectively. Therefore by (2), no error can be raised during the execution. □

Chapter 4

Modelling PCF

We will now move on to constructing a model of PCF based on Extended Addressing Machines, which we will find to be fully abstract. To do so, we first define a *translation* from EPCF to Extended Addressing Machines. We then prove that both the typing and the operational semantics of EPCF programs are preserved under translation. This translation will later be used to define the interpretation function $[\![-]\!]$ introduced in Definition 1.3.5.

4.1 Translating EPCF Terms into EAMs

4.1.1 Auxiliary EAMs

We start by defining some auxiliary Extended Addressing Machines implementing the main instructions of PCF.

Definition 4.1.1 (Auxiliary EAMs)**.**
Define the following Extended Addressing Machines (for $k > 0$ and $n \geq 0$):

$$
\begin{aligned}
\mathsf{Pr}_i^k &= \left\langle R_0, (\texttt{Load}\ 1)^{i-1}; \texttt{Load}\ 0; (\texttt{Load}\ 1)^{k-i}; \texttt{Call}\ 0; [\,] \right\rangle, \textit{ for } 1 \leq i \leq k; \\
\mathsf{Pred} &= \left\langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Pred}(0); \texttt{Call}\ 0; [\,] \right\rangle; \\
\mathsf{Succ} &= \left\langle R_0, \texttt{Load}\ 0; 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0; [\,] \right\rangle; \\
\mathsf{Ifz} &= \left\langle R_0, R_1, R_2, \texttt{Load}\ 0; \texttt{Load}\ 1; \texttt{Load}\ 2; 0 \leftarrow \texttt{Test}(0,\ 1,\ 2); \texttt{Call}\ 0; [\,] \right\rangle; \\
\mathsf{Apply}_0^k &= \mathsf{Pr}_1^1, \textit{ for } k > 0. \textit{ Recall that the EAM } \mathsf{Pr}_1^1 = \mathsf{I} \textit{ represents the identity}; \\
\mathsf{Apply}_{n+1}^k &= \left\langle \begin{array}{c} R_0 = \#\mathsf{Apply}_n^k, R_1, \ldots, R_{k+2}, \texttt{Load}\ (1, \ldots, k+2); \\ 2 \leftarrow \texttt{App}(2,\ k+2); \cdots; k+1 \leftarrow \texttt{App}(k+1,\ k+2); \\ 0 \leftarrow \texttt{App}(0,\ 1); \cdots; 0 \leftarrow \texttt{App}(0,\ k+1); \texttt{Call}\ 0, [\,] \end{array} \right\rangle.
\end{aligned}
$$

The registers whose values are not specified are assumed to be uninitialized.

By Remark 3.3.9(5), the above machines exist regardless of the choice of address table map. The purpose of the machines $\mathsf{Pred}, \mathsf{Succ}, \mathsf{Ifz}$ are self-evident, trivially implementing their respective instructions in machine form. The Pr machines can be understood as "projection" machines – extracting the i-th address from a tuple (represented as a list, or a section of a list) of length k.

The Apply machines are a bit more complex. An Apply_n^k machine is stuck, waiting for $n+k+1$ arguments $a, d_1, \ldots, d_k, e_1, \ldots, e_n$, where k is the arity of a and n is the arity of each d_i. Once all arguments are available in the tape, Apply_n^k applies $\vec{e}$ to each d_i and then feeds the machine $\#^{-1}(a)$ the resulting list of arguments $d_1 \cdot \vec{e}, \ldots, d_k \cdot \vec{e}$.

Lemma 4.1.2.
For all $a, b, c, d_1, \ldots, d_k, e_1, \ldots, e_n \in \mathbb{A}$, the Extended Addressing Machines below reduce as follows:

(1) $\mathsf{Pr}_i^k \mathbin{@} [d_1, \ldots, d_k] \twoheadrightarrow_{\mathsf{c}}^{k+1} \#^{-1}(d_i)$, *for* $1 \leq i \leq k > 0$;

(2) $\mathsf{Apply}_n^k \mathbin{@} [a, d_1, \ldots, d_k, e_1, \ldots, e_n] \twoheadrightarrow_{\mathsf{c}}^{(3k+4)n+2} \#^{-1}(a) \mathbin{@} [d_1 \cdot e_1 \cdots e_n, \ldots, d_k \cdot e_1 \cdots e_n]$;

(3) $\mathsf{Pred} \mathbin{@} [a] \to_{\mathsf{c}} \left\langle R_0 = a, 0 \leftarrow \texttt{Pred}(0); \texttt{Call}\ 0, [\,] \right\rangle$;

(4) $\mathsf{Succ} \mathbin{@} [a] \to_{\mathsf{c}} \left\langle R_0 = a, 0 \leftarrow \texttt{Succ}(0); \texttt{Call}\ 0, [\,] \right\rangle$;

(5) $\mathsf{Ifz} \mathbin{@} [a, b, c] \twoheadrightarrow_{\mathsf{c}}^{3} \left\langle R_0 = a, R_1 = b, R_2 = c, 0 \leftarrow \texttt{Test}(0,\ 1,\ 2); \texttt{Call}\ 0, [\,] \right\rangle$;

(6) $\mathsf{Y} \mathbin{@} [a] \twoheadrightarrow_{\mathsf{c}}^{5} \#^{-1}(a) \mathbin{@} [\#(\mathsf{Y} \mathbin{@} [a]\,)]$.

Proof. The only interesting cases are the application and fixed point combinator.

(2) Concerning Apply_n^k, we proceed by induction on n.

- Base case. If $n = 0$ then $\mathsf{Apply}_0^k = \mathsf{I}$, and $\mathsf{I} \mathbin{@} [a, d_1, \ldots, d_k] \twoheadrightarrow_{\mathsf{c}}^{2} \#^{-1}(a) \mathbin{@} [d_1, \ldots, d_k]$.

- Induction case. Easy calculations give:

  $\mathsf{Apply}_{n+1}^k \mathbin{@} [a, d_1, \ldots, d_k, e_1, \ldots, e_{n+1}] \twoheadrightarrow_{\mathsf{c}}^{3k+4} \mathsf{Apply}_n^k \mathbin{@} [a, d_1 \cdot e_1, \ldots, d_k \cdot e_1, e_2, \ldots, e_{n+1}]$

  This case follows from the IH since $3k + 4 + (3k + 4)n + 2 = (3k + 4)(n + 1) + 2$.

(6) Recall that Y has been introduced in Definitions 3.3.6(2) and 3.3.7(2).

$$\begin{array}{rcl}
\mathsf{Y} \mathbin{@} a & = & \langle R_0, R_1, \mathtt{Load}\ (0,1); 0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1, [\#\mathsf{Y}, a] \rangle \\
& \to_{\mathsf{c}}^2 & \langle R_0 = \#\mathsf{Y}, R_1 = a, 0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1, [\,] \rangle \\
& \to_{\mathsf{c}} & \langle R_0 = \#\mathsf{Y} \cdot a, R_1 = a, 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1, [\,] \rangle \\
& \to_{\mathsf{c}} & \langle R_0 = \#\mathsf{Y} \cdot a, R_1 = a \cdot (\#\mathsf{Y} \cdot a), \mathtt{Call}\ 1, [\,] \rangle \\
& \to_{\mathsf{c}} & \#^{-1}(a \cdot (\#\mathsf{Y} \cdot a)) = \#^{-1}(a) \mathbin{@} [\#(\mathsf{Y} \mathbin{@} [a]\,)]\,.
\end{array}$$

This concludes the proof. □

We show that the above machines are actually typable using the type assignment system. This property is needed to show that the translation is type-preserving.

Lemma 4.1.3.
The EAMs introduced in Definition 4.1.1 can be typed as follows (for all $\alpha, \beta_i, \delta_i \in \mathbb{T}$, using the notation $\vec{\delta} \to \beta_i = \delta_1 \to \cdots \to \delta_n \to \beta_i$):

(1) $\mathsf{Pr}_i^k : \beta_1 \to \cdots \to \beta_k \to \beta_i$, *for* $1 \le i \le k > 0$;

(2) $\mathsf{Apply}_n^k : (\beta_1 \to \cdots \to \beta_k \to \alpha) \to (\vec{\delta} \to \beta_1) \to \cdots \to (\vec{\delta} \to \beta_k) \to \vec{\delta} \to \alpha$;

(3) $\mathsf{Pred} : \mathsf{int} \to \mathsf{int}$;

(4) $\mathsf{Succ} : \mathsf{int} \to \mathsf{int}$;

(5) $\mathsf{Ifz} : \mathsf{int} \to \alpha \to \alpha \to \alpha$.

Proof. It follows easily from Definition 4.1.1. We will show Apply_n^k by induction on n. All other cases are trivial.

Base case: if $n = 1$, then $\mathsf{Apply}_n^k = \mathsf{Pr}_1^1$, and we have $\mathsf{Pr}_1^1 : \alpha \to \alpha$ by (1).

Induction case: We use the following notations in addition to those already named:

$$\begin{array}{rcl}
\vec{\delta'} \to \beta & = & \delta_2 \to \cdots \to \delta_n \to \beta \\
\sigma & = & (\beta_1 \to \cdots \to \beta_k \to \alpha) \to (\vec{\delta} \to \beta_1) \to \cdots \to (\vec{\delta} \to \beta_k) \to \vec{\delta} \to \alpha \\
\tau & = & (\beta_1 \to \cdots \to \beta_k \to \alpha) \to (\vec{\delta'} \to \beta_1) \to \cdots \to (\vec{\delta'} \to \beta_k) \to \vec{\delta'} \to \alpha \\
\Delta & = & 0 : \tau, 1 : \beta_1 \to \cdots \to \beta_k \to \alpha, 2 : \vec{\delta} \to \beta_1, \ldots, k+1 : \vec{\delta} \to \beta_k, k+2 : \delta_1 \\
\Delta' & = & 0 : \tau, 1 : \beta_1 \to \cdots \to \beta_k \to \alpha, 2 : \vec{\delta'} \to \beta_1, \ldots, k+1 : \vec{\delta'} \to \beta_k, k+2 : \delta_1 \\
\Delta'' & = & 0 : \alpha, 1 : \beta_1 \to \cdots \to \beta_k \to \alpha, 2 : \vec{\delta'} \to \beta_1, \ldots, k+1 : \vec{\delta'} \to \beta_k, k+2 : \delta_1
\end{array}$$

Using the rules $(\vec{R})$, $(R_\varnothing)$, and $(R_\mathbb{T})$, we get $\mathsf{Apply}_j^k : \tau$ by IH, and have to show

$$0 : \tau \Vdash^{k+3} (\mathsf{Apply}_j^k.P, [\,]) : \sigma$$

Using $k+2$ $(\mathrm{load}_\varnothing)$ cases, we obtain

$$\Delta \Vdash^{k+2} \left(\begin{array}{l} 2 \leftarrow \mathtt{App}(2,\, k+2); \cdots ; k+1 \leftarrow \mathtt{App}(k+1,\, k+2); \\ 0 \leftarrow \mathtt{App}(0,\, 1); \cdots ; 0 \leftarrow \mathtt{App}(0,\, k+1); \mathtt{Call}\ 0 \end{array} , [\,] \right) : \alpha$$

We use k (app) cases to obtain

$$\Delta \Vdash^{k+2} (0 \leftarrow \mathtt{App}(0,\, 1); \cdots ; 0 \leftarrow \mathtt{App}(0,\, k+1); \mathtt{Call}\ 0, [\,]) : \alpha$$

With another $k+1$ (app) cases we get

$$\Delta' \Vdash^{k+2} (\mathtt{Call}\ 0, [\,]) : \alpha$$

We then conclude with a (call). □

4.1.2 Translating (E)PCF programs to EAMs

Using the auxiliary EAM introduced above, we can translate an EPCF term M having $x_1, \ldots, x_n$ as free variables as an EAM M loading their values from the input tape.

Definition 4.1.4 (Translation)**.**
Let M be an EPCF *term such that* $\mathrm{FV}(M) \subseteq \{x_1, \ldots, x_n\}$*. The* translation *of M (w.r.t. $\vec{x}$) is an EAM $|M|_{\vec{x}}$ defined by structural induction on M:*

$$\begin{array}{lcl}
|x_i|_{\vec{x}} & = & \mathsf{Pr}_i^n,\ \textit{where } i \in \{1, \ldots, n\}; \\
|\lambda y.M|_{\vec{x}} & = & |M|_{\vec{x},y},\ \textit{where wlog } y \notin \vec{x}; \\
|M\langle N/y\rangle|_{\vec{x}} & = & |M|_{y,\vec{x}} \mathbin{@} [\#|N|]\,; \\
|MN|_{\vec{x}} & = & \mathsf{Apply}_n^2 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}]\,; \\
|\mathbf{0}|_{\vec{x}} & = & \mathsf{Pr}_1^{n+1} \mathbin{@} [0]\,; \\
|\mathbf{pred}\, M|_{\vec{x}} & = & \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Pred}, \#|M|_{\vec{x}}]\,; \\
|\mathbf{succ}\, M|_{\vec{x}} & = & \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Succ}, \#|M|_{\vec{x}}]\,; \\
|\mathbf{ifz}(L, M, N)|_{\vec{x}} & = & \mathsf{Apply}_n^3 \mathbin{@} [\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}]\,; \\
|\mathbf{fix}\, M|_{\vec{x}} & = & \begin{cases} \mathsf{Y} \mathbin{@} [\#|M|_{\vec{x}}]\,, & \textit{if } n = 0, \\ \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Y}, \#|M|_{\vec{x}}]\,, & \text{otherwise.} \end{cases}
\end{array}$$

Extended Addressing Machines do not really have free variables – the closest counterpart are the internal registers, but these cannot be "accessed" externally. We take advantage of the duality between free variables and abstractions to treat them the same in translation – they are both "empty slots" in the input tape. Whereas we usually treat the free variables of M as a set,

the translation sees them as list where the order is crucial to ensure that we project to the correct machine when encountering a variable.

The multiple appearances of Apply_n^k machines also deserves discussion. Essentially, these machines exist to propagate free variables between terms. Rather than identifying which free variables are present within which subterm, we propagate all free variables among all sub-Extended Addressing Machines, using the projections to obtain the machine corresponding to a variable when a variable is encountered. This is in spirit exactly the same functionality as explicit substitutions. Explicit substitutions being the internal language of Extended Addressing Machines is very apparent when comparing the definitions of $|M\langle N/y\rangle|_{\vec{x}}$ and $|MN|_{\vec{x}}$.

The definition of the translation also sheds additional light on some choices made earlier on – specifically, the strange closure requirement on EPCF. To remove the closure requirement, we would need to propagage the free variables in the explicit substitution case as well. This approach would lead to $|M\langle N/y\rangle|_{\vec{x}}$ and $|MN|_{\vec{x}}$ being defined in a very similar manner, so there would be no EPCF term which corresponds to application in the Extended Addressing Machine sense. The result of such an approach is issues appearing with the consistency of the fixed point.

Example 4.1.5.
Recall the EPCF *terms introduced in Examples 1.1.2 and 1.2.6. The translation of some of said* EPCF *terms produces the following machines:*

(1) $|\mathbf{I}| = |x|_x = \mathsf{Pr}_1^1$.

(2) $|\Omega^{\mathsf{PCF}}| = \mathsf{Y} \mathbin{@} [\#|\mathbf{I}|] = \mathsf{Y} \mathbin{@} [\#\mathsf{Pr}_1^1]$.

(3) $|\mathtt{succ1}| = |\lambda x.\mathbf{succ}\, x| = |\mathbf{succ}\, x|_x = \mathsf{Apply}_1^1 \mathbin{@} [\#\mathsf{Succ}, \#\mathsf{Pr}_1^1]$,

(4)
$$\begin{aligned} |\mathtt{succ2}| &= \mathsf{Apply}_2^0 \mathbin{@} [\#|s \cdot (s \cdot n)|_{s,n} \cdot \#|\mathtt{succ1}|] \\ &= \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot \#|s \cdot n|_{s,n}, \#|\mathtt{succ1}|] \\ &= \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot (\#\mathsf{Apply}_2^2 \cdot \mathsf{Pr}_1^2 \cdot \mathsf{Pr}_2^2), \#|\mathtt{succ1}|] \\ &= \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Apply}_2^2 \cdot \#\mathsf{Pr}_1^2 \cdot (\#\mathsf{Apply}_2^2 \cdot \mathsf{Pr}_1^2 \cdot \mathsf{Pr}_2^2), \#\mathsf{Apply}_1^1 \cdot \#\mathsf{Succ} \cdot \#\mathsf{Pr}_1^1]\,. \end{aligned}$$

In the translation above typing information was deliberately ignored for the sake of generality. We now show that our translation preserves the typings in the following sense.

Theorem 4.1.6.
Let M *be an* (E)PCF *term and* $\Gamma = x_1 : \delta_1, \ldots, x_n : \delta_n$. *Then*

$$\Gamma \vdash M : \alpha \quad \Rightarrow \quad |M|_{\vec{x}} : \delta_1 \to \cdots \to \delta_n \to \alpha.$$

Proof. As the type assignment systems of EPCF and of PCF coincide on PCF terms, we prove the above statement for EPCF and obtain PCF for free. Proceed by induction on a derivation of

$\Gamma \vdash M : \alpha$, using the notation $\vec{\delta} = \delta_1 \to \cdots \to \delta_n$. We split into cases depending on the last applied rule.

- Case (0). In this case $M = \mathbf{0}$ and $\alpha = \mathsf{int}$. By Definition 4.1.4, we have $|\mathbf{0}|_{\vec{x}} = \mathsf{Pr}_1^{n+1} \mathbin{@} [0]$. By Lemma 4.1.3(1), we know that $\mathsf{Pr}_1^{n+1} : \mathsf{int} \to \vec{\delta} \to \mathsf{int}$. By rule (nat), we get $\mathsf{0} : \mathsf{int}$. By Proposition 3.3.21(1), we conclude $\mathsf{Pr}_1^{n+1} \mathbin{@} [0] : \vec{\delta} \to \mathsf{int}$;

- Case (ax). Then $M = x_i$ for some $x_i \in \vec{x}$ and $|x_i|_{\vec{x}} = \mathsf{Pr}_i^n$. By Lemma 4.1.3(1).

- Case $(\to_{\mathrm{E}})$. Then $M = M_1 M_2$ and there is $\beta \in \mathbb{T}$ such that $\Gamma \vdash M_1 : \beta \to \alpha, \Gamma \vdash M_2 : \beta$. From the induction hypothesis, we obtain that $|M_1|_{\vec{x}} : \vec{\delta} \to \beta \to \alpha$ and $|M_2|_{\vec{x}} : \vec{\delta} \to \beta$. By Lemma 4.1.3(1), we get $\mathsf{Pr}_1^1 : (\beta \to \alpha) \to \beta \to \alpha$. By Lemma 4.1.3(2), we know that $\mathsf{Apply}_n^2 : ((\beta \to \alpha) \to \beta \to \alpha) \to (\vec{\delta} \to \beta \to \alpha) \to \vec{\delta} \to \beta$. By Definition 4.1.4, we have $|M_1 \cdot M_2|_{\vec{x}} = \mathsf{Apply}_n^2 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}]$ so we conclude by Proposition 3.3.21(1).

- Case (σ). Then $M = M'\langle N/y \rangle$ with $\Gamma, y : \beta \vdash M' : \alpha$ and $\vdash N : \beta$, for some $\beta \in \mathbb{T}$. By induction hypothesis, we get $|M'|_{\vec{x},y} : \beta \to \vec{\delta} \to \alpha$ and $|N| : \beta$. By Definition 4.1.4, we have $|M'\langle N/y \rangle|_{\vec{x}} = |M'|_{y,\vec{x}} \mathbin{@} [\#|N|]$. Conclude by Proposition 3.3.21(1).

- Case $(\to_{\mathrm{I}})$. Then $M = \lambda y.N$ and $\alpha = \alpha_1 \to \alpha_2$, with $\Gamma, y : \alpha_1 \vdash N : \alpha_2$. By Definition 4.1.4, we have $|\lambda y.N|_{\vec{x}} = |N|_{\vec{x},y}$ so the case follows from the induction hypothesis.

- Case $(+)$. Then $M = \mathbf{succ}\, N$ and $\alpha = \mathsf{int}$, with $\Gamma \vdash N : \mathsf{int}$. By induction hypothesis $|N|_{\vec{x}} : \vec{\delta} \to \mathsf{int}$ and, by Lemma 4.1.3(4), we have $\mathsf{Succ} : \mathsf{int} \to \mathsf{int}$. By Definition 4.1.4, $|\mathbf{succ}\, N|_{\vec{x}} = \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Succ}, \#|N|_{\vec{x}}]$. Conclude by Lemma 4.1.3(2) and Proposition 3.3.21(1).

- Case $(-)$. Analogous to the previous, using Lemma 4.1.3(3) instead of Lemma 4.1.3(2).

- Case (ifz). Then $M = \mathbf{ifz}(L, N_1, N_2)$ with $\Gamma \vdash L : \mathsf{int}$ and $\Gamma \vdash N_i : \alpha$, for each $i = 1, 2$. By induction hypothesis, we get $|L|_{\vec{x}} : \vec{\delta} \to \mathsf{int}$ and $|N_i|_{\vec{x}} : \vec{\delta} \to \alpha$, for every such i. By Lemma 4.1.3(5), we have $\mathsf{Ifz} : \mathsf{int} \to \alpha \to \alpha \to \alpha$. Since, by Definition 4.1.4, $|\mathbf{ifz}(L, N_1, N_2)|_{\vec{x}} = \mathsf{Apply}_n^3 \mathbin{@} [\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|N_1|_{\vec{x}}, \#|N_2|_{\vec{x}}]$ we conclude by applying Lemma 4.1.3(2) and Proposition 3.3.21(1).

- Case (Y). Then $M = \mathbf{fix}\, N$ with $\Gamma \vdash N : \alpha \to \alpha$. By induction hypothesis, we have $|N|_{\vec{x}} : \vec{\gamma} \to \alpha \to \alpha$. By rule (fix) we know that $\mathsf{Y} : (\alpha \to \alpha) \to \alpha$. The result follows from $|\mathbf{fix}\, N|_{\vec{x}} = \mathsf{Apply}_n^1 \mathbin{@} [\#\mathsf{Y}, \#|N|_{\vec{x}}]$ using Lemma 4.1.3(2) and Proposition 3.3.21(1). □

Ideally, one would like that an EPCF step $M \to_{\mathtt{wh}} N$ becomes a reduction $|M| \twoheadrightarrow_{\mathtt{c}} |N|$ in the corresponding EAMs. Unfortunately, the situation is more complicated—the translation

$|N|$ may contain auxiliary EAMs that are not generated by $|M|$ along reduction. For example, consider the reduction $\mathbf{pred}\,\underline{0} \to_{\mathtt{wh}} \underline{0}$. Translating $\mathbf{pred}\,\underline{0}$ and reducing it we have

$$\begin{array}{rcl}
|\mathbf{pred}\,\underline{0}| & = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Pred}, \#|\underline{0}|] \\
& = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Pred}, \#(\mathsf{Pr}_1^1 \mathbin{@} [0]\,)] \\
& \to_{\mathsf{c}} & \mathsf{Pred} \mathbin{@} [\#(\mathsf{Pr}_1^1 \mathbin{@} [0]\,)] \\
& \to_{\mathsf{c}} & \Big\langle R_0 = \#(\mathsf{Pr}_1^1 \mathbin{@} [0]\,), 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [\,] \Big\rangle \\
& \to_{\mathsf{c}} & \Big\langle R_0 = 0, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [\,] \Big\rangle \\
& \to_{\mathsf{c}} & \Big\langle R_0 = 0, \mathtt{Call}\ 0, [\,] \Big\rangle \\
& \to_{\mathsf{c}} & \#^{-1}(0)
\end{array}$$

$|\underline{0}| = (\mathsf{Pr}_1^1 \mathbin{@} [0]\,)$ which does not appear as a standalone machine in the above reduction sequence (although its address does appear).

The property that actually holds is that the two EAMs are interconvertible $|M| \leftrightarrow_{\mathsf{c}} |N|$, and $|N|$ is 'closer' to their common reduct. The next definition captures this intuition.

Definition 4.1.7 (Directional Interconvertability)**.**
For $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$*, define the relation:*

$$\mathsf{M} \succ_{\mathsf{c}} \mathsf{N} \iff \exists \mathsf{Z} \in \mathcal{M}_{\mathbb{A}}\,.\,[\,(\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z} \mathrel{{}_{\mathsf{c}}\!\twoheadleftarrow} \mathsf{N}) \wedge (|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}| \sqsubset |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}|)\,].$$

Note that there cannot be an empty $\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}$ *reduction path. Moreover, recall that* $\mathsf{M} \succ_{\mathsf{c}} \mathsf{N}$ *entails* $\mathsf{M} \leftrightarrow_{\mathsf{c}} \mathsf{N}$.

Lemma 4.1.8.

(1) $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}| \sqsubset |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}|$ *and* $\mathsf{Z} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'$ *imply* $|\mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'| \sqsubset |\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{Z}'|$.

(2) The relation $\succ_{\mathsf{c}}$ *is transitive.*

Proof.

(1) By confluence and determinism of $\to_{\mathsf{c}}$ (Lemma 3.3.12).

(2) follows from (1) and Corollary 3.3.13. □

Lemma 4.1.9.
$|\underline{n}| \twoheadrightarrow_{\mathtt{wh}}^{4n+2} \#^{-1}(n)$

Proof. By induction on n.

- $n = 0$: we have $|\mathbf{0}| = \mathsf{Pr}_1^1 \mathbin{@} 0 \twoheadrightarrow_{\mathtt{wh}}^{2} 0$.

- $n > 0$: we have

$$\begin{array}{rll} |\underline{n+1}| = & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Succ}, \#|\underline{n}|] & \\ \twoheadrightarrow_{\mathtt{wh}}^2 & \mathsf{Succ} \mathbin{@} [\#|\underline{n}|] & \text{by Lemma 4.1.2(1)} \\ \to_{\mathtt{wh}} & \left\langle R_0 = \#|\underline{n}|, 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [\,] \right\rangle & \text{by Lemma 4.1.2(4)} \\ \twoheadrightarrow_{\mathtt{wh}}^{4n+2} & \left\langle R_0 = n, 0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [\,] \right\rangle & \text{using the IH} \\ \twoheadrightarrow_{\mathtt{wh}}^2 & \#^{-1}(n+1) & \end{array}$$

  Conclude as $4n + 2 + 4 = 4(n + 1) + 2$.

□

Proposition 4.1.10.
Given an EPCF *program* M *of type* int*, we have* $M \to_{\mathtt{wh}} N \Rightarrow |M| \succ_{\mathsf{c}} |N|$.

Proof. By induction on a derivation of $M \to_{\mathtt{wh}} N$. As a matter of notation, we use $\vec{x} = x_1, \ldots, x_n$ and $\#\vec{\sigma} = \#|N_1|, \ldots, \#|N_n|$.

- Case $M = y^\sigma$ for $y \notin \mathrm{dom}(\sigma)$. Vacuous, since $M \in \mathcal{P}_{\mathsf{E}}$.

- Case $M = x_i^\sigma$ and $N = \sigma(x_i) = N_i$. So $|x^\sigma| = \mathsf{Pr}_i^n \mathbin{@} [\#\vec{\sigma}] \twoheadrightarrow_{\mathsf{c}}^{n+1} |N_i|$ by Lemma 4.1.2(1).

- Case $M = \mathbf{0}^\sigma$ and $N = \mathbf{0}$. On the one side, $|\mathbf{0}^\sigma| = |\mathbf{0}|_{\vec{x}} = \mathsf{Pr}_1^{n+1} \mathbin{@} [0, \#\vec{\sigma}]$ whence, by Lemma 4.1.2(1), we get $\mathsf{Pr}_1^{n+1} \mathbin{@} [0, \#\vec{\sigma}] \twoheadrightarrow_{\mathsf{c}}^{n+2} \#^{-1}(0) = \mathbf{0}$. On the other side, by Lemma 4.1.2(1), we obtain $|\mathbf{0}| = \mathsf{Pr}_1^1 \mathbin{@} [0] \twoheadrightarrow_{\mathsf{c}}^2 \mathbf{0}$. Since $n > 0$, we conclude $|\mathbf{0}^\sigma| \succ_{\mathsf{c}} |\mathbf{0}|$.

- Case $M = ((\lambda y.M_1)^\sigma)M_2$ and $N = M_1^\sigma\langle M_2/y\rangle$. Wlog $y \notin \mathrm{dom}(\sigma)$ and since $M_1 \in \mathcal{P}_{\mathsf{E}}$, we must have $\mathrm{FV}(M_1) \subseteq \{y\}$. Therefore

$$\begin{array}{rcll} |M| & = & |((\lambda y.M_1)^\sigma)M_2| & \\ & = & \mathsf{Apply}_0^2 \mathbin{@} [\#|M_1^\sigma|_y, \#|M_2|]\,, & \text{by Definition 4.1.4,} \\ & = & \mathsf{Pr}_1^1 \mathbin{@} [\#|M_1^\sigma|_y, \#|M_2|]\,, & \text{since } \mathsf{Apply}_0^2 = \mathsf{Pr}_1^1 \text{ by Definition 4.1.1,} \\ & \twoheadrightarrow_{\mathsf{c}}^2 & |M_1^\sigma|_y \mathbin{@} [\#|M_2|]\,, & \text{by Lemma 4.1.2(1),} \\ & = & |M_1^\sigma\langle M_2/y\rangle|, & \text{by Definition 4.1.4.} \end{array}$$

- Case $M = M_1M_2$ and $N = M_1'M_2$, where $M_1 \to_{\mathtt{wh}} M_1'$. By IH $|M_1| \succ_{\mathsf{c}} |M_1'|$, i.e. there

exists an EAM X such that $|M_1| \twoheadrightarrow_{\mathsf{c}}^{k} \mathsf{X}$ and $|M_1'| \twoheadrightarrow_{\mathsf{c}}^{k'} \mathsf{X}$, for some $k > k'$. Then we get

$$\begin{array}{rcll}
|M_1M_2| & = & \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1|, \#|M_2|], & \text{by Definitions 4.1.4 and 4.1.1,} \\
& \twoheadrightarrow_{\mathsf{c}}^{4} & |M_1|@[\#|M_2|], & \text{by Lemma 4.1.2(2)} \\
& \twoheadrightarrow_{\mathsf{c}}^{k} & \mathsf{X}@[\#|M_2|] & \\
& {}_{\mathsf{c}}^{k'}\!\!\twoheadleftarrow & |M_1'|@[\#|M_2|] & \\
& {}_{\mathsf{c}}^{4}\!\!\twoheadleftarrow & \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1'|, \#|M_2|], & \text{by Lemma 4.1.2(2),} \\
& = & |M_1'M_2|, & \text{by Definition 4.1.4.}
\end{array}$$

- Case $M = (M_1M_2)^\sigma$ and $N = M_1^\sigma M_2^\sigma$. Since $M \in \mathcal{P}_{\mathsf{E}}$, each M_i^σ is closed, whence $\mathrm{FV}(M_i) \subseteq \{x_1, \ldots, x_n\}$. On the one side, we get

$$\begin{array}{rcll}
& & |(M_1M_2)^\sigma| = \mathsf{Apply}_n^2@[\#\mathsf{Pr}_1^1, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}, \#\vec{\sigma}] & \\
& \twoheadrightarrow_{\mathsf{c}}^{10n+2} & \mathsf{Pr}_1^1 \,@\, [\#(|M_1|_{\vec{x}}@[\#\vec{\sigma}]), \#(|M_2|_{\vec{x}}@[\#\vec{\sigma}])]\,, & \text{by Lemma 4.1.2(2),} \\
& \twoheadrightarrow_{\mathsf{c}}^{2} & |M_1^\sigma|@[\#|M_2^\sigma|], & \text{by Lemma 4.1.2(1).}
\end{array}$$

  On the other side, we get $|M_1^\sigma M_2^\sigma| = \mathsf{Pr}_1^1@[\#\mathsf{Pr}_1^1, \#|M_1^\sigma|, \#|M_2^\sigma|] \twoheadrightarrow_{\mathsf{c}}^{4} |M_1^\sigma|@[\#|M_2^\sigma|]$ by applying Lemma 4.1.2(1) (twice). Conclude as this case only applies when σ is non-empty.

- Case $M = \mathbf{pred}\,\mathbf{0}$ and $N = \mathbf{0}$. By Lemma 4.1.2(1) and (3), we have

$$\begin{array}{rcl}
|\mathbf{pred}\,\mathbf{0}| = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Pred}, 0] & \twoheadrightarrow_{\mathsf{c}}^{2} \mathsf{Pred} \,@\, [0] \\
& \to_{\mathsf{c}} & \Big\langle R_0 = 0, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [\,] \Big\rangle \\
& \to_{\mathsf{c}} & \Big\langle R_0 = 0, \mathtt{Call}\ 0, [\,] \Big\rangle \to_{\mathsf{c}} \mathsf{0}
\end{array}$$

  Conclude using Lemma 4.1.9.

- Case $M = \mathbf{pred}\,(\mathbf{succ}\,\underline{n})$ and $N = \underline{n}$. Note that $\mathbf{succ}\,\underline{n} = \underline{n+1}$. We have

$$\begin{array}{rcll}
|\mathbf{pred}\,(\underline{n+1})| = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Pred}, \#|\underline{n+1}|] & & \\
& \twoheadrightarrow_{\mathsf{c}}^{2} & \mathsf{Pred} \,@\, [\#|\underline{n+1}|] & \text{by Lemma 4.1.2(3)} \\
& \to_{\mathsf{c}} & \Big\langle R_0 = \#|\underline{n+1}|, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [\,] \Big\rangle & \\
& \twoheadrightarrow_{\mathtt{wh}}^{4(n+1)+2} & \Big\langle R_0 = n+1, 0 \leftarrow \mathtt{Pred}(0); \mathtt{Call}\ 0, [\,] \Big\rangle & \text{by Lemma 4.1.9} \\
& \twoheadrightarrow_{\mathsf{c}} & \Big\langle R_0 = n, \mathtt{Call}\ 0, [\,] \Big\rangle \to_{\mathsf{c}} \mathsf{n} &
\end{array}$$

  Conclude using Lemma 4.1.9.

- Case $M = \mathbf{ifz}(\underline{0}, M_1, M_2)$ and $N = M_1$. Therefore $|M|$ is equal to

$$\begin{array}{rcll}
|\mathbf{ifz}(\underline{0}, M_1, M_2)| & = & \mathsf{Apply}_0^3 \,@\, [\#\mathsf{Ifz}, \#|\underline{0}|, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.1.4,} \\
& = & \mathsf{Pr}_1^1 \,@\, [\#\mathsf{Ifz}, \#|\underline{0}|, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.1.1,} \\
& \twoheadrightarrow_{\mathsf{c}}^{2} & \mathsf{Ifz} \,@\, [\#|\underline{0}|, \#|M_1|, \#|M_2|]\,, & \text{by Lemma 4.1.2(1),} \\
& \twoheadrightarrow_{\mathsf{c}} & |M_1|. &
\end{array}$$

- Case $M = \mathbf{ifz}(\underline{k+1}, M_1, M_2)$, for some $k \geq 0$, and $N = M_2$. Therefore $|M|$ is equal to

$$\begin{array}{lll}
& |\mathbf{ifz}(\underline{k+1}, M_1, M_2)| & \\
= & \mathsf{Apply}_0^3 \mathbin{@} [\#\mathsf{Ifz}, \#|\underline{k+1}|, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.1.4,} \\
= & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Ifz}, \#|\underline{k+1}|, \#|M_1|, \#|M_2|]\,, & \text{by Definition 4.1.1,} \\
\twoheadrightarrow_{\mathsf{c}}^2 & \mathsf{Ifz} \mathbin{@} [\#|\underline{k+1}|, \#|M_1|, \#|M_2|]\,, & \text{by Lemma 4.1.2(1),} \\
\twoheadrightarrow_{\mathsf{c}} & |M_2|. &
\end{array}$$

- Case $M = \mathbf{ifz}(L, M_1, M_2)$ and $N = \mathbf{ifz}(L', M_1, M_2)$, where $L \to_{\mathtt{wh}} L'$. By IH $|L| \succ_{\mathsf{c}} |L'|$, i.e. there is an EAM X such that $|L| \twoheadrightarrow_{\mathsf{c}}^k \mathsf{X}$ and $|L'| \twoheadrightarrow_{\mathsf{c}}^{k'} \mathsf{X}$ for some $k > k'$. Then we get

$$\begin{array}{lll}
& |\mathbf{ifz}(L, M_1, M_2)| & \\
= & \mathsf{Pr}_1^1@[\#\mathsf{Ifz}, \#|L|, \#|M_1|, \#|M_2|], & \text{by Def. 4.1.4 and 4.1.1,} \\
\twoheadrightarrow_{\mathsf{c}}^2 & \mathsf{Ifz}@[\#|L|, \#|M_1|, \#|M_2|], & \text{by Lemma 4.1.2(2)} \\
\twoheadrightarrow_{\mathsf{c}}^3 & \left\langle \begin{array}{r} R_0 = \#|L|, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\ 1,\ 2); \mathtt{Call}\ 0, [\,] \end{array} \right\rangle, & \text{by Lemma 4.1.2(5)} \\
\twoheadrightarrow_{\mathsf{c}}^k & \left\langle \begin{array}{r} R_0 = \#\mathsf{X}, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\ 1,\ 2); \mathtt{Call}\ 0, [\,] \end{array} \right\rangle, & \\
{\scriptstyle k'}\!\!\twoheadleftarrow_{\mathsf{c}} & \left\langle \begin{array}{r} R_0 = \#|L'|, R_1 = \#|M_1|, R_2 = \#|M_2|, \\ 0 \leftarrow \mathtt{Test}(0,\ 1,\ 2); \mathtt{Call}\ 0, [\,] \end{array} \right\rangle & \\
{\scriptstyle 3}\!\!\twoheadleftarrow_{\mathsf{c}} & \mathsf{Ifz}@[\#|L|, \#|M_1|, \#|M_2|], & \text{by Lemma 4.1.2(5),} \\
{\scriptstyle 2}\!\!\twoheadleftarrow_{\mathsf{c}} & \mathsf{Pr}_1^1@[\#\mathsf{Ifz}, \#|M_1'|, \#|M_2|], & \text{by Lemma 4.1.2(2),} \\
= & |\mathbf{ifz}(L', M_1, M_2)|, & \text{by Definition 4.1.4.}
\end{array}$$

- Case $M = \mathbf{pred}\, M'$ and $N = \mathbf{pred}\, N'$, where $M' \to_{\mathtt{wh}} N'$. Analogous to the above.

- Case $M = \mathbf{succ}\, M'$ and $N = \mathbf{succ}\, N'$, where $M' \to_{\mathtt{wh}} N'$. Analogous to the above.

- Case $M = (\mathbf{ifz}(L, M_1, M_2))^\sigma$ and $N = \mathbf{ifz}(L^\sigma, M_1^\sigma, M_2^\sigma)$. Note that $M \in \mathcal{P}_{\mathsf{E}}$ entails $L^\sigma, M_1^\sigma, M_2^\sigma$ closed, thus $\mathrm{FV}(L) \subseteq \{\vec{x}\}$ and $\mathrm{FV}(M_i) \subseteq \{\vec{x}\}$. By Lemma 4.1.2(2), we get

$$\begin{array}{rcl}
|(\mathbf{ifz}(L, M_1, M_2))^\sigma| & = & \mathsf{Apply}_n^3@[\#\mathsf{Ifz}, \#|L|_{\vec{x}}, \#|M_1|_{\vec{x}}, \#|M_2|_{\vec{x}}, \#\vec{\sigma}] \\
& \twoheadrightarrow_{\mathsf{c}}^{13n+2} & \mathsf{Ifz} \mathbin{@} [\#|L^\sigma|, \#|M_1^\sigma|, \#|M_2^\sigma|] \\
& {\scriptstyle 2}\!\!\twoheadleftarrow_{\mathsf{c}} & \mathsf{Pr}_1^1@[\#\mathsf{Ifz}, \#|L^\sigma|, \#|M_1^\sigma|, \#|M_2^\sigma|] \\
& = & |\mathbf{ifz}(L^\sigma, M_1^\sigma, M_1^\sigma)|
\end{array}$$

  Conclude as this case only applies when σ is non-empty.

- Case $M = (\mathbf{pred}\, M')^\sigma$ and $N = \mathbf{pred}\, ((M')^\sigma)$. Analogous to the above.

- Case $M = (\mathbf{succ}\, M')^\sigma$ and $N = \mathbf{succ}\, ((M')^\sigma)$. Analogous to the above.

- Case $M = \mathbf{fix}\, M'$ and $N = M'(\mathbf{fix}\, M')$. In this case, we get

$$\begin{array}{rcll} |\mathbf{fix}\, M'| & = & \mathsf{Y} \mathbin{@} [\#|M|]\,, & \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}}^{5} & |M'| \mathbin{@} [\#\mathsf{Y} \mathbin{@} [\#|M'|]]\,, & \text{by Lemma 4.1.2(6),} \\ & = & |M'| \mathbin{@} [\#|\mathbf{fix}\, M'|]\,, & \text{by Definition 4.1.4,} \\ & \overset{2}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Pr}_1^1 \mathbin{@} [\#|M'|, \#|\mathbf{fix}\, M'|]\,, & \text{by Lemma 4.1.2(1),} \\ & \overset{2}{\underset{\mathsf{c}}{\twoheadleftarrow}} & \mathsf{Pr}_1^1 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M'|, \#|\mathbf{fix}\, M'|]\,, & \text{by Lemma 4.1.2(1),} \\ & = & |M'(\mathbf{fix}\, M')|, & \text{by Definition 4.1.4.} \end{array}$$

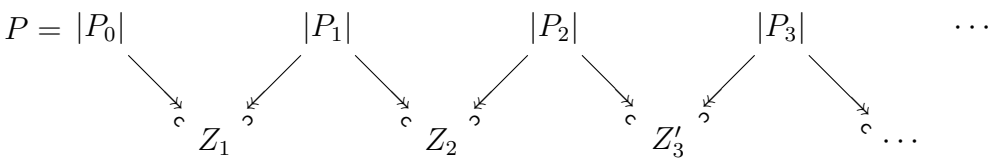
- Case $M = (\mathbf{fix}\, M')^{\sigma}$ and $N = |\mathbf{fix}\, ((M')^{\sigma})|$. By Lemma 4.1.2(2), we obtain $|(\mathbf{fix}\, M')^{\sigma}| = \mathsf{Apply}_n^1 \mathbin{@} [\mathsf{Y}, \#|M'|_{\vec{x}}, \#\vec{\sigma}] \twoheadrightarrow_{\mathsf{c}}^{7n+2} \mathsf{Y} \mathbin{@} [\#|(M')^{\sigma}|] = |\mathbf{fix}\, (M'^{\sigma})|$. □

Since every PCF program P of type int is also an EPCF program of the same type, and in this case the operational semantics of PCF and EPCF coincide (Theorem 2.2.20), we can use the above proposition to prove that the EAM $|P|$ faithfully simulates the behavior of P.

Theorem 4.1.11.
For a PCF *program* P *having type* int*, the following are equivalent:*

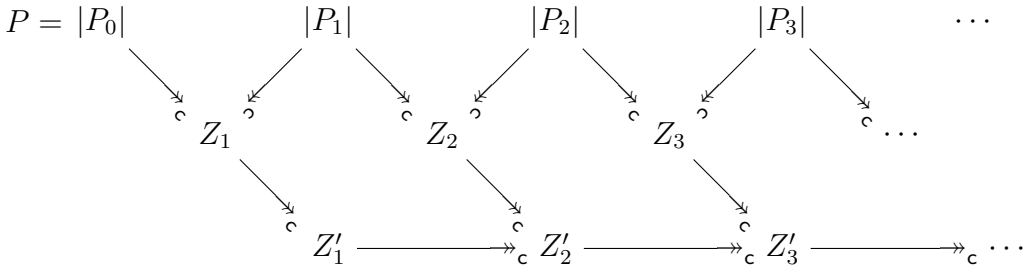
1. $P \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$;
2. $|P| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$.

Proof. $(1 \Rightarrow 2)$ Let $P \twoheadrightarrow_{\mathsf{PCF}} \underline{n}$. Equivalently, $P \twoheadrightarrow_{\mathtt{wh}} \underline{n}$ (by Theorem 2.2.20). By Proposition 4.1.10, we get $|P| \leftrightarrow_{\mathsf{c}} |\underline{n}|$ and by Lemma 4.1.9 we get $|\underline{n}| \leftrightarrow_{\mathsf{c}} \mathsf{n}$. Since n is in final state, this entails $|P| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$ by confluence (Lemma 3.3.12(2)). $(2 \Rightarrow 1)$ We prove the contrapositive. From $(1 \Rightarrow 2)$ we cannot have $P \twoheadrightarrow_{\mathsf{PCF}} \underline{m}$ and $|P| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$ for $m \neq n$. Thus we assume that $\vdash P : \mathsf{int}$, but P does not reduce to a numeral. As PCF enjoys subject reduction [Ong95] and numerals are the only PCF programs of type int in a final state by Remark 1.2.12, P must have an infinite $\rightarrow_{\mathsf{PCF}}$ reduction path. By Theorem 2.2.20, this generates an infinite w.h. reduction path $P = P_0 \rightarrow_{\mathtt{wh}} P_1 \twoheadrightarrow_{\mathtt{wh}} P_k \twoheadrightarrow_{\mathtt{wh}} \cdots$. By Proposition 4.1.10, this translates to an infinite $\succ_{\mathsf{c}}$-chain $|P_k| \succ_{\mathsf{c}} |P_{k+1}|$, in other words we have $\mathsf{Z_k}$ such that



where for all k, $|P_{k-1} \twoheadrightarrow_{\mathsf{c}} Z_k| \sqsubset |P_k \twoheadrightarrow_{\mathsf{c}} Z_k|$.

By confluence, there are Z'_k such that



We will now prove by contradiction that the above diagram presents a non-terminating reduction sequence, i.e. there cannot exist an $x \geq 0$ such that Z'_x is in a final state. Let us assume the existence of such a Z'_x. Then, for all $y > x$, we have $Z'_y = Z'_x$. Using Lemma 4.1.8(1) we obtain $||P_x| \twoheadrightarrow_{\mathsf{c}} Z'_x| \sqsubset ||P_{x+1}| \twoheadrightarrow_{\mathsf{c}} Z'_x| \sqsubset ||P_{x+2}| \twoheadrightarrow_{\mathsf{c}} Z'_x| \sqsubset \ldots$. By Corollary 3.3.13, Z'_x either loops, in which case it is not in final state, or we have an infinitely strictly decreasing sequence of natural numbers, which cannot exist. Thus, relying on the reduction of Extended Addressing Machines being deterministic, we conclude that $|P|$ does not terminate. □

4.2 The Model

4.2.1 Machine Equivalence

We now move on to the construction of the model of PCF based on Extended Addressing Machines. For a reminder of the definition of such a model, see Definition 1.3.5. The idea is to focus on the subset $\mathcal{D} \subseteq \mathbb{A}$ containing addresses of typable EAMs, which is then stratified $(\mathcal{D}_\alpha)_{\alpha \in \mathbb{T}}$ following the inductive syntax of simple types. In other words, the set $\mathcal{D}_\alpha$ contains the addresses of those EAM having type α. The well-typedness condition allows to get rid of those EAMs containing addresses of infinite chains of 'pointers'—a phenomenon described in Remark 3.3.9(3)—that might exist in $\mathcal{M}_{\mathbb{A}}$, but cannot be typed. Subsequently, we are going to impose that two addresses $a, b \in \mathcal{D}_\alpha$ are equal in the model whenever the corresponding EAMs exhibit the same applicative behavior: at ground type ($\alpha = \mathsf{int}$) this simply means that either $\#^{-1}(a)$ and $\#^{-1}(b)$ compute the same numeral, or they are both non-terminating. This equality is then lifted at higher order types following the well-established tradition of logical relations [Plo73].

Definition 4.2.1 (Machine Logical Equivalence)**.**
For all EAMs M, N *of type* α*, define* $\mathsf{M} \equiv_\alpha \mathsf{N}$ *inductively with the following rules:*

$$\mathsf{M} \equiv_{\mathsf{int}} \mathsf{N} \quad \iff \quad \forall n \in \mathbb{N} \,.\, \big[\, \mathsf{M} \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \iff \mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \,\big]$$

$$\mathsf{M} \equiv_{\alpha\to\beta} \mathsf{N} \quad \iff \quad \forall \mathsf{L}, \mathsf{L}' \in \mathcal{M}_{\mathbb{A}} \,.\, \big[\, \mathsf{L} \equiv_\alpha \mathsf{L}' \Rightarrow \mathsf{M} @ [\#\mathsf{L}] \equiv_\beta \mathsf{N} @ [\#\mathsf{L}'] \,\big]$$

We extend the notion to tapes and registers:

- Given tapes $T = [a_1, \ldots, a_n], T' = [a'_1, \ldots, a'_n]$, we write $T \equiv T'$ to mean that there exist types $\alpha_1, \ldots, \alpha_n$ such that $\#a_1 \equiv_{\alpha_1} \#a'_1, \ldots, \#a_n \equiv_{\alpha_n} \#a'_n$.
- Given lists of registers $\vec{R}, \vec{R'}$ of length r, we write $\vec{R} \equiv \vec{R'}$ to mean that for $0 \leq i \leq r$, either $!R_i = !R'_i = \varnothing$, or there exists a type α_i such that $\#!R_i \equiv_{\alpha_i} \#!R'_i$.

We will later find that this relation is indeed an equivalence, but the proof for this is a bit tricky. In particular, proving reflexivity of the relation with respect to the machine Y is troublesome due to its self-referential nature. We will sidestep the issue by introducing the notion of approximations to Y, and making use of the approximations to show that Y is indeed reflexive.[1]

Definition 4.2.2.
Recall the machine $|\Omega^{\mathsf{PCF}}| = \mathsf{Y} \mathbin{@} [\#\mathsf{Pr}^1_1]$ *from Example 4.1.5. We define the following machines by induction on* $n \in \mathbb{N}$*:*

$$\begin{aligned} \mathsf{fix}_0 &= |\Omega^{\mathsf{PCF}}| \\ \mathsf{fix}_{n+1} &= \langle \mathsf{Y}.\vec{R}, \mathsf{Y}.P, [\#\mathsf{fix}_n] \rangle \end{aligned}$$

Note that for all $n \in \mathbb{N}$ *and for all types* α, $\mathsf{fix}_n : (\alpha \to \alpha) \to \alpha$.

Definition 4.2.3 (Approximations to Y)**.**
Let $\mathsf{M} : \alpha$ *and* $\mathsf{N} : \alpha$*. We define the approximation of* M *to* N *with respect to* Y*,* $\mathsf{M} \leq_\alpha \mathsf{N}$*, with the following rules:*

$$\frac{n \in \mathbb{N}}{\mathsf{n} \leq_{\mathsf{int}} \mathsf{n}} \qquad \frac{n \leq m}{\mathsf{fix}_n \leq_{(\alpha\to\alpha)\to\alpha} \mathsf{fix}_m} \qquad \frac{\mathsf{M} \leq_{\beta\to\alpha} \mathsf{M}' \quad \#^{-1}(a) \leq_\beta \#^{-1}(a')}{\mathsf{M} \mathbin{@} [a] \ \leq_\alpha \mathsf{M}' \mathbin{@} [a']}$$

$$\frac{}{\mathsf{Y} \leq_{(\alpha\to\alpha)\to\alpha} \mathsf{Y}} \qquad \frac{n \in \mathbb{N}}{\mathsf{fix}_n \leq_{(\alpha\to\alpha)\to\alpha} \mathsf{Y}} \qquad \frac{\forall i \leq r, \exists \beta_i \in \mathbb{T}. \#^{-1}(!R_i) \leq_{\beta_i} \#^{-1}(!R'_i)}{\langle R_0, \ldots, R_r, P, [\,] \rangle \leq_\alpha \langle R'_0, \ldots, R'_r, P, [\,] \rangle}$$

Definition 4.2.4.
Define the set of machines with only approximations $\mathcal{M}^\Omega_{\mathbb{A}}$ *as the set of machines typable when substituting the* (fix) *rule with the following rules for all types* α*:*

$$\frac{}{|\Omega^{\mathsf{PCF}}| : \alpha} \ (\Omega)$$

Of particular note is that we can type $\mathsf{fix}_0 = |\Omega^{\mathsf{PCF}}| : (\alpha \to \alpha) \to \alpha$ using the above rules.

Remark 4.2.5.

(1) If $n \in \mathbb{N}$ *and* $\mathsf{M} \leq \mathsf{n}$ *or* $n \leq \mathsf{M}$*, then* $\mathsf{M} = \mathsf{n}$.

(2) If $\mathsf{M} \leq \mathsf{M}'$*, then* $\mathsf{M}.P = \mathsf{M}'.P$.

[1] This approach to the problem was inspired by [MP21].

(3) If $\mathsf{M} \leq \mathsf{M}'$ *yet the length of* $\mathsf{M}.T$ *is not equal to the length of* $\mathsf{M}'.T$*, then* $\mathsf{M} = |\Omega^{\mathsf{PCF}}| \mathbin{@} T''$ *for some tape* T''*.*

(4) For all machines $\mathsf{M} : \alpha$*, there exists a* $\mathsf{M}' \in \mathcal{M}_{\mathbb{A}}^{\Omega}$ *such that* $\mathsf{M}' : \alpha$ *and* $\mathsf{M}' \leq_\alpha \mathsf{M}$*.*

Lemma 4.2.6 (Monotonicity)**.**
If $\mathsf{M} \leq \mathsf{N}$ *and* $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}'$*, then there exist* L, N' *such that* $\mathsf{M}' \twoheadrightarrow_{\mathsf{c}} \mathsf{L}$*,* $\mathsf{N} \twoheadrightarrow_{\mathsf{c}} \mathsf{N}'$ *and* $\mathsf{L} \leq \mathsf{N}'$*.*

Proof. By induction on $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}'$.

- Case $\mathsf{M}.P = j \leftarrow \mathtt{Pred}(i); P'$: There are two subcases:
  - Case $!\mathsf{M}.R_i \in \mathbb{N}$: Then $!\mathsf{M}.R_i = !\mathsf{N}.R_i$, so $\mathsf{N}' = \langle \mathsf{N}.\vec{R}[R_j := !\mathsf{N}.R_i \ominus 1], P', \mathsf{N}.T \rangle$.
  - Case $!\mathsf{M}.R_i \notin \mathbb{N}$: Then $\#^{-1}(!\mathsf{M}.R_i) \to_{\mathsf{c}} \#^{-1}(!\mathsf{M}'.R_i)$ and $\#^{-1}(!\mathsf{M}.R_i) \leq \#^{-1}(!\mathsf{N}.R_i)$. We use the IH to obtain L', K such that $\#^{-1}(!\mathsf{M}'.R_i) \twoheadrightarrow_{\mathsf{c}} \mathsf{L}'$, $\#^{-1}(!\mathsf{N}.R_i) \twoheadrightarrow_{\mathsf{c}} \mathsf{K}$ and $\mathsf{L}' \leq \mathsf{K}$. We then have $\mathsf{N}' = \langle \mathsf{N}.\vec{R}[R_i := \#\mathsf{K}], j \leftarrow \mathtt{Pred}(i); P', \mathsf{N}.T \rangle$ for which the desired properties hold.
- Cases $\mathsf{M}.P = j \leftarrow \mathtt{Succ}(i); P'$ and $\mathsf{M}.P = l \leftarrow \mathtt{Test}(i,\, j,\, k); P'$ are analogous.
- Case $\mathsf{M}.P = \mathtt{Load}\ i; P'$: We have two subcases:
  - The length of $\mathsf{M}.T$ is equal to the length of $\mathsf{M}'.T$: Then $\mathsf{M}.T = [a_1, \ldots, a_n]$ and $\mathsf{N}.T = [b_1, \ldots, b_n]$ such that for all $1 \leq i \leq n$, $\#^{-1}(a_i) \leq \#^{-1}(b_i)$. We then have $\mathsf{N}' = \langle \mathsf{N}.\vec{R}[R_i := b_1], P', [b_1, \ldots, b_n] \rangle$ for which the desired properties hold.
  - The length of $\mathsf{M}.T$ is not equal to the length of $\mathsf{M}'.T$: By Proposition 4.2.5(3), for some tape T'' we have $\mathsf{M} = |\Omega^{\mathsf{PCF}}| \mathbin{@} T''$. We have $|\Omega^{\mathsf{PCF}}| \mathbin{@} T'' \twoheadrightarrow_{\mathsf{c}}^{7} |\Omega^{\mathsf{PCF}}| \mathbin{@} T''$ and $\mathsf{N} \twoheadrightarrow_{\mathsf{c}}^{0} \mathsf{N}$, so we conclude.
- Case $\mathsf{M}.P = k \leftarrow \mathtt{App}(i,\, j); P'$: Then $\mathsf{M}.R_i \leq \mathsf{N}.R_i$ and $\mathsf{M}.R_j \leq \mathsf{N}.R_j$. We then have $\mathsf{N}' = \langle \mathsf{N}.\vec{R}[R_k := !\mathsf{N}.R_i \cdot !\mathsf{N}.R_j], P', \mathsf{N}.T \rangle$ for which the desired properties hold.
- Case $\mathsf{M}.P = \mathtt{Call}\ i$: Let $\mathsf{M}.T = [a_1, \ldots, a_n]$ and $\mathsf{N}.T = [b_1, \ldots, b_n]$. For all $1 \leq j \leq n$, we have $\#^{-1}(a_j) \leq \#^{-1}(b_j)$, and we also have $\#^{-1}(!\mathsf{M}.R_i) \leq \#^{-1}(!\mathsf{N}.R_i)$. Then there is $\mathsf{N}' = \#^{-1}(!\mathsf{N}.R_i) \mathbin{@} [b_1, \ldots, b_n]$ for which the desired properties hold. □

Lemma 4.2.7 (Continuity)**.**
If $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}'$ *and* $\mathsf{N}' \leq \mathsf{M}'$*, then there exists* N *such that* $\mathsf{N} \leq \mathsf{M}$ *and* $\mathsf{N} \to_{\mathsf{c}} \mathsf{N}'$*. Moreover, if* $\mathsf{N}' \in \mathcal{M}_{\mathbb{A}}^{\Omega}$*, then* $\mathsf{N} \in \mathcal{M}_{\mathbb{A}}^{\Omega}$*.*

Proof. By induction on $\mathsf{M} \to_{\mathsf{c}} \mathsf{M}'$.

- Case $\mathsf{M}.P = j \leftarrow \mathtt{Pred}(i); P'$: There are two subcases:

  - Case $!\mathsf{M}.R_i \in \mathbb{N}$: We have $M' = \langle \mathsf{M}.\vec{R}[R_j := !\mathsf{M}.R_i \ominus 1], P', \mathsf{M}.T\rangle$. Let $a \in \mathbb{A}$ such that $\#^{-1}(a) \leq \#^{-1}(!\mathsf{M}.R_j)$ and $\#^{-1}(a) \in \mathcal{M}^{\Omega}_{\mathbb{A}}$. We then have the machine $\mathsf{N} = \langle \mathsf{N}'.\vec{R}[R_j := a], j \leftarrow \mathtt{Pred}(i); P', \mathsf{N}'.T\rangle$ for which the desired properties hold.
  - Case $!\mathsf{M}.R_i \notin \mathbb{N}$: Then $\#^{-1}(!\mathsf{M}.R_i) \to_{\mathsf{c}} \#^{-1}(!\mathsf{M}'.R_i)$, $\#^{-1}(!\mathsf{N}'.R_i) \leq \#^{-1}(!\mathsf{M}'.R_i)$. We use the IH to obtain L such that $\mathsf{L} \leq \#^{-1}(!\mathsf{M}.R_i), \mathsf{L} \to_{\mathsf{c}} \#^{-1}(!\mathsf{N}'.R_i)$, and if $\#^{-1}(!\mathsf{N}'.R_i) \in \mathcal{M}^{\Omega}_{\mathbb{A}}$ then $\mathsf{L} \in \mathcal{M}^{\Omega}_{\mathbb{A}}$. We then find the desired properties hold for the machine $\mathsf{N} = \langle \mathsf{N}'.\vec{R}[R_i := \#\mathsf{L}], j \leftarrow \mathtt{Pred}(i); P', \mathsf{N}'.T\rangle$.

- Cases $\mathsf{M}.P = j \leftarrow \mathtt{Succ}(i); P'$ and $\mathsf{M}.P = l \leftarrow \mathtt{Test}(i,\ j,\ k); P'$ are analogous.

- Case $\mathsf{M}.P = \mathtt{Load}\ i; P'$: Then $\mathsf{M}.T = a :: T'$. Let $b \in \mathbb{A}$ such that $\#^{-1}(b) \leq \#^{-1}(!\mathsf{M}.R_i)$ and $\#^{-1}(b) \in \mathcal{M}^{\Omega}_{\mathbb{A}}$. We then have $\mathsf{N} = \langle \mathsf{N}'.\vec{R}[R_i := b], \mathtt{Load}\ i; P', !\mathsf{N}'.R_i :: \mathsf{N}'.T\rangle$ for which the desired properties hold.

- Case $\mathsf{M}.P = k \leftarrow \mathtt{App}(i,\ j); P'$: Consider an $a \in \mathbb{A}$ such that $\#^{-1}(a) \leq \#^{-1}(!\mathsf{M}.R_j)$ and $\#^{-1}(a) \in \mathcal{M}^{\Omega}_{\mathbb{A}}$. We then have $\mathsf{N} = \langle \mathsf{N}'.\vec{R}[R_k := a], k \leftarrow \mathtt{App}(i,\ j); P', \mathsf{N}'.T\rangle$ for which the desired properties hold.

- Case $\mathsf{M}.P = \mathtt{Call}\ i$: Let $r+1$ be the number of registers of M. Let $a_0, \ldots, a_r \in \mathbb{A}_{\varnothing}$ such that for all $0 \leq j \leq r$,

$$a_j = \begin{cases} \varnothing, & \text{if } !\mathsf{M}.R_j = \varnothing, \\ c \text{ s.t. } c \in \mathbb{A}, \#^{-1}(c) \in \mathcal{M}^{\Omega}_{\mathbb{A}}, \#^{-1}(c) \leq \#^{-1}(!\mathsf{M}.R_j), & \text{otherwise.} \end{cases}$$

  Let n and m be the lengths of $\mathsf{M}.T$ and $\mathsf{N}'.T$ respectively. Let T' be the list of the first $m-n$ addresses in $\mathsf{N}'.T$ and T'' be the list of the last n addresses in $\mathsf{N}'.T$. Let b be the address of the machine $\langle \mathsf{N}'.\vec{R}, \mathsf{N}'.P, T'\rangle$. We then find the desired properties hold for the machine $\mathsf{N} = \langle (R_0 = a_0, \ldots, R_r = a_r)[R_i := b], \mathtt{Call}\ i, T''\rangle$. □

Lemma 4.2.8.
Given $\mathsf{N} : (\beta_1 \to \cdots \to \beta_k \to \mathsf{int}) \to (\beta_1 \to \cdots \to \beta_k \to \mathsf{int}), \mathsf{L_1} : \beta_1, \ldots, \mathsf{L_k} : \beta_k$, $n \in \mathbb{N}$,

$$\mathsf{Y} \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \quad \Leftrightarrow \quad \exists m \in \mathbb{N} \text{ such that } \mathsf{fix}_m \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

Proof. ($\Leftarrow$) As $\mathsf{fix}_m \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \leq \mathsf{Y} \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}]$, we apply Lemma 4.2.6 and conclude by Remark 4.2.5(1).

($\Rightarrow$) We iterate Lemma 4.2.7 to get $\mathsf{fix}_m \mathbin{@} [\#\mathsf{N}', \#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_k] \in \mathcal{M}^{\Omega}_{\mathbb{A}}$ for some $m \in \mathbb{N}$ such that

$$\begin{array}{ccc} \mathsf{Y} \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} & \Rightarrow & \mathsf{fix}_m \mathbin{@} [\#\mathsf{N}', \#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_k] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \\ \mathsf{fix}_m \mathbin{@} [\#\mathsf{N}', \#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_k] & \leq & \mathsf{Y} \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \end{array}$$

From Remark 4.2.5(3), we have

$$\mathsf{fix}_m \mathbin{@} [\#\mathsf{N}', \#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_k] \quad \leq \quad \mathsf{fix}_m \mathbin{@} [\#\mathsf{N}, \#\mathsf{L}_1, \ldots, \#\mathsf{L}_k]$$

By yet again applying Lemma 4.2.6 and Remark 4.2.5(1), we get

$$\mathsf{fix}_m \mathbin{@} [\#\mathsf{N}, \#\mathsf{L}_1, \ldots, \#\mathsf{L}_k] \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

□

Lemma 4.2.9.
For all $k \in \mathbb{N}$, $\mathsf{fix}_k \equiv_{(\alpha\to\alpha)\to\alpha} \mathsf{fix}_k$.

Proof. Let $\alpha = \gamma_1 \to \cdots \to \gamma_m \to \mathsf{int}$. We proceed by induction on k.

- Case $k = 0$: Let $\mathsf{N}_0 : \alpha \to \alpha, \mathsf{N}_1 : \gamma_1, \ldots, \mathsf{N}_m : \gamma_m$. We have

$$|\Omega^{\mathsf{PCF}}| \mathbin{@} [\#\mathsf{N}_0, \#\mathsf{N}_1, \ldots, \#\mathsf{N}_m] \twoheadrightarrow^7_{\mathsf{c}} |\Omega^{\mathsf{PCF}}| \mathbin{@} [\#\mathsf{N}_0, \#\mathsf{N}_1, \ldots, \#\mathsf{N}_m]$$

  As for all $\mathsf{N}_0 : \alpha \to \alpha, \mathsf{N}_1 : \gamma_1, \ldots, \mathsf{N}_m : \gamma_m$, we have $|\Omega^{\mathsf{PCF}}| \mathbin{@} [\#\mathsf{N}_1, \ldots, \#\mathsf{N}_m] \not\twoheadrightarrow_{\mathsf{c}} \mathsf{n}$ for all $n \in \mathbb{N}$, we obtain $|\Omega^{\mathsf{PCF}}| \equiv_{\gamma_1\to\cdots\to\gamma_m\to\mathsf{int}} |\Omega^{\mathsf{PCF}}|$.

- Case $k > 0$: Let $\mathsf{N}_0 \equiv_{\alpha\to\alpha} \mathsf{N}'_0, \mathsf{N}_1 \equiv_{\gamma_1} \mathsf{N}'_1, \ldots, \mathsf{N}_m \equiv_{\gamma_m} \mathsf{N}'_m$. We have:

$$\mathsf{fix}_{k+1} \mathbin{@} [\#\mathsf{N}_0, \ldots, \#\mathsf{N}_m] \twoheadrightarrow_{\mathsf{c}} \mathsf{N}_0 \mathbin{@} [\#(\mathsf{fix}_k \mathbin{@} [\#\mathsf{N}_0]), \#\mathsf{N}_1, \ldots, \#\mathsf{N}_m]$$
$$\mathsf{fix}_{k+1} \mathbin{@} [\#\mathsf{N}'_0, \ldots, \#\mathsf{N}'_m] \twoheadrightarrow_{\mathsf{c}} \mathsf{N}'_0 \mathbin{@} [\#(\mathsf{fix}_k \mathbin{@} [\#\mathsf{N}'_0]), \#\mathsf{N}'_1, \ldots, \#\mathsf{N}'_m]$$

  By IH, we have $\mathsf{fix}_k \mathbin{@} [\#\mathsf{N}_0] \equiv_\alpha \mathsf{fix}_k \mathbin{@} [\#\mathsf{N}'_0]$. As $\mathsf{N}_0 \equiv_{\alpha\to\alpha} \mathsf{N}'_0$, we have

$$\mathsf{N}_0 \mathbin{@} [\#(\mathsf{fix}_k \mathbin{@} [\#\mathsf{N}_0]), \#\mathsf{N}_1, \ldots, \#\mathsf{N}_m] \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$
$$\Leftrightarrow$$
$$\mathsf{N}'_0 \mathbin{@} [\#(\mathsf{fix}_k \mathbin{@} [\#\mathsf{N}'_0]), \#\mathsf{N}'_1, \ldots, \#\mathsf{N}'_m] \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

  We conclude using determinism. □

Lemma 4.2.10.
If $\mathsf{M} : \beta_1 \to \cdots \to \beta_n \to \mathsf{int}$ *and* $\mathsf{N} : \beta_1 \to \cdots \to \beta_n \to \mathsf{int}$, *then*

$$\mathsf{M} \equiv_{\beta_1\to\cdots\to\beta_n\to\mathsf{int}} \mathsf{N}$$

if and only if for all $\mathsf{L}_1 \equiv_{\beta_1} \mathsf{L}'_1, \ldots, \mathsf{L}_n \equiv_{\beta_n} \mathsf{L}'_n$,

$$\mathsf{M} \mathbin{@} [\#\mathsf{L}_1, \ldots, \#\mathsf{L}_n] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(\underline{m}) \quad \Leftrightarrow \quad \mathsf{N} \mathbin{@} [\#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_n] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(\underline{m})$$

Proof. Trivial extension of the definition. □

Lemma 4.2.11.

(1) The relation $\equiv_\alpha$ is symmetric and transitive.

(2) The relation $\equiv_\alpha$ is reflexive.

(3) Let $\mathsf{M}, \mathsf{N} \in \mathcal{M}_{\mathbb{A}}$ and $\alpha \in \mathbb{T}$ be such that $\mathsf{M} : \alpha$ and $\mathsf{N} : \alpha$. If $\mathsf{M} \to_{\mathsf{c}} \mathsf{N}$ then $\mathsf{M} \equiv_\alpha \mathsf{N}$.

(4) Assume $\mathsf{M} : \alpha \to \beta, \mathsf{N}_1 : \alpha$ and $\mathsf{N}_2 : \alpha$. If $\mathsf{N}_1 \to_{\mathsf{c}} \mathsf{N}_2$ then $\mathsf{M} \mathbin{@} [\#\mathsf{N}_1] \equiv_\beta \mathsf{M} \mathbin{@} [\#\mathsf{N}_2]$.

Proof. The above statements hold trivially for $\alpha = \mathsf{int}$ thanks to determinism (Lemma 3.3.12(1)).

(1) We prove symmetry and transitivity by induction on α.

- $\alpha = \mathsf{int}$: Holds trivially, as mentioned previously.
- $\alpha = \beta_1 \to \beta_2$:
  - Transitivity:
    Let $\mathsf{M}_1 \equiv_{\beta_1 \to \beta_2} \mathsf{M}_2$ and $\mathsf{M}_2 \equiv_{\beta_1 \to \beta_2} \mathsf{M}_3$ and let $\mathsf{L}_1 \equiv_{\beta_1} \mathsf{L}_2 \equiv_{\beta_1} \mathsf{L}_3$. By definition we have $\mathsf{M}_1 \mathbin{@} [\#\mathsf{L}_1] \equiv_{\beta_2} \mathsf{M}_2 \mathbin{@} [\#\mathsf{L}_2]$ and $\mathsf{M}_2 \mathbin{@} [\#\mathsf{L}_2] \equiv_{\beta_2} \mathsf{M}_3 \mathbin{@} [\#\mathsf{L}_3]$. From IH we have $\mathsf{M}_1 \mathbin{@} [\#\mathsf{L}_1] \equiv_{\beta_2} \mathsf{M}_3 \mathbin{@} [\#\mathsf{L}_3]$, so we conclude $\mathsf{M}_1 \equiv_{\beta_1 \to \beta_2} \mathsf{M}_3$.
  - Symmetry:
    Let $\mathsf{M} \equiv_{\beta_1 \to \beta_2} \mathsf{M}'$ and $\mathsf{N} \equiv_{\beta_1} \mathsf{N}'$. By IH we have $N' \equiv_{\beta_1} N$. By definition we get $\mathsf{M} \mathbin{@} [\#\mathsf{N}'] \equiv_{\beta_2} \mathsf{M}' \mathbin{@} [\#\mathsf{N}]$. Through another application of the IH we get $\mathsf{M}' \mathbin{@} [\#\mathsf{N}] \equiv_{\beta_2} \mathsf{M} \mathbin{@} [\#\mathsf{N}']$, so we conclude.

(2) We prove reflexivity by proving the following statement:

Given any $\mathsf{M} : \beta_1 \to \cdots \to \beta_k \to \mathsf{int}$, for all $\mathsf{L}_1 \equiv_{\beta_1} \mathsf{L}'_1, \ldots, \mathsf{L}_k \equiv_{\beta_k} \mathsf{L}'_k$, we have

$$\mathsf{M} \mathbin{@} [\#\mathsf{L}_1, \ldots, \#\mathsf{L}_k] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \quad \Rightarrow \quad \mathsf{M} \mathbin{@} [\#\mathsf{L}'_1, \ldots, \#\mathsf{L}'_k] \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

By symmetry, Lemma 4.2.10, and determinism (Lemma 3.3.12(1)) the above statement and reflexivity are one and the same. The proof is done by induction on a type derivation of $\mathsf{M} : \beta_1 \to \cdots \to \beta_k \to \mathsf{int}$. We set $\equiv_\alpha$ is reflexive as the first inductive hypothesis (IH1), and make use of the following simultaneous sister statement (IH2):

Given $\Delta \Vdash^r (P, T) : \beta_1 \to \cdots \to \beta_k \to \mathsf{int}$, for all $\vec{R} \models \Delta$, $\vec{R'} \models \Delta$, if $\vec{R} \equiv \vec{R'}$ then we have $\langle \vec{R}, P, T \rangle \equiv_{\beta_1 \to \cdots \to \beta_k \to \mathsf{int}} \langle \vec{R'}, P, T \rangle$.

- Case $\mathsf{M} : \mathsf{int}$: Holds trivially.
- Case $\mathsf{M} = \langle \vec{R}, P, T \rangle$: We have $\vec{R} \models \Delta$ and $\Delta \Vdash^r (P, T) : \beta_1 \to \cdots \to \beta_k \to \mathsf{int}$ from the type derivation, where r is the length of $\vec{R}$. From IH1 we have $T \equiv T$ and $\vec{R} \equiv \vec{R}$, so we conclude with IH2.

- Case $P = \texttt{Load}\ i; P'$: We have $\Delta[i : \gamma] \Vdash^r (P', T) : \beta_2 \to \cdots \to \beta_k \to \mathsf{int}$ from the type derivation. We have two subcases:

  - $T = [\,]$: Then $\gamma = \beta_1$. Let $\mathsf{L_1} \equiv_{\beta_1} \mathsf{L'_1}, \ldots, \mathsf{L_k} \equiv_{\beta_k} \mathsf{L'_k}$. Let $\vec{R}, \vec{R'}$ of length r such that $\vec{R} \models \Delta$, $\vec{R'} \models \Delta$, $\vec{R} \equiv \vec{R'}$. We use IH2 to obtain

    $$\langle \vec{R}[R_i := \#\mathsf{L_1}], P', [\,] \rangle \equiv_{\beta_2 \to \cdots \to \beta_k \to \mathsf{int}} \langle \vec{R'}[R'_i := \#\mathsf{L'_1}], P', [\,] \rangle$$

    By the definition of $\equiv_\alpha$, we have

    $$\langle \vec{R}[R_i := \#\mathsf{L_1}], P', [\mathsf{L_2}, \ldots, \mathsf{L_k}] \rangle \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

    if and only if

    $$\langle \vec{R'}[R'_i := \#\mathsf{L'_1}], P', [\mathsf{L'_2}, \ldots, \mathsf{L'_k}] \rangle \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

    By (3) for int and the definition of $\equiv_\alpha$, we conclude that

    $$\langle \vec{R}, \texttt{Load}\ i; P', [\,] \rangle \equiv_{\beta_1 \to \cdots \to \beta_k \to \mathsf{int}} \langle \vec{R'}, \texttt{Load}\ i; P', [\,] \rangle$$

  - $T = a :: T'$: We use IH1 to obtain $\#^{-1}(a) \equiv_\gamma \#^{-1}(a)$, for some α. Let $\vec{R}, \vec{R'}$ of length r such that $\vec{R} \models \Delta$, $\vec{R'} \models \Delta$, $\vec{R} \equiv \vec{R'}$. We use IH2 to obtain

    $$\langle \vec{R}[R_i := a], P', T \rangle \equiv_{\beta_1 \to \cdots \to \beta_k \to \mathsf{int}} \langle \vec{R'}[R'_i := a], P', T \rangle$$

    Let $\mathsf{L_1} \equiv_{\beta_1} \mathsf{L'_1}, \ldots, \mathsf{L_k} \equiv_{\beta_k} \mathsf{L'_k}$. By the definition of $\equiv_\alpha$, we have

    $$\langle \vec{R}[R_i := a], P', T \mathbin{@} [\mathsf{L_1}, \ldots, \mathsf{L_k}] \rangle \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

    if and only if

    $$\langle \vec{R'}[R'_i := a], P', T \mathbin{@} [\mathsf{L'_1}, \ldots, \mathsf{L'_k}] \rangle \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$$

    By (3) for int and the definition of $\equiv_\alpha$, we conclude that

    $$\langle \vec{R}, \texttt{Load}\ i; P', T \rangle \equiv_{\beta_1 \to \cdots \to \beta_k \to \mathsf{int}} \langle \vec{R'}, \texttt{Load}\ i; P', T \rangle$$

- Case $P = \texttt{Call}\ i$: From the type derivation we have

  $$\Delta[i : \alpha_1 \to \alpha_j \to \beta_1 \to \beta_k \to \mathsf{int}] \Vdash^r (\texttt{Call}\ i, [\mathsf{N_1}, \ldots, \mathsf{N_j}]) : \beta_1 \to \beta_k \to \mathsf{int}$$

  and $\mathsf{N_1} : \alpha_1, \ldots, \mathsf{N_j} : \alpha_j$. Let $\vec{R}, \vec{R'}$ of length r such that

  $$\begin{gathered} \vec{R} \models \Delta[i : \alpha_1 \to \alpha_j \to \beta_1 \to \beta_k \to \mathsf{int}] \\ \vec{R'} \models \Delta[i : \alpha_1 \to \alpha_j \to \beta_1 \to \beta_k \to \mathsf{int}] \\ \vec{R} \equiv \vec{R'} \end{gathered}$$

Then we have $\#^{-1}(R_i) \equiv_{\alpha_1\to\alpha_j\to\beta_1\to\beta_k\to\mathsf{int}} \#^{-1}(R'_i)$, and by IH1 we must have $\mathsf{N}_1 \equiv_{\alpha_1} \mathsf{N}_1, \ldots, \mathsf{N}_j \equiv_{\alpha_j} \mathsf{N}_j$, so by Lemma 4.2.10 we have

$$\#^{-1}(R_i) \mathbin{@} [\mathsf{N}_1, \ldots, \mathsf{N}_j] \equiv_{\beta_1\to\cdots\to\beta_k\to\mathsf{int}} \#^{-1}(R_i) \mathbin{@} [\mathsf{N}_1, \ldots, \mathsf{N}_j]$$

Let $\mathsf{L}_1 \equiv_{\beta_1} \mathsf{L}'_1, \ldots, \mathsf{L}_\mathsf{k} \equiv_{\beta_k} \mathsf{L}'_\mathsf{k}$. By definition, we have

$$\#^{-1}(R_i) \mathbin{@} [\mathsf{N}_1, \ldots, \mathsf{N}_j, \mathsf{L}_1 \ldots, \mathsf{L}_\mathsf{k}] \twoheadrightarrow_\mathsf{c} \mathsf{n}$$

if and only if

$$\#^{-1}(R'_i) \mathbin{@} [\mathsf{N}_1, \ldots, \mathsf{N}_j, \mathsf{L}'_1 \ldots, \mathsf{L}'_\mathsf{k}] \twoheadrightarrow_\mathsf{c} \mathsf{n}$$

By (3) for int and the definition of $\equiv_\alpha$, we conclude that

$$\langle \vec{R}, \mathtt{Call}\ i, [\mathsf{N}_1, \ldots, \mathsf{N}_j] \rangle \equiv_{\beta_1\to\cdots\to\beta_k\to\mathsf{int}} \langle \vec{R}'[R'_i := a], \mathtt{Call}\ i, [\mathsf{N}_1, \ldots, \mathsf{N}_j] \rangle$$

- Case $P = j_3 \leftarrow \mathtt{App}(j_1,\ j_2); P'$: From the type derivation we have

$$(\Delta, j_1 : \alpha_1 \to \alpha_2, j_2 : \alpha_1)[j_3 : \alpha_2] \Vdash^r (P', T) : \beta_1 \to \cdots \to \beta_k \to \mathsf{int}$$

Let $\vec{R}, \vec{R}'$ of length r such that

$$\begin{gathered} \vec{R} \models \Delta, j_1 : \alpha_1 \to \alpha_2, j_2 : \alpha_1 \\ \vec{R}' \models \Delta, j_1 : \alpha_1 \to \alpha_2, j_2 : \alpha_1 \\ \vec{R} \equiv \vec{R}' \end{gathered}$$

As $\vec{R} \equiv \vec{R}'$, by definition, $\#^{-1}(!R_{j_1} \cdot !R_{j_2}) \equiv_{\alpha_2} \#^{-1}(!R'_{j_1} \cdot !R'_{j_2})$. Thus we can use IH2 to obtain

$$\langle \vec{R}[R_{j_3} := !R_{j_1} \cdot !R_{j_2}], P', T \rangle \equiv_{\beta_1\to\cdots\to\beta_k\to\mathsf{int}} \langle \vec{R}'[R'_{j_3} := !R'_{j_1} \cdot !R'_{j_2}], P', T \rangle$$

Let $\mathsf{L}_1 \equiv_{\beta_1} \mathsf{L}'_1, \ldots, \mathsf{L}_\mathsf{k} \equiv_{\beta_k} \mathsf{L}'_\mathsf{k}$. By definition, we have

$$\langle \vec{R}[R_{j_3} := !R_{j_1} \cdot !R_{j_2}], P', T \mathbin{@} \mathsf{L}_1 \ldots, \mathsf{L}_\mathsf{k} \rangle \twoheadrightarrow_\mathsf{c} \mathsf{n}$$

if and only if

$$\langle \vec{R}'[R'_{j_3} := !R'_{j_1} \cdot !R'_{j_2}], P', T \mathbin{@} \mathsf{L}'_1 \ldots, \mathsf{L}'_\mathsf{k} \rangle \twoheadrightarrow_\mathsf{c} \mathsf{n}$$

By (3) for int and the definition of $\equiv_\alpha$, we conclude that

$$\langle \vec{R}, j_3 \leftarrow \mathtt{App}(j_1,\ j_2); P', T \rangle \equiv_{\beta_1\to\cdots\to\beta_k\to\mathsf{int}} \langle \vec{R}', j_3 \leftarrow \mathtt{App}(j_1,\ j_2); P', T \rangle$$

- Cases $P = j_2 \leftarrow \mathtt{Pred}(j_1); P'$, $P = j_2 \leftarrow \mathtt{Succ}(j_1); P'$, $P = j_4 \leftarrow \mathtt{Test}(j_1,\ j_2,\ j_3); P'$

are analogous.

- Case $\mathsf{M} = \mathsf{Y}$. Let $\alpha = \beta_1 \to \cdots \to \beta_k \to \mathsf{int}, \mathsf{L_1} \equiv_{\beta_1} \mathsf{L'_1}, \ldots, \mathsf{L_k} \equiv_{\beta_k} \mathsf{L'_k}$, and $\mathsf{N} \equiv_{(\beta_1 \to \cdots \to \beta_k \to \mathsf{int}) \to (\beta_1 \to \cdots \to \beta_k \to \mathsf{int})} \mathsf{N'}$. We have:

$$\begin{array}{rl} & \mathsf{Y} \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \\ \Rightarrow & \text{Lemma 4.2.8} \\ & \exists m \in \mathbb{N}.\ \mathsf{fix}_m \mathbin{@} [\#\mathsf{N}, \#\mathsf{L_1}, \ldots, \#\mathsf{L_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \\ \Rightarrow & \text{Lemma 4.2.9} \\ & \mathsf{fix}_m \mathbin{@} [\#\mathsf{N'}, \#\mathsf{L'_1}, \ldots, \#\mathsf{L'_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \\ \Rightarrow & \text{Lemma 4.2.8} \\ & \mathsf{Y} \mathbin{@} [\#\mathsf{N'}, \#\mathsf{L'_1}, \ldots, \#\mathsf{L'_k}] \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \end{array}$$

  Thus we can conclude that for all α, $\mathsf{Y} \equiv_{(\alpha \to \alpha) \to \alpha} \mathsf{Y}$.

(3) By induction on α.

- $\alpha = \mathsf{int}$: Holds trivially.
- $\alpha = \beta_1 \to \beta_2$: Let $\mathsf{L}, \mathsf{L'} \in \mathcal{M}_{\mathbb{A}}$ such that $\mathsf{L} \equiv_{\beta_1} \mathsf{L'}$. By Lemma 3.3.12.(3) we have $\mathsf{M} \mathbin{@} [\#L] \to_{\mathtt{wh}} \mathsf{N} \mathbin{@} [\#L]$, so by IH we have $\mathsf{M} \mathbin{@} [\#L] \equiv_{\beta_2} \mathsf{N} \mathbin{@} [\#L]$. As $N \equiv_{\beta_1 \to \beta_2} N$ (reflexivity) we have $\mathsf{N} \mathbin{@} [\#L] \equiv_{\beta_2} \mathsf{N} \mathbin{@} [\#L']$, so we have

$$\mathsf{M} \mathbin{@} [\#L] \equiv_{\beta_2} \mathsf{N} \mathbin{@} [\#L] \equiv_{\beta_2} \mathsf{N} \mathbin{@} [\#L']$$

  We conclude by transitivity.

(4) From (3) we have $N_1 \equiv_\alpha N_2$. As $M \equiv_{\alpha \to \beta} M$ by reflexivity, $\mathsf{M} \mathbin{@} [\#\mathsf{N_1}] \equiv_\beta \mathsf{M} \mathbin{@} [\#\mathsf{N_2}]$ by definition. □

Corollary 4.2.12.
The relation $\equiv_\alpha$ is an equivalence.

Lemma 4.2.13.
For (E)PCF *programs $M, N_1, \ldots, N_n$ such that $\vdash M : \alpha_1 \to \cdots \to \alpha_n \to \beta$ and $\vdash N_i : \alpha_i$ for all i $(1 \leq i \leq n)$, we have*

$$|M| \mathbin{@} [\#|N_1|, \ldots, \#|N_n|] \equiv_\beta |M N_1 \cdots N_n|.$$

Proof. By induction on n, the case $n = 0$ being trivial.

Case $n > 0$. By Definitions 4.1.4 and 4.1.1, we get

$$|M N_1 \cdots N_n| = \mathsf{Pr}^1_1 \mathbin{@} [\#\mathsf{Pr}^1_1, \#|M N_1 \cdots N_{n-1}|, \#|N_n|] \twoheadrightarrow_{\mathsf{c}} |M N_1 \cdots N_{n-1}| \mathbin{@} [\#|N_n|],$$

whence $|MN_1 \cdots N_n| \equiv_\beta |MN_1 \cdots N_{n-1}| \mathbin{@} [\#|N_n|]$ by Lemma 4.2.11(3). By IH, we have $|M| \mathbin{@} [\#|N_1|, \ldots, \#|N_{n-1}|] \equiv_\beta |MN_1 \cdots N_{n-1}|$, so we conclude by Corollary 4.2.12. □

Lemma 4.2.14 (Weakening for EAMs)**.**

Let $M \in \Lambda^{\mathsf{E}}$ be such that $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \beta$. Assume that $x_i \notin \mathrm{FV}(M)$ for some i (where $1 \leq i \leq n$). Then for all $a_1 \in \mathcal{D}_{\alpha_1}, \ldots, a_n \in \mathcal{D}_{\alpha_n}$:

$$|M|_{x_1,\ldots,x_n} \mathbin{@} [a_1, \ldots, a_n] \equiv_\beta |M|_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n} \mathbin{@} [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n]$$

Proof. By induction on a derivation of $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \beta$. As a matter of notation, we let $\Gamma = x_1 : \alpha_1, \ldots, x_n : \alpha_n$ and introduce the abbreviations

$$\begin{array}{lcllcl} \vec{a} & = & a_1, \ldots, a_n; & \vec{x} & = & x_1, \ldots, x_n; \\ \vec{a}^- & = & a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n; & \vec{x}^- & = & x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n. \end{array}$$

- Case $\Gamma \vdash \underline{0} : \mathsf{int}$. We get

$$\begin{array}{rcll} |\underline{0}|_{\vec{x}} @ [\vec{a}] & = & \mathsf{Pr}_1^{n+1} \mathbin{@} [0, \vec{a}]\,, & \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \#^{-1}(0), & \text{by Lemma 4.1.2(1),} \\ & {}_{\mathsf{c}}\!\twoheadleftarrow & \mathsf{Pr}_1^n \mathbin{@} [0, \vec{a}^-]\,, & \text{by Lemma 4.1.2(1),} \\ & = & |\underline{0}|_{\vec{x}^-} @ [\vec{a}^-], & \text{by Definition 4.1.4.} \end{array}$$

  Conclude by Lemma 4.2.11(3).

- Case $\Gamma, y : \beta \vdash y : \beta$. Let $b \in \mathcal{D}_\beta$. By Definition 4.1.4 and Lemma 4.1.2(1), we have $|y|_{\vec{x},y} \mathbin{@} [\vec{a}, b] = \mathsf{Pr}_{n+1}^{n+1} \mathbin{@} [\vec{a}, b] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(b)$ and $|y|_{\vec{x}^-,y} @ [\vec{a}^-, b] = \mathsf{Pr}_n^n \mathbin{@} [\vec{a}^-, b] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(b)$. Conclude by Lemma 4.2.11(3).

- Case $\Gamma \vdash MN : \beta$ since $\Gamma \vdash M : \alpha \to \beta$ and $\Gamma \vdash N : \alpha$. Then, we have

$$\begin{array}{rcll} |MN|_{\vec{x}} \mathbin{@} [\vec{a}] & = & \mathsf{Apply}_n^2 \mathbin{@} [\#\mathsf{Pr}_1^1, \#|M|_{\vec{x}}, \#|N|_{\vec{x}}, \vec{a}]\,, & \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & |M|_{\vec{x}} \mathbin{@} [\vec{a}, \#(|N|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,, & \text{by Lemma 4.1.2(2).} \end{array}$$

  By IH, we have $|N|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_\alpha |N|_{\vec{x}^-} \mathbin{@} [\vec{a}^-]$ and $|M|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\alpha\to\beta} |M|_{\vec{x}^-} \mathbin{@} [\vec{a}^-]$, so by Lemma 4.2.11(2) (reflexivity), $|M|_{\vec{x}} \mathbin{@} [\vec{a}, \#(|N|_{\vec{x}} @ [\vec{a}])] \equiv_\beta |M|_{\vec{x}} \mathbin{@} [\vec{a}, \#(|N|_{\vec{x}^-} @ [\vec{a}^-])]$. Conclude by Corollary 4.2.12, Lemma 4.2.11(3) and Definition 4.1.4.

- Case $\Gamma \vdash M\langle N/z\rangle : \beta$ with $\Gamma, z : \alpha \vdash M : \beta$ and $\vdash N : \alpha$. By Definition 4.1.4 we get $|M\langle N/z\rangle|_{\vec{x}} @ [\vec{a}] = |M|_{z,\vec{x}} \mathbin{@} [\#|N|, \vec{a}]$. Conclude by applying the IH.

- Case $\Gamma \vdash \lambda z.M : \gamma \to \delta$ since $\Gamma, z : \gamma \vdash M : \delta$. By Definition 4.1.4, $|\lambda z.M|_{\vec{x}} = |M|_{\vec{x},z}$. This case follows straightforwardly from the IH.

- Case $\Gamma \vdash \mathbf{succ}\, M : \mathsf{int}$, since $\Gamma \vdash M : \mathsf{int}$. We get

$$\begin{array}{rcll} |\mathbf{succ}\, M|_{\vec{x}} \mathbin{@} [\vec{a}] & = & \mathsf{Apply}^1_n \mathbin{@} [\#\mathsf{Succ}, \#|M|_{\vec{x}}, \vec{a}]\,, & \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \mathsf{Succ} \mathbin{@} [\#(|M|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,, & \text{by Lemma 4.1.2(2).} \end{array}$$

  From the IH we obtain $|M|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\mathsf{int}} |M|_{\vec{x}^-} \mathbin{@} [\vec{a}^-]$, so by Lemma 4.2.11(2) (reflexivity),

$$\mathsf{Succ} \mathbin{@} [\#(|M|_{\vec{x}} @ [\vec{a}])] \equiv_{\mathsf{int}} \mathsf{Succ} \mathbin{@} [\#(|M|_{\vec{x}^-} @ [\vec{a}^-])]$$

  Conclude by Corollary 4.2.12, Lemma 4.2.11(3) and Definition 4.1.4.

- Case $\Gamma \vdash \mathbf{pred}\, M : \mathsf{int}$. Analogous.

- Case $\Gamma \vdash \mathbf{ifz}(L, M, N) : \beta$. Analogous.

- Case $\Gamma \vdash \mathbf{fix}\, M : \beta$ since $\Gamma \vdash M : \beta \to \beta$. Then,

$$\begin{array}{rcll} |\mathbf{fix}\, M|_{\vec{x}} \mathbin{@} [\vec{a}] & = & \mathsf{Apply}^1_n \mathbin{@} [\#\mathsf{Y}, \#|M|_{\vec{x}}, \vec{a}]\,, & \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \mathsf{Y} \mathbin{@} [\#(|M|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,, & \text{by Lemma 4.1.2(2).} \end{array}$$

  By IH, we have $|M|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\beta\to\beta} |M|_{\vec{x}^-} \mathbin{@} [\vec{a}^-]$, so by Lemma 4.2.11(2) (reflexivity) we get

$$\mathsf{Y} \mathbin{@} [\#(|M|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] \equiv_{\beta} \mathsf{Y} \mathbin{@} [\#(|M|_{\vec{x}^-} \mathbin{@} [\vec{a}^-]\,)]$$

  Conclude by Definition 4.1.4, applying Corollary 4.2.12 and Lemma 4.2.11(3) if $n > 1$, and Corollary 4.2.12 when $n = 1$. □

Corollary 4.2.15.
Let $M \in \mathcal{P}_{\mathsf{E}}$ and $\alpha_1, \ldots, \alpha_n, \beta \in \mathbb{T}$. If $\vdash M : \beta$ then, for all $x_1, \ldots, x_n \in \mathrm{Var}$ and $a_i \in \mathcal{D}_{\alpha_i}$ $(1 \le i \le n)$, we have

$$|M|_{x_1,\ldots,x_n} \mathbin{@} [a_1, \ldots, a_n] \equiv_{\beta} |M|\,.$$

4.2.2 An Adequate Model

The intuitive description of our model from earlier is formalized in the following definition.

Definition 4.2.16 ($\mathcal{D}_\alpha/_{\simeq_\alpha}$)**.**

(1) For all types $\alpha \in \mathbb{T}$, define $\mathcal{D}_\alpha = \{a \in \mathbb{A} \mid \#^{-1}(a) : \alpha\}$.

(2) For $\alpha \in \mathbb{T}$ and $a, b \in \mathcal{D}_\alpha$, we write $a \simeq_\alpha b$ whenever $\#^{-1}(a) \equiv_\alpha \#^{-1}(b)$ holds.

(3) We write $[a]_{\simeq_\alpha}$ for the equivalence class of a modulo $\simeq_\alpha$.

(4) We let $\mathcal{D}_\alpha/_{\simeq_\alpha} = \{[a]_{\simeq_\alpha} \mid a \in \mathcal{D}_\alpha\}$.

The model which is shown to be fully abstract is constructed as follows.

Definition 4.2.17.
Define the model $\mathscr{D} = \langle (\mathcal{D}_\alpha/_{\simeq_\alpha})_{\alpha\in\mathbb{T}}, (\ \cdot^{\alpha,\beta})_{\alpha,\beta\in\mathbb{T}}, [\![-]\!]\rangle$ *where*

$$\begin{array}{rcl} [a]_{\simeq_{\alpha\to\beta}} \cdot^{\alpha,\beta} [b]_{\simeq_\alpha} & = & [a\cdot b]_{\simeq_\beta} \\ [\![x_1:\beta_1,\ldots,x_n:\beta_n \vdash P:\alpha]\!] & = & [\#|P|_{\vec{x}}]_{\simeq_{\beta_1\to\cdots\to\beta_n\to\alpha}} \end{array}$$

The application $\cdot^{\alpha,\beta}$ is well defined by Proposition 3.3.21(1) and the interpretation function $[\![-]\!]$ by Theorem 4.1.6. Note that two PCF programs P, Q of type α have the same interpretation in the model $\mathscr{D}$, i.e. $[\![P]\!]^\alpha = [\![Q]\!]^\alpha$, exactly when $|P| \equiv_\alpha |Q|$. Hence, we mainly work with translations of PCF terms modulo $\simeq_\alpha$ and draw conclusions for $\mathscr{D}$ at the end (Theorem 4.2.26).

Everything is now in place to prove that the model is sound and adequate.

Corollary 4.2.18 (Soundness)**.**
For all PCF *programs* P_1, P_2 *of type* α*, we have*

$$|P_1| \leftrightarrow_{\mathsf{c}} |P_2| \ \Rightarrow\ |P_1| \equiv_\alpha |P_2|$$

Proof. By definition of interconvertibility $\leftrightarrow_{\mathsf{c}}$ and Lemma 4.2.11(3). □

Theorem 4.2.19 (Adequacy)**.**
Given two PCF *programs* P_1, P_2 *of type* α*, we have*

$$|P_1| \equiv_\alpha |P_2| \ \Rightarrow\ P_1 \equiv_{\mathrm{obs}} P_2.$$

Proof. By Proposition 1.3.4 it is sufficient to show $P_1 \equiv_{\mathrm{app}} P_2$. Proceed by induction on α. Base case $\alpha = \mathsf{int}$. For $i = 1, 2$, we have $P_i \twoheadrightarrow_{\mathsf{PCF}} \underline{n} \iff |P_i| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$, by Theorem 4.1.11. Then, this case follows from the assumption $|P_1| \equiv_{\mathsf{int}} |P_2|$, that is $|P_1| \twoheadrightarrow_{\mathsf{c}} \mathsf{n} \iff |P_2| \twoheadrightarrow_{\mathsf{c}} \mathsf{n}$.

Case $\alpha = \beta_1 \to \beta_2$. Take any PCF program $\vdash Q_1 : \beta_1$. Using Lemma 4.2.13 we get $|P_1Q_1| \equiv_{\beta_2} |P_1| \mathbin{@} [\#|Q_1|]$ and $|P_2Q_1| \equiv_{\beta_2} |P_2| \mathbin{@} [\#|Q_1|]$. From $|P_1| \equiv_{\beta_1\to\beta_2} |P_2|$ and $|Q_1| \equiv_{\beta_1} |Q_1|$ (reflexivity), we get $|P_1| \mathbin{@} [\#|Q_1|] \equiv_{\beta_2} |P_2| \mathbin{@} [\#|Q_1|]$. By transitivity of $\equiv_{\beta_2}$, we get $|P_1Q_1| \equiv_{\beta_2} |P_2Q_1|$ and by IH $P_1Q_1 \equiv_{\mathrm{app}} P_2Q_1$. As Q_1 is arbitrary, conclude $P_1 \equiv_{\mathrm{app}} P_2$. □

4.2.3 Definability and Full Abstraction

The adequacy result established above gives one implication of the Full Abstraction property. In order to prove the converse implication, namely completeness, we need to show that the model does not contain any undefinable element (*junk*). This amounts to associate with any typable

EAM, a PCF program of the same type exhibiting the same observational behavior (*mutatis mutandi*). The problem is that an EAM might perform useless computations (e.g., updating the value of a register that is subsequently never used) that could however prevent the machine from terminating. To simulate the same behavior in the corresponding PCF term, we use a 'convergency' test $\mathtt{Ifc}(x,y)$ (*if-converges-then*) defined by: $\mathtt{Ifc}(x,y) = \mathbf{ifz}(x,y,y)$. Indeed, given PCF programs P and Q such that $\vdash P : \mathsf{int}$, the program $\mathtt{Ifc}(P,Q)$ is observationally indistinguishable from Q exactly when P is terminating, independently from its result.

Recall that an EAM typing context $\Delta = i_1 : \beta_{i_1}, \dots, i_k : \beta_{i_k}$ is a list of associations between indices of registers and types. Moreover, the judgments $\mathsf{M} : \alpha$, $\Delta \Vdash^r (P,T) : \alpha$ and $\vec{R} \models \Delta$ have been defined in Definition 3.3.16.

Definition 4.2.20 (Reverse Translation)**.**
Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$, P be an EAM program, $T \in \mathcal{T}_{\mathbb{A}}$, $\alpha \in \mathbb{T}$. Given $\mathsf{M} : \alpha$ (resp. $\Delta \Vdash^r (P,T) : \alpha$), we associate a PCF term $(\!|\mathsf{M}|\!)_\alpha$ (resp. $(\!|P,T|\!)^\Delta_\alpha$) defined by induction on the type-derivation as follows:

$$(\!|\mathsf{n}|\!)_{\mathsf{int}} = \underline{n};$$
$$(\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha} = \lambda x.\mathbf{fix}\, x;$$
$$(\!|\langle \vec{R}, P, T\rangle|\!)_\alpha = (\lambda x_{i_1}\dots x_{i_k}.(\!|P,T|\!)^{i_1:\beta_{i_1},\dots,i_k:\beta_{i_k}}_\alpha) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_k})|\!)_{\beta_{i_k}},$$
$$\text{where } \vec{R} \models i_1 : \beta_{i_1}, \dots, i_k : \beta_{i_k}, \text{ for } 1 \le k \le r;$$
$$(\!|\mathtt{Load}\ i; P, [\,]|\!)^\Delta_{\beta\to\alpha} = \lambda x_i.(\!|P,[\,]|\!)^{\Delta[i:\beta]}_\alpha;$$
$$(\!|\mathtt{Load}\ i; P, a :: T|\!)^\Delta_\alpha = \left(\lambda x_i.(\!|P,T|\!)^{\Delta[i:\beta]}_\alpha\right) \cdot (\!|\#^{-1}(a)|\!)_\beta;$$
$$(\!|j \leftarrow \mathtt{Pred}(i); P, T|\!)^{\Delta,i:\mathsf{int}}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_j.(\!|P,T|\!)^{(\Delta,i:\mathsf{int})[j:\mathsf{int}]}_\alpha) \cdot (\mathbf{pred}\, x_i));$$
$$(\!|j \leftarrow \mathtt{Succ}(i); P, T|\!)^{\Delta,i:\mathsf{int}}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_j.(\!|P,T|\!)^{(\Delta,i:\mathsf{int})[j:\mathsf{int}]}_\alpha) \cdot (\mathbf{succ}\, x_i));$$
$$(\!|l \leftarrow \mathtt{Test}(i,\ j,\ k); P, T|\!)^{\Delta,i:\mathsf{int},j:\beta,k:\beta}_\alpha = \mathtt{Ifc}(x_i, (\lambda x_l.(\!|P,T|\!)^{(\Delta,i:\mathsf{int},j:\beta,k:\beta)[l:\beta]}_\alpha) \cdot \mathbf{ifz}(x_i,x_j,x_k));$$
$$(\!|k \leftarrow \mathtt{App}(i,\ j); P, T|\!)^{\Delta,i:\beta\to\alpha,j:\beta}_\gamma = (\lambda x_k.(\!|P,T|\!)^{(\Delta,i:\beta\to\alpha,j:\beta)[k:\alpha]}_\gamma) \cdot (x_i \cdot x_j);$$
$$(\!|\mathtt{Call}\ i, [a_1,\dots,a_n]|\!)^{\Delta,i:\alpha_1\to\cdots\to\alpha_n\to\alpha}_\alpha = x_i \cdot (\!|\#^{-1}(a_1)|\!)_{\alpha_1} \cdots (\!|\#^{-1}(a_n)|\!)_{\alpha_n}.$$

It is easy to check that the PCF term associated with $\mathsf{M} : \alpha$ is actually a PCF program.

Example 4.2.21.
Consider the EAMs introduced in Example 3.3.10. The reverse translation applied to said machines produces the following PCF programs:

(1) $(\!|\mathsf{I}|\!)_{\alpha\to\alpha} = (\!|\mathtt{Load}\ 0; \mathtt{Call}\ 0, [\,]|\!)_{\alpha\to\alpha} = \lambda x_0.(\!|\mathtt{Call}\ 0, [\,]|\!)^{0:\alpha}_\alpha = \lambda x_0.x_0.$

(2) $$\begin{aligned}(\!|\mathsf{Y} \mathbin{@} [\#\mathsf{I}]|\!)_\alpha &= (\!|\mathtt{Load}\ 0; \mathtt{Load}\ 1; 0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1; [\#\mathsf{Y}, \#\mathsf{I}]|\!)_\alpha \\ &= (\lambda x_0.(\lambda x_1.(\!|0 \leftarrow \mathtt{App}(0,\ 1); 1 \leftarrow \mathtt{App}(1,\ 0); \mathtt{Call}\ 1; [\,]|\!)_\alpha) \cdot (\!|\mathsf{I}|\!)_{\alpha\to\alpha}) \cdot (\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha} \\ &= (\lambda x_0.(\lambda x_1.(\lambda x'_0.(\lambda x'_1.x'_1) \cdot (x_1 \cdot x'_0)) \cdot (x_0 \cdot x_1)) \cdot (\lambda y_0.y_0)) \cdot (\lambda x.\mathbf{fix}\, x).\end{aligned}$$

(3) $(\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}} = \lambda x_0.(\!|0 \leftarrow \mathtt{Succ}(0); \mathtt{Call}\ 0, [\,]|\!)^{0:\mathsf{int}}_{\mathsf{int}} = \lambda x_0.\mathtt{Ifc}(x_0, (\lambda x_0'.x_0') \cdot \mathbf{succ}\, x_0)$.

(4) $(\!|\mathsf{Succ2}|\!)_{\mathsf{int}\to\mathsf{int}} = \big(\lambda x_0.(\lambda x_1.(\lambda x_1'.(\lambda x_1''.x_1'') \cdot (x_0 \cdot x_1')) \cdot (x_0 \cdot x_1))\big) \cdot (\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}}$.

Note that, for all PCF *programs* P *of type* int*, the program* $(\!|\mathsf{Succ1}|\!)_{\mathsf{int}\to\mathsf{int}} \cdot P$ *is well typed and converges to a natural number exactly when* P *does.*

Proposition 4.2.22.

(1) Let $\mathsf{M} \in \mathcal{M}_{\mathbb{A}}$ *and* $\alpha \in \mathbb{T}$*. If* $\mathsf{M} : \alpha$ *then* $\vdash (\!|\mathsf{M}|\!)_\alpha : \alpha$*.*

(2) Let P *be an EAM program,* $T \in \mathcal{T}_{\mathbb{A}}$*,* $\Delta = i_1 : \alpha_{i_1}, \ldots, i_k : \alpha_{i_k}$ *be a type environment and* $\alpha \in \mathbb{T}$*. Then,*

$$\Delta \Vdash^r (P, T) : \alpha \qquad \Rightarrow \qquad x_{i_1} : \alpha_{i_1}, \ldots, x_{i_k} : \alpha_{i_k} \vdash (\!|P, T|\!)^{i_1:\alpha_{i_1},\ldots,i_k:\alpha_{i_k}}_\alpha : \alpha$$

Proof. Both (1) and (2) follow by mutual induction on a derivation of $\mathsf{M} : \alpha$ and $\Delta \Vdash^r (P, T) : \alpha$ and call IH1, IH2 the respective induction hypothesis.

As a matter of notation, if $\Delta = i_1 : \alpha_{i_1}, \ldots, i_k : \alpha_{i_k}$ we let $\Delta^* = x_{i_1} : \alpha_{i_1}, \ldots, x_{i_k} : \alpha_{i_k}$.

(1) Case (nat). Then $\mathsf{M} = \mathsf{n}$ and $\alpha = \mathsf{int}$. Conclude since $(\!|\mathsf{n}|\!)_{\mathsf{int}} = \underline{n}$ and $\vdash \underline{n} : \mathsf{int}$ holds.

Case (fix). Then $\mathsf{M} = \mathsf{Y}$ and $\alpha = (\beta \to \beta) \to \beta$. We type $(\!|\mathsf{Y}|\!)^\Delta_\alpha = \lambda x.\mathbf{fix}\, x$ as follows:

$$\dfrac{\dfrac{\dfrac{}{x : \beta \to \beta \vdash x : \beta \to \beta}\,(\mathrm{ax})}{x : \beta \to \beta \vdash \mathbf{fix}\, x : \beta}\,(\mathrm{Y})}{\vdash \lambda x.\mathbf{fix}\, x : (\beta \to \beta) \to \beta}\,(\to_{\mathrm{I}})$$

Case $(\vec{R})$. Then $\mathsf{M} = \langle R_0, \ldots, R_r, P, T\rangle$ with $\vec{R} \models \Delta$ and $\Delta \Vdash^r (P, T) : \alpha$, for some $\Delta = i_1 : \alpha_{i_1}, \ldots, i_k : \alpha_{i_k}$. From the former condition, by rules $(R_\varnothing)$ and $(R_{\mathbb{T}})$, we get a derivation of $\#^{-1}(!R_{i_j}) : \alpha_{i_j}$ having smaller size, for all $1 \le j \le k$. By applying IH1, we obtain a derivation of $\vdash (\!|\#^{-1}(!R_{i_j})|\!)_{\alpha_{i_j}} : \alpha_{i_j}$. From the latter condition and IH2, we have $\Delta^* \vdash (\!|P, T|\!)^\Delta_\alpha : \alpha$. Therefore, we construct a derivation

$$\dfrac{\dfrac{\Delta^* \vdash (\!|P, T|\!)^\Delta_\alpha : \alpha}{\vdash \lambda x_{i_1} \ldots x_{i_k}.(\!|P, T|\!)^\Delta_\alpha : \alpha_{i_1} \to \cdots \to \alpha_{i_k} \to \alpha} \qquad \vdash (\!|\#^{-1}(!R_{i_j})|\!)_{\alpha_{i_j}} : \alpha_{i_j} \quad 1 \le j \le k}{\vdash (\lambda x_{i_1} \ldots x_{i_k}.(\!|P, T|\!)^\Delta_\alpha) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_k})|\!)_{\beta_{i_k}} : \alpha}$$

(2) Case $(\mathrm{load}_\varnothing)$. Then $P = \mathtt{Load}\ j; P'$, $T = [\,]$, $\alpha = \beta_1 \to \beta_2$ and $\Delta[j : \beta_1] \Vdash^r (P', [\,]) : \beta_2$ has a derivation of smaller size. There are two subcases.

- Case $j \notin \mathrm{dom}(\Delta)$, whence $\Delta[j : \beta_1] = \Delta, j : \beta_1$. By IH2 we get $\Delta^* \vdash (\!|P', [\,]|\!)^{\Delta, j:\beta_1}_{\beta_2} : \beta_2$. Simply apply rule $(\to_{\mathrm{I}})$.

- Case $j \in \mathrm{dom}(\Delta)$, say, $j = i_k$. From the IH2 we get $\Gamma, x_{i_k} : \beta_1 \vdash (\!|P', [\,]|\!)_{\beta_2}^{\Delta[i_k:\beta_1]} : \beta_2$ for $\Gamma = x_{i_1} : \alpha_{i_1}, \ldots, x_{i_{k-1}} : \alpha_{i_{k-1}}$. By applying the rule $(\to_{\mathrm{I}})$, we obtain a derivation of $\Gamma \vdash \lambda x_{i_k}.(\!|P', [\,]|\!)_{\beta_2}^{\Delta[i_k:\beta_1]} : \beta_1 \to \beta_2$ whence, by weakening (Lemma 2.2.13(2)), we conclude $\Gamma, x_{i_k} : \alpha_{i_k} \vdash \lambda x_{i_k}.(\!|P', [\,]|\!)_{\beta_2}^{\Delta[i_k:\beta_1]} : \beta_1 \to \beta_2$.

Case $(\mathrm{load}_{\mathbb{T}})$. Then $P = \mathtt{Load}\ j; P'$, $T = a :: T'$. Moreover, $\Delta[j : \beta] \Vdash^r (P', T') : \alpha$ and $\#^{-1}(a) : \beta$ have a derivation of smaller size, for some β. From the former, one obtains a derivation of $\Delta^* \vdash \lambda x_j.(\!|P', T'|\!)_{\alpha}^{\Delta[j:\beta]} : \beta \to \alpha$ proceeding as above. By the IH1 applied to the latter, we obtain a derivation of $\vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$, whence $\Delta^* \vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$ holds by strengthening (Lemma 2.2.13(2)). By applying the rule $(\to_{\mathrm{E}})$, we conclude that $\Delta^* \vdash \left(\lambda x_j.(\!|P', T'|\!)_{\alpha}^{\Delta[j:\beta]}\right) \cdot (\!|\#^{-1}(a)|\!)_{\beta} : \alpha$ is derivable.

Case (pred). Then $P = j \leftarrow \mathtt{Pred}(i); P'$ and $(\Delta, i : \mathsf{int})[j : \mathsf{int}] \Vdash^r (P', T') : \alpha$ has a smaller derivation. We assume $j \notin \mathrm{dom}(\Delta, i : \mathsf{int})$, otherwise proceed as in case $(\mathrm{load}_{\varnothing})$. By IH2, we obtain a derivation of $\Delta^*, x_i : \mathsf{int}, x_j : \mathsf{int} \vdash (\!|(P', T')|\!)_{\alpha}^{\Delta, i:\mathsf{int}, j:\mathsf{int}} : \alpha$, thus:

$$\dfrac{\dfrac{\dfrac{\Delta^*, x_i : \mathsf{int}, x_j : \mathsf{int} \vdash (\!|P', T'|\!)_{\alpha}^{\Delta, i:\mathsf{int}, j:\mathsf{int}} : \alpha}{\Delta^*, x_i : \mathsf{int} \vdash \lambda x_j.(\!|P', T'|\!)_{\alpha}^{\Delta, i:\mathsf{int}, j:\mathsf{int}} : \alpha} \quad \dfrac{\Delta^*, x_i : \mathsf{int} \vdash x_i : \mathsf{int}}{\Delta^*, x_i : \mathsf{int} \vdash \mathbf{pred}\, x_i : \mathsf{int}}}{\Delta^*, x_i : \mathsf{int} \vdash x_i : \mathsf{int} \qquad \Delta^*, x_i \vdash (\lambda x_j.(\!|P', T'|\!)_{\alpha}^{\Delta, i:\mathsf{int}, j:\mathsf{int}}) \cdot (\mathbf{pred}\, x_i) : \alpha}}{\Delta^*, x_i : \mathsf{int} \vdash \mathtt{Ifc}(x_i, (\lambda x_j.(\!|P', T'|\!)_{\alpha}^{\Delta, i:\mathsf{int}, j:\mathsf{int}}) \cdot (\mathbf{pred}\, x_i)) : \alpha}$$

Case (succ). Analogous.

Case (call). Then $P = \mathtt{Call}\ i$ and $T = [a_1, \ldots, a_n]$ with $\#^{-1}(a_j) : \beta_j$, for $j\,(1 \le j \le n)$. Call $\Gamma = \Delta^*, x_i : \beta_1 \to \cdots \to \beta_n \to \alpha$. By IH1, we get $\vdash (\!|\#^{-1}(a_j)|\!)_{\beta_j} : \beta_j$ whence $\Gamma \vdash (\!|\#^{-1}(a)|\!)_{\beta} : \beta$ holds by strengthening (Lemma 2.2.13(2)). Derive

$$\dfrac{\Gamma \vdash x_i : \beta_1 \to \cdots \to \beta_n \to \alpha \quad \Gamma \vdash (\!|\#^{-1}(a_1)|\!)_{\beta_1} : \beta_1 \cdots \Gamma \vdash (\!|\#^{-1}(a_n)|\!)_{\beta_n} : \beta_n}{\Gamma \vdash x_i \cdot (\!|\#^{-1}(a_1)|\!)_{\beta_1} \cdots (\!|\#^{-1}(a_n)|\!)_{\beta_n} : \alpha}$$

(3) Case (app). Then $P = l \leftarrow \mathtt{App}(i,\ j); P'$ and $(\Delta, i : \beta_1 \to \beta_2, j : \beta_1)[l : \beta_2] \Vdash^r (P', T) : \alpha$ has a smaller derivation, for some β_1, β_2. Assume that $l \notin \mathrm{dom}(\Delta, i : \beta_1 \to \beta_2, j : \beta_1)$, otherwise proceed as in case $(\mathrm{load}_{\varnothing})$. Setting $\Gamma' = \Delta^*, x_i : \beta_1 \to \beta_2, x_j : \beta_1$, we get:

$$\dfrac{\dfrac{\Gamma, x_l : \beta_2 \vdash (\!|P', T|\!)_{\alpha}^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2} : \alpha}{\Gamma \vdash \lambda x_l.(\!|P', T|\!)_{\alpha}^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2} : \beta_2 \to \alpha} \quad \dfrac{\Gamma \vdash x_i : \beta_1 \to \beta_2 \quad \Gamma \vdash x_j : \beta_1}{\Gamma \vdash x_i \cdot x_j : \beta_2}}{\Gamma \vdash (\lambda x_l.(\!|P', T|\!)_{\alpha}^{\Delta, i:\beta_1\to\beta_2, j:\beta_1, l:\beta_2}) \cdot (x_i \cdot x_j) : \alpha}$$

Case (test). Then $P = m \leftarrow \mathtt{Test}(i,\ j,\ l); P'$ and for some $\beta \in \mathbb{T}$ there is a smaller derivation of $(\Delta, i : \mathsf{int}, j : \beta, l : \beta)[m : \beta] \Vdash^r (P, T) : \alpha$. If $m \in \mathrm{dom}(\Delta, i : \mathsf{int}, j : \beta, l : \beta)$, proceed as in case $(\mathrm{load}_{\varnothing})$. Otherwise, letting $\Gamma = \Delta^*, x_i : \mathsf{int}, x_j : \beta, x_l : \beta, x_m : \beta$, we

get:

$$\dfrac{\dfrac{\dfrac{\Gamma, x_m : \beta \vdash (\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta} : \alpha}{\Gamma \vdash \lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta} : \beta \to \alpha} \quad \dfrac{\Gamma \vdash x_n : \mathsf{int}, n \in \{i, j, k\}}{\Gamma \vdash \mathbf{ifz}(x_i, x_j, x_l) : \beta}}{\Gamma \vdash x_i : \mathsf{int} \quad \Gamma \vdash (\lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta}) \cdot \mathbf{ifz}(x_i, x_j, x_l) : \alpha}}{\Gamma \vdash \mathtt{Ifc}(x_i, (\lambda x_m.(\!|P', T|\!)_\alpha^{\Delta, i:\mathsf{int}, j:\beta, l:\beta, m:\beta}) \cdot \mathbf{ifz}(x_i, x_j, x_l)) : \alpha}$$

This concludes the proof. □

Theorem 4.2.23.
For all EAMs $\mathsf{M} : \alpha$*, we have* $|(\!|\mathsf{M}|\!)_\alpha| \equiv_\alpha \mathsf{M}$.

Proof. One needs to consider the additional statement:

"For all $i_1 : \alpha_{i_1}, \ldots, i_n : \alpha_{i_n} \Vdash^r (P, T) : \alpha$, $a_{i_1} \in \mathcal{D}_{\alpha_{i_1}}, \ldots, a_{i_n} \in \mathcal{D}_{\alpha_{i_n}}$,

$$|(\!|P, T|\!)_\alpha^{i_1:\alpha_{i_1}, \ldots, i_n:\alpha_{i_n}}|_{x_{i_1}, \ldots, x_{i_n}} @ [a_{i_1}, \ldots, a_{i_n}] \equiv_\alpha \langle \vec{R}^r_{a_{i_1}, \ldots, a_{i_n}}, P, T \rangle,$$

where $\vec{R}^r_{a_{i_1}, \ldots, a_{i_n}}$ denotes the list of registers $R_0, \ldots, R_r$ such that, for all j $(0 \leq j \leq r)$, $!R_j = a_j$ if $j \in \{i_1, \ldots, i_n\}$, and $!R_j = \varnothing$ otherwise."

The two statements are proven by mutual induction on a derivation of $\mathsf{M} : \alpha$ and a derivation of $\Delta \Vdash^r (P, T) : \alpha$, respectively. We refer to the former induction hypothesis as IH1 and to the latter as IH2. As a matter of notation, we let $\Delta = i_1 : \beta_{i_1}, \ldots, i_n : \beta_{i_n}$, $\vec{x} = x_{i_1}, \ldots, x_{i_n}$, and $\vec{a} = a_{i_1}, \ldots, a_{i_n}$ such that for all $j \in \{i_1, \ldots, i_n\}$, $a_j \in \mathcal{D}_{\beta_j}$.

- Case $\mathsf{k} : \mathsf{int}$. We prove this case by induction on $k \in \mathbb{N}$ (and call this IH1′).
  - Case $k = 0$: By Definition 4.1.4 we get $|(\!|\mathsf{0}|\!)_{\mathsf{int}}| = |\underline{0}| = \mathsf{Pr}^1_1@[0]$. Conclude by Lemmas 4.1.2(1) and 4.2.11(3).
  - Case $k = m + 1$, for some $m \in \mathbb{N}$. Then by Definitions 4.1.4 and 4.2.20, we have

    $$\begin{array}{rcl} |(\!|\mathsf{k}|\!)_{\mathsf{int}}| & = & |\underline{m+1}| = |\mathbf{succ}\,\underline{m}| \\ & = & \mathsf{Apply}^1_0 @ [\#\mathsf{Succ}, \#|\underline{m}|] = \mathsf{Pr}^1_1@[\#\mathsf{Succ}, \#|\underline{m}|]. \end{array}$$

    By IH1′ we have $|\underline{m}| \equiv_{\mathsf{int}} \mathsf{m}$, so $\mathsf{Pr}^1_1@[\#\mathsf{Succ}, \#|\underline{m}|] \equiv_{\mathsf{int}} \mathsf{Pr}^1_1@[\#\mathsf{Succ}, m]$, by reflexivity. Conclude by Lemma 4.1.2(1),(4), Lemma 4.2.11(3), and transitivity.
- Case $\mathsf{Y} : (\alpha \to \alpha) \to \alpha$. Let $a, b \in \mathcal{D}_{\alpha \to \alpha}$ such that $a \simeq_{\alpha \to \alpha} b$. Since by Lemma 4.1.3(1)

and Lemma 4.2.11(3) we have $\mathsf{Pr}_1^1 \mathbin{@} [a] \equiv_{\alpha\to\alpha} \#^{-1}(a)$, we obtain

$$\begin{array}{lcll}
|(\!|\mathsf{Y}|\!)_{(\alpha\to\alpha)\to\alpha}| \mathbin{@} [a] & = & |\lambda x.\mathbf{fix}\, x| \mathbin{@} [a]\,, & \text{by Definition 4.2.20,}\\
& = & \mathsf{Apply}_1^1 \mathbin{@} [\#\mathsf{Y}, \#\mathsf{Pr}_1^1, a]\,, & \text{by Definition 4.1.4,}\\
& \twoheadrightarrow_{\mathsf{c}} & \#^{-1}(a) \mathbin{@} [\#\mathsf{Y} \cdot (\#\mathsf{Pr}_1^1 \cdot a)]\,, & \text{by Lemma 4.1.2(6),}\\
& \equiv_\alpha & \#^{-1}(a) \mathbin{@} [\#\mathsf{Y} \cdot a]\,, & \text{by Lemma 4.2.11(4),}\\
& \equiv_\alpha & \#^{-1}(a) \mathbin{@} [\#\mathsf{Y} \cdot b]\,, & \text{by reflexivity,}\\
& \equiv_\alpha & \#^{-1}(b) \mathbin{@} [\#\mathsf{Y} \cdot b]\,, & \text{by Definition 4.2.16,}\\
& {}_{\mathsf{c}}\!\!\twoheadleftarrow & \mathsf{Y} \mathbin{@} [b]\,, & \text{by Lemma 4.1.2(6).}
\end{array}$$

Conclude by Lemma 4.2.11(3) and transitivity.

- Case $\langle \vec{R}, P, T\rangle : \alpha$. Let $\vec{R} \models \Delta$. By Definition 4.2.20, Lemma 4.1.2, and Definition 4.1.4 we get

$$\begin{array}{rcl}
|(\!|\langle \vec{R}, P, T\rangle|\!)_\alpha| & = & |(\lambda x_{i_1} \ldots x_{i_n}.(\!|P, T|\!)_\alpha^\Delta) \cdot (\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}} \cdots (\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}|\\
& = & |(\!|P, T|\!)_\alpha^\Delta|_{\vec{x}} \mathbin{@} [\#|(\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}}|, \ldots, \#|(\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}|]
\end{array}$$

By IH1, for all $k \in \mathrm{dom}(\Delta)$, we have $\#|(\!|\#^{-1}(!R_k)|\!)_{\beta_k}| \simeq_\alpha !R_k$. Then by reflexivity,

$$\begin{array}{l}
|(\!|P, T|\!)_\alpha^\Delta|_{\vec{x}} \mathbin{@} [\#|(\!|\#^{-1}(!R_{i_1})|\!)_{\beta_{i_1}}|, \ldots, \#|(\!|\#^{-1}(!R_{i_n})|\!)_{\beta_{i_n}}|]\\
\equiv_\alpha |(\!|P, T|\!)_\alpha^\Delta|_{\vec{x}} \mathbin{@} [!R_{i_1}, \ldots, !R_{i_n}]
\end{array}$$

Conclude by IH2, Lemma 4.2.11(3), and transitivity.

- Case $\Delta \Vdash^r (\mathtt{Load}\ k; P, [\,]) : \beta \to \alpha$. There are two subcases.

  - Subcase $k \notin \mathrm{dom}(\Delta)$. Let $b, c \in \mathcal{D}_\beta$ such that $b \simeq_\beta c$. We get

$$\begin{array}{lcll}
& & |(\!|\mathtt{Load}\ k; P, [\,]|\!)_{\beta\to\alpha}^\Delta|_{\vec{x}} \mathbin{@} [\vec{a}, b] & \\
& = & |\lambda x_k.(\!|P, [\,]|\!)_\alpha^{\Delta, k:\beta}|_{\vec{x}} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 4.2.20,}\\
& = & |(\!|P, [\,]|\!)_\alpha^{\Delta, k:\beta}|_{\vec{x}, x_k} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 4.1.4,}\\
& \equiv_\alpha & \langle \vec{R}^r_{\vec{a}, b}, P, [\,]\rangle, & \text{by IH2,}\\
& {}_{\mathsf{c}}\!\!\leftarrow & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [b]\rangle, & \text{by Definition 3.3.11,}\\
& \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, \mathtt{Load}\ k; P, [c]\rangle, & \text{by reflexivity.}
\end{array}$$

    Conclude by Lemma 4.2.11(3) and transitivity.

  - Subcase $k \in \mathrm{dom}(\Delta)$. Let $k = i_m$, and let $b, c \in \mathcal{D}_\beta$ such that $b \simeq_\beta c$. We also fix

$\vec{x'} = x_{i_1}, \ldots, x_{i_{m-1}}, x_{i_{m+1}}, \ldots, x_{i_n}$ and $\vec{a'} = a_{i_1}, \ldots, a_{i_{m-1}}, a_{i_{m+1}}, \ldots, a_{i_n}$. We get

$$\begin{array}{lll}
& |(\!|\texttt{Load}\ k; P, [\,]|\!)^{\Delta}_{\beta\to\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}, b] & \\
= & |\lambda x_k.(\!|P, [\,]|\!)^{\Delta[k:\beta]}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}, b]\,, & \text{by Definition 4.2.20,} \\
\equiv_\alpha & |\lambda x_k.(\!|P, [\,]|\!)^{\Delta[k:\beta]}_{\alpha}|_{\vec{x'}} \mathbin{@} [\vec{a'}, b]\,, & \text{by Lemma 4.2.14,} \\
= & |(\!|P, [\,]|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x'},x_k} \mathbin{@} [\vec{a'}, b]\,, & \text{by Definition 4.1.4,} \\
\equiv_\alpha & \langle \vec{R}^r_{\vec{a'},b}, P, [\,]\rangle, & \text{by IH2,} \\
{}_{\mathsf{c}}\!\!\twoheadleftarrow & \langle \vec{R}^r_{\vec{a}}, \texttt{Load}\ k; P, [b]\rangle, & \text{by Definition 3.3.11,} \\
\equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, \texttt{Load}\ k; P, [c]\rangle, & \text{by reflexivity.}
\end{array}$$

Conclude by Lemma 4.2.11(3) and transitivity.

In the cases following, we assume that $k \notin \mathrm{dom}(\Delta)$. If $k \in \mathrm{dom}(\Delta)$, one proceeds as above.

- Case $\Delta \Vdash^r (\texttt{Load}\ k; P, b :: T) : \alpha$. By Definition 4.2.20, Lemma 4.1.2, and Definition 4.1.4, we get

$$\begin{array}{lll}
& |(\!|\texttt{Load}\ k; P, b :: T|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] & \\
= & |(\lambda x_k.(\!|P, T|\!)^{\Delta,k:\beta}_{\alpha}) \cdot (\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}] & \\
= & \mathsf{Apply}^2_n \mathbin{@} [\#\mathsf{Pr}^1_1, \#|(\!|P, T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k}, \#|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}}, \vec{a}] & \\
\twoheadrightarrow_{\mathsf{c}} & |(\!|P, T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,. &
\end{array}$$

By Proposition 4.2.22, $\vdash (\!|\#^{-1}(b)|\!)_\beta : \beta$, so $\#|(\!|\#^{-1}(b)|\!)_\beta| \mathbin{@} \vec{a} \simeq_\alpha b$ by Corollary 4.2.15 and IH1. By reflexivity, we obtain

$$|(\!|P, T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\#^{-1}(b)|\!)_\beta|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] \equiv_\alpha |(\!|P, T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, b]\,.$$

Conclude by IH2, Lemma 4.2.11(3), and transitivity.

- Case $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \texttt{Pred}(i_m); P, T) : \alpha$. Fix the notation $\mathsf{M} = |(\lambda x_k.(\!|P, T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}) \cdot (\mathbf{pred}\, x_{i_m})|_{\vec{x}}$. By Definition 4.2.20 and Lemma 4.1.2 we get

$$\begin{array}{lll}
& |(\!|k \leftarrow \texttt{Pred}(i_m); P, T|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] & \\
= & |\texttt{Ifc}(x_{i_m}, (\lambda x_k.(\!|P, T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}) \cdot (\mathbf{pred}\, x_{i_m}))|_{\vec{x}} \mathbin{@} [\vec{a}] & \\
= & \mathsf{Apply}^3_n \mathbin{@} [\#\mathsf{Ifz}, \#\mathsf{Pr}^n_{i_m}, \#\mathsf{M}, \#\mathsf{M}, \vec{a}] & \\
\twoheadrightarrow_{\mathsf{c}} & \mathsf{Ifz} \mathbin{@} [\#(\mathsf{Pr}^n_{i_m} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}] & \\
\equiv_\alpha & \mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}]\,. &
\end{array}$$

There are two subcases from this point.

  - Subcase $\#^{-1}(a_{i_m})$ does not terminate. Then, by Definition 3.3.11, we have that the machine $\mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}]$ and $\langle \vec{R}^r_{\vec{a}}, l \leftarrow \texttt{Pred}(k); P, T\rangle$ cannot

terminate either. By Definition 3.3.16, we have $\langle \vec{R}^r_{\vec{a}}, l \leftarrow \texttt{Pred}(k); P, T\rangle : \alpha$. Conclude by Lemma 4.2.11(3) and transitivity.

– Subcase $\#^{-1}(a_{i_m}) \twoheadrightarrow_{\mathsf{c}} \mathsf{t} = \#^{-1}(t)$, for some $t \in \mathbb{N}$. Now, easy calculations give $|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}] \twoheadrightarrow_{\mathsf{c}} \mathsf{t}' := \#^{-1}(t \ominus 1)$, from which $|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\mathsf{int}} \mathsf{t}'$ follows by Lemma 4.2.11(3). Then we get

$$
\begin{array}{lll}
 & \mathsf{Ifz} \mathbin{@} [a_{i_m}, \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \#(\mathsf{M} \mathbin{@} [\vec{a}]\,), \vec{a}] & \\
\twoheadrightarrow_{\mathsf{c}} & |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}) \cdot (\mathbf{pred}\, x_{i_m})|_{\vec{x}} \mathbin{@} [\vec{a}]\,, & \text{by Lem. 4.1.2(6),} \\
= & \mathsf{Apply}^2_n \mathbin{@} \left[\begin{array}{l} \#\mathsf{Pr}^1_1, \#|(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}|_{\vec{x},x_k}, \\ \#|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}}, \vec{a} \end{array}\right], & \text{by Def. 4.1.4,} \\
\twoheadrightarrow_{\mathsf{c}} & |(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|(\!|\mathbf{pred}\, x_{i_m}|\!)^{\Delta}_{\mathsf{int}}|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]\,, & \text{by Lem. 4.1.2(2),} \\
\equiv_{\alpha} & |(\!|P,T|\!)^{\Delta,k:\mathsf{int}}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, t \ominus 1, 0)]\,, & \text{by reflexivity,} \\
\equiv_{\alpha} & \langle \vec{R}^r_{\vec{a}}[R_k := t \ominus 1], P, T\rangle, & \text{by IH2,} \\
\equiv_{\alpha} & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \texttt{Pred}(i_m); P, T\rangle, & \text{by Lem. 4.2.11(3).}
\end{array}
$$

Conclude by Lemma 4.2.11(3) and transitivity.

- Case $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \texttt{Succ}(i_m); P, T) : \alpha$. Analogous.

- Case $\Delta \Vdash^r (k \leftarrow \texttt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T) : \alpha$, where $\Delta(l) = \mathsf{int}$, $\Delta(i_{m_1}) = \beta$, $\Delta(i_{m_2}) = \beta$. There are two subcases.

  – Subcase $\#^{-1}(a_{i_l})$ does not terminate. Proceed as in the case for $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \texttt{Pred}(i_m); P, T) : \alpha$.

  – Subcase $\#^{-1}(a_{i_l}) \twoheadrightarrow_{\mathsf{c}} \#^{-1}(t), t \in \mathbb{N}$. Proceed as in the case for $x_{i_1} : \beta_{i_1}, \ldots, x_{i_m} : \mathsf{int}, \ldots, x_n : \beta_{i_n} \Vdash^r (k \leftarrow \texttt{Pred}(i_m); P, T) : \alpha$ to get

$$
\begin{array}{c}
|(\!|k \leftarrow \texttt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] \\
\equiv_{\alpha} \\
|(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|\vec{x}, x_k \mathbin{@} [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}]\,)]
\end{array}
$$

  There are two subcases.

  * Subcase $t = 0$. We have $|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_{\beta} \#^{-1}(a_{i_{m_1}})$ by applying Lemma 4.2.11(3). Then we get

$$
\begin{array}{lll}
 & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] & \\
\equiv_{\alpha} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, a_{i_{m_1}}]\,, & \text{by reflexivity,} \\
\equiv_{\alpha} & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_{m_1}}], P, T\rangle, & \text{by IH2,} \\
\equiv_{\alpha} & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \texttt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T\rangle, & \text{by Lemma 4.2.11(3).}
\end{array}
$$

  Conclude by Lemma 4.2.11(3) and transitivity.

∗ Subcase $t > 0$. We have $|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_\beta \#^{-1}(a_{i_{m_2}})$ by applying Lemma 4.2.11(3). Then we get

$$\begin{array}{lll} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|\mathbf{ifz}(x_{i_l}, x_{i_{m_1}}, x_{i_{m_2}})|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] & \\ \equiv_\alpha & |(\!|P,T|\!)^{\Delta,k:\beta}_{\alpha}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, a_{i_{m_2}}]\,, & \text{by reflexivity,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_{m_2}}], P, T\rangle, & \text{by IH2,} \\ \equiv_\alpha & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{Test}(i_l,\, i_{m_1},\, i_{m_2}); P, T\rangle, & \text{by Lemma 4.2.11(3).} \end{array}$$

Conclude by Lemma 4.2.11(3) and transitivity.

- Case $\Delta \Vdash^r (k \leftarrow \mathtt{App}(i_l,\, i_m); P, T) : \gamma$, where $\Delta(i_l) = \beta \to \alpha$ and $\Delta(i_m) = \beta$. By Definition 4.2.20, Definition 4.1.4, and Lemma 4.1.2(2), we get

$$\begin{array}{lll} & |(\!|m \leftarrow \mathtt{App}(i_l,\, i_m); P, T|\!)^{\Delta}_{\gamma}|_{\vec{x}} \mathbin{@} [\vec{a}] & \\ = & |(\lambda x_k.(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}) \cdot (x_{i_l} \cdot x_{i_m})|_{\vec{x}} \mathbin{@} [\vec{a}] & \\ = & \mathsf{Apply}^2_n \mathbin{@} [\#\mathsf{Pr}^1_1, \#|(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k}, \#|x_{i_l} \cdot x_{i_m}|_{\vec{x}}, \vec{a}] & \\ \twoheadrightarrow_{\mathsf{c}} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|x_{i_l} \cdot x_{i_m}|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] & \end{array}$$

By Lemma 4.2.11(3) we have $\#\mathsf{Pr}^n_{i_m} \mathbin{@} [\vec{a}] \equiv_\beta \#^{-1}(a_{i_m})$. We then have

$$\begin{array}{lll} |x_{i_l} \cdot x_{i_m}|_{\vec{x}} \mathbin{@} [\vec{a}] & = & \mathsf{Apply}^2_n \mathbin{@} [\#\mathsf{Pr}^1_1, \#\mathsf{Pr}^n_{i_l}, \#\mathsf{Pr}^n_{i_m}, \vec{a}]\,, \quad \text{by Definition 4.1.4,} \\ & \twoheadrightarrow_{\mathsf{c}} & \mathsf{Pr}^n_{i_l} \mathbin{@} [\vec{a}, \#(\#\mathsf{Pr}^n_{i_m} \mathbin{@} [\vec{a}]\,)]\,, \quad \text{by Lemma 4.1.2(2),} \\ & \equiv_\alpha & \mathsf{Pr}^n_{i_l} \mathbin{@} [\vec{a}, a_{i_m}]\,, \quad \text{by reflexivity,} \\ & \equiv_\alpha & \#^{-1}(a_{i_l}) \mathbin{@} [a_{i_m}]\,, \quad \text{by Lemma 4.2.11(3).} \end{array}$$

Thus we get

$$\begin{array}{lll} & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, \#(|x_{i_l} \cdot x_{i_m}|_{\vec{x}} \mathbin{@} [\vec{a}]\,)] & \\ \equiv_\gamma & |(\!|P,T|\!)^{\Delta,k:\beta}_{\gamma}|_{\vec{x},x_k} \mathbin{@} [\vec{a}, a_{i_l} \cdot a_{i_m}]\,, & \text{by reflexivity,} \\ \equiv_\gamma & \langle \vec{R}^r_{\vec{a}}[R_k := a_{i_l} \cdot a_{i_m}], P, T\rangle, & \text{by IH2,} \\ \equiv_\gamma & \langle \vec{R}^r_{\vec{a}}, k \leftarrow \mathtt{App}(i_l,\, i_m); P, T\rangle, & \text{by Lemma 4.2.11(3).} \end{array}$$

Conclude by Lemma 4.2.11(3) and transitivity.

- Case $\Delta \Vdash^r (\mathtt{Call}\ k, [b_1, \ldots, b_m]) : \alpha$, where we have $\Delta(k) = \gamma_1 \to \cdots \to \gamma_m \to \alpha$ and for all $0 < j \leq m$, $b_j \in \mathcal{D}_{\gamma_j}$. Let $\vec{b} = b_1, \ldots, b_m$. By Definition 4.2.20 we get $|(\!|\mathtt{Call}\ k, [\vec{b}]|\!)^{\Delta}_{\alpha}|_{\vec{x}} \mathbin{@} [\vec{a}] = |x_k \cdot (\!|\#^{-1}(b_1)|\!)_{\gamma_1} \cdots (\!|\#^{-1}(b_m)|\!)_{\gamma_m}|_{\vec{x}} \mathbin{@} [\vec{a}]$. By an easy induction on m, one shows the following:

$$|x_k \cdot (\!|\#^{-1}(b_1)|\!)_{\gamma_1} \cdots (\!|\#^{-1}(b_m)|\!)_{\gamma_m}|_{\vec{x}} \mathbin{@} [\vec{a}] \equiv_\alpha |x_k|_{\vec{x}} \mathbin{@} [\vec{a}, \vec{b}] \text{ (cf. Lemma 4.2.13).}$$

By Definition 4.1.4 and Lemma 4.1.2(1), $|x_k|_{\vec{x}} \mathbin{@} [\vec{a}, \vec{b}] = \mathsf{Pr}^n_k \mathbin{@} [\vec{a}, \vec{b}] \twoheadrightarrow_{\mathsf{c}} \#^{-1}(a_k) \mathbin{@} [\vec{b}]$. Conclude by Lemma 4.2.11(3) and transitivity, as $\langle \vec{R}^r_{\vec{a}}, \mathtt{Call}\ k, [\vec{b}]\rangle \to_{\mathsf{c}} \#^{-1}(a_k) \mathbin{@} [\vec{b}]$. □

Corollary 4.2.24.
For all $a \in \mathcal{D}_\alpha$, there is a PCF *program $\vdash P_a : \alpha$ such that $|P_a| \equiv_\alpha \#^{-1}(a)$.*

Theorem 4.2.25 (Completeness)**.**
Given two PCF *programs P_1, P_2 of type α, we have*

$$P_1 \equiv_{\text{obs}} P_2 \Rightarrow |P_1| \equiv_\alpha |P_2|$$

Proof. Assume $P_1 \equiv_{\text{obs}} P_2$. Let $\alpha{=}\alpha_1 \to \cdots \to \alpha_n \to \mathsf{int}$ and $a_1, b_1 \in \mathcal{D}_{\alpha_1}, \ldots, a_n, b_n \in \mathcal{D}_{\alpha_n}$ be such that $\#^{-1}(a_i) \equiv_{\alpha_i} \#^{-1}(b_i)$. By Corollary 4.2.24, there are PCF programs $Q_1, \ldots, Q_n$, and $Q'_1, \ldots, Q'_n$ such that Q_i, Q'_i have type α_i and both $\#^{-1}(a_i) \equiv_{\alpha_i} |Q_i|$ and $\#^{-1}(b_i) \equiv_{\alpha_i} |Q'_i|$ hold, for every such i. We get

$$\begin{array}{rcll}
|P_1| \mathbin{@} [a_1, \ldots, a_n] & \equiv_{\mathsf{int}} & |P_1| \mathbin{@} [\#|Q_1|, \ldots, \#|Q_n|]\,, & \text{as } |P_1| \equiv_\alpha |P_1| \text{ (Corollary 4.2.12),} \\
& \equiv_{\mathsf{int}} & |P_1 \cdot Q_1 \cdots Q_n|, & \text{by Lemma 4.2.13,} \\
|P_1 \cdot Q_1 \cdots Q_n| \to_{\mathsf{c}} \mathsf{k} & \Leftrightarrow & P_1 \cdot Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{k}, & \text{by Theorem 4.1.11,} \\
& \Leftrightarrow & P_2 \cdot Q_1 \cdots Q_n \twoheadrightarrow_{\mathsf{PCF}} \underline{k}, & \text{as } P_1 \equiv_{\text{obs}} P_2, \\
& \Leftrightarrow & |P_2 \cdot Q_1 \cdots Q_n| \to_{\mathsf{c}} \mathsf{k}, & \text{by Theorem 4.1.11,} \\
|P_2 \cdot Q_1 \cdots Q_n| & \equiv_{\mathsf{int}} & |P_2| \mathbin{@} [\#|Q_1|, \ldots, \#|Q_n|]\,, & \text{by Lemma 4.2.13,} \\
& \equiv_{\mathsf{int}} & |P_2| \mathbin{@} [b_1, \ldots, b_n]\,, & \text{as } |P_2| \equiv_\alpha |P_2| \text{ (Corollary 4.2.12).}
\end{array}$$

Conclude by transitivity (Lemma 4.2.114.2.12). □

Adequacy and completeness together yield full abstraction.

Theorem 4.2.26 (Full Abstraction)**.**
The model $\mathscr{D}$ is fully abstract for PCF.

Proof. By Definition 4.2.17, the interpretation in $\mathscr{D}$ is defined by $[\![P]\!]^\alpha = [\#|P|]_{\simeq_\alpha}$. Therefore, the full abstraction property follows directly from Theorems 4.2.19 and 4.2.25. □

Chapter 5

Categorical Interlude

In the previous chapters, we have focused on fully abstract models for PCF. We will now move on to models which, while not being fully abstract, are still of interest. Due to the constraints required of fully abstract models (they may not contain "junk"), there are many (quantitative) aspects of program behaviour which are unable to be analysed by them – runtime, i.e. the number of reduction steps, to name but an example. We will be presenting and analysing a particular group of categorical models. A *categorical model* is a model created from a *category* taken from the field of *category theory*. Thus, we will need to begin with an introduction to category theory.

5.1 Category Theory Preliminaries

Category theory is unto categories as set theory is unto sets. Whereas set theory focuses on the study of objects, category theory focuses on the study of arrows – directed connections between objects. The traditional way of presenting categories is by using set theory as a foundation. We will encounter issues related to sizing in the definition - for a (brief) discussion on this, ZFC set theory, and the definition of a *class*, please see the appendix. The sources used for the basic definitions are [EGRS04, BW95, ML78, Com].

Definition 5.1.1 (Categories)**.**
A category $\mathbb{C}$ *consists of*

- *a class of* objects $\mathrm{Ob}(\mathbb{C})$*;*
- *for every pair of objects* $A, B \in \mathrm{Ob}(\mathbb{C})$*, a class of* arrows *from* A *to* B*:* $\mathbb{C}(A, B)$*. These arrows are often depicted graphically; if* $\varphi \in \mathbb{C}(A, B)$*, then we also depict it as*

$$A \xrightarrow{\varphi} B$$

We also require the following to be satisfied:

- *for every object $A \in \mathrm{Ob}(\mathbb{C})$, there must be an* identity arrow $\mathrm{id}_A \in \mathbb{C}(A, A)$*;*
- *for every pair of arrows $f \in \mathbb{C}(A, B), g \in \mathbb{C}(B, C)$, there must be a composite arrow $f; g \in \mathbb{C}(A, C)$, also written $g \circ f$;*
- *composition must be associative: for every triple $f \in \mathbb{C}(A, B), g \in \mathbb{C}(B, C), h \in \mathbb{C}(C, D)$ we must have $(f; g); h = f; (g; h)$;*
- *the identity arrows must satisfy left and right unit laws: for every $f \in \mathbb{C}(A, B)$, we must have $\mathrm{id}_A; f = f = f; \mathrm{id}_B$.*

We generally write $a \in \mathbb{C}$ to mean $a \in \mathrm{Ob}(\mathbb{C})$, writing $\varphi \in \mathbb{C}(A, B)$ when referring to elements of an arrow class rather than elements of the objects class.

Definition 5.1.2 (Monoid)**.**
A monoid *is a triple $(M, \cdot^M, 1^M)$ consisting of:*

- *A set M;*
- *A binary operator $\cdot^M : M \times M \to M$;*
- *An identity element 1^M;*

such that the binary operator is associative, and the identity element acts as a left and right unit for the operator. If the binary operator is also commutative, then the monoid is a commutative monoid*.*

Example 5.1.3.
The following are some common examples of categories:

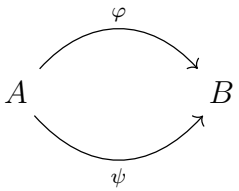
- *The category* $\mathbf{Set}$*, whose objects are (small) sets, and whose arrows are functions. Composition is given by function composition, and the identity arrows are identity functions.*
- *The category* $\mathbf{Rel}$*, whose objects are (small) sets, and whose arrows are binary relations – for $A, B \in \mathbf{Rel}$, $\mathbf{Rel}(A, B) = \mathcal{P}(A \times B)$ the powerset of $A \times B$. Intuitively, a relation is a set of "valid" pairings between elements of two sets. One can also see a relation as a matrix where A indexes the columns, B the rows, and with elements populated by "true" if a pairing is valid and "false" otherwise. Using this perspective, composition in* $\mathbf{Rel}$ *is given by matrix multiplication of the underlying boolean semiring*[1]*, with the identities are given by the identity matrix.*
- *The category* $\mathbf{Poset}$*, whose objects are partially ordered sets, and whose arrows are monotone (order preserving/reversing) maps.*[2]

[1] Semirings and the boolean semiring will be defined later.
[2] If the reader is unfamiliar with partially ordered sets, a definition is given later in Definition 5.1.31.

- *The category* **CMonoid***, whose objects are commutative monoids and whose arrows are monoid homomorphisms.*

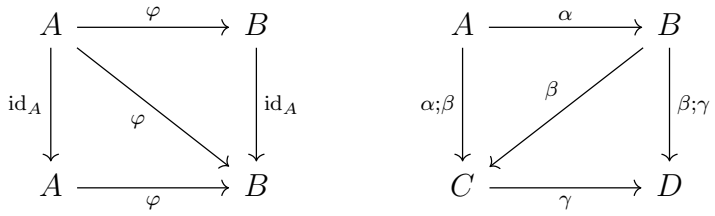
- *A category can trivially be constructed from any monoid by considering a single element set as the set of objects, the objects of the monoid as the set of arrows, and composition given by the monoid operator.*

Definition 5.1.4 (Categorical Diagrams)**.**
A categorical diagram is a graphical representation of morphisms in a category. Let us assume the presence of two morphisms $\varphi, \psi \in \mathbb{C}(A, B)$, and the diagram



We say that the diagram commutes *if $\varphi = \psi$. This convention is extended in multiple ways; φ and ψ could be the composite of other arrows, and multiple equalities can be expressed in the same diagram. If we wish to name a particular diagram, then we write the name in the centre of the diagram. The identity arrow is often replaced by a double line similar to $=$ to increase readability.*

Example 5.1.5.
The following diagrams commute as a result of the definition of a category:



The first diagram presents the left and right unit laws. The second diagram is unnamed and simultaneously presents composition and associativity of composition.

Definition 5.1.6 (Product Category)**.**
Much like sets, one can define a category from the cartesian product of two categories. Given categories $\mathbb{C}, \mathbb{D}$, the category $\mathbb{C} \times \mathbb{D}$ is defined with:

- $\mathrm{Ob}(\mathbb{C} \times \mathbb{D}) = \{(a, b) | a \in \mathbb{C}, b \in D\}$;

- $\mathbb{C} \times \mathbb{D}(A \times B, A' \times B') = \{(\varphi, \psi) | \varphi \in \mathbb{C}(A, A'), \psi \in \mathbb{D}(B, B')\}$.

Composition, identities, and associativity are inherited from $\mathbb{C}$ and $\mathbb{D}$.

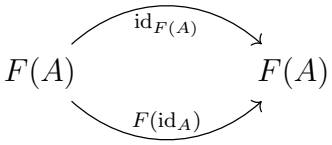
The cartesian product for categories is an example of a trend in category theory called *categorification*. It is common for notions from other branches of mathematics to be "lifted" into a categorical setting. The original notion is then treated as a particular example of the categorified notion. Often notions can be lifted into the categorical setting in multiple different ways to very different outcomes - we will see some examples of this later on.

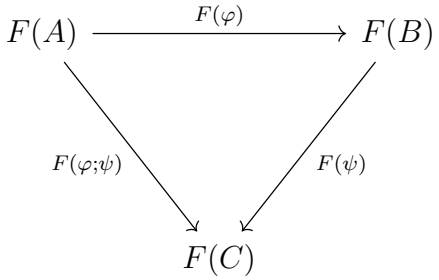
Definition 5.1.7 (Opposite Category)**.**
Given a category $\mathbb{C}$, the opposite category $\mathbb{C}^{\mathsf{op}}$ is defined with $\mathrm{Ob}(\mathbb{C}^{\mathsf{op}}) = \mathrm{Ob}(\mathbb{C})$, and the arrows swapped $\mathbb{C}^{\mathsf{op}}(A, B) = \mathbb{C}(B, A)$. Composition and identities are the reversed form of the corresponding arrows in $\mathbb{C}$.

Definition 5.1.8 (Functor)**.**
A functor is an arrow between categories which preserves their structure. That is to say, given categories $\mathbb{C}, \mathbb{D}$ a functor $F : \mathbb{C} \to \mathbb{D}$ behaves as follows:

- *For every $A \in \mathrm{Ob}(\mathbb{C})$, $F(A) \in \mathrm{Ob}(D)$;*
- *For every $\varphi \in \mathbb{C}(A, B)$, $F(\varphi) \in \mathbb{D}(F(A), F(B))$;*

such that F preserves composition and identities, i.e.





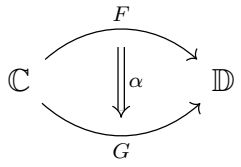
Unless it causes clarity issues, we typically omit the brackets when discussing functors applied to objects, i.e. $FA = F(A)$. Note that functors can always be composed to form another functor.

Example 5.1.9.
A common example of functors are so-called "forgetful functors" which discard structure present in one category. For example, there exists a forgetful functor from **Poset** *to* **Set** *which simply discards all orderings.*

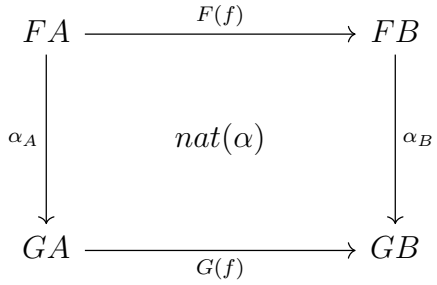
Definition 5.1.10 (Endofunctor)**.**
An endofunctor *is a functor from a category to itself.*

Definition 5.1.11 (Contravariant Functor)**.**
A contravariant functor *$F : \mathbb{C} \to \mathbb{D}$ is a functor from $\mathbb{C}^{\mathsf{op}}$ to $\mathbb{D}$.*

Definition 5.1.12 (Natural Transformations)**.**
A natural transformation is an arrow between functors which preserves their structure. Given functors $F, G : \mathbb{C} \to \mathbb{D}$, a natural transformation $\alpha : F \to G$, denoted



consists of an assignment to every $A \in \mathbb{C}$ of an arrow $\alpha_A \in \mathbb{D}(FA, GA)$, such that for every morphism $f \in \mathbb{C}(A, B)$, the following diagram commutes (in $\mathbb{D}$):



When defining natural transformations it is common to write "$\alpha_A : FA \to GA$ is a natural transformation" or even "$\alpha_A : FA \to GA$ is natural," rather than the full definition denoted above. Composition of natural transformations makes use of composition in $\mathbb{D}$.

Definition 5.1.13 (Adjoint Functors)**.**
Consider two categories $\mathbb{C}, \mathbb{D}$ and two functors $F : \mathbb{C} \to \mathbb{D}, G : \mathbb{D} \to \mathbb{C}$. If we have the following:

- *A natural transformation $\eta_A : A \to GFA$ called the* unit*;*
- *A natural transformation $\epsilon_A : FGA \to A$ called the* counit*;*
- *The following diagrams commute (in $\mathbb{D}$ and $\mathbb{C}$ respectively):*

Then we say that F and G are adjoint functors*, with F being the* left adjoint *to G and G the* right adjoint *to F. This is written $F \dashv G$, or alternatively*

$$\mathbb{D} \underset{G}{\overset{F}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbb{C}$$

Example 5.1.14.
Many of the forgetful functors mentioned in Example 5.1.9 have left adjoints – such functors

are called "free functors", and they introduce some additional structure in the simplest manner possible. The forgetful functor from $\mathbf{Poset}$ *to* $\mathbf{Set}$ *has such a left adjoint: It takes a set* $A \in \mathbf{Set}$ *to the poset* $(A, \leq)$ *where for all* $a, b \in A$, $a \leq b \Rightarrow a = b$, *and leaves functions unchanged. Both the unit and counit are then the identity functions in the corresponding categories.*

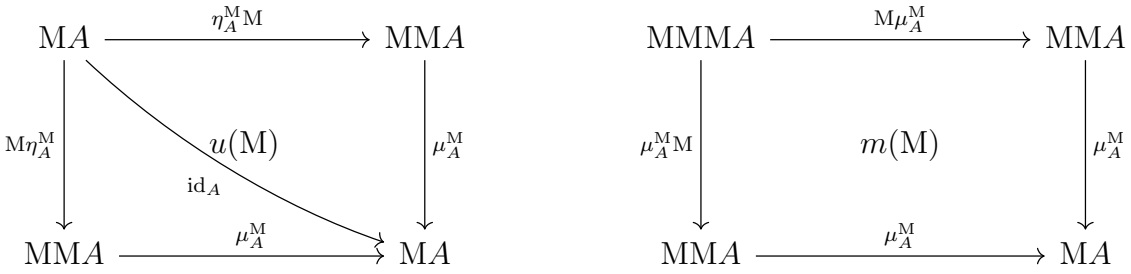
Definition 5.1.15 (Isomorphisms)**.**
An isomorphism is the categorification of the notion of a bijection. Given an arrow $f : A \to B$ *in a category* $\mathbb{C}$, *we say that* f *is an isomorphism if there also exists an arrow* $f^{-1} : B \to A$ *such that* $f; f^{-1} = \mathrm{id}_A$ *and* $f^{-1}; f = \mathrm{id}_B$.

If f *is a natural transformation, then it is called a* natural isomorphism*, written* $f : A \simeq B$.

Definition 5.1.16 (Monads)**.**
A monad on a category $\mathbb{C}$ *consists of:*

- *An endofunctor* $\mathrm{M} : \mathbb{C} \to \mathbb{C}$
- *A natural transformation* $\eta^{\mathrm{M}}_A : A \to \mathrm{M}A$ *called the* unit*;*
- *A natural transformation* $\mu^{\mathrm{M}}_A : \mathrm{MM}A \to \mathrm{M}A$ *called* multiplication*;*

such that the following diagrams commute:



We can also obtain the dual notion of a *comonad* by reversing the arrows.

Definition 5.1.17.
A comonad *on* $\mathbb{C}$ *is a monad on* $\mathbb{C}^{\mathsf{op}}$.

Monads can be seen as a form of categorifying monoids. The diagrams can be seen as providing the left and right unit and associativity respectively. Monads are particularly interesting due to their close relationship with adjoint functors:

Proposition 5.1.18.
Let $F \dashv G$ *be a pair of adjoint functors where* $F : \mathbb{C} \to \mathbb{D}$, *with unit* η_A *and counit* ϵ_A. *Then* $F; G$ *is a monad on* $\mathbb{C}$ *and* $G; F$ *is a comonad on* $\mathbb{D}$.

The unit of the adjunction is then the same as the unit of the monad $F;G$, hence they are given the same name.

The converse also holds, in a way. What is important for the converse is the choice of $\mathbb{D}$, as it is not given. One possibility is the so-called *Kleisli* Category of a monad.

Definition 5.1.19 (Kleisli Category)**.**
Given a monad M *on a category* $\mathbb{C}$*, the* Kleisli category $\mathbb{C}_{\mathrm{M}}$ *of* M *on* $\mathbb{C}$ *is defined as follows:*

- *The objects of* $\mathbb{C}_{\mathrm{M}}$ *are the same as the objects of* $\mathbb{C}$*;*
- *The morphisms of* $\mathbb{C}_{\mathrm{M}}$ *are of the form* $A \to \mathrm{M}B$ *in* $\mathbb{C}$*, i.e.* $\mathbb{C}_{\mathrm{M}}(A,B) = \mathbb{C}(A, \mathrm{M}B)$.
- *Composition of morphisms* $f : A \to \mathrm{M}B, g : B \to \mathrm{M}C$ *in the Kleisli category are defined using* μ^{M}_C*:*
$$g \circ^{\mathbb{C}_{\mathrm{M}}} f = \mu^{\mathrm{M}}_C \circ^{\mathbb{C}} \mathrm{M}g \circ^{\mathbb{C}} f$$
- *The identity is given by* η^{M}_A.

When $\mathbb{C}$ *and* M *are both clearly stated, we will often write morphisms* $f : A \to \mathrm{M}B$ *as* $f : A \longrightarrow\!\!\!\!\!\bullet\; B$ *. In other words, the two diagrams below are treated as one and the same.*

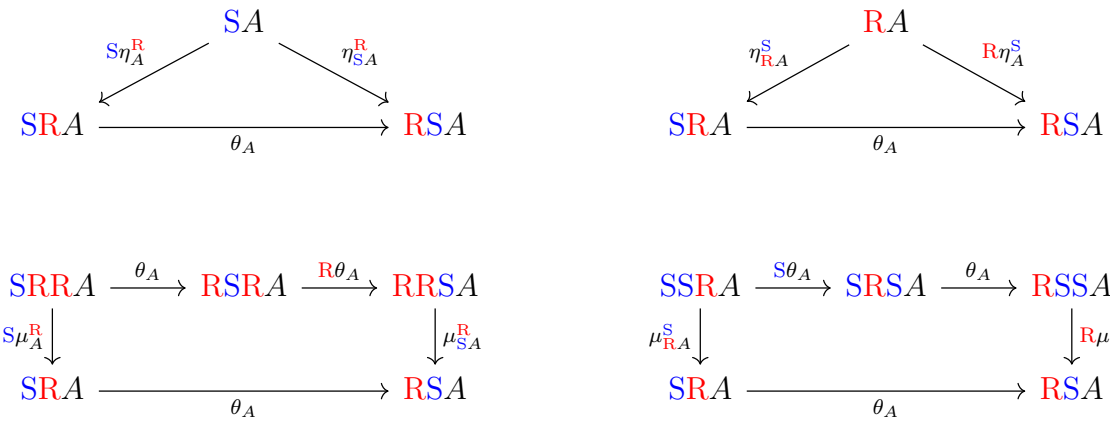
$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle u} \\ C & \xrightarrow[h]{} & D \end{array} \qquad\qquad \begin{array}{ccccc} A & & \xrightarrow{\qquad\qquad f \qquad\qquad} & & \mathrm{M}B \\ {\scriptstyle g}\downarrow & & & & \downarrow{\scriptstyle \mathrm{M}u} \\ \mathrm{M}C & \xrightarrow[\mathrm{M}h]{} & \mathrm{MM}D & \xrightarrow[\mu^{\mathrm{M}}_D]{} & \mathrm{M}D \end{array}$$

Proposition 5.1.20.
Let $\mathrm{M} : \mathbb{C} \to \mathbb{C}$ *be a monad. Then* $\mathrm{M} = F;G$ *for some functors* $F : \mathbb{C} \to \mathbb{C}_{\mathrm{M}}, G : \mathbb{C}_{\mathrm{M}} \to \mathbb{C}$*, where* $F \dashv G$.

Kleisli categories are not the only categories for which the above property holds, but they are certainly the simplest. We also have co-Kleisli categories which are to comonads as Kleisli categories are to monads.

Definition 5.1.21.
Given a comonad M *on a category* $\mathbb{C}$*, the* co-Kleisli category *of* M *on* $\mathbb{C}$ *is the category* $((\mathbb{C}^{\mathsf{op}})_{\mathrm{M}})^{\mathsf{op}}$.

Proposition 5.1.22 (Distribution of Monads)**.**
Given two monads R, S*, the composite functor* RS *can also form a monad if a distributive law* $\theta : \mathrm{SR} \to \mathrm{RS}$ *can be defined such that* θ *is a natural transformation and the following diagrams commute:*



In this case, $\eta^{\mathrm{RS}}_A = \eta^{\mathrm{S}}_A ; \eta^{\mathrm{R}}_A$ *and* $\mu^{\mathrm{RS}}_A = \mathrm{R}\theta_{\mathrm{S}A} ; \mu^{\mathrm{R}}_A ; \mathrm{R}\mu^{\mathrm{S}}_A$*.*

Proposition 5.1.23.
Given two monads R, S *on a category* $\mathbb{C}$ *such that* RS *is also a monad, then* $\mathrm{S}(-) ; \theta$ *is a monad on* $\mathbb{C}_{\mathrm{R}}$*.*

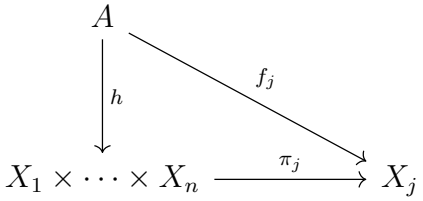
5.1.1 Modelling PCF Using Categories

With the basic notions introduced, we move on to the notions necessary for modelling PCF. We must first discuss the categories used to model the simply typed λ-calculus, and then we can further narrow down the requirements placed upon such categories until we can model PCF. To model the simply typed λ-calculus, we need to represent abstraction and application in categories.

Definition 5.1.24 (Finite Products and Coproducts)**.**
A category $\mathbb{C}$ *has* finite products *if for every finite list* $X_1, \ldots, X_n$ *of objects of* $\mathbb{C}$*, there is an object* $X_1 \times \cdots \times X_n \in \mathbb{C}$ *and, for* $1 \leq i \leq n$*, there is a map*

$$\pi_i \in \mathbb{C}(X_1 \times \cdots \times X_n, X_i)$$

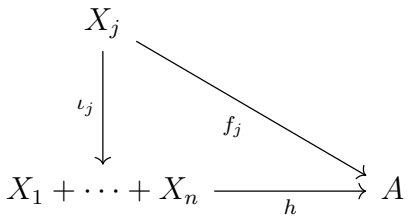
such that for every object $A \in \mathbb{C}$ *and every family of morphisms* $f_i \in \mathbb{C}(A, X_i), i \leq n$*, there is a unique morphism* $h \in \mathbb{C}(A, X_1 \times \cdots \times X_n)$ *such that for* $1 \leq j \leq n$*, the following diagram commutes:*



A category $\mathbb{C}$ *has* finite coproducts *if* $\mathbb{C}^{\mathsf{op}}$ *has finite products. Spelled out, this means that for every finite list* $X_1, \ldots, X_n$ *of objects of* $\mathbb{C}$*, there is an object* $X_1 + \cdots + X_n \in \mathbb{C}$ *and, for* $1 \leq i \leq n$*, there is a map*

$$\iota_i \in \mathbb{C}(X_i, X_1 + \cdots + X_n)$$

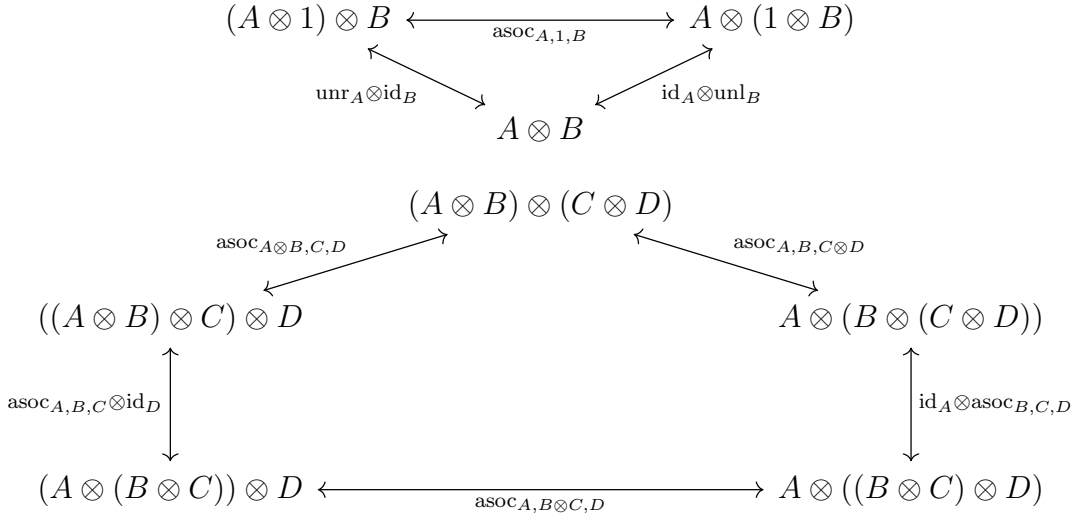
such that for every object $A \in \mathbb{C}$ *and every family of morphisms* $f_i \in \mathbb{C}(X_i, A), i \leq n$*, there is a unique morphism* $h \in \mathbb{C}(X_1 + \cdots + X_n, A)$ *such that for* $1 \leq j \leq n$*, the following diagram commutes:*



Definition 5.1.25 (Symmetric Monoidal Category)**.**
A symmetric monoidal category consists of a category $\mathbb{C}$*, together with:*

- *A functor* $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$*, called the* tensor product*;*
- *An object* $1 \in \mathbb{C}$*, called the* unit object*;*
- *A natural isomorphism* $\mathrm{asoc}_{A,B,C} : (A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$ *called the* associator*;*
- *Natural isomorphisms* $\mathrm{unl}_A : 1 \otimes A \simeq A$ *and* $\mathrm{unr}_A : A \otimes 1 \simeq A$*, called the* left unitor *and* right unitor *respectively;*
- *A natural isomorphism* $\mathrm{sym}_{A,B} : A \otimes B \simeq B \otimes A$ *traditionally called the* braiding*;*

such that for all $A, B \in \mathbb{C}$*,* $\mathrm{sym}_{A,B}; \mathrm{sym}_{B,A} = \mathrm{id}_{A \otimes B}$*, and the following diagrams commute in all directions:*



$$\begin{array}{ccccc} (A \otimes B) \otimes C & \overset{\mathrm{asoc}_{A,B,C}}{\longleftrightarrow} & A \otimes (B \otimes C) & \overset{\mathrm{sym}_{A,B\otimes C}}{\longleftrightarrow} & (B \otimes C) \otimes A \\ \Big\updownarrow{\scriptstyle \mathrm{sym}_{A,B}\otimes \mathrm{id}_C} & & & & \Big\updownarrow{\scriptstyle \mathrm{asoc}_{B,C,A}} \\ (B \otimes A) \otimes C & \underset{\mathrm{asoc}_{B,A,C}}{\longleftrightarrow} & B \otimes (A \otimes C) & \underset{\mathrm{id}_B\otimes \mathrm{sym}_{A,C}}{\longleftrightarrow} & B \otimes (C \otimes A) \end{array}$$

Example 5.1.26.
Any category with finite products is a symmetric monoidal category, with $\otimes$ being given by the cartesian product.

Notice that symmetric monoidal categories can be seen as an alternate way to categorify commutative monoids.[3]

One notion of interest is that of *closed categories*. Informally, a category $\mathbb{C}$ can be called *closed* if for all $A, B \in \mathbb{C}$, $\mathbb{C}(A, B) \in \mathrm{Ob}(\mathbb{C})$, or more accurately phrased for all $A, B \in \mathbb{C}$, there exists an object $A \multimap B \in \mathrm{Ob}(\mathbb{C})$ acting as the "internal" version of $\mathbb{C}(A, B)$. When the category has some additional structure, then the closedness should be compatible with said structure. Of particular interest are *symmetric monoidal closed categories*.

Definition 5.1.27 (Symmetric Monoidal Closed Category)**.**
Given any object $A \in \mathbb{C}$, where $\mathbb{C}$ is a symmetric monoidal category, we can define the tensor product endofunctor $(-) \otimes A : \mathbb{C} \to \mathbb{C}$ *as follows:*

- *For all $B \in \mathbb{C}$, $((-) \otimes A)(B) = B \otimes A$;*
- *For all $\varphi \in \mathbb{C}(B, C)$, $((-) \otimes A)(\varphi) = \varphi \otimes \mathrm{id}_A$.*

If for all $A \in \mathbb{C}$, $(-) \otimes A : \mathbb{C} \to \mathbb{C}$ has a right adjoint, which we will denote $A \multimap (-) : \mathbb{C} \to \mathbb{C}$, then we say that the category is closed *with respect to its tensor product. The consequence of this closedness is that for all $A, B \in \mathbb{C}$, $\mathbb{C}(A, B) \in \mathrm{Ob}(\mathbb{C})$, which we will write as $A \multimap B$ when considering the object $\mathbb{C}(A, B)$ rather than the collection of arrows, and for all $A, B, C \in \mathbb{C}$, there is a natural isomorphism $\mathbb{C}(A \otimes B, C) \simeq \mathbb{C}(A, B \multimap C)$.*

It is precisely this isomorphism that "enables currying" for a category – as a result, when considering models of λ-calculus and its derivative calculi, we are generally exclusively interested in symmetric monoidal closed categories.

Definition 5.1.28 (Cartesian Closed Category)**.**
A cartesian closed category (CCC) *is a special case of a symmetric monoidal closed category where the tensor product is given by the cartesian product.*

[3]There exists a similar alternative categorification, *braided monoidal categories*, and *monoidal categories* as a more general variant of both, but these are not of interest to this work.

Rather than checking all of the details of a symmetric monoidal closed category, a cartesian closed category is usually identified via the following proposition:

Proposition 5.1.29.
A category $\mathbb{C}$ is cartesian closed if and only if it has finite products and for all $A \in \mathbb{C}$, the functor $(-) \times A : \mathbb{C} \to \mathbb{C}$ has a right adjoint.

Example 5.1.30.
The categories **Set** *and* **Poset** *are cartesian closed. The categories* **CMonoid** *and* **Rel** *are not, but they are symmetric monoidal closed categories.*

Cartesian closed categories have a very close relationship with the simply typed λ-calculus. One can construct a sound model of the simply typed λ-calculus using any CCC.[4] As PCF is essentially an extension of the simply typed λ-calculus, it stands to reason that to model PCF with a category, we require a CCC with some additional restrictions that enable it to model numerals and fixed points. The addition of numerals is comparatively simple – an object of the category which behaves similar enough to numerals suffices. Fixed points require a bit more nuance.

Definition 5.1.31 (Complete Partial Order [BW95])**.**

- *A partial order on a set X is a relation $\preceq$ between elements of X that is reflexive, anti-symmetric, and transitive. The "order" comes from the anti-symmetry and transitivity, and it is "partial" as not all pairs of elements of X must be related with $\preceq$.*
- *A bottom element $\bot$ is an element of X such that for all $a \in X$, $\bot \preceq a$.*
- *Given a subset $Y \subseteq X$, a supremum of Y is an element $\bigvee Y \in X$ such that for all $a \in Y$, $a \preceq \bigvee Y$, and $\forall b \in X$ if $\forall a \in Y, a \preceq b$, then $\bigvee Y \preceq b$.*
- *A directed set $D \subseteq X$ is a set where for every $a, b \in D$ there exists a $c \in D$ such that $a \preceq c$ and $b \preceq c$.*

A complete partial order (CPO) is a partially ordered set $(X, \preceq)$ having a bottom element and any directed subset $D \subseteq X$has a supremum $\bigvee D$.[5]

Definition 5.1.32 (Scott-continuous Functions)**.**
Structure preserving functions on CPOs are called Scott-continuous functions*. Given CPOs X and Y, a Scott-continuous function $f : X \to Y$ is a function where for every directed subset $D \subseteq X$, $f(\bigvee D) = \bigvee f(D)$.*

Modelling fixed points, i.e. representing them in some mathematical structure, is usually done via complete partial orders or some related work. Without going into too much detail, the

[4] See the appendix for proper definitions and elaborations.
[5] There are notions sometimes called CPOs which are weaker than this definition.

structure of CPOs allows us to find a fixed point element $x \in X$ for every Scott-continuous function $h : X \to X$. These fixed point elements can then be used to model recursion, giving us fix. What remains is the task of combining the structure of complete partial orders which cartesian closed categories. To do so, we consider the category $\mathbf{CPO}$ of complete partial orders and Scott-continuous functions.

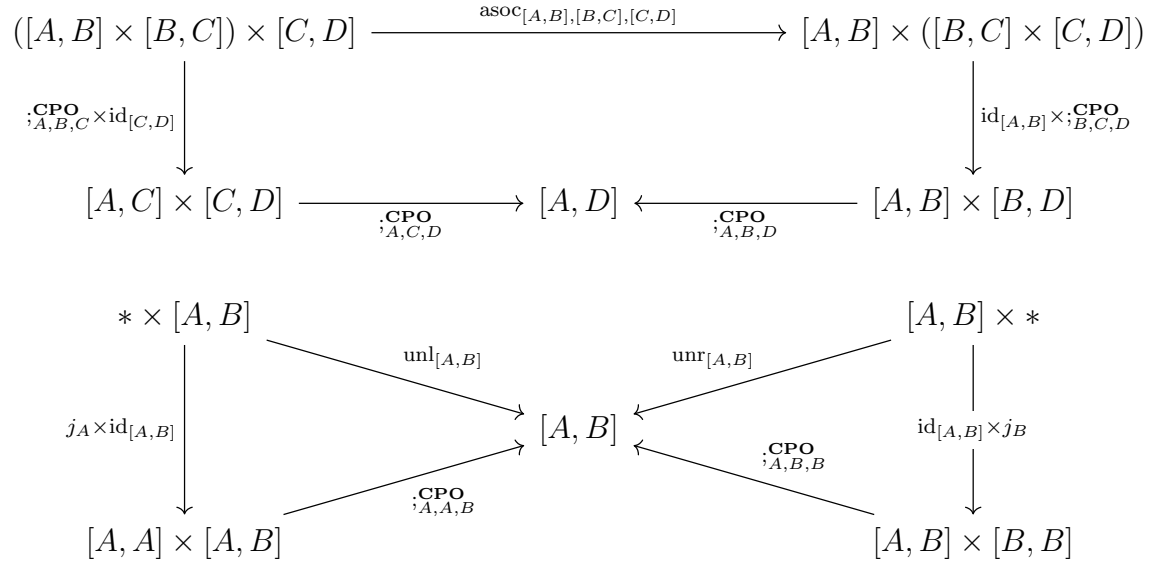
Proposition 5.1.33.
The category $\mathbf{CPO}$ *is cartesian closed.*

We can make use of the cartesian closed property to define the notion of a CCC *enriched* over $\mathbf{CPO}$. A category $\mathbb{C}$ enriched by $\mathbb{V}$ is, in essence, a category where the classes of arrows in $\mathbb{C}$ are objects in $\mathbb{V}$, and composition of arrows is given by morphisms in $\mathbb{V}$. We are only interested in the special case where $\mathbb{C}$ is a CCC and $\mathbb{V}$ is $\mathbf{CPO}$. We will be referring to the arrow objects in $\mathbf{CPO}$ as $[A, B]$ rather than $A \multimap B$, and the unit object of $\mathbf{CPO}$ as $*$.

Definition 5.1.34 ($\mathbf{CPO}$-enriched Cartesian Closed Categories)**.**
A $\mathbf{CPO}$*-enriched cartesian closed category* $\mathbb{C}$ *consists of:*

- *A class* $\mathrm{Ob}(\mathbb{C})$ *of objects of* $\mathbb{C}$*, where* $\mathbb{C}$ *is a cartesian closed category;*
- *For every* $A, B \in \mathrm{Ob}(\mathbb{C})$*, an object* $[A, B] \in \mathrm{Ob}(\mathbf{CPO})$*;*
- *For every* $A, B, C \in \mathrm{Ob}(\mathbb{C})$*, an arrow* $;^{\mathbf{CPO}}_{A,B,C} \in \mathbf{CPO}([A, B] \times [B, C], [A, C])$*;*
- *For every* $A \in \mathrm{Ob}(\mathbb{C})$*, an arrow* $j_A \in \mathbf{CPO}(*, [A, A])$ *called the* identity element*;*

such that the following diagrams commute:



Notice how the first diagram provides associativity of composition while the second provides the left and right unit of the identity. The result is a category whose sets of arrows are CPOs in a

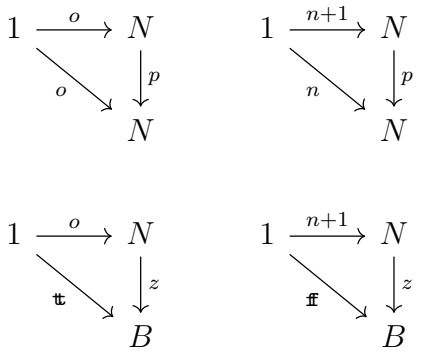
manner that is internally consistent. Given a λ-function $h : A \to A$, we can then interpret $\mathsf{fix}(h)$ as the least fixed point of the arrow $(-);^{\mathbf{CPO}}_{A,A,A} h \in \mathbf{CPO}([1, A], [1, A])$, where 1 is the cartesian unit of $\mathbb{C}$.

The final puzzle piece is the object of numerals. Strictly speaking, we are defining an object of booleans, and then defining the object of numerals with respect to the object of booleans. This is necessary to model "if-then-else".

Definition 5.1.35 (Simple Object of Numerals [HO00])**.**
Let $\mathbb{C}$ be a cartesian closed category with unit object 1. Let $B \in \mathbb{C}$ such that there are two distinct maps $\mathtt{tt} : 1 \to B$ and $\mathtt{ff} : 1 \to B$ in $\mathbb{C}$. Let $N \in \mathbb{C}$ such that there are distinct maps $o : 1 \to N, s : N \to N, p : N \to N, z : N \to B$ in $\mathbb{C}$. For all $n \in \mathbb{N}$, we define the arrow $n : 1 \to N$ as:

$$(n+1)(*) = \begin{cases} o(*), & \text{if } n = 0, \\ s(n(*)), & \text{otherwise.} \end{cases}$$

We then say that N comes equipped with a simple object of numerals[6] *(relative to B, t, f) if the following diagrams commute:*



It is clear to see how such an object can represent the numerals in PCF - the object B represents the boolean hidden in $\mathbf{ifz}(-,-,-)$, with $\mathtt{tt}$ and $\mathtt{ff}$ being true and false respectively, and z being "check for zero." We then have the following:

Proposition 5.1.36 (Categorical models of PCF [AMJ94])**.**
Every $\mathbf{CPO}$-enriched cartesian closed category equipped with a simple object of numerals is a model of PCF.

The typical example of such a categorical model of PCF is the game semantics based one found in [AMJ94] and [HO00], which has been brought up in previous chapters.

[6] An object of numerals is different from the well-known concept of a "natural numbers object" [Law63], roughly corresponding to a weak natural number object.

5.2 Semiring Monads and Categories

Knowing how to *identify* categorical models of PCF is different from the process of *finding* categorical models of PCF. The above proposition does not help in identifying new categories which could be used to model PCF. The remaining work presented in this thesis can be seen as an attempt to expand on one particular collection of categorical models of PCF by identifying the crossover between it and a particular collection of categories which are (among other things) categorical models of the simply typed λ-calculus. We now present the notions required to understand these collections.

5.2.1 Semirings

The categories we will be working with are all based on semirings, with multiple semirings often being present within the same category.

Definition 5.2.1 (Semirings [DK09])**.**
A semiring $\mathcal{S}$ *is a 5-tuple* $\langle |\mathcal{S}|, +^{\mathcal{S}}, \cdot^{\mathcal{S}}, 0^{\mathcal{S}}, 1^{\mathcal{S}} \rangle$ *such that:*

- $0^{\mathcal{S}}, 1^{\mathcal{S}} \in |\mathcal{S}|$*;*
- $\langle |\mathcal{S}|, +^{\mathcal{S}}, 0^{\mathcal{S}} \rangle$ *is a commutative monoid (*$+^{\mathcal{S}}$ *is semiring addition);*
- $\langle |\mathcal{S}|, \cdot^{\mathcal{S}}, 1^{\mathcal{S}} \rangle$ *is a monoid (*$\cdot^{\mathcal{S}}$ *is semiring multiplication),;*
- *for all* $a \in |\mathcal{S}|$, $a \cdot^{\mathcal{S}} 0^{\mathcal{S}} = 0 \cdot^{\mathcal{S}} a = 0^{\mathcal{S}}$*;*
- *for all* $a, b, c \in |\mathcal{S}|$, $a \cdot^{\mathcal{S}} (b +^{\mathcal{S}} c) = (a \cdot^{\mathcal{S}} b) +^{\mathcal{S}} (a \cdot^{\mathcal{S}} c)$ *and* $(a +^{\mathcal{S}} b) \cdot^{\mathcal{S}} c = (a \cdot^{\mathcal{S}} c) +^{\mathcal{S}} (a \cdot^{\mathcal{S}} c)$.

$\mathcal{S}$ *is referred to as commutative if* $\langle |\mathcal{S}|, \cdot^{\mathcal{S}}, 1^{\mathcal{S}} \rangle$ *is commutative. When the semiring being used is clear, the semiring addition and multiplication are written without the* $\mathcal{S}$ *superscript:* $a + b$ *for* $a +^{\mathcal{S}} b$ *and* $a \cdot b$*, or simply* ab*, for* $a \cdot b$*. We also often write* $a \in \mathcal{S}$ *to mean* $a \in |\mathcal{S}|$.

Example 5.2.2.
Some examples of semirings:

- *A semiring can trivially be constructed from the single-element set* $\{*\}$*;*
- *The semiring of natural numbers:* $\langle \mathbb{N}, +, \cdot, 0, 1 \rangle$*;*
- *Non-negative rational numbers form a semiring:* $\langle \mathbb{Q}^{+}, +, \cdot, 0, 1 \rangle$*;*
- *Real numbers form a semiring:* $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$.

Definition 5.2.3.
Given $A \in \mathbf{Set}$ *and* $\alpha : A \to \mathcal{S}$*, define* $\mathrm{supp}(\alpha) = \{a | a \in A, \alpha(a) \neq 0\}$*, the* support *of* α*. If the support of* α *is finite, then we say that* α *has* finite support.

Notation 5.2.4.

- *Given a semiring* $\mathcal{S}$ *and* $A \in \mathbf{Set}$*,* χ_A *refers to the characteristic function of* A *on* $\mathcal{S}$*, defined as:*
$$\chi_A(a) = \begin{cases} 1^{\mathcal{S}} & \text{if } a \in A, \\ 0^{\mathcal{S}} & \text{otherwise}. \end{cases}$$
- $\delta_{a,b}$ *refers to the Kronecker delta over a semiring* $\mathcal{S}$*, which is defined as:*
$$\delta_{a,b} = \begin{cases} 1^{\mathcal{S}} & \text{if } a = b, \\ 0^{\mathcal{S}} & \text{otherwise}. \end{cases}$$

In the upcoming parts, we will encounter scenarios where we are summing over an infinite number of elements of a semiring. We require this sum to be well-defined. Semirings with infinite sums are also called *complete semirings*.

Definition 5.2.5 (Complete Semirings [DK09])**.**
A complete semiring $\mathcal{S}$ *is a semiring where for every* $s \in S$*, every pair of index sets* I, J*, and every function* $f : I \to \mathcal{S}$*:*

$$\sum_{i \in I} f(i) \in \mathcal{S}, \qquad \sum_{i \in \emptyset} f(i) = 0^{\mathcal{S}}, \qquad \sum_{i \in \{j\}} f(i) = f(j),$$
$$\sum_{i \in \{j,k\}} f(i) = f(j) + f(k), \text{ where } j \neq k,$$
$$\sum_{j \in J} \left(\sum_{i \in I_j} f(i) \right) = \sum_{i \in I} f(i), \text{ if } \bigcup_{j \in J} I_j = I \text{ and } I_j \cap I_{j'} = \emptyset \text{ for } j \neq j',$$
$$\sum_{i \in I} (s \cdot f(i)) = s \cdot \left(\sum_{i \in I} f(i) \right) \qquad \sum_{i \in I} (f(i) \cdot s) = \left(\sum_{i \in I} f(i) \right) \cdot s$$

In other words, a complete semiring is a semiring with an infinite sum operation that is:

- An extension of the finite sum;
- Associative and commutative;
- Obeys distribution rules.

If $\mathcal{S}$ is not complete yet we encounter a scenario where we sum over an infinite set, then the

sum is only well-defined if it is with respect to a finitely supported function $f : B \to \mathcal{S}$. Given such a function, we set $\sum_{b \in B} f(b) = \sum_{\{b | b \in \mathrm{supp}(f)\}} f(b)$, which is well-defined.

Most of the complete semirings one would commonly think of actually belong to a subset of complete semirings called *continuous semirings*. To define these, we first define continuous operators.

Definition 5.2.6 (Continuous operators [DK09])**.**
A (unary) operator $\star$ on cpo's is continuous *if it is monotone and preserves directed suprema, i.e. $\star(\bigvee_{i \in I}(x_i)) = \bigvee_{i \in I} \star(x_i)$. Similarly, we say that an n-ary operator $\star$ is continuous if it is continuous in each component.*

Definition 5.2.7 (Continuous Semirings [DK09])**.**
A continuous *semiring $(|\mathcal{S}|, +^{\mathcal{S}}, \cdot^{\mathcal{S}}, 0^{\mathcal{S}}, 1^{\mathcal{S}})$ is a semiring equipped with a partial order $\preceq^{\mathcal{S}}$ such that:*

- *$(|\mathcal{S}|, \preceq^{\mathcal{S}})$ is a complete partial order;*
- *the operators $+^{\mathcal{S}}, \cdot^{\mathcal{S}}$ are continuous;*
- *$a \preceq^{\mathcal{S}} b$ if and only if there exists a $s \in \mathcal{S}$ such that $a +^{\mathcal{S}} s = b$.*

Proposition 5.2.8 ([DK09])**.**
Every continuous semiring is a complete semiring.

Proof. Let $\mathcal{S}$ be a continuous semiring with the partial ordering denoted by $\preceq$. Let I be an index set and $f : I \to \mathcal{S}$. The set $D_s(I, f) = \{\sum_{p \in F} f(p) | F \subseteq_{finite} I\}$ is directed – given any two elements $\sum_{p \in F_1} f(p), \sum_{q \in F_2} f(q) \in D_s(I, f)$ we have $\sum_{r \in F_1 \cup F_2} f(r) \in D_s(I, f)$, and since $\sum_{r \in F_1 \cup F_2} f(r) = \sum_{p \in F_1} f(p) + \sum_{q \in F_2} f(q)$ by the third property of continuous semirings we have $\sum_{p \in F_1} f(p) \preceq \sum_{r \in F_1 \cup F_2} f(r)$ and $\sum_{q \in F_2} q \preceq \sum_{r \in F_1 \cup F_2} f(r)$. As a consequence, $D_s(I, f)$ has a supremum in $\mathcal{S}$ which we can use to define the infinite sum:

$$\sum_{i \in I} f(i) = \bigvee D_s(I, f)$$

Consistency with $+^{\mathcal{S}}$ and $\cdot^{\mathcal{S}}$ is given by their continuous nature. □

Example 5.2.9.
The following are all examples of continuous (and thus complete) semirings.

- *The boolean semiring* $\mathrm{bool} = \langle \{\mathtt{ff}, \mathtt{tt}\}, \vee, \wedge, \mathtt{ff}, \mathtt{tt} \rangle$*, with $\mathtt{tt}$ as the top element.;*
- *The natural numbers completed with a top element ∞,* $\bar{\mathbb{N}} = \langle \mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1 \rangle$ *(where $\infty + \infty = \infty$ and for all $a \in \mathbb{N}, a + \infty = \infty$);*

- *The positive real numbers completed* $\bar{\mathbb{R}} = \langle \mathbb{R}^+ \cup \{\infty\}, +, \cdot, 0, 1\rangle$*;*
- *The tropical semiring* $\mathrm{trop} = \langle \mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0\rangle$*;*
- *The arctic semiring* $\mathrm{arc} = \langle \mathbb{R}^+ \cup \{-\infty\}, \max, +, -\infty, 0\rangle$*;*
- *For all sets* U*, the semiring of binary relations over* U*:* $\langle \mathcal{P}(U \times U), \cup, \diamond, \emptyset, \Delta\rangle$*, where for two relations* R_1, R_2*,* $R_1 \diamond R_2 = \{(u_1, u_2) | \exists (u_1, u) \in R_1, (u, u_2) \in R_2\}$*, and* $\Delta = \{(u, u) | u \in U\}$*.*
- *The idempotent naturally ordered commutative semiring* $\langle \{0, 1, a, \infty\}, +, \cdot, 0, 1\rangle$*, with* $a \cdot a = a$ *and ordering* $0 \preceq 1 \preceq a \preceq \infty$*.*

Definition 5.2.10 (Multiplicity Semirings [CES10, BP15])**.**

- *A* positive *semiring* $\mathcal{S}$ *is one where, for all* $s, s' \in \mathcal{S}$*,* $s + s' = 0 \Rightarrow s = s' = 0$*.*
- *A* discrete *semiring* $\mathcal{S}$ *is one where, for all* $s, s' \in \mathcal{S}$*,* $s + s' = 1 \Rightarrow s = 0 \vee s' = 0$*.*
- *A semiring* $\mathcal{S}$ *has the* additive splitting *property if, for all* $a_1, a_2, b_1, c_2 \in \mathcal{S}$*,* $a_1 + a_2 = b_1 + b_2$ *implies that there exist* $n_{11}, n_{12}, n_{21}, n_{22}$ *such that* $a_1 = n_{11} + n_{12}$*,* $a_2 = n_{21} + n_{22}, b_1 = n_{11} + n_{21}, b_2 = n_{12} + n_{22}$*.*
- *A semiring* $\mathcal{S}$ *has the* multiplicative splitting *property if, for all* $r, s, n_1, n_2 \in \mathcal{S}$*,* $r \cdot s = n_1 + n_2$ *implies that there must exist* $r_1, r_2, s_{11}, s_{12}, s_{21}, s_{22} \in \mathcal{S}$ *such that* $s_{11} + s_{21} = s_{12} + s_{22} = s, r_1 \cdot s_{11} + r_2 \cdot s_{12} = n_1, r_1 \cdot s_{21} + r_2 \cdot s_{22} = n_2$ *and* $r_1 + r_2 = r$*.*

A multiplicity semiring is a semiring which is commutative, has a multiplicative unit, and satisfies the above 4 properties.

Proposition 5.2.11 ([CES10])**.**
Any multiplicity semiring contains an isomorphic copy of $\mathbb{N}$*.*

Example 5.2.12.
The following are some examples of multiplicity semirings.

- *The semiring of natural numbers* $\mathbb{N}$*;*
- *The semiring of natural numbers completed with a top element* ∞*:* $\bar{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$*;*
- *The semiring* $\mathrm{TwoN} = \langle (\mathbb{N}^+ \times \mathbb{N}) \cup \{0\}, +, \cdot, 0, (1, 0)\rangle$*, where* $(n, i) \cdot (p, j) = (n \cdot p, i + j)$*,* 0 *is neutral for* $+$*, and*

$$(n, i) + (p, j) = \begin{cases} (n + p, i), & \text{if } i = j, \\ (n, i), & \text{if } i > j, \\ (p, j), & \text{otherwise.} \end{cases}$$

Lemma 5.2.13 (Generalised additive splitting [CES10])**.**
Let $\mathcal{S}$ be a semiring which has the additive splitting property.
Let $A, B \in \mathbf{Set}, \alpha : A \to \mathcal{S}, \beta : B \to \mathcal{S}$ such that:

$$\#(\mathrm{supp}(\alpha)) \in \mathbb{N}, \#(\mathrm{supp}(\beta)) \in \mathbb{N}$$
$$\textstyle\sum_{a \in A} \alpha(a) = \sum_{b \in B} \beta(b)$$

Then there exists $\sigma : (A \times B) \to \mathcal{S}$ such that:

$$\begin{aligned} &\forall a \in A, \quad \textstyle\sum_b \sigma(a,b) = \alpha(a), \\ &\forall b \in B, \quad \textstyle\sum_a \sigma(a,b) = \beta(b) \end{aligned}$$

Proof. By induction on the cardinality of $\mathrm{supp}(\alpha)$.

- $\#(\mathrm{supp}(\alpha)) = 0$: Trivial.
- $\#(\mathrm{supp}(\alpha)) = 1$: Simplified case of $\#(\mathrm{supp}(\alpha)) = 2$, detailed below.
- $\#(\mathrm{supp}(\alpha)) = 2$: By induction on the cardinality of $\mathrm{supp}(\beta)$. Here the induction hypothesis is a restricted version of the overall lemma being proven.
  - $\#(\mathrm{supp}(\beta)) = 1$: Trivial.
  - $\#(\mathrm{supp}(\beta)) = 2$: Let $\mathrm{supp}(\alpha) = \{a_1, a_2\}$ and $\mathrm{supp}(\beta) = \{b_1, b_2\}$. By the additive splitting property, there will exist $n_{11}, n_{12}, n_{21}, n_{22}$ such that $\alpha(a_1) = n_{11} + n_{12}$, $\alpha(a_2) = n_{21} + n_{22}$, $\beta(b_1) = n_{11} + n_{21}$, $\beta(b_2) = n_{12} + n_{22}$. Then we finish this case by defining σ as:

$$\sigma(x,y) = \begin{cases} n_{11} & \text{if } x = a_1, y = b_1, \\ n_{12} & \text{if } x = a_1, y = b_2, \\ n_{21} & \text{if } x = a_2, y = b_1, \\ n_{22} & \text{if } x = a_2, y = b_2, \\ 0 & \text{if } (x,y) \notin \mathrm{supp}(\alpha) \times \mathrm{supp}(\beta). \end{cases}$$

  - $\#(\mathrm{supp}(\beta)) = n + 1, n > 1$: Assume that the property holds for n.
    Let $\mathrm{supp}(\alpha) = \{a_1, a_2\}$ and $\mathrm{supp}(\beta) = \{b_1, \ldots, b_n, b_{n+1}\}$. By the additive splitting property, there exist $n_{11}, n_{12}, n_{21}, n_{22}$ such that

$$\begin{aligned} \alpha(a_1) &= n_{11} + n_{12} \\ \alpha(a_2) &= n_{21} + n_{22} \\ \textstyle\sum_{i=1}^{n} \beta(b_i) &= n_{11} + n_{21} \\ \beta(b_{n+1}) &= n_{12} + n_{22} \end{aligned}$$

By induction hypothesis we can find a σ_n such that for all $a \in A, \sum_b \sigma_n(a,b) = n_{11}$ and for all $b \in B, \sum_a \sigma_n(a,b) = n_{21}$. Then we finish this case by defining σ as:

$$\sigma(a_i, b_j) = \begin{cases} \sigma_n(a_i, b_j) & \text{if } j \leq n, \\ n_{21} & \text{if } i = 1, \\ n_{22} & \text{if } i = 2, \\ 0 & \text{if } (a_i, b_j) \notin \text{supp}(\alpha) \times \text{supp}(\beta). \end{cases}$$

- $\#(\text{supp}(\alpha)) = m{+}1$: Assume that the erty holds for m. Let $\text{supp}(\alpha) = \{a_1, \ldots, a_m, a_{m+1}\}$. Define α' as:

$$\alpha'(a_i) = \begin{cases} \sum_{j=1}^{m} \alpha(a_j) & \text{if } i = 1, \\ \alpha(a_{m+1}) & \text{if } i = m+1 \\ 0 & \text{otherwise.} \end{cases}$$

We can then apply the lemma for $\#(\text{supp}(\alpha')) = 2$, proven previously, to get σ_1 such that

$$\begin{array}{rcl} \sum_{b \in B} \sigma_1(a_1, b) & = & \sum_{j=1}^{m} \alpha(a_j) \\ \sum_{b \in B} \sigma_1(a_{m+1}, b) & = & \alpha(a_{m+1}) \\ \forall b \in B, \sum_a \sigma_1(a, b) & = & \beta(b) \end{array}$$

By induction hypothesis, we can obtain σ_2 such that $\sum_{b \in B} \sigma_2((a_1, b), a) = \alpha(a)$ for all $a \in A \setminus \{a_{m+1}\}$, and for all $b \in B, \sum_{a \in A \setminus \{a_{m+1}\}} \sigma_2((a_1, b), a) = \sigma_1(a_1, b)$. We then conclude by defining σ as:

$$\sigma(a_i, b) = \begin{cases} \sigma_2((a_1, b), a_i) & \text{if } i \leq m \\ \sigma_1(a_i, b) & \text{if } i = m+1, \\ 0 & \text{if } a, b \notin \text{supp}(\alpha) \times \text{supp}(\beta). \end{cases}$$

□

The above lemma is incredibly important, as without it elements cannot be paired together in a consistant manner. Consider a function $f \in \mathbf{Rel}(A, B)$. To evaluate f, it is necessary to give it a pair of elements. The above property is required to pair together elements (a, b) from two separate "collections" $\alpha : A \to \mathcal{S}, \beta : B \to \mathcal{S}$ to then be evaluated by f.

It is useful to note that the above lemma and the additive splitting property are one and the same:

Proposition 5.2.14.
If a semiring does not have the additive splitting property, then it also does not have the property described in Lemma 5.2.13.

Proof. Trivial proof by contradiction, taking the case where the cardinality of $\mathrm{supp}(\alpha)$ is 2. □

5.2.2 Semiring monad

Every semiring induces a monad on **Set**. If the semiring is complete, then we can also define a second monad from said semiring. The second monad is simply a more general version of the first, so we present them together.

Definition 5.2.15.
Given any semiring $\mathcal{S}$, there is a functor $\mathrm{S}^{\mathsf{fin}} : \mathbf{Set} \to \mathbf{Set}$ acting as follows:

- *Given an object $A \in \mathbf{Set}$, $\mathrm{S}^{\mathsf{fin}}A = \{\phi \mid \phi : A \to \mathcal{S}, \mathrm{supp}(\phi)$ is finite$\}$, the set of finitely supported functions from A to the semiring $\mathcal{S}$,*
- *Given a function $f : A \to B \in \mathbf{Set}(A, B)$, along with left and right "inputs" to the function $\phi \in \mathrm{S}^{\mathsf{fin}}A, b \in B$, $\mathrm{S}^{\mathsf{fin}}f(\phi)(b) = \sum_{a \in A} \phi(a) \cdot \delta_{f(a),b} \in \mathcal{S}$.*

If $\mathcal{S}$ is complete, there also exists the functor $\mathrm{S}^{\mathsf{inf}} : \mathbf{Set} \to \mathbf{Set}$:

- *Given an object $A \in \mathbf{Set}$, $\mathrm{S}^{\mathsf{inf}}A = \{\phi \mid \phi : A \to \mathcal{S}\}$, the set of functions from A to the semiring $\mathcal{S}$,*
- *Given a function $f : A \to B \in \mathbf{Set}(A, B)$, along with left and right "inputs" to the function $\phi \in \mathrm{S}^{\mathsf{inf}}A, b \in B$, $\mathrm{S}^{\mathsf{inf}}f(\phi)(b) = \sum_{a \in A} \phi(a) \cdot \delta_{f(a),b} \in \mathcal{S}$.*

We will write S to mean either $\mathrm{S}^{\mathsf{inf}}$ or $\mathrm{S}^{\mathsf{fin}}$ depending on context. Generally, if $\mathcal{S}$ is complete, we use $\mathrm{S}^{\mathsf{inf}}$ unless otherwise specified.

For both choices of functor, there exist identically acting unit $\eta^{\mathrm{S}}_A : A \to \mathrm{S}A$ and multiplication $\mu^{\mathrm{S}}_A : \mathrm{SS}A \to \mathrm{S}A$ natural transformations :

- *Given any $a, a' \in A$, $\eta^{\mathrm{S}}_A(a)(a') = \delta_{a,a'}$,*
- *Given any $\varphi \in \mathrm{SS}A, a \in A$, $\mu^{\mathrm{S}}_A(\varphi)(a) = \sum_{\alpha \in \mathrm{S}A} \varphi(\alpha) \cdot \alpha(a)$.*

Lemma 5.2.16.
$(\mathrm{S}, \eta^{\mathrm{S}}, \mu^{\mathrm{S}})$ is a monad.

Proof. Proof proceeds by simple solving of equations.[7]

[7] When definitions are trivial or multiple definitions are used simultaneously, we omit the number.

- Functoriality of S:
  - Given $\varphi \in \mathrm{S}A, a \in A$,

$$\begin{array}{rcll} \mathrm{S}\mathrm{id}_A(\varphi)(a) & = & \sum_{a' \in A} \varphi(a') \cdot \delta_{\mathrm{id}_A(a'),a} & \text{Definition} \\ & = & \varphi(a) & a = a' \\ & = & \mathrm{id}_{\mathrm{S}A}(\varphi)(a) & \text{Definition} \end{array}$$

  - Given $f : A \to B, g : B \to C$,

$$\begin{array}{rcll} \mathrm{S}(g \circ f)(\varphi)(c) & = & \sum_{\{a \mid g(f(a))=c\}} \varphi(a) & \text{Definition} \\ & = & \sum_{\{a,b \mid f(a)=b, g(b)=c\}} \varphi(a) & \text{Function preimages are disjoint} \\ & = & \sum_{\{b \mid g(b)=c\}} \sum_{\{a \mid f(a)=b\}} \varphi(a) & \text{Rearranged} \\ & = & (\mathrm{S}g)(\mathrm{S}f(\varphi))(c) & \text{Definition} \end{array}$$

- Naturality of η^{S}: Given $f : A \to B, a \in A, b \in B$,

$$\begin{array}{rcll} \mathrm{S}f(\eta^{\mathrm{S}}_A(a))(b) & = & \sum_{a' \in A} \eta^{\mathrm{S}}_A(a)(a') \cdot \delta_{f(a'),b} & \text{Definition} \\ & = & \sum_{a' \in A} \delta_{a,a'} \cdot \delta_{f(a'),b} & \text{Definition} \\ & = & \delta_{f(a),b} & a = a' \\ & = & \eta^{\mathrm{S}}_B(f(a))(b) & \text{Definition} \end{array}$$

- Naturality of μ^{S}: Given $f : A \to B, a \in A, b \in B$,

$$\begin{array}{rcll} \mu^{\mathrm{S}}_A(\mathrm{SS}f(\varphi))(b) & = & \sum_{\beta \in \mathrm{S}B} \mathrm{SS}f(\varphi)(\beta) \cdot \beta(b) & \text{Definition} \\ & = & \sum_{\beta \in \mathrm{S}B} \sum_{\alpha \in \mathrm{S}A} \varphi(\alpha) \cdot \delta_{\mathrm{S}f(\alpha),\beta} \cdot \beta(b) & \text{Definition} \\ & = & \sum_{\alpha \in \mathrm{S}A} \varphi(\alpha) \cdot \mathrm{S}f(\alpha)(b) & \beta = \mathrm{S}f(\alpha) \\ & = & \sum_{a \in A} \sum_{\alpha \in \mathrm{S}A} \varphi(\alpha) \cdot \alpha(a) \cdot \delta_{f(a),b} & \text{Definition} \\ & = & \sum_{a \in A} \mu^{\mathrm{S}}_A(\varphi)(a) \cdot \delta_{f(a),b} & \text{Definition} \\ & = & \mathrm{S}f(\mu^{\mathrm{S}}_A(\varphi))(b) & \text{Definition} \end{array}$$

- Monad Diagrams:
  - Diagram 1: Given $\alpha, \alpha' \in \mathrm{S}A$,

$$\begin{array}{rcll} \mu^{\mathrm{S}}_A(\mathrm{S}\eta^{\mathrm{S}}_A(\alpha))(a) & = & \sum_{\alpha' \in \mathrm{S}A} \mathrm{S}\eta^{\mathrm{S}}_A(\alpha)(\alpha') \cdot \alpha'(a) & \text{Definition} \\ & = & \sum_{\alpha' \in \mathrm{S}A} \sum_{a' \in A} \alpha(a) \cdot \delta_{\eta^{\mathrm{S}}_A(a'),\alpha'} \cdot \alpha'(a) & \text{Definition} \\ & = & \sum_{a \in A} \alpha(a) \cdot \eta^{\mathrm{S}}_A(a')(a) & \alpha' = \eta^{\mathrm{S}}_A(a') \\ & = & \alpha(a) & a = a' \\ & = & \sum_{\alpha' \in \mathrm{S}A} \delta_{\alpha,\alpha'} \cdot \alpha'(a) & \alpha = \alpha' \\ & = & \sum_{\alpha' \in \mathrm{S}A} \eta^{\mathrm{S}}_{\mathrm{S}A}(\alpha)(\alpha') \cdot \alpha'(a) & \text{Definition} \\ & = & \mu^{\mathrm{S}}_A(\eta^{\mathrm{S}}_{\mathrm{S}A}(\alpha))(a) & \text{Definition} \end{array}$$

– Diagram 2: Given $\bar{\varphi} \in \mathrm{SSS}A, a \in A$,

$$\begin{array}{rcll}
\mu^{\mathrm{S}}_A(\mathrm{S}\mu^{\mathrm{S}}_A(\bar{\varphi}))(a) & = & \sum_{\alpha \in \mathrm{S}A} \mathrm{S}\mu^{\mathrm{S}}_A(\bar{\varphi})(\alpha) \cdot \alpha(a) & \text{Definition} \\
& = & \sum_{\alpha \in \mathrm{S}A} \sum_{\varphi \in \mathrm{SS}A} \bar{\varphi}(\varphi) \cdot \delta_{\mu^{\mathrm{S}}_A(\varphi),\alpha} \cdot \alpha(a) & \text{Definition} \\
& = & \sum_{\varphi \in \mathrm{SS}A} \bar{\varphi}(\varphi) \cdot \mu^{\mathrm{S}}_A(\varphi)(a) & \alpha = \mu^{\mathrm{S}}_A(\varphi) \\
& = & \sum_{\alpha' \in \mathrm{S}A} \sum_{\varphi \in \mathrm{SS}A} \bar{\varphi}(\varphi) \cdot \varphi(\alpha') \cdot \alpha'(a) & \text{Definition} \\
& = & \sum_{\alpha' \in \mathrm{S}A} \mu^{\mathrm{S}}_{\mathrm{S}A}(\bar{\varphi})(\alpha') \cdot \alpha(a') & \text{Definition} \\
& = & \mu^{\mathrm{S}}_A(\mu^{\mathrm{S}}_{\mathrm{S}A}(\bar{\varphi}))(a) & \text{Definition}
\end{array}$$

□

Notation 5.2.17.
Given a semiring monad S*, we write* $\mu^{\mathrm{S}-1}_A(\alpha)$ *for the set* $\{\bar{\alpha} | \mu^{\mathrm{S}}_A(\bar{\alpha}) = \alpha\}$.

The Kleisli categories induced by this monad may seem familiar. If $\mathcal{S} = \mathrm{bool}$ the boolean semiring, then the corresponding Kleisli category is exactly **Rel**. More generally, for a given semiring $\mathcal{S}$, the Kleisli category $\mathbf{Set}_{\mathrm{S}}$ of a semiring monad has sets as objects and the arrows $\varphi : A \to B$ are matrices where the columns are indexed by A, rows by B, and cells populated by elements of $\mathcal{S}$. The identity is the identity matrix, and composition is given by matrix multiplication.

$\mathbf{Set}_{\mathrm{S}}$ has a number of useful and interesting properties, two of which are particularly relevant at this point. The first is that they are not cartesian closed, so they cannot be used to model PCF – the cartesian product functor fails to have a right adjoint. $\mathbf{Set}_{\mathrm{S}}$ is still a symmetric monoidal closed category, but with a different tensor product:

For $A, B \in \mathbf{Set}_{\mathrm{S}}$ and $\varphi \in \mathbf{Set}_{\mathrm{S}}(A, B), \psi \in \mathbf{Set}_{\mathrm{S}}(C, D)$, we have

$$A \otimes B = A \times B \qquad (\varphi \otimes \psi)((a, c), (b, d)) = \varphi(a, b) \cdot^{\mathcal{S}} \psi(c, d)$$

Remark 5.2.18.
$\mathbf{Set}_{\mathrm{S}}{}^{\mathrm{op}}$ *and* $\mathbf{Set}_{\mathrm{S}}$ *are isomorphic. As a consequence, any monad on* $\mathbf{Set}_{\mathrm{S}}$ *is also a comonad on* $\mathbf{Set}_{\mathrm{S}}$.

Corollary 5.2.19.
Given two semiring monads R, S*, there is a composite functor* RS*:*

$$\forall A \in \mathbf{Set}, \mathrm{RS}A = \{\psi | \psi : (A \to \mathcal{S}) \to \mathcal{R}\}$$
$$\forall f : A \to B \in \mathbf{Set}, \psi \in \mathrm{RS}A, \beta \in \mathrm{S}B,$$
$$\mathrm{RS}f(\psi)(\beta) = \sum_{\alpha \in \mathrm{S}A} \psi(\alpha) \cdot \delta_{\lambda b. \sum_{a \in A} \alpha(a) \cdot \delta_{f(a),b},\beta}$$

While the composite functor always exists, the composite functor may not be a monad. For the composite to be a monad, there must be a distribution between the monads.

We are now able to present the following previous results:

- Given any continuous semiring with a commutative product $\mathcal{R}$, the co-Kleisli category of the comonad $\mathbb{N}$ (using finite support) on the category $\mathbf{Set}_{\mathrm{R}}$ is a CPO-enriched cartesian closed category with an object of numerals [LMMP13].
- Given any multiplicity semiring $\mathcal{S}$, S (with finite support) is a comonad on the category $\mathbf{Set}_{\mathrm{bool}}$ [CES10]. Moreover, the resulting co-Kleisli category is a CPO-enriched cartesian closed category with an object of numerals.[8]

Given the similarities between these, one could reasonably assume that they can be combined in some way. The remainder of the work to be presented is an attempt to do so by answering the following question:

"Given two semirings $\mathcal{R}, \mathcal{S}$ and corresponding monads R, S on $\mathbf{Set}$, what conditions are necessary for RS to be a monad?"

[8] While this category being CPO-enriched and having an object of numerals is not stated in the source paper for this result, it is a trivial consequence of the first paper.

Chapter 6

Semiring Monad Distributions

6.1 Idempotent Sum

We will begin by discussing the cases where $\mathcal{R}$ has an idempotent sum. This significantly simplifies the problem. A better intuition for why we require this restriction will be given later; from a purely technical perspective, idempotency means that for any surjective function $f : A \to B$ and a $\beta \in B \to \mathcal{R}$, we would have

$$\sum_{a \in A} \beta(f(a)) = \sum_{b \in B} \beta(b)$$

This is clearly not true for the non-idempotent case, but holds for the idempotent case.

For a distribution to exist, we require the semirings to interact with one another – to be precise, we require $\mathcal{S}$ to act on $\mathcal{R}$.

Definition 6.1.1 (Exponential Action)**.**
Given two semirings $\mathcal{R}$ and $\mathcal{S}$, consider fixed a function $(-)^- : \mathcal{R} \times \mathcal{S} \to \mathcal{R}$ satisfying the following properties (for all $r \in \mathcal{R}, s \in \mathcal{S}$):

$$\begin{aligned} r^0 &= 1 & r^1 &= r \\ 0^s &= 0 & 1^s &= 1 \\ (r^{s_1})^{s_2} &= & r^{s_1 * s_2} & \\ \forall X \subseteq_{\text{finite}} \mathcal{R}, (\textstyle\prod_{x \in X} x)^s &= & \textstyle\prod_{x \in X} x^s & \\ \forall Y \subseteq_{\text{finite}} \mathcal{S}, \textstyle\prod_{y \in Y} r^y &= & r^{\sum_{y \in Y} y} & \end{aligned}$$

This function mirrors the exponent in natural numbers. If $\mathcal{R} = \mathcal{S} = \mathbb{N}$, then $(-)^-$ is exactly the exponential action of natural numbers. This function will be referred to as the exponential action *of $\mathcal{S}$ on $\mathcal{R}$.*

We also require the notion of *witnesses*. A witness is essentially a generalisation of what can intuitively be described as pairing elements of multisets together. Consider a finite multiset m as an unordered list $[1, 1, 2]$, and another $n = [1, 2, 2]$. When considering the ways that we can pair the elements of these multisets together, we arrive at the following possible multisets:

$$[(1,1),(1,2),(2,2)][(1,2),(1,2),(2,1)]$$

Witnesses generalise this from multisets (functions $f : A \to \mathbb{N}$) to any semiring $\mathcal{S}$ (functions $g : A \to \mathcal{S}$).

Definition 6.1.2 (Witness Sets)**.**
Given a positive and discrete semiring $\mathcal{S}$ with corresponding monad S, the set of witnesses *of (α, β), denoted $\mathcal{W}(\alpha, \beta)$, is defined as follows:*

$$\mathcal{W}(\alpha,\beta) = \left\{ \sigma \;\middle|\; \begin{array}{l} \sigma : A \times B \to \mathcal{S}, \\ \forall a \in A, \sum_{b \in B} \sigma(a,b) = \alpha(a), \\ \forall b \in B, \sum_{a \in A} \sigma(a,b) = \beta(b) \end{array} \right\}$$

We define $\mathcal{W} : \mathrm{S}A \times \mathrm{S}B \to \mathcal{P}(\mathrm{S}(A \times B))$ as the function taking two functions $\alpha : \mathrm{S}A, \beta : \mathrm{S}B$ to their set of witnesses. We call any $\sigma \in \mathcal{W}(\alpha, \beta)$ a witness *of (α, β).*

Lemma 6.1.3.
Witness sets are associative and commutative: Given

$$\begin{gathered} \alpha : A \to \mathcal{S}, \beta : B \to \mathcal{S}, \gamma : C \to \mathcal{S}, \\ \sigma_{a,b} \in \mathcal{W}(\alpha,\beta), \rho_{(a,b),c} \in \mathcal{W}(\sigma_{a,b},\gamma) \end{gathered}$$

there exist unique $\sigma_{b,a} \in \mathcal{W}(\beta,\alpha), \sigma_{a,c} \in \mathcal{W}(\alpha,\gamma), \rho_{(a,c),b} \in \mathcal{W}(\sigma_{a,c},\beta)$ such that:

$$\begin{array}{rrcl} \forall a \in A, b \in B, & \sigma_{a,b}(a,b) & = & \sigma_{b,a}(b,a) \\ \forall a \in A, b \in B, c \in C, & \rho_{(a,b),c}((a,b),c) & = & \rho_{(a,c),b}((a,c),b) \end{array}$$

Proof. Trivial, by definition of the sets. □

Lemma 6.1.4.
$\mathcal{W}$ is a natural transformation.

Proof. Let $f : A \to B, g : C \to D, \alpha \in \mathrm{S}A, \gamma \in \mathrm{S}C$. By definition, $\mathcal{W}$ is natural if

$$\sigma \in \mathcal{W}(\mathrm{S}f(\alpha), \mathrm{S}g(\gamma)) \Leftrightarrow \exists \sigma' \in \mathcal{W}(\alpha,\gamma) \text{ such that } \mathrm{S}(f \times g)(\sigma') = \sigma$$

By Lemma 6.1.3, it suffices to prove that

$$\sigma \in \mathcal{W}(\mathrm{S}f(\alpha), \gamma) \Leftrightarrow \exists \sigma' \in \mathcal{W}(\alpha,\gamma) \text{ such that } \mathrm{S}(f \times \mathrm{id}_C)(\sigma') = \sigma$$

We will prove one direction first, and then the converse.

$\Rightarrow$ Let $\sigma \in \mathcal{W}(\mathrm{S}f(\alpha), \gamma)$. Define for each b in the image of f a set

$$\left\{ \sigma_b \middle| \sigma_b \in \mathrm{S}(A \times C), \sigma_b(a,c) \neq 0 \Rightarrow f(a) = b, \sum_{a \in A} \sigma_b(a,c) = \sigma(b,c) \right\}$$

The above set is always non-empty: by definition of $\mathcal{W}$, if $\sigma(b,c) \neq 0$, then we have $Sf(\alpha)(b) = \sum_c \sigma(b,c) \neq 0$ as $\mathcal{S}$ is positive. Thus, if $\sigma(b,c) \neq 0$, then by definition of S there exists an $a \in A$ such that $f(a) = b$, and so we have a σ_b where $\sigma_b(a,c) = \sigma(b,c)$.

We can then define a $\sigma' \in \mathcal{W}(\alpha, \gamma)$ by selecting an arbitrary σ_b for each $b \in B$ and setting $\sigma'(a,c) = \sigma_{f(a)}(a,c)$.

Then for all $b \in B$, $\sum_{c \in C} \sigma(b,c) = \sum_{a \in A} \sigma'(a,c) \cdot {\delta^{\mathcal{S}}}_{f(a),b}$, so $\mathrm{S}(f \times \mathrm{id}C)(\sigma') = \sigma$.

$\Leftarrow$ Let $\sigma' \in \mathcal{W}(\alpha, \gamma)$ such that $\mathrm{S}(f \times \mathrm{id}_C)(\sigma') = \sigma$. Then $\lambda(b,c). \sum_{a \in A} \sigma'(a,c) \cdot {\delta^{\mathcal{S}}}_{f(a),b}$ is an element of $\mathcal{W}(\mathrm{S}f(\alpha), \gamma)$, and $\lambda(b,c). \sum_{a \in A} \sigma'(a,c) \cdot {\delta^{\mathcal{S}}}_{f(a),b} = \mathrm{S}(f \times \mathrm{id}_C)(\sigma')$ by definition. □

With the additional notation presented, we can also rephrase Lemma 5.2.13 to the following:

Lemma 6.1.5 (Generalised Additive Splitting)**.**
Let $\mathcal{S}$ be a semiring with additive splitting*, and let S be the corresponding semiring monad using finitely supported functions. For all $f \in \mathrm{S}A, g \in \mathrm{S}B$, if $\sum_{a \in A} f(a) = \sum_{b \in B} g(b)$, then there exists $\sigma \in \mathcal{W}(f,g)$.*

Lemma 6.1.6.
Let $\mathcal{S}$ be a semiring with additive splitting. Given two witness sets $\mathcal{W}(\alpha,\beta), \mathcal{W}(\beta,\gamma)$, where $\alpha \in \mathrm{S}A, \beta \in \mathrm{S}B, \gamma \in \mathrm{S}C$, then for all $\sigma_1 \in \mathcal{W}(\alpha,\beta), \sigma_2 \in \mathcal{W}(\beta,\gamma)$ there exists $\tau \in \mathcal{W}(\sigma_1,\gamma)$ such that for all $b \in B, c \in C$,

$$\sum_{a \in A} \tau((a,b),c) = \sigma_2(b,c)$$

Proof. First, by the definition of witness sets, the existence of σ_1, σ_2 forces

$$\sum_{a \in A} \alpha(a) = \sum_{b \in B} \beta(b) = \sum_{c \in C} \gamma(c)$$

Using Lemma 6.1.5, we have $\tau_1 \in \mathcal{W}(\sigma_1,\gamma), \tau_2 \in \mathcal{W}(\alpha,\sigma_2)$. Using symmetry and associativity, we get $\sigma_{31} \in \mathcal{W}(\alpha,\gamma), \tau_{31} \in \mathcal{W}(\sigma_{31},\beta)$ and $\sigma_{32} \in \mathcal{W}(\alpha,\gamma), \tau_{32} \in \mathcal{W}(\sigma_{32},\beta)$.
Define for $i \in \{1,2\}$ $b \in B$ the morphisms $\frac{\tau_{3i}}{b} \in \mathrm{S}(A \times C)$:

$$\tfrac{\tau_{3i}}{b}(a,c) = \tau_{3i}((a,c),b)$$

Using Lemma 6.1.5, we get for each $b \in B$ a morphism $\frac{\rho}{b} \in \mathcal{W}(\frac{\tau_{31}}{b}, \frac{\tau_{32}}{b})$. We then define $\rho \in \mathrm{S}(A \times C \times A \times C \times B)$ as:

$$\rho(a_1, c_1, a_2, c_2, b) = \frac{\rho}{b}((a_1, c_1), (a_2, c_2))$$

We then have $\tau((a, b), c) = \sum_{a', c'} \rho(a, c', a', c, b)$ satisfying our requirements. □

Lemma 6.1.7 (Generalised Multiplicative Splitting [CES10])**.**
Let $\mathcal{S}$ be a multiplicity semiring with the corresponding monad S using finite support. Let A be an arbitrary set. Given $p \in \mathcal{S}, m \in \mathrm{S}A, f : \mathrm{S}(A \times B)$, if $pm(a) = \sum_{b \in B} f(a, b)$, then there exist $k \in \mathbb{N}$, $\wp \in \mathrm{S}\mathbb{N}, \flat \in \mathrm{S}(\mathbb{N} \times A \times B)$ such that

$$\begin{aligned}
&&\sum_{n \in \mathbb{N}} \wp(n) &= p, \\
\forall a \in A, n \in \mathbb{N}, &\quad \sum_{b \in B} \flat(n, a, b) &= m(a), \\
\forall a \in A, b \in B, &\sum_{n \in \mathbb{N}} \wp(n) \cdot \flat(n, a, b) &= f(a, b), \\
\forall a \in A, b \in B, &\quad \flat(n, a, b) = \wp(n) &= 0, \qquad \text{if } n > k.
\end{aligned}$$

Proof. By induction on $\#\mathsf{supp}(f)$.

- Cases $\#\mathsf{supp}(f) = 0$ and $\#\mathsf{supp}(f) = 1$ are trivial.
- Case $\#\mathsf{supp}(f) = l + 1$: Select arbitrary $(a_1, b_1), (a_2, b_2) \in \mathsf{supp}(f)$ such that we have $(a_1, b_1) \neq (a_2, b_2)$ and define $g \in \mathrm{S}(\{0, 1\} \times A \times B)$ such that

$$g(i, a', b') = \begin{cases} f(a', b') \cdot \chi_{A \times B \setminus \{a_2, b_2\}}, & \text{if } i = 0, \\ f(a_2, b_2) \cdot \delta_{a', a_1} \cdot \delta_{b', b_1}, & \text{if } i = 1. \end{cases}$$

  We have $pm(a) = \sum_{b \in B} \sum_{i \in \{0,1\}} g(n, a, b)$ with $\#\mathsf{supp}(\lambda a, b. \sum_{i \in \{0,1\}} g(n, a, b)) = l$, so by inductive hypothesis we have $k \in \mathbb{N}$, $\wp \in \mathrm{S}\mathbb{N}, \flat \in \mathrm{S}(\mathbb{N} \times A \times B)$ such that

$$\begin{aligned}
&&\sum_{n \in \mathbb{N}} \wp(n) &= p, \\
\forall a \in A, n \in \mathbb{N}, &\quad \sum_{b \in B} \flat(n, a, b) &= m(a), \\
\forall a \in A, b \in B, &\sum_{n \in \mathbb{N}} \wp(n) \cdot \flat(n, a, b) &= \sum_{i \in \{0,1\}} g(i, a, b), \\
\forall a \in A, b \in B, &\quad \flat(n, a, b) = \wp(n) &= 0, \qquad \text{if } n > k.
\end{aligned}$$

  By Lemma 6.1.5 there exists $\rho \in \mathcal{W}(\lambda n. \wp(n) \cdot \flat(n, a_1, b_1), \lambda i. g(i, a_1, b_1))$. We have

$$\wp(n) \cdot \flat(n, a_1, b_1) = \rho(n, 0) + \rho(n, 1)$$

As $\mathcal{S}$ has the multiplicative splitting property, there exist $r_1, r_2, s_{11}, s_{12}, s_{21}, s_{22} \in \mathrm{S}\mathbb{N}$ such that for all $n \in \mathbb{N}$,

$$\begin{aligned} r_1(n) + r_2(n) &= \wp(n), \\ s_{11}(n) + s_{21}(n) = s_{12}(n) + s_{22}(n) &= \flat(n, a_1, b_1), \\ r_1(n) \cdot s_{11}(n) + r_2(n) \cdot s_{12}(n) &= \rho(n, 0), \\ r_1(n) \cdot s_{21}(n) + r_2(n) \cdot s_{22}(n) &= \rho(n, 1). \end{aligned}$$

We will define a $\wp' \in \mathrm{S}\mathbb{N}, \flat' \in \mathrm{S}(\mathbb{N} \times A \times B)$ as follows (for all $a \in A, b \in B, n \in \mathbb{N}$):

$$\begin{aligned} \wp'(2 \cdot n) &= r_1(n), \\ \wp'(2 \cdot n + 1) &= r_2(n), \\ \flat'(2 \cdot n, a, b) &= \flat(n, a, b), \quad \text{if } (a, b) \notin \{(a_1, b_1), (a_2, b_2)\}, \\ \flat'(2 \cdot n + 1, a, b) &= \flat(n, a, b), \quad \text{if } (a, b) \notin \{(a_1, b_1), (a_2, b_2)\}, \\ \flat'(2 \cdot n, a_1, b_1) &= s_{11}(n), \\ \flat'(2 \cdot n + 1, a_1, b_1) &= s_{12}(n), \\ \flat'(2 \cdot n, a_2, b_2) &= s_{21}(n), \\ \flat'(2 \cdot n + 1, a_2, b_2) &= s_{22}(n). \end{aligned}$$

Simple calculations show that the required properties hold true (with $k' \geq k * 2$), so we conclude. □

Definition 6.1.8 (Coefficientless Expansion I)**.**
Given two semirings $\mathcal{R}$ and $\mathcal{S}$ such that $\mathcal{R}$ has infinite sums and $\mathcal{S}$ is positive and discrete, if an exponential action of $\mathcal{S}$ on $\mathcal{R}$ exhibits the following property (for all $A \in \mathbf{Set}, s \in \mathcal{S}, f : A \to \mathcal{R}$)

$$(\sum_{a \in A} f(a))^s = \sum_{\left\{\alpha \middle| \sum_{a \in A} \alpha(a) = s\right\}} \prod_{a \in A} (f(a))^{\alpha(a)}$$

then we say that the exponential action has a coefficientless expansion.

The term *coefficientless expansion* arises from its similarity to the multinomial expansion (generalised binomial expansion) rule in the natural numbers, but lacking the multinomial coefficient. An expansion with a coefficient that directly mirrors the natural numbers will be presented later. If the exponent is seen as a form of product, then there is also a resemblance to *Skolemization*, for readers familiar with the concept.

Notation 6.1.9 (Product of Exponents)**.**
To improve readability, $\prod_{a \lhd \alpha} f(a)$ is used as syntactic sugar for $\prod_{a \in A} (f(a))^{\alpha(a)} = \prod_{a \in \mathrm{dom}(\alpha)} (f(a))^{\alpha(a)}$.

Intuitively, $\prod_{a \lhd \alpha} f(a)$ can be thought of as "pulling" a out of α $\alpha(a)$-many times, and multiplying $f(a)$ with itself for each "pull".

Lemma 6.1.10 (Coefficientless Expansion II)**.**
Given two semirings $\mathcal{R}$ and $\mathcal{S}$ such that $\mathcal{S}$ is a multiplicity semiring with an exponential action of $\mathcal{S}$ on $\mathcal{R}$ that has a coefficientless expansion, then ($\forall A, B \in \mathbf{Set}, \beta : B \to \mathcal{S}, f : A \times B \to \mathcal{R}$):

$$\prod_{b \lhd \beta} \sum_{a \in A} f(a,b) = \sum_{\alpha \in A \to \mathcal{S}} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \prod_{a,b \lhd \sigma} f(a,b)$$

Proof.

$$\begin{aligned}
\prod_{b \lhd \beta} \sum_{a \in A} f(a,b) &= \prod_{b \in B} \sum_{\left\{\alpha \middle| \substack{\alpha \in \mathcal{S}A, \\ \sum_{a \in A} \alpha(a) = \beta(b)}\right\}} \prod_{a \lhd \alpha} f(a,b) && \text{Definition 6.1.8} \\
&= \prod_{b \in B} \sum_{\alpha \in \mathcal{S}A} \delta^{\mathcal{R}}{}_{\sum_{a \in A} \alpha(a), \beta(b)} \prod_{a \lhd \alpha} f(a,b) && \text{rearranged} \\
&= \sum_{h : B \to \mathcal{S}A} \prod_{b \in B} \delta^{\mathcal{R}}{}_{\sum_{a \in A} h(b)(a), \beta(b)} \prod_{a \lhd h(b)} f(a,b) && +^{\mathcal{R}} \text{ distributes} \\
&= \sum_{\sigma \in \mathcal{S}(A \times B)} \prod_{b \in B} \delta^{\mathcal{R}}{}_{\sum_{a \in A} \sigma(a,b), \beta(b)} \prod_{a \in A} (f(a,b))^{\sigma(a,b)} && \text{rearranged} \\
&= \sum_{\sigma \in \mathcal{S}(A \times B)} \delta^{\mathcal{R}}{}_{\lambda b. \sum_{a \in A} \sigma(a,b), \beta} \prod_{b \in B} \prod_{a \in A} (f(a,b))^{\sigma(a,b)} && \text{rearranged} \\
&= \sum_{\alpha \in \mathcal{S}A} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \prod_{a,b \lhd \sigma} f(a,b) && \text{Definition}
\end{aligned}$$

□

Notation 6.1.11.
To improve readability, $\sum_{\sigma \in \mathcal{W}(A,\beta)} f(\sigma)$ is used as a syntactic sugar for $\sum_{\alpha \in \mathcal{S}A} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} f(\sigma)$.

Lemma 6.1.12.
Let $\mathcal{R}$ and $\mathcal{S}$ be semirings such that $\mathcal{S}$ is a multiplicity semiring, with an exponential action of $\mathcal{S}$ on $\mathcal{R}$ that has a coefficientless expansion. Given $\alpha : A \to \mathcal{S}, \beta : B \to \mathcal{S}, \tau : A \to \mathcal{R}, f : A \to B$, then

$$\sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \prod_{(a,b) \lhd \sigma} \tau(a) \cdot \delta^{\mathcal{R}}{}_{f(a),b} = \prod_{a \lhd \alpha} \tau(a) \cdot \delta^{\mathcal{R}}{}_{\beta, \mathcal{S}f(\alpha)}$$

Essentially, because $\tau(a) \cdot \delta_{f(a),b}$ is forcing a functional relation between the elements of α and β, there is at most one $\sigma \in \mathcal{W}(\alpha,\beta)$ where the value does not equal 0 – where there is a comparable functional relation between α and β, and all the elements are paired together according to the functional relation $\delta_{f(a),b}$.

Proof. The case where $\alpha = \beta = \chi^{\mathcal{S}}_{\emptyset}$ is trivial, as then both sides are equal to $0^{\mathcal{R}}$.

Otherwise, consider $\mathcal{W}_f(\alpha,\beta) = \{\sigma | \sigma \in \mathcal{W}(\alpha,\beta), \forall a \in A, b \in B, \sigma(a,b) \neq 0 \Rightarrow b = f(a)\}$

and $\mathcal{W}_{\neg f}(\alpha,\beta) = \mathcal{W}(\alpha,\beta)/\mathcal{W}_f(\alpha,\beta)$. Simple arithmetic shows that

$$\sum_{\sigma\in\mathcal{W}_{\neg f}(\alpha,\beta)} \prod_{a,b\lhd\sigma} \tau(a)\cdot {\delta^{\mathcal{R}}}_{f(a),b} = 0$$

Next we show that $\mathcal{W}_f(\alpha,\beta)\neq\emptyset \Leftrightarrow \beta = \mathrm{S}f(\alpha)$:

- $\Rightarrow$ Assume there exists $\sigma\in\mathcal{W}_f(\alpha,\beta)$, and $\beta\neq \mathrm{S}f(\alpha)$. By definition of $\mathcal{W}_f(\alpha,\beta)$, there would exist at least one $a\in A$ such that $\sigma(a,f(a))\neq 0$ and $\sum_{a\in A}\sigma(a,f(a))\neq\beta(f(a))$. This is a contradiction, as $\sigma\in\mathcal{W}(\alpha,\beta)$ implies that for all $b\in B$, $\sum_{a\in A}\sigma(a,b)=\beta(b)$.
- $\Leftarrow$ $\lambda(a,b).\alpha(a)\cdot {\delta^{\mathcal{S}}}_{f(a),b}\in\mathcal{W}_f(\alpha,\lambda b.\sum_{a\in A}\alpha(a)\cdot {\delta^{\mathcal{S}}}_{f(a),b})$.

In fact, $\mathcal{W}_f(\alpha,\mathrm{S}f(\alpha)) = \{\lambda(a,b).\alpha(a)\cdot {\delta^{\mathcal{S}}}_{f(a),b}\}$, a singleton set. We then have

$$\begin{aligned}\textstyle\sum_{\sigma\in\mathcal{W}_f(\alpha,\beta)}\prod_{a,b\lhd\sigma}\tau(a)\cdot {\delta^{\mathcal{R}}}_{f(a),b} &= \textstyle\prod_{(a,b)\in A\times B}(\tau(a))^{\alpha(a)\cdot {\delta^{\mathcal{S}}}_{f(a),b}}\cdot {\delta^{\mathcal{R}}}_{\beta,\mathrm{S}f(\alpha)}\\ &= \textstyle\prod_{a\lhd\alpha}\tau(a)\cdot {\delta^{\mathcal{R}}}_{\beta,\mathrm{S}f(\alpha)}\end{aligned}$$

We conclude, as $\mathcal{W}(\alpha,\beta)=\mathcal{W}_f(\alpha,\beta)\cup\mathcal{W}_{\neg f}(\alpha,\beta)$. □

The following are some additional technical lemmas to be used later on, with $\mathcal{R}$ having infinite sums, $\mathcal{S}$ being a multiplicity semiring, and S using finite support.

Lemma 6.1.13.
Given $\varphi\in\mathrm{R}A,\alpha\in\mathrm{S}A$ where $\alpha\neq\lambda a'.{\delta^{\mathcal{S}}}_{a,a'}$ for some $a\in A$, then $\mathcal{W}(\eta^{\mathrm{S}}_{\mathrm{R}A}(\varphi),\alpha)=\emptyset$.

Proof. By definition, $\sum_{\psi\in\mathrm{R}A}\eta^{\mathrm{S}}_{\mathrm{R}A}(\varphi)(\psi) = 1^{\mathcal{S}}$. As $\alpha\neq\lambda a'.{\delta^{\mathcal{S}}}_{a,a'}$ for some $a\in A$, either $\#(\mathrm{supp}(\alpha))\neq 1$, or there exists an $a\in A$ such that $\alpha=\lambda a'.{\delta^{\mathcal{S}}}_{a,a'}\cdot s$, for some $s\in\mathcal{S}, s\neq 1$. In the first case, as $\mathcal{S}$ is positive and discrete, we cannot have $\sum_{a\in A}\alpha(a)=1^{\mathcal{S}}$. In the second case, this is by definition. □

Lemma 6.1.14.
Let $\mathcal{S}$ be a multiplicity semiring with the corresponding monad S using finite support.

(1) Given $\varphi\in\mathrm{SS}A,\beta\in\mathrm{S}B,\sigma\in\mathcal{W}(\mu^{\mathrm{S}}_A(\varphi),\beta)$, there exist $\psi\in\mathrm{SS}B$, $\tau\in\mathcal{W}(\varphi,\psi)$, and $\rho\in\mathrm{SS}(A\times B)$ such that $\mu^{\mathrm{S}}_A(\psi)=\beta$, $\mu^{\mathrm{S}}_{A\times B}(\rho)=\sigma$, and for all $\sigma'\in\mathrm{S}(A\times B)$,

$$\rho(\sigma') = \tau(\lambda a.\sum_{b\in B}\sigma'(a,b),\lambda b.\sum_{a\in A}\sigma'(a,b))$$

(2) Given $\rho\in\mathrm{SS}(A\times B),\varphi\in\mathrm{SS}A,\beta\in\mathrm{S}B,\psi\in\mathrm{SS}B,\tau\in\mathcal{W}(\varphi,\psi)$, if $\mu^{\mathrm{S}}_A(\psi)=\beta$ and for all $\sigma\in\mathrm{S}(A\times B)$, $\rho(\sigma)=\tau(\lambda a.\sum_{b\in B}\sigma(a,b),\lambda b.\sum_{a\in A}\sigma(a,b))$, then we have $\mu^{\mathrm{S}}_{A\times B}(\rho)\in\mathcal{W}(\mu^{\mathrm{S}}_A(\varphi),\beta)$.

Proof.

(1) For each $a \in A$, we have

$$\sum_{\alpha \in \mathrm{S}A} \varphi(\alpha) \cdot \alpha(a) = \sum_{b \in B} \sigma(a,b)$$

By Lemma 6.1.5, for each $a \in A$ we can find a $\zeta(a) \in \mathcal{W}(\lambda\alpha.\varphi(\alpha) \cdot \alpha(a), \lambda b.\sigma(a,b))$. We then have (for each $\alpha \in \mathrm{S}A, a \in A$):

$$\varphi(\alpha) \cdot \alpha(a) = \sum_{b \in B} \zeta(a)(\alpha, b)$$

We apply the generalised form of multiplicative splitting (Lemma 6.1.7) for each $\alpha \in \mathrm{S}A$ to obtain $k \in \mathbb{N}, \wp(\alpha) \in \mathrm{S}\mathbb{N}, \flat(\alpha) \in \mathrm{S}(\mathbb{N} \times A \times B)$ such that:

$$\begin{array}{rrcl}
\forall \alpha \in \mathrm{S}A, & \sum_{n \in \mathbb{N}} \wp(\alpha)(n) & = & \varphi(\alpha), \\
\forall \alpha \in \mathrm{S}A, a \in A, n \in \mathbb{N}, & \sum_{b \in B} \flat(\alpha)(n,a,b) & = & \alpha(a), \\
\forall \alpha \in \mathrm{S}A, a \in A, b \in B, & \sum_{n \in \mathbb{N}} \wp(\alpha)(n) \cdot \flat(\alpha)(n,a,b) & = & \zeta(a)(\alpha,b), \\
\forall \alpha \in \mathrm{S}A, a \in A, b \in B, & \wp(\alpha)(n) = \flat(\alpha)(n,a,b) & = & 0, \text{if } n > k.
\end{array}$$

We define $\rho \in \mathrm{SS}(A \times B)$ as

$$\rho(\sigma') = \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\sigma', \lambda a', b'.\flat(\alpha')(n,a',b') \cdot \delta_{a,a'}}$$

Then we have

$$\begin{array}{rcl}
\mu^{\mathrm{S}}_{A \times B}(\rho)(a,b) & = & \sum_{\sigma' \in \mathrm{S}(A \times B)} \rho(\sigma') \cdot \sigma'(a,b) \\
& = & \sum_{\sigma' \in \mathrm{S}(A \times B)} \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\sigma', \lambda a', b'.\flat(\alpha')(n,a',b') \cdot \delta_{a,a'}} \cdot \sigma'(a,b) \\
& = & \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\alpha')(n) \cdot \flat(\alpha')(n,a',b) \cdot \delta_{a,a'} \\
& = & \sum_{\alpha', n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\alpha')(n) \cdot \flat(\alpha')(n,a,b) \\
& = & \sum_{\alpha' \in \mathrm{S}A} \zeta(a)(\alpha,b) = \sigma(a,b)
\end{array}$$

We define $\tau \in \mathrm{S}(\mathrm{S}A \times \mathrm{S}B), \psi \in \mathrm{SS}B$ as follows:

$$\begin{array}{rcl}
\tau(\alpha', \beta') & = & \sum_{n \in \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\lambda b. \sum_{a \in A} \flat(\alpha')(n,a,b), \beta'} \\
\psi(\beta') & = & \sum_{\alpha' \in \mathrm{S}A} \tau(\alpha', \beta')
\end{array}$$

Then we have $\tau \in \mathcal{W}(\varphi, \psi)$, as

$$\begin{aligned}\sum_{\beta' \in \mathrm{S}B} \tau(\alpha', \beta') &= \sum_{\beta', n \in \mathrm{S}B \times \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\lambda b. \sum_{a \in A} \flat(\alpha')(n,a,b), \beta'} \\ &= \sum_{n \in \mathbb{N}} \wp(\alpha')(n) \\ &= \varphi(\alpha)\end{aligned}$$

We also have $\mu_B^{\mathrm{S}}(\psi) = \beta$:

$$\begin{aligned}\mu_B^{\mathrm{S}}(\psi)(b) &= \sum_{\beta' \in \mathrm{S}B} \psi(\beta') \cdot \beta'(b) \\ &= \sum_{\alpha', \beta' \in \mathrm{S}A \times \mathrm{S}B} \tau(\alpha', \beta') \cdot \beta'(b) \\ &= \sum_{\alpha', \beta' \in \mathrm{S}A \times \mathrm{S}B} \sum_{n \in \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\lambda b. \sum_{a \in A} \flat(\alpha')(n,a,b), \beta'} \cdot \beta'(b) \\ &= \sum_{\alpha' \in \mathrm{S}A} \sum_{n \in \mathbb{N}} \wp(\alpha')(n) \cdot \sum_{a \in A} \flat(\alpha')(n, a, b) \\ &= \sum_{a \in A} \sum_{\alpha' \in \mathrm{S}A} \zeta(a)(\alpha, b) \\ &= \sum_{a \in A} \sigma(a, b) = \beta(b)\end{aligned}$$

Finally, we need to check that for all $\sigma' \in \mathrm{S}(A \times B)$,

$$\rho(\sigma') = \tau(\lambda a. \sum_{b \in B} \sigma(a, b), \lambda b. \sum_{a \in A} \sigma(a, b))$$

This is indeed the case:

$$\begin{aligned}\rho(\sigma') &= \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\alpha')(n) \cdot \delta_{\sigma', \lambda a', b'. \flat(\alpha')(n,a',b') \cdot \delta_{a,a'}} \\ &= \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\lambda a'. \sum_{b \in B} \flat(\alpha')(n, a, b) \cdot \delta_{a,a'})(n) \cdot \delta_{\sigma', \lambda a', b'. \flat(\alpha')(n,a',b') \cdot \delta_{a,a'}} \\ &= \sum_{\alpha', a, n \in \mathrm{S}A \times A \times \mathbb{N}} \wp(\lambda a'. \sum_{b \in B} \sigma(a', b))(n) \cdot \delta_{\sigma', \lambda a', b'. \flat(\alpha')(n,a',b') \cdot \delta_{a,a'}} \\ &= \sum_{a, n \in A \times \mathbb{N}} \wp(\lambda a'. \sum_{b \in B} \sigma(a', b))(n) \cdot \delta_{\sigma', \lambda a', b'. \flat(\lambda a'. \sum_{b \in B} \sigma(a', b))(n,a',b') \cdot \delta_{a,a'}} \\ &= \sum_{n \in \mathbb{N}} \wp(\lambda a. \sum_{b \in B} \sigma(a, b))(n) \cdot \delta_{\lambda b. \sum_{a \in A} \flat(\lambda a. \sum_{b \in B} \sigma(a,b))(n,a,b), \lambda b. \sum_{a \in A} \sigma(a,b)} \\ &= \tau(\lambda a. \textstyle\sum_{b \in B} \sigma(a, b), \lambda b. \textstyle\sum_{a \in A} \sigma(a, b))\end{aligned}$$

(2) This direction is comparatively significantly simpler. As $\tau \in \mathcal{W}(\varphi, \psi)$, we have

$$\begin{aligned}\text{for all } \alpha \in \mathrm{S}A, \quad \sum_{\beta' \in \mathrm{S}B} \tau(\alpha, \beta') = \varphi(\alpha), \\ \text{for all } \beta' \in \mathrm{S}B, \quad \sum_{\alpha \in \mathrm{S}A} \tau(\alpha, \beta') = \psi(\beta').\end{aligned}$$

By applying μ^{S} to both sides we get

$$\begin{aligned}\text{for all } a \in A,&\quad \sum_{\alpha,\beta' \in \mathrm{S}A \times \mathrm{S}B} \tau(\alpha,\beta') \cdot \alpha(a) = \mu^{\mathrm{S}}_A(\varphi)(a),\\ \text{for all } b \in B,&\quad \sum_{\alpha,\beta' \in \mathrm{S}A \times \mathrm{S}B} \tau(\alpha,\beta') \cdot \alpha(a) = \beta(b).\end{aligned}$$

As for all $(\alpha,\beta') \in \mathrm{S}A \times \mathrm{S}B$, $\tau(\alpha,\beta') \neq 0$ implies that $\sum_{a\in A} \alpha(a) = \sum_{b\in B} \beta'(b)$, we can use Lemma 6.1.5 to obtain

$$\begin{aligned}\text{for all } a \in A,&\quad \sum_{\sigma \in \mathrm{S}(A\times B)} \tau(\lambda a.\sum_{b\in B}\sigma(a,b), \lambda b.\sum_{a\in A}\sigma(a,b)) \cdot \sum_{b\in B}\sigma(a,b) = \mu^{\mathrm{S}}_A(\varphi)(a),\\ \text{for all } b \in B,&\quad \sum_{\sigma \in \mathrm{S}(A\times B)} \tau(\lambda a.\sum_{b\in B}\sigma(a,b), \lambda b.\sum_{a\in A}\sigma(a,b)) \cdot \sum_{a\in A}\sigma(a,b) = \beta(b).\end{aligned}$$

By substituting $\rho(\sigma) = \tau(\lambda a.\sum_{b\in B}\sigma(a,b), \lambda b.\sum_{a\in A}\sigma(a,b))$ and distributing the product over the sum, we are able to conclude. □

Lemma 6.1.15.
Given $\tau : A \to \mathrm{R}B, \alpha \in \mathrm{S}A, \beta \in \mathrm{S}B$*:*

$$\sum_{\sigma \in \mathcal{W}(\mathrm{S}\tau(\alpha),\beta)} \prod_{g,b\lhd\sigma} g(b) = \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{a,b\lhd\sigma} \tau(a)(b)$$

Proof. This is a simple application of the naturality of $\mathcal{W}$ (Lemma 6.1.4).

$$\begin{aligned}
\sum_{\sigma \in \mathcal{W}(\mathrm{S}\tau(\alpha),\beta)} \prod_{g,b\lhd\sigma} g(b) &= \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{g,b\lhd\mathrm{S}(\tau\times \mathrm{id}_B)(\sigma)} g(b) && \text{Lemma 6.1.4}\\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{g\in\mathrm{R}B} \prod_{b\in B} (g(b))^{\sum_{a\in A}\sigma(a,b)\cdot\delta^{\mathcal{S}}_{\tau(a),g}} && \text{Definition}\\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{a\in\mathrm{R}A} \prod_{g\in\mathrm{R}B} \prod_{b\in B} (g(b))^{\sigma(a,b)\cdot\delta^{\mathcal{S}}_{\tau(a),g}} && \text{Definition 6.1.1}\\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{a\in\mathrm{R}A} \prod_{b\in B} (\tau(a)(b))^{\sigma(a,b)} && g = \tau(a)\\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \prod_{a,b\lhd\sigma} \tau(a)(b) && \text{Definition}
\end{aligned}$$

□

Thus far, we have not really encountered idempotency, despite the title of this section. The upcoming theorem requires that $\mathcal{R}$ has an idempotent sum. This is crucial, as when we are summing over the elements of a set such as $\mathcal{W}(\alpha,\alpha')$, for some $\alpha,\alpha' \in \mathrm{S}A$, idempotency allows us to only consider whether a particular σ exists within $\mathcal{W}(\alpha,\alpha')$, not how many times a particular σ could be created from α and α'. This would not be the case with non-idempotent sums – there will be more on this in the next section, where we will tackle the non-idempotent case. For now, we have all the tools necessary to form new distributions in the idempotent case.

Theorem 6.1.16.
Let $\mathcal{R}$ be an arbitrary continuous commutative semiring such that $+^{\mathcal{R}}$ is idempotent. Let $\mathcal{S}$ be an arbitrary multiplicity semiring, with the corresponding monad S using finite support. Also consider fixed an exponential action of $\mathcal{S}$ on $\mathcal{R}$ with a coefficientless expansion. Under these conditions, $\mathcal{R}$ distributes over $\mathcal{S}$ to form the composite monad RS, with the distribution $\theta : \mathrm{SR} \to \mathrm{RS}$ defined as (for all $\varphi \in \mathrm{SR}A, \alpha \in \mathrm{S}A$):

$$\theta_A(\varphi)(\alpha) = \sum_{\sigma \in \mathcal{W}(\varphi,\alpha)} \prod_{(f,a) \lhd \sigma} f(a)$$

Proof of Theorem 6.1.16. We prove this by solving the equations of the 5 requisite diagrams.

- θ is a natural transformation: Given $\varphi \in \mathrm{SR}A, \beta \in \mathrm{S}B, f : A \to B$,

$$\begin{aligned}
\theta_B(\mathrm{SR}f(\varphi))(\beta) &= \sum_{\sigma \in \mathcal{W}(\mathrm{SR}f(\varphi),\beta)} \prod_{g,b \lhd \sigma} g(b) && \text{Definition} \\
&= \sum_{\sigma_1 \in \mathcal{W}(\varphi,\beta)} \prod_{\tau,b \lhd \sigma_1} \mathrm{R}f(\tau)(b) && \text{Lemma 6.1.15} \\
&= \sum_{\sigma_1 \in \mathcal{W}(\varphi,\beta)} \prod_{\tau,b \lhd \sigma_1} \sum_{a \in A} \tau(a) \cdot {\delta^{\mathcal{R}}}_{f(a),b} && \text{Definition} \\
&= \sum_{\sigma_1 \in \mathcal{W}(\varphi,\beta)} \sum_{\alpha \in \mathrm{S}A} \sum_{\sigma_2 \in \mathcal{W}(\alpha,\sigma_1)} \prod_{\tau,b,a \lhd \sigma_2} \tau(a) \cdot {\delta^{\mathcal{R}}}_{f(a),b} && \text{Definition 6.1.10} \\
&= \sum_{\sigma \in \mathcal{W}(\varphi,A)} \sum_{\sigma_2 \in \mathcal{W}(\sigma,\beta)} \prod_{\tau,b,a \lhd \sigma_2} \tau(a) \cdot {\delta^{\mathcal{R}}}_{f(a),b} && \begin{bmatrix}\text{Idempotent } +^{\mathcal{R}} \\ \text{Lemma 6.1.3}\end{bmatrix} \\
&= \sum_{\sigma \in \mathcal{W}(\varphi,A)} \cdot {\delta^{\mathcal{R}}}_{\mathrm{S}f(\alpha),\beta} \cdot \prod_{f,a \lhd \sigma} f(a) && \text{Lemma 6.1.12} \\
&= \sum_{\alpha \in \mathrm{S}A} \theta_A(\varphi)(\alpha) \cdot {\delta^{\mathcal{R}}}_{\mathrm{S}f(\alpha),\beta} && \text{Definition} \\
&= \mathrm{RS}f(\theta_A(\varphi))(\beta) && \text{Definition}
\end{aligned}$$

  In the fourth line, we rely on idempotency to allow us to adjust the sets we are summing over – thanks to idempotency, the sizes of the sets we sum over do not matter, as long as the content remains the same. We use Lemma 6.1.3 to guarantee that the content will not change, enabling the step.

- $\theta_A \circ \mathrm{S}\eta^{\mathrm{R}}_A = \eta^{\mathrm{R}}_A$: Given $\alpha, \alpha' \in \mathrm{S}A$,

$$\begin{aligned}
\theta(\mathrm{S}\eta^{\mathrm{R}}_A(\alpha))(\alpha') &= \sum_{\sigma \in \mathcal{W}(\mathrm{S}\eta^{\mathrm{R}}_A(\alpha),\alpha')} \prod_{f,a \lhd \sigma} f(a) && \text{Definition} \\
&= \sum_{\sigma \in \mathcal{W}(\alpha,\alpha')} \prod_{a,a' \lhd \sigma} \eta^{\mathrm{R}}_A(a)(a') && \text{Lemma 6.1.15} \\
&= \sum_{\sigma \in \mathcal{W}(\alpha,\alpha')} \prod_{a,a' \lhd \sigma} {\delta^{\mathcal{R}}}_{a,a'} && \text{Definition} \\
&= {\delta^{\mathcal{R}}}_{\alpha,\alpha'} \cdot \prod_{a \lhd \alpha} 1 && \text{Lemma 6.1.12}
\end{aligned}$$

$$\begin{aligned} &= \delta^{\mathcal{R}}{}_{\alpha,\alpha'} && \textstyle\prod_{a \triangleleft \alpha} 1 = 1 \\ &= \eta^{\mathrm{R}}_A(\alpha)(\alpha') && \text{Definition} \end{aligned}$$

Here we rely on $\mathcal{S}$ being positive. This is not immediately obvious – Lemma 6.1.15 relies on the naturality of $\mathcal{W}$, which requires a positive $\mathcal{S}$.

- $\theta_A \circ \eta^{\mathrm{S}}_{\mathrm{R}A} = \mathrm{R}\eta^{\mathrm{S}}_A$: Given $\varphi \in \mathrm{R}A, \alpha \in A$,

$$\begin{aligned} \theta(\eta^{\mathrm{S}}_{\mathrm{R}A}(\varphi))(\alpha) &= \sum_{a \in A} \delta_{\alpha,\lambda a'.\delta_{a,a'}} \cdot \theta(\eta^{\mathrm{S}}_{\mathrm{R}A}(\varphi))(\lambda a'.\delta_{a,a'}) && \text{Lemma 6.1.13} \\ &= \sum_{a \in A} \delta_{\alpha,\lambda a'.\delta_{a,a'}} \cdot \sum_{\sigma \in \mathcal{W}(\eta^{\mathrm{S}}_{\mathrm{R}A}(\varphi), \lambda a'.\delta_{a,a'})} \prod_{f, a \triangleleft \sigma} f(a) && \text{Definition} \\ &= \sum_{a \in A} \delta_{\alpha,\lambda a'.\delta_{a,a'}} \cdot \prod_{f, a \triangleleft \lambda(\varphi', a').\delta_{\varphi,\varphi'} \cdot \delta_{a,a'}} f(a) && \text{Singleton } \mathcal{W} \\ &= \sum_{a' \in A} \varphi(a') \cdot \delta^{\mathcal{R}}{}_{\alpha,\lambda a.\delta^{\mathcal{S}}{}_{a,a'}} && \text{Definition} \\ &= \mathrm{R}\eta^{\mathrm{S}}_A(\varphi)(\alpha) && \text{Definition} \end{aligned}$$

Here, we rely on the semiring $\mathcal{S}$ being discrete. Both of the previous two diagrams are used to show the consistency of the units when passing through the distribution. This involves a sort of "splitting and rearranging", where it is important that we can only do this in one way. If one considers the hypothetical comonad that would be induced by $\mathcal{S}$, these requirements are also quite intuitive: Using a resource a negative amount of times or a fraction of an amount of times makes no sense.

The basic properties of the exponential action are also linked to these two diagrams, ensuring a consistent mapping between the identities of $\mathcal{R}$ and $\mathcal{S}$. Going back to the comonad intuition, a resource which isn't present shouldn't have an affect, and if a resource is only present once it shouldn't change the value it produces. Meanwhile, 0 and 1 values should be untouched.

- $\mu^{\mathrm{R}}_{\mathrm{S}A} \circ \mathrm{R}\theta_A \circ \theta_A = \theta \circ \mathrm{S}\mu^{\mathrm{R}}_A$: Given $\varphi \in \mathrm{SRR}A, \alpha \in \mathrm{S}A$,

$$\begin{aligned} \mu^{\mathrm{R}}_{\mathrm{S}A}(\mathrm{R}\theta_A(\theta_A(\varphi)))(\alpha) &= \sum_{\psi \in \mathrm{RS}A} \mathrm{R}\theta_A(\theta_A(\varphi)) \cdot \psi(\alpha) && \text{Definition} \\ &= \sum_{\psi \in \mathrm{RS}A} \sum_{\rho \in \mathrm{SR}A} \theta_A(\varphi)(\rho) \cdot \delta^{\mathcal{R}}{}_{\theta_A(\rho),\psi} \cdot \psi(\alpha) && \text{Definition} \\ &= \sum_{\rho \in \mathrm{SR}A} \theta_A(\varphi)(\rho) \cdot \theta_A(\rho)(\alpha) && \psi = \theta_A(\rho) \\ &= \sum_{\rho \in \mathrm{SR}A} \left(\sum_{\sigma_l \in \mathcal{W}(\varphi,\rho)} \prod_{g, f \triangleleft \sigma_l} g(f) \right) \cdot \left(\sum_{\sigma_r \in \mathcal{W}(\rho,\alpha)} \prod_{f', a \triangleleft \sigma_r} f(a) \right) && \text{Definition} \\ &= \sum_{\rho \in \mathrm{SR}A} \sum_{\sigma_r \in \mathcal{W}(\rho,\alpha)} \sum_{\sigma_l \in \mathcal{W}(\varphi,\rho)} \left(\prod_{g, f \triangleleft \sigma_l} g(f) \right) \cdot \left(\prod_{f', a \triangleleft \sigma_r} f(a) \right) && +^{\mathcal{R}} \text{ distributes} \end{aligned}$$

$$= \sum_{\sigma_l \in \mathcal{W}(\varphi, \mathrm{S}\mathrm{R}A)} \sum_{\sigma^3 \in \mathcal{W}(\sigma_l, \alpha)} \left(\begin{array}{c} \left[\prod_{g \in \mathrm{R}\mathrm{R}A} \prod_{f \in \mathrm{R}A} (g(f))^{\sum_{a \in A} \sigma^3((g,f),a)} \right] \\ \cdot^{\mathcal{R}} \\ \left[\prod_{f \in \mathrm{R}A} \prod_{a \in A} (f(a))^{\sum_{g \in \mathrm{R}\mathrm{R}A} \sigma^3((g,f),a)} \right] \end{array} \right)$$ Lemma 6.1.6

$$= \sum_{\sigma_l \in \mathcal{W}(\varphi, \mathrm{S}\mathrm{R}A)} \sum_{\sigma^3 \in \mathcal{W}(\sigma_l, \alpha)} \prod_{(g,f), a \lhd \sigma^3} g(f) \cdot f(a)$$ Definition 6.1.1

$$= \sum_{\sigma \in \mathcal{W}(\varphi, \alpha)} \sum_{\sigma^3 \in \mathcal{W}(\mathrm{S}\mathrm{R}A, \sigma)} \prod_{f, (g,a) \lhd \sigma^3} g(f) \cdot f(a)$$ Lemma 6.1.3

$$= \sum_{\sigma \in \mathcal{W}(\varphi, \alpha)} \prod_{g, a \lhd \sigma} \sum_{f \in \mathrm{R}A} g(f) \cdot f(a)$$ Definition 6.1.8

$$= \sum_{\sigma \in \mathcal{W}(\varphi, \alpha)} \prod_{g, a \lhd \sigma} \mu_A^{\mathrm{R}}(g)(a)$$ Definition

$$= \sum_{\sigma \in \mathcal{W}(\mathrm{S}\mu_A^{\mathrm{R}}(\varphi), \alpha)} \prod_{f, a \lhd \sigma} f(a)$$ Lemma 6.1.15

$$= \theta_A(\mathrm{S}\mu_A^{\mathrm{R}}(\varphi))(\alpha)$$ Definition

This diagram could also be described as the "additive splitting diagram". Alongside said property, this diagram forces the remainder of the properties of the exponential action, along with the coefficientless expansion. The equation on the 7th line of the proof is the "purest" form of the equation – from our $\varphi \in \mathrm{S}\mathrm{R}\mathrm{R}A$ and $\alpha \in \mathrm{S}A$ we need to consider all arrangements of $g \in \mathrm{R}\mathrm{R}A$, $f \in \mathrm{R}A$, $a \in A$ from the contents of φ and α. This is the only way that the equation truly "makes sense" – the other properties are then forced to ensure that both halves meet this "middle ground".

- $\mathrm{R}\mu_A^{\mathrm{S}} \circ \theta_A \circ \mathrm{S}\theta_A = \theta_A \circ \mu_{\mathrm{R}A}^{\mathrm{S}}$: Given $\varphi \in \mathrm{S}\mathrm{S}\mathrm{R}A$, $\alpha \in \mathrm{S}A$,

$$\mathrm{R}\mu_A^{\mathrm{S}}(\theta_A(\mathrm{S}\theta_A(\varphi)))(\alpha) = \sum_{\psi \in \mu_A^{\mathrm{S}-1}(\alpha)} \sum_{\tau \in \mathcal{W}(\mathrm{S}\theta_A(\varphi), \psi)} \prod_{\zeta, \alpha_2 \lhd \tau} \zeta(\alpha_2)$$ Definition

$$= \sum_{\psi \in \mu_A^{\mathrm{S}-1}(\alpha)} \sum_{\tau \in \mathcal{W}(\varphi, \psi)} \prod_{\psi_2, \alpha_2 \lhd \tau} \sum_{\sigma \in \mathcal{W}(\psi_2, \alpha_2)} \prod_{f, a \lhd \sigma} f(a)$$ Lemma 6.1.15

$$= \sum_{\psi \in \mu_A^{\mathrm{S}-1}(\alpha)} \sum_{\tau \in \mathcal{W}(\varphi, \psi)} \prod_{\psi_2, \alpha_2 \lhd \tau} \sum_{\sigma \in \mathrm{S}(\mathrm{R}A \times A)} \chi^{\mathcal{R}}_{\mathcal{W}(\psi_2, \alpha_2)}(\sigma) \prod_{f, a \lhd \sigma} f(a)$$ Rearranged

$$= \sum_{\left\{ \psi, \tau \middle| \begin{smallmatrix} \psi \in \mu_A^{\mathrm{S}-1}(\alpha), \\ \tau \in \mathcal{W}(\varphi, \psi) \end{smallmatrix} \right\}} \sum_{\rho \in \mathcal{W}(\mathrm{S}\mathrm{S}(\mathrm{R}A \times A), \tau)} \prod_{\sigma, (\psi_2, \alpha_2) \lhd \rho} \chi^{\mathcal{R}}_{\mathcal{W}(\psi_2, \alpha_2)}(\sigma) \prod_{f, a \lhd \sigma} f(a)$$ Lemma 6.1.10

$$= \sum_{\left\{ \begin{smallmatrix} \psi, \\ \tau, \\ \rho \end{smallmatrix} \middle| \begin{smallmatrix} \psi \in \mu_A^{\mathrm{S}-1}(\alpha), \\ \tau \in \mathcal{W}(\varphi, \psi) \end{smallmatrix} \right\}} \left\{ \rho \middle| \begin{pmatrix} \rho \in \mathcal{W}(\mathrm{S}\mathrm{S}(\mathrm{R}A \times A), \tau), \\ \left(\forall (\sigma, (\psi_2, \alpha_2)) \in \mathrm{supp}(\rho), \right. \\ \left. \sigma \in \mathcal{W}(\psi_2, \alpha_2) \right) \end{pmatrix} \right\} \prod_{\sigma, (\psi_2, \alpha_2) \lhd \rho} \prod_{f, a \lhd \sigma} f(a)$$ Rearranged

$$= \sum_{\left\{ \begin{smallmatrix} \psi, \\ \tau \end{smallmatrix} \middle| \begin{smallmatrix} \psi \in \mu_A^{\mathrm{S}-1}(\alpha), \\ \tau \in \mathcal{W}(\varphi, \psi) \end{smallmatrix} \right\}} \left\{ \rho \middle| \begin{pmatrix} \rho \in \mathrm{S}\mathrm{S}(\mathrm{R}A \times A), \\ \left(\forall \sigma \in \mathrm{supp}(\rho) \ \exists (\psi_2, \alpha_2) \in \mathrm{supp}(\tau) \right. \\ \left. \text{s.t. } \sigma \in \mathcal{W}(\psi_2, \alpha_2),\ \rho(\sigma) = \tau(\psi_2, \alpha_2) \right) \end{pmatrix} \right\} \prod_{\sigma \lhd \rho} \prod_{f, a \lhd \sigma} f(a)$$ $+^{\mathcal{R}}$ idempotent

$$= \sum_{\left\{\psi,\tau \,\middle|\, \psi\in\mu_A^{\mathrm{S}-1}(\alpha),\ \tau\in\mathcal{W}(\varphi,\psi)\right\}} \sum_{\left\{\rho \,\middle|\, \rho\in\mathrm{SS}(\mathrm{R}A\times A),\forall\sigma\in\mathrm{S}(\mathrm{R}A\times A),\ \rho(\sigma)=\tau\left(\begin{smallmatrix}\lambda f.\ \sum_{a\in A}\sigma(f,a),\\ \lambda a.\ \sum_{f\in\mathrm{R}A}\sigma(f,a)\end{smallmatrix}\right)\right\}} \prod_{\sigma\lhd\rho}\prod_{f,a\lhd\sigma} f(a) \qquad \sigma\in\mathcal{W}(\psi_2,\alpha_2)$$

$$= \sum_{\left\{\begin{smallmatrix}\psi,\\ \tau,\\ \rho\end{smallmatrix} \,\middle|\, \begin{smallmatrix}\psi\in\mu_A^{\mathrm{S}-1}(\alpha),\tau\in\mathcal{W}(\varphi,\psi),\\ \rho\in\mathrm{SS}(\mathrm{R}A\times A),\forall\sigma\in\mathrm{S}(\mathrm{R}A\times A),\\ \rho(\sigma)=\tau\left(\begin{smallmatrix}\lambda f.\ \sum_{a\in A}\sigma(f,a),\\ \lambda a.\ \sum_{f\in\mathrm{R}A}\sigma(f,a)\end{smallmatrix}\right)\end{smallmatrix}\right\}} \prod_{\sigma,f,a\in\mathrm{S}(\mathrm{R}A\times A)\times\mathrm{R}A\times A} (f(a))^{\rho(\sigma)\cdot\sigma(f,a)} \qquad \text{Definition 6.1.1}$$

$$= \sum_{\left\{\begin{smallmatrix}\psi,\\ \tau,\\ \rho\end{smallmatrix} \,\middle|\, \begin{smallmatrix}\psi\in\mu_A^{\mathrm{S}-1}(\alpha),\tau\in\mathcal{W}(\varphi,\psi),\\ \rho\in\mathrm{SS}(\mathrm{R}A\times A),\forall\sigma\in\mathrm{S}(\mathrm{R}A\times A),\\ \rho(\sigma)=\tau\left(\begin{smallmatrix}\lambda f.\ \sum_{a\in A}\sigma(f,a),\\ \lambda a.\ \sum_{f\in\mathrm{R}A}\sigma(f,a)\end{smallmatrix}\right)\end{smallmatrix}\right\}} \prod_{f,a\in\mathrm{R}A\times A} (f(a))^{\sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times A)}\rho(\sigma)\cdot\sigma(f,a)} \qquad \text{Definition}$$

$$= \sum_{\left\{\psi,\tau \,\middle|\, \psi\in\mu_A^{\mathrm{S}-1}(\alpha),\ \tau\in\mathcal{W}(\varphi,\psi)\right\}} \sum_{\left\{\rho \,\middle|\, \rho\in\mathrm{SS}(\mathrm{R}A\times A),\forall\sigma\in\mathrm{S}(\mathrm{R}A\times A),\ \rho(\sigma)=\tau\left(\begin{smallmatrix}\lambda f.\ \sum_{a\in A}\sigma(f,a),\\ \lambda a.\ \sum_{f\in\mathrm{R}A}\sigma(f,a)\end{smallmatrix}\right)\right\}} \prod_{f,a\lhd\mu_A^{\mathrm{S}}(\rho)} f(a) \qquad \text{Definition}$$

$$= \sum_{\sigma\in\mathcal{W}(\mu_{\mathrm{R}A}^{\mathrm{S}}(\varphi),\alpha)} \prod_{f,a\lhd\sigma} f(a) = \theta_A(\mu_{\mathrm{R}A}^{\mathrm{S}}(\varphi))(\alpha) \qquad \left[\begin{array}{l}\text{Idempotent} +\\ \text{Lemma 6.1.14}\end{array}\right]$$

Though this is the most complex equation, it merely introduces the requirement for multiplicative splitting – all other properties are already present. Multiplicative splitting was originally introduced by the authors of [CES10] out of purely technical reasons [personal communication]. We start to get a better feeling for it here – though the presentation remains separate for familiarity, additive splitting and multiplicative splitting are in fact two sides of the same coin and may be better presented through a joint "multiplicative-additive splitting". We discuss this in more detail later – for now, consider that the semiring μ_A^{M} contains both an additive and multiplicative component. To "split and rearrange" this, we need to split both components. □

Conjecture 6.1.17.

There exists a definition of infinite product which, when used correctly together with the infinite sum, allows for the finiteness requirements to be removed without affecting the results to come.

6.2 Non-idempotent Sum

In the previous section, we have shown that it is possible to find a distribution between $\mathcal{R}$ and $\mathcal{S}$, under certain conditions. Forcing $\mathcal{S}$ to be a multiplicity semiring is due to the properties described in Lemmas 6.1.5 and 6.1.14 being necessary, along with the definition of witnesses in general. It is possible that there may be a more general definition which still allows these lemmas to be satisfied. The monad based on $\mathcal{S}$ being the finitely supported version may not be mandatory – a version without finite support would require $\mathcal{R}$ to have a well-defined notion of infinite product, but there are no other obvious changes to the requirements.

The previous results presented after Corollary 5.2.19 seem to imply that idempotency is not a requirement, as it shows that $\mathbb{N}$ is a (co)monad on $\mathbf{Set}_{\mathrm{R}}$ regardless of whether $\mathcal{R}$ has an idempotent sum. In the next section, we will show that idempotency is actually a requirement for the distribution (at least, in a form akin to the previous sections) – however, there is a way to obtain monads for the non-idempotent case despite the lack of a proper distribution.

6.2.1 Non-Idempotent Distributions are a Lie

Let us consider a particular case presented in the paper by Laird et. al. [LMMP13]: We set $\mathcal{R} = \bar{\mathbb{N}}$ and $\mathcal{S} = \mathbb{N}$, with the corresponding $\mathrm{R} = \bar{\mathrm{N}}$ monad using infinite sums and the $\mathrm{S} = \mathrm{N_f}$ monad using finite support. By representing multisets as unordered lists, we have (from the results in the paper)

$$!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}\varphi(\alpha, [b_1, \ldots, b_n]) = \sum_{\substack{(a_1,\ldots,a_n) \\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{i=1}^{n} \varphi(a_i, b_i)$$

Where $!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$ is the comonad produced when considering the Kleisli category $\mathbf{Set}_{\bar{\mathrm{N}}}$ as a Lafont category[1]. This comonad uses the same functor as $\mathrm{N_f}$, and when viewed as a monad (due to $(\mathbf{Set}_{\bar{\mathrm{N}}})^{\mathsf{op}} = \mathbf{Set}_{\bar{\mathrm{N}}}$) has identical unit and multiplication natural transformations to $\mathrm{N_f}$. This implies that there should be a distribution $\theta : \mathrm{N_f}\bar{\mathrm{N}} \to \bar{\mathrm{N}}\mathrm{N_f}$, from which $!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$ is constructed.

The odd presentation of the above equation is due to the tuple under the sum. Instead of $\mathcal{R} = \bar{\mathbb{N}}$, let us consider $\mathcal{R} = \mathrm{Tr}$, the tropical semiring. The above equation would then simplify to

$$!^{\mathrm{Tr}}_{\mathrm{N_f}}\varphi(m, [b_1, \ldots, b_n]) = \sum_{\substack{[a_1,\ldots,a_n] \\ \text{s.t. } [a_1]+\cdots+[a_n]=m}} \prod_{i=1}^{n} \varphi(a_i, b_i)$$

The tuple would become a multiset. Essentially, when the semiring chosen for $\mathcal{R}$ has an idempotent sum, we are keeping track of the ways that we can pair together the elements of two multisets.

[1] Lafont categories have not been defined, nor is the definition relevant to this work. For those unfamilar with the concept, it suffices to know that said comonad exists.

If the semiring has a non-idempotent sum, then we also need to keep track of the number of ways each pairing can occur. This is the intuitive explanation that was promised in the previous section for why non-idempotent sums significantly complicate the issue at hand.

For the remainder of this subsection, we will assume that the exponential action between $\bar{\mathbb{N}}$ and $\mathbb{N}$ is the common exponent of natural numbers.

Notation 6.2.1.
Given $\alpha : A \to \mathbb{N}$, let $\sqcup\!\sqcup(\alpha) = \prod_{a \in A} \mathsf{fact}(\alpha(a))$, where $\mathsf{fact}(i)$ is used to denote the factorial of i (as opposed to $!$ which may create confusion).

Intuitively, $\sqcup\!\sqcup(\alpha)$ (pronounced "shuffle α") is counting the number of internal rearrangements of α. This is *not* the number of tuples which can be created from a multiset, simply a measure of the "repetitions" in α. The relation to the tuples and justification for the usage of the notation is provided with the following proposition:

Proposition 6.2.2.
Let $\varphi : A \times B \to \mathbb{N}, \alpha : \mathrm{N_f}A, \beta : \mathrm{N_f}B$, and let us fix an ordering $(b_1, \ldots, b_n)$ for β (i.e. $\beta = [b_1] + \cdots + [b_n]$). Then

$$\sum_{\substack{(a_1,\ldots,a_n) \\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{i=1}^{n} \varphi(a_i, b_i) = \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)} \prod_{a,b \lhd \sigma} \varphi(a, b)$$

The $\frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)}$ part of the right hand side makes use of division of natural numbers – something which is undefined for semirings in general.

Proof. We will separate this proof into steps, to refer back to later.

(1) To begin, let us first rearrange the left half slightly.

$$\sum_{\substack{(a_1,\ldots,a_n) \\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{i=1}^{n} \varphi(a_i, b_i) = \sum_{\substack{(a_1,\ldots,a_n) \\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{a,b \lhd [(a_1,b_1),\ldots,(a_n,b_n)]} \varphi(a, b)$$

(2) As a reminder, given an $\alpha : \mathrm{N_f}A$, the number of permutations of α is $\frac{\mathsf{fact}(\sum_{a \in A} \alpha(a))}{\sqcup\!\sqcup(\alpha)}$ [Bru12]. In other words, we have

$$\sum_{\substack{(a_1,\ldots,a_n) \\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} 1 = \frac{\mathsf{fact}(\sum_{a \in A} \alpha(a))}{\sqcup\!\sqcup(\alpha)}$$

(3) As a consequence of (2), for any $n \in \mathbb{N}$ and any $f : \bar{\mathrm{N}}(A_1 \times \cdots \times A_n)$ (where for all $i \leq n$, $A_i = A$), if f is commutative with respect to its arguments (i.e. for all $a_1, \ldots, a_n \in A$ and

all bijective functions $g : \{1,\dots,n\} \to \{1,\dots,n\}$, $f(a_1,\dots,a_n) = f(a_{g(1)},\dots,a_{g(n)})$), then for all fixed orderings $(a'_1,\dots,a'_n)$ such that $[a'_1]+\cdots+[a'_n]=\alpha$, we have

$$\sum_{\substack{(a_1,\dots,a_n)\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} f(a_1,\dots,a_n) = \frac{\mathsf{fact}(\sum_{a\in A}\alpha(a))}{\sqcup\!\sqcup(\alpha)} f(a'_1,\dots,a'_n)$$

(4) Since the product and sum of $\mathbb{N}$ are commutative, the following function is also commutative:

$$(b_1,\dots,b_n) \to \sum_{\substack{(a_1,\dots,a_n)\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{a,b\lhd[(a_1,b_1),\dots,(a_n,b_n))]} \varphi(a,b)$$

(5) We make use of (4) to apply (3) to the right hand half of (1) to obtain

$$\sum_{\substack{(a_1,\dots,a_n)\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{i=1}^{n} \varphi(a_i,b_i) = \sum_{\substack{(b_1,\dots,b_n)\\ \text{s.t. } [b_1]+\cdots+[b_n]=\beta}} \frac{\sqcup\!\sqcup(\beta)}{\mathsf{fact}(\sum_{b\in B}\beta(b))} \sum_{\substack{(a_1,\dots,a_n)\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{a,b\lhd[(a_1,b_1),\dots,(a_n,b_n))]} \varphi(a,b)$$

Essentially, the above step is "dividing" by the ordering. We rearrange a bit to obtain

$$\sum_{\substack{(a_1,\dots,a_n)\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha}} \prod_{i=1}^{n} \varphi(a_i,b_i) = \sum_{\substack{((a_1,b_1),\dots,(a_n,b_n))\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha,\\ \text{s.t. } [b_1]+\cdots+[b_n]=\beta}} \frac{\sqcup\!\sqcup(\beta)}{\mathsf{fact}(\sum_{b\in B}\beta(b))} \prod_{a,b\lhd[(a_1,b_1),\dots,(a_n,b_n))]} \varphi(a,b)$$

We apply (3) again to obtain

$$\begin{aligned} &\sum_{\substack{((a_1,b_1),\dots,(a_n,b_n))\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha,\\ \text{s.t. } [b_1]+\cdots+[b_n]=\beta}} \frac{\sqcup\!\sqcup(\beta)}{\mathsf{fact}(\sum_{b\in B}\beta(b))} \prod_{a,b\lhd[(a_1,b_1),\dots,(a_n,b_n))]} \varphi(a,b) \\ = &\sum_{\substack{\sigma=[(a_1,b_1),\dots,(a_n,b_n)]\\ \text{s.t. } [a_1]+\cdots+[a_n]=\alpha,\\ \text{s.t. } [b_1]+\cdots+[b_n]=\beta}} \frac{\mathsf{fact}(\sum_{a,b\in A\times B}\sigma(a,b))}{\sqcup\!\sqcup(\sigma)} \frac{\sqcup\!\sqcup(\beta)}{\mathsf{fact}(\sum_{b\in B}\beta(b))} \prod_{a,b\lhd\sigma} \varphi(a,b) \end{aligned}$$

For $\sigma \in \mathcal{W}(\alpha,\beta)$ we have $(\sum_{a,b\in A\times B}\sigma(a,b)) = (\sum_{b\in B}\beta(b))$, so by the definition of $\mathcal{W}(\alpha,\beta)$ the above is equal to $\sum_{\sigma\in\mathcal{W}(\alpha,\beta)} \frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)} \prod_{a,b\lhd\sigma} \varphi(a,b)$ and we conclude. □

The multinomial coefficient mentioned previously also reappears here. As a reminder, the multinomial expansion rule for the natural numbers is (for $n\in\mathbb{N}, f : A \to \mathbb{N}, \#(\mathrm{supp}(f)) \in \mathbb{N}$):

$$(\sum_{a\in A} f(a))^n = \sum_{\left\{\alpha \middle| \alpha\in \mathrm{N_f}A, \sum_{a\in A}\alpha(a)=n\right\}} \frac{\mathsf{fact}(n)}{\sqcup\!\sqcup(\alpha)} \prod_{a\lhd\alpha} f(a)$$

Proposition 6.2.3.
The multinomial expansion rule for the natural numbers can be generalised to the following (for $\beta : \mathrm{N_f}B, f : A \to \mathbb{N}, \#(\mathrm{supp}(f)) \in \mathbb{N}$*):*

$$\prod_{b \lhd \beta} \sum_{a \in A} f(a) = \sum_{\alpha \in \mathrm{N_f}A} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)} \prod_{a,b \lhd \sigma} f(a)$$

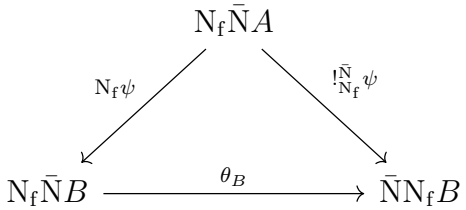
Proof.

$$\begin{aligned}
\prod_{b \lhd \beta} \sum_{a \in A} f(a) &= \prod_{b \in B} \sum_{\left\{\alpha \middle| \alpha \in \mathrm{N_f}A, \sum_{a \in A} \alpha(a) = \beta(b)\right\}} \frac{\mathsf{fact}(\beta(b))}{\sqcup\!\sqcup(\alpha)} \prod_{a \lhd \alpha} f(a) && \text{Definition} \\
&= \prod_{b \in B} \sum_{\alpha \in \mathrm{N_f}A} \delta^{\mathbb{N}}_{\sum_{a \in A} \alpha(a), \beta(b)} \frac{\mathsf{fact}(\beta(b))}{\sqcup\!\sqcup(\alpha)} \prod_{a \lhd \alpha} f(a) && \text{Rearranging} \\
&= \sum_{h : B \to \mathrm{N_f}A} \prod_{b \in B} \delta^{\mathbb{N}}_{\sum_{a \in A} h(b)(a), \beta(b)} \frac{\mathsf{fact}(\beta(b))}{\sqcup\!\sqcup(h(b))} \prod_{a \lhd h(b)} f(a) && + \text{ distributes} \\
&= \sum_{\sigma \in \mathrm{N_f}(A \times B)} \prod_{b \in B} \delta^{\mathbb{N}}_{\lambda b. \sum_{a \in A} \sigma(a,b), \beta} \frac{\mathsf{fact}(\beta(b))}{\sqcup\!\sqcup(\lambda a.\sigma(a,b))} \prod_{a \in A} (f(a))^{\sigma(a,b)} && \text{Rearranging} \\
&= \sum_{\sigma \in \mathrm{N_f}(A \times B)} \delta^{\mathbb{N}}_{\lambda b. \sum_{a \in A} \sigma(a,b), \beta} \frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)} \prod_{a,b \lhd \sigma} f(a) && \text{Rearranging} \\
&= \sum_{\alpha \in \mathrm{N_f}A} \sum_{\sigma \in \mathcal{W}(\alpha,\beta)} \frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)} \prod_{a,b \lhd \sigma} f(a) && \text{Rearranging}
\end{aligned}$$

□

When considering $\frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)}$ as it appears in Proposition 6.2.2, note that α, β, and σ are in $\mathrm{N_f}$, yet the result of the fraction is in $\bar{\mathrm{N}}$. The coefficient transforms the types from one to another, which is hard to spot due to the similarity in their types. If one was to consider other non-idempotent $\mathcal{R}$ candidates with the same $\mathcal{S} = \mathrm{N_f}$, one could write $\sum_{i=1}^{\frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)}}$ instead of $\frac{\sqcup\!\sqcup(\beta)}{\sqcup\!\sqcup(\sigma)}$.

Let us assume now that there exists a distribution $\theta : \mathrm{N_f}\bar{\mathrm{N}} \to \bar{\mathrm{N}}\mathrm{N_f}$, from which $!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$ is constructed. Then the following diagram would commute:



Proposition 6.2.4.
The comonad $!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$ *is not constructed from a distribution* $\theta : \mathrm{N_f}\bar{\mathrm{N}} \to \bar{\mathrm{N}}\mathrm{N_f}$.

Proof. Let us assume that such a θ exists. By setting $\varphi = \mathrm{id}_{\bar{\mathrm{N}}A}$ the identity function in $\mathbf{Set}$ in the previous diagram, we obtain $\mathrm{N_f}\mathrm{id}_{\bar{\mathrm{N}}A}; \theta_A = \theta_A = !^{\bar{\mathrm{N}}}_{\mathrm{N_f}}\mathrm{id}_{\bar{\mathrm{N}}A}$. We can make use of this to "recover" what such a θ would be:

$$\begin{array}{rcl} \theta_A(\varphi)(\alpha) & = & !^{\bar{\mathrm{N}}}_{\mathrm{N_f}}\mathrm{id}_{\bar{\mathrm{N}}A}(\varphi)(\alpha) \\ & = & \sum_{\sigma\in\mathcal{W}(\varphi,\alpha)} \frac{\sqcup(\alpha)}{\sqcup(\sigma)} \prod_{f,a\lhd\sigma} \mathrm{id}_{\bar{\mathrm{N}}A}(f)(a) \\ & = & \sum_{\sigma\in\mathcal{W}(\varphi,\alpha)} \frac{\sqcup(\alpha)}{\sqcup(\sigma)} \prod_{f,a\lhd\sigma} f(a) \end{array}$$

This definition of θ mirrors the definition used in the idempotent case. This process is, in fact, how the definition of θ for the idempotent case was identified.

Next, consider the following choices:

$$A = \{a_1, a_2\},\ B = \{*\},$$
$$\psi = \lambda * .1^{\mathbb{N}},\ \varphi = [a_1, a_2],\ \alpha = [*, *]$$

Plugging these into $!^{\bar{\mathrm{N}}}_{\mathrm{N_f}}$, we obtain the following:

$$\begin{array}{rcl} !^{\bar{\mathrm{N}}}_{\mathrm{N_f}}\psi(\varphi)(\alpha) & = & \sum_{\sigma\in\mathcal{W}(\varphi,\alpha)} \frac{\sqcup(\alpha)}{\sqcup(\sigma)} \prod_{f,a\lhd\sigma} \psi(f)(a) \\ & = & \frac{\sqcup([*,*])}{\sqcup([(a_1,*),(a_2,*)])} \cdot \psi(a_1)(*) \cdot \psi(a_2)(*) \\ & = & 2 \cdot \psi(a_1)(*) \cdot \psi(a_2)(*) = 2 \end{array}$$

while with $\theta_A(\mathrm{N_f}\psi(\varphi))(\alpha)$ we obtain a different result:

$$\begin{array}{rcl} \theta_A(\mathrm{N_f}\psi(\varphi))(\alpha) & = & \sum_{\sigma\in\mathcal{W}(\mathrm{N_f}\psi(\varphi),\alpha)} \frac{\sqcup(\alpha)}{\sqcup(\sigma)} \prod_{f,a\lhd\sigma} f(a) \\ & = & \sum_{\sigma\in\mathcal{W}([[*],[*]],[*,*])} \frac{\sqcup([*,*])}{\sqcup(\sigma)} \prod_{f,a\lhd\sigma} f(a) \\ & = & \frac{\sqcup([*,*])}{\sqcup([([*],*),([*],*)])} \cdot 1 \cdot 1 = 1 \end{array}$$

□

Notation 6.2.5.
Let us denote $\frac{\alpha \| \beta}{\sigma} = \frac{\sqcup(\beta)}{\sqcup(\sigma)} \cdot \chi^{\mathcal{R}}_{\mathcal{W}(\alpha,\beta)}(\sigma)$.

Proposition 6.2.6.
$\frac{- \| -}{-}$ *is not a natural transformation.*

Proof. By contradiction. Let $\bot : A \to *$ be the terminal arrow. If $\frac{- \| -}{-}$ was a natural transformation, then $\frac{\mathrm{N_f}\bot(\alpha) \| \beta}{\sigma} = \bar{\mathrm{N}}\mathrm{N_f}(\bot \times \mathrm{id})(\lambda\tau.\frac{\alpha\|\beta}{\tau})(\sigma)$. Consider $\alpha = [a_1, a_2], \beta = [b, b], \sigma = [(*, b), (*, b)]$. Then $\frac{\mathrm{N_f}\bot(\alpha) \| \beta}{\sigma} = \frac{\sqcup(\beta)}{\sqcup(\sigma)} = \frac{2}{2} = 1$, while

$$\begin{array}{rcl} \bar{\mathrm{N}}\mathrm{N_f}(\bot \times \mathrm{id})(\lambda\tau.\frac{\alpha\|\beta}{\tau})(\sigma) & = & \sum_{\tau\in\mathrm{N_f}(A\times B)} \frac{\alpha\|\beta}{\tau} \cdot \delta_{\lambda(*,b)\sum_{a\in A}\tau(a,b),\sigma} \\ & = & \frac{[a_1,a_2]\|[b,b]}{[(a_1,b),(a_2,b)]} \\ & = & 2 \end{array}$$

□

The above proposition is the underlying reason for why $\bar{\mathrm{N}}\mathrm{N}_\mathrm{f}$ is not a monad on $\mathbf{Set}$. We are lead to the following conjecture:

Conjecture 6.2.7.
Given two semirings $\mathcal{R}, \mathcal{S}$ where $\mathcal{R}$ is continuous and non-idempotent, if R is the corresponding semiring monad of $\mathcal{R}$ using infinite support and S is a corresponding semiring monad of $\mathcal{S}$, then there cannot exist a distribution $\theta : \mathrm{SR} \to \mathrm{RS}$.[2][3]

Despite the above, we know that N_f is a monad on $\mathbf{Set}$, and we obtain $!_{\mathrm{N}_\mathrm{f}}$ from N_f, which is a monad (and comonad) on $\mathbf{Set}_{\bar{\mathrm{N}}}$. We will now attempt to identify the method by which we obtain $!_{\mathrm{N}_\mathrm{f}}$ from N_f, and in the process of doing so discover what is happening at a more abstract level.

[2]It is known that the multiset monad distributes over the multiset monad [MM07]. Though this has not been put into context with these results, it seems a reasonable conjecture that finiteness is crucial for that case.

[3]There is a sort of duality here with recent results from Maaike Zwart and Dan Marsden [ZM22]. They show cases where a distribution is impossible, relying on a notion of idempotency for at least one of the monads. Meanwhile, we require idempotency for distributions to exist in our case.

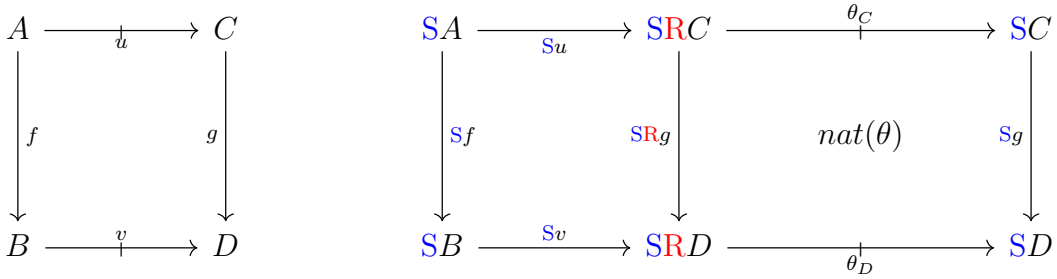
Chapter 7

Unnatural Distributions

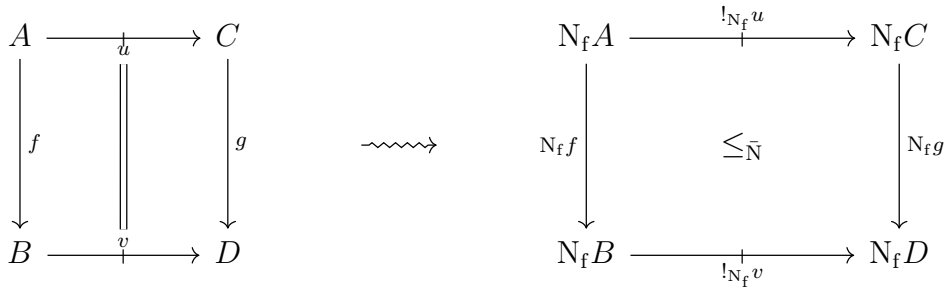
The work presented here is a part of work in progress and subject to mistakes – what is presented is the fragment which is expected to work, but the presentation of it is quite horrible. It is expected that continued investigations would allow this content to be presented in a cleaner way.

7.1 Unnatural Distributive Laws

In the previous chapter, we discovered that $!_{\mathrm{N_f}}$ does not arise from the monad $\mathrm{N_f}$ and a distribution. The underlying cause is that the supposed distribution which would be obtained is not natural. However, we can still find a pattern in its "unnaturality". Consider for a moment arbitrary monads R, S, with a distributive law $\theta : \mathrm{SR} \to \mathrm{RS}$. Given the below left hand diagram, we can then form the right hand diagram by naturality (Reminder: we write $u : A \longrightarrow\!\!\!\!\!\!+\; C$ for $u \in \mathbb{C}_{\mathrm{R}}(A, C)$).



In the case of $\mathrm{N_f}$ and $!_{\mathrm{N_f}}$, what we have instead is an ordering in the right hand diagram.



This inequality can in fact be quantified, and thus corrected. To do so, we first prove a minor technical lemma.

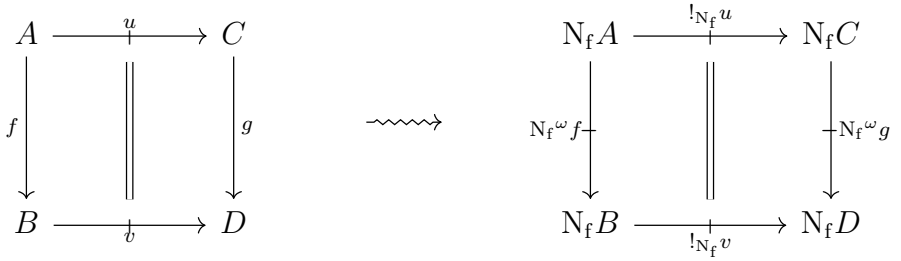
Lemma 7.1.1.
Let $\alpha : \mathrm{N_f}A, \gamma : \mathrm{N_f}C, f : A \to B$. *Then* $\#(\mathsf{supp}(\mathcal{W}(\alpha,\gamma))) = \frac{\sqcup\!\sqcup(\mathrm{N_f}f(\alpha))}{\sqcup\!\sqcup(\alpha)} \cdot \#(\mathsf{supp}(\mathcal{W}(\mathrm{N_f}f(\alpha),\gamma)))$.

Proof. Let us fix an order for γ: We consider $(c_1,\dots,c_n)$ such that $\sum_{i=1}^n [c_i] = \gamma$. By definition, $\{[(a_1,c_1),\dots,(a_n,c_n)] | \sum_{i=1}^n [a_i] = \alpha\} = \mathcal{W}(\alpha,\gamma)$. The number of permutations of $\mathrm{N_f}f(\alpha)$ is $\frac{\sum_{b\in B}\mathrm{N_f}f(\alpha)(b)}{\sqcup\!\sqcup(\mathrm{N_f}f(\alpha))}$, while the number of permutations of α is $\frac{\sum_{a\in A}\alpha(a)}{\sqcup\!\sqcup(\alpha)}$. Thus, $\#(\mathsf{supp}(\mathcal{W}(\alpha,\gamma))) = \frac{\sum_{a\in A}\alpha(a)}{\sqcup\!\sqcup(\alpha)}$ and $\#(\mathsf{supp}(\mathcal{W}(\mathrm{N_f}f(\alpha),\gamma))) = \frac{\sum_{b\in B}\mathrm{N_f}f(\alpha)(b)}{\sqcup\!\sqcup(\mathrm{N_f}f(\alpha))}$. Divide one by the other. □

We now move on to the correction of the inequality. Let us define the functor $\mathrm{N_f}^\omega : \mathbf{Set} \to \mathbf{Set}_{\bar{\mathbb{N}}}$:

$$\begin{array}{rrcl} \forall A \in \mathrm{Ob}(\mathbf{Set}), & \mathrm{N_f}^\omega A & = & \mathrm{N_f}A \\ \forall f \in \mathbf{Set}(A,B), & \mathrm{N_f}^\omega f(\alpha,\beta) & = & \eta_B^{\bar{\mathbb{N}}}(\mathrm{N_f}f(\alpha))(\beta) \cdot \frac{\sqcup\!\sqcup \mathrm{N_f}f(\alpha)}{\sqcup\!\sqcup \alpha} \end{array}$$

Proposition 7.1.2.
Given the left hand below commuting diagram, the below right hand diagram commutes.



Proof. Let $f : A \to B, g : C \to D, u : A \to \bar{\mathbb{N}}C, v : B \to \bar{\mathbb{N}}D$ such that for all $a \in A, d \in D$, $v(f(a),d) = \sum_{c\in C} u(a,c) \cdot \delta^{\mathbb{N}}_{g(c),d}$.

$$\begin{aligned}
\sum_{\beta\in\mathrm{N_f}B} !_{\mathrm{N_f}}v(\beta)(\tau) \cdot \mathrm{N_f}^\omega f(\alpha)(\beta) &= !_{\mathrm{N_f}}v(\mathrm{N_f}f(\alpha))(\tau) \cdot \frac{\sqcup\!\sqcup(\mathrm{N_f}f(\alpha))}{\sqcup\!\sqcup(\alpha)} && \text{Definition} \\
&= \sum_{\sigma\in\mathcal{W}(\mathrm{N_f}f(\alpha),\tau)} \frac{\sqcup\!\sqcup(\mathrm{N_f}f(\alpha)) \cdot \sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\alpha)\cdot\sqcup\!\sqcup(\sigma)} \prod_{b,d\lhd\sigma} v(b)(d) && \text{Definition} \\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\tau)} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\sigma)} \prod_{a,d\lhd\sigma} v(f(a))(d) && \text{Lemma 7.1.1} \\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\tau)} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\sigma)} \prod_{a,d\lhd\sigma} \sum_{c\in C} u(a,c)\cdot\delta^{\mathbb{N}}_{g(c),d} && \text{Definition} \\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\tau)} \sum_{\gamma\in\mathrm{N_f}C} \sum_{\rho\in\mathcal{W}(\gamma,\sigma)} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\rho)} \prod_{c,a,d\lhd\rho} u(a,c)\cdot\delta_{g(c),d} && \text{Proposition 6.2.3} \\
&= \sum_{\sigma\in\mathcal{W}(\alpha,\tau)} \sum_{\gamma\in\mathrm{N_f}C} \delta_{\mathrm{N_f}g(\gamma),\tau} \sum_{\left\{\rho \middle| \begin{smallmatrix}\rho\in\mathcal{W}(\gamma,\sigma),\\ \rho(c,a,d)\neq 0\\ \Rightarrow d=g(c)\end{smallmatrix}\right\}} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\rho)} \prod_{c,a,d\lhd\rho} u(a,c) && g(c)=d
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\gamma\in \mathrm{N_f}C} \delta_{\mathrm{N_f}g(\gamma),\tau} \sum_{\sigma_2\in\mathcal{W}(\alpha,\gamma)} \sum_{\left\{\rho \middle| \substack{\rho\in\mathcal{W}(\tau,\sigma),\\ \rho(d,a,c)\neq 0\\ \Rightarrow d=g(c)}\right\}} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\rho)} \prod_{d,a,c\lhd\rho} u(a,c) && \text{Lemma 6.1.3}\\
&= \sum_{\gamma\in \mathrm{N_f}C} \delta_{\mathrm{N_f}g(\gamma),\tau} \sum_{\sigma_2\in\mathcal{W}(\alpha,\gamma)} \frac{\sqcup\!\sqcup(\tau)}{\sqcup\!\sqcup(\sigma_2)} \prod_{a,c\lhd\sigma_2} u(a)(c) && \text{Isomorphic sets}\\
&= \sum_{\gamma\in \mathrm{N_f}C} \delta_{\mathrm{N_f}g(\gamma),\tau} \sum_{\sigma_2\in\mathcal{W}(\alpha,\gamma)} \frac{\sqcup\!\sqcup(\mathrm{N_f}g(\gamma))}{\sqcup\!\sqcup(\sigma_2)} \prod_{a,c\lhd\sigma_2} u(a)(c) && \mathrm{N_f}g(\gamma)=\tau\\
&= \sum_{\gamma\in \mathrm{N_f}C} \delta_{\mathrm{N_f}g(\gamma),\tau} \sum_{\sigma_2\in\mathcal{W}(\alpha,\gamma)} \frac{\sqcup\!\sqcup(\mathrm{N_f}g(\gamma))\cdot\sqcup\!\sqcup(\gamma)}{\sqcup\!\sqcup(\gamma)\cdot\sqcup\!\sqcup(\sigma_2)} \prod_{a,c\lhd\sigma_2} u(a)(c) && \text{Arithmetic}\\
&= \sum_{\gamma\in \mathrm{N_f}C} !_{\mathrm{N_f}}u(\alpha)(\gamma)\cdot\delta_{\mathrm{N_f}g(\gamma),\tau}\cdot\frac{\sqcup\!\sqcup(\mathrm{N_f}g(\gamma))}{\sqcup\!\sqcup(\gamma)} && \text{Definition} \quad \square
\end{aligned}
$$

Note that we also have $\mathrm{N_f}^{\omega} f = !_{\mathrm{N_f}}(f;\eta^{\mathrm{R}}_B)$, which would result in a trivial presentation of the above proof.

Normally, distributions between monads are defined using a distributive law $\theta : \mathrm{SR}\to\mathrm{RS}$, which allows us to identify the monad $!_{\mathrm{S}}$. In $\bar{\mathrm{N}}$ and $\mathrm{N_f}$ we have two monads which do not distribute, yet *almost* distribute. Consider also the following proposition:

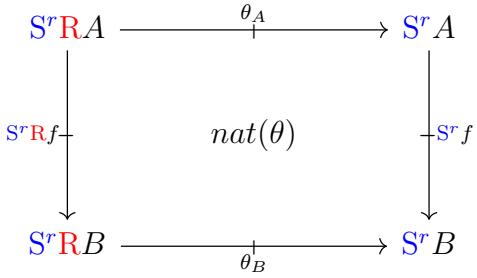
Proposition 7.1.3.
Let R *be a monad on a category* $\mathbb{C}$*. All endofunctors* $F:\mathbb{C}_{\mathrm{R}}\to\mathbb{C}_{\mathrm{R}}$ *arise from the composition of a related functor* $F^r:\mathbb{C}\to\mathbb{C}_{\mathrm{R}}$ *with a distributing natural transformation in* $\mathbb{C}_{\mathrm{R}}$*,* $\theta : F^r\mathrm{R}\to\mathrm{R}F^r$*.*

Proof. Let R be a monad on a category $\mathbb{C}$, and let $F:\mathbb{C}_{\mathrm{R}}\to\mathbb{C}_{\mathrm{R}}$. Let $T\dashv U$ be adjoint functors forming R with respect to the Kleisli category, i.e. $T:\mathbb{C}\to\mathbb{C}_{\mathrm{R}}, U:\mathbb{C}_{\mathrm{R}}\to\mathbb{C}, T;U=\mathrm{R}$. The morphism $\mathrm{id}_{\mathrm{R}A}:\mathrm{R}A\to\mathrm{R}A$ (where id is the identity morphism of $\mathbb{C}$) is clearly an element of $\mathbb{C}_{\mathrm{R}}(\mathrm{R}A,A)$, and we apply F to it to obtain $F(\mathrm{id}_{\mathrm{R}A})=\theta_A : F\mathrm{R}A\to\mathrm{R}FA$. Meanwhile, we set $F^r = T;F$. That $F^r;\theta = F$ for morphisms in $\mathbb{C}_{\mathrm{R}}$ is given by functoriality of F. $\square$

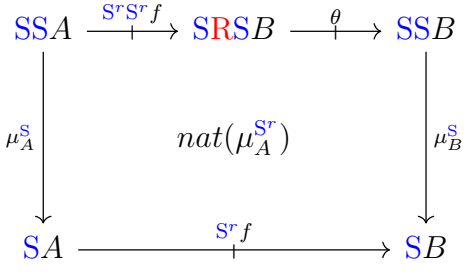
By building on the above, we can define new notion of a distribution matching the scenario before us. When we have two monads R,S such that RS is not a monad, yet there exists a monad $?_{\mathrm{S}}$ on $\mathbb{C}_{\mathrm{R}}$ that is adjacent to S (having the same objects, unit, and multiplication), then we say that R and S *unnaturally distribute*, and $?_{\mathrm{S}}$ is the monad arising from said unnatural distribution. We can define the corresponding notion of an *unnatural distributive law* to identify unnatural distributions.

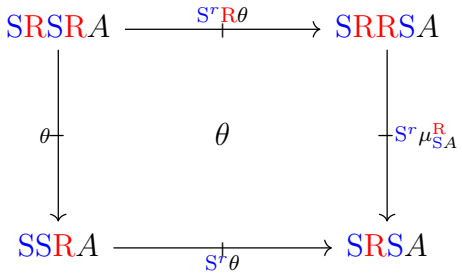
Definition 7.1.4 (Unnatural Distributive Law)**.**
Given two monads R,S *on* $\mathbb{C}$*, an unnatural distributive law of* R *over* S *consists of a functor* $\mathrm{S}^r:\mathbb{C}\to\mathbb{C}_{\mathrm{R}}$ *taking* $f:A\to B\in\mathbf{Set}$ *to* $\mathrm{S}^r f:\mathrm{S}A\to\mathrm{RS}B\in\mathbb{C}_{\mathrm{R}}$*, and a morphism* $\theta_A:\mathrm{S}^r\mathrm{R}\to\mathrm{RS}^r$ *such that*

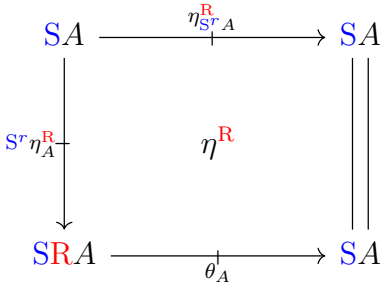
- *for all* $A \in \mathrm{Ob}(\mathbb{C}), \mathrm{S}^r A = \mathrm{S}A$*;*
- θ_A *is a natural transformation in* $\mathbb{C}_{\mathrm{R}}$*, i.e. the following diagram should commute (for all* $f : A \to B$*):*

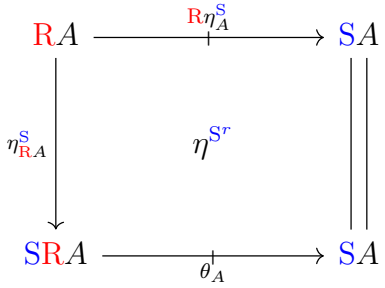


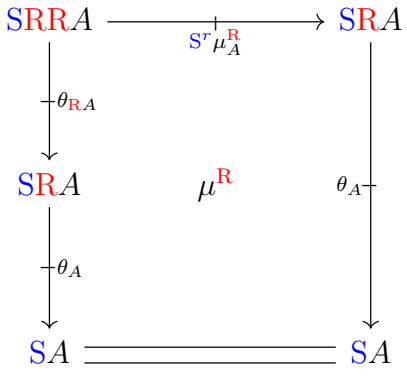
- *all of the following diagrams commute:*

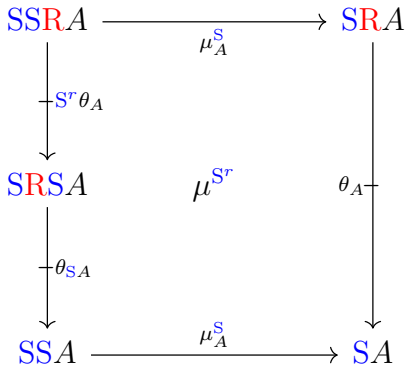


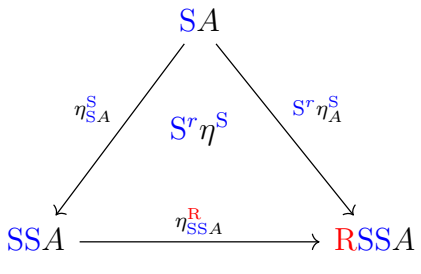


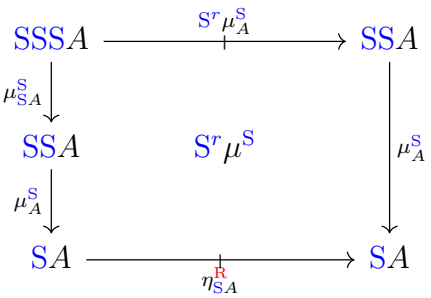










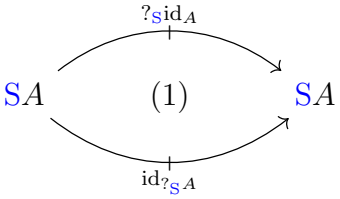


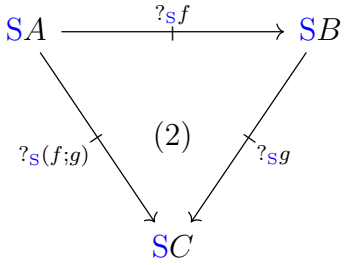
Theorem 7.1.5.
Let $\mathbb{C}$ be a category, and let R, S be monads on $\mathbb{C}$ such that there exists an unnatural distribution (S^r, θ_A) of R over S. Then there exists a monad $?_{\mathrm{S}}$ on $\mathbb{C}_{\mathrm{R}}$ acting as follows:

$$\begin{aligned} \forall A \in \mathrm{Ob}(\mathbb{C}_{\mathrm{R}}), \quad ?_{\mathrm{S}}A &= \mathrm{S}A \\ \forall \varphi \in \mathbb{C}_{\mathrm{R}}(A,B), \quad ?_{\mathrm{S}}\varphi &= \mathrm{S}^r\varphi ; \mathrm{R}\theta \\ \eta^{?_{\mathrm{S}}} &= \eta^{\mathrm{R}} \circ \eta^{\mathrm{S}} = \eta^{\mathrm{R}} ; \mathrm{R}\eta^{\mathrm{S}} \\ \mu^{?_{\mathrm{S}}} &= \eta^{\mathrm{R}} \circ \mu^{\mathrm{S}} = \eta^{\mathrm{R}} ; \mathrm{R}\mu^{\mathrm{S}} \end{aligned}$$

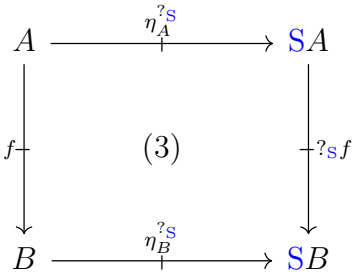
Proof. We will prove this diagrammically. We have the following diagrams (in $\mathbf{Set}_{\mathrm{R}}$) to check:

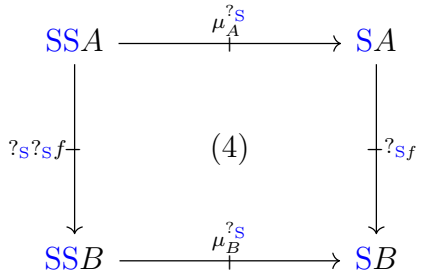
- Diagrams showing the functoriality of $?_{\mathrm{S}}$:



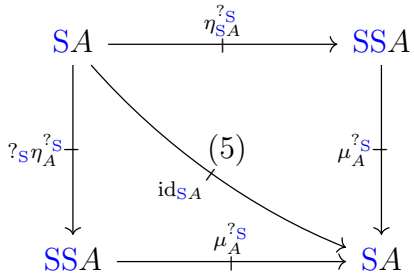


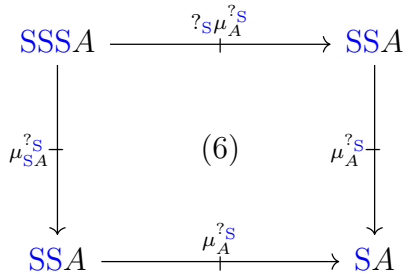
- Diagrams showing the naturality of $\eta^{?_{\mathrm{S}}}$ and $\mu^{?_{\mathrm{S}}}$:





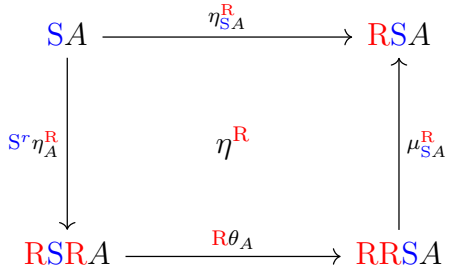
- The monad diagrams for $\eta^{?_{\mathrm{S}}}$ and $\mu^{?_{\mathrm{S}}}$:



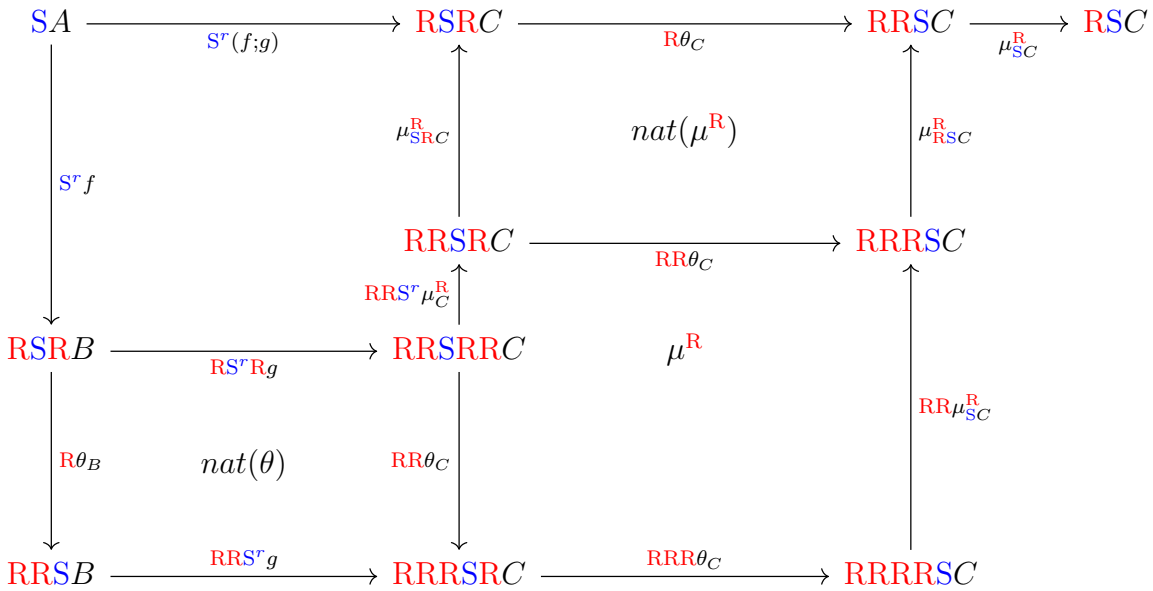


By breaking the $?_{\mathrm{S}}$ monad into its components, we obtain the following commuting diagrams (in $\mathbf{Set}$) proving the above diagrams:

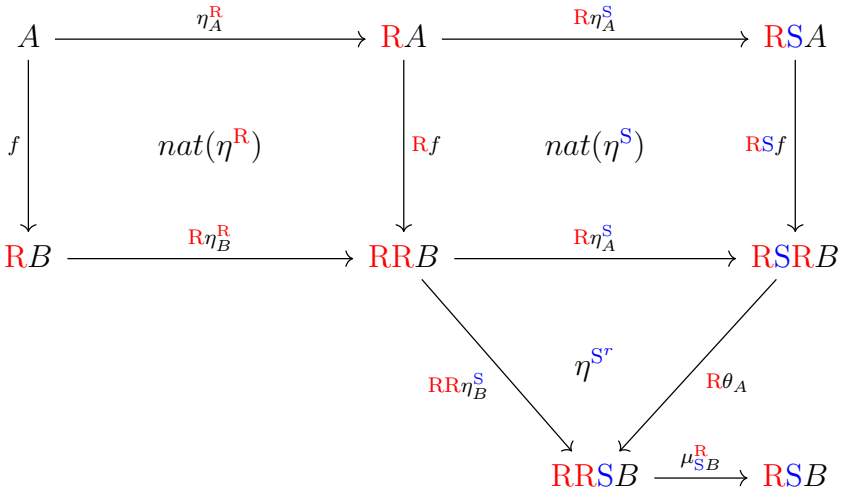
1.



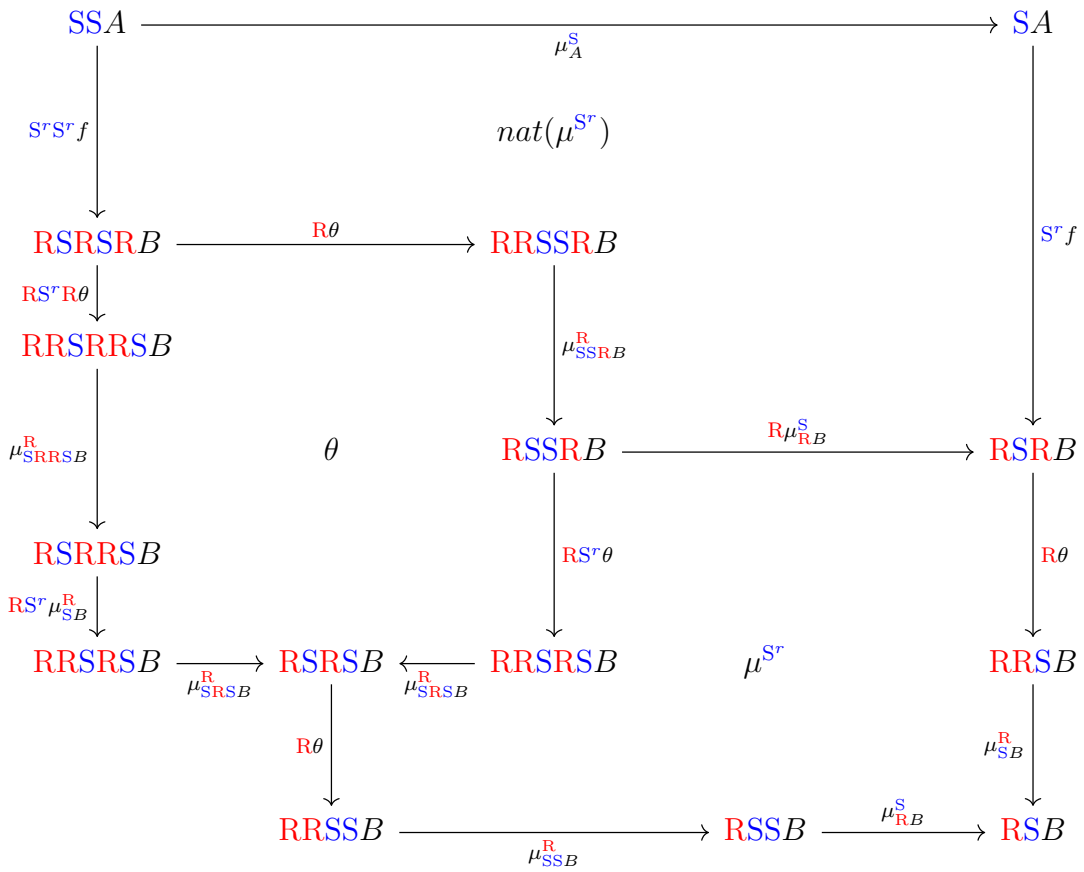
2. The unnamed square is given by composition in $\mathbf{Set}_{\mathrm{R}}$ and functoriality of S^r.



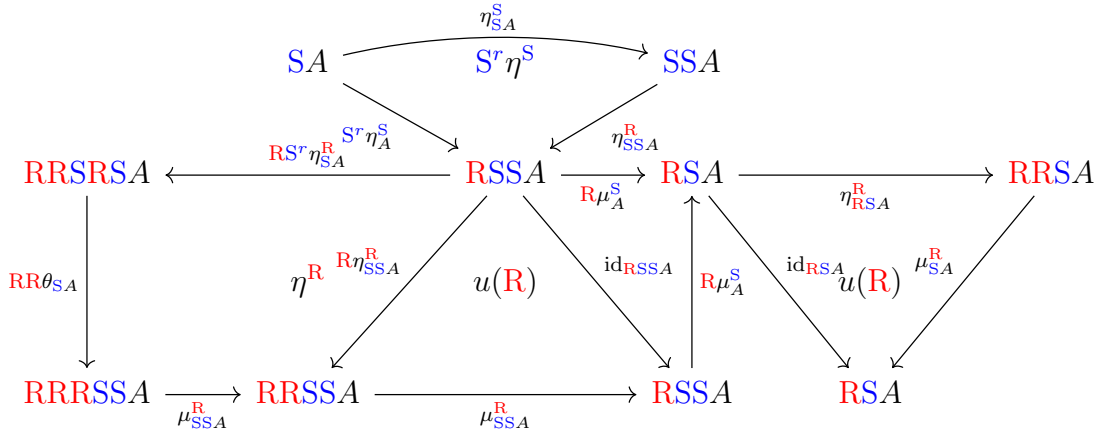
3.



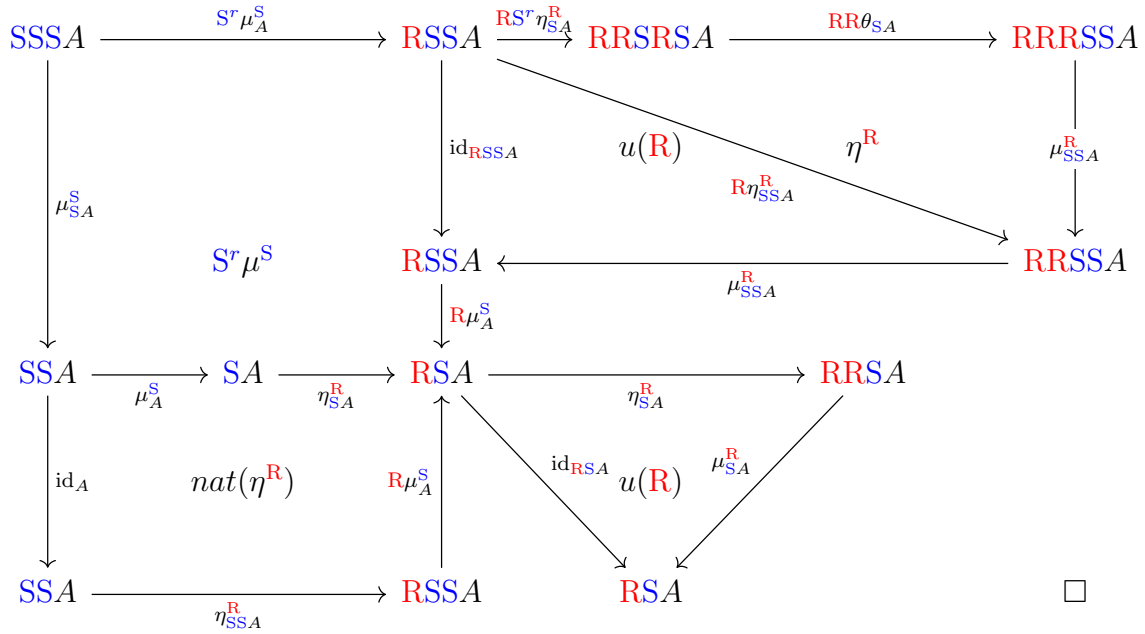
4.


SSA
SA
μ^{S}_{A}
$\mathrm{S}^r\mathrm{S}^r f$
$nat(\mu^{\mathrm{S}^r})$
$\mathrm{S}^r f$
RSRSRB
$\mathrm{R}\theta$
RRSSRB
$\mathrm{RS}^r\mathrm{R}\theta$
$\mu^{\mathrm{R}}_{\mathrm{SSR}B}$
RRSRRSB
$\mu^{\mathrm{R}}_{\mathrm{SRRS}B}$
θ
RSSRB
$\mathrm{R}\mu^{\mathrm{S}}_{\mathrm{R}B}$
RSRB
RSRRSB
$\mathrm{RS}^r\theta$
$\mathrm{R}\theta$
$\mathrm{RS}^r\mu^{\mathrm{R}}_{\mathrm{S}B}$
RRSRSB
$\mu^{\mathrm{R}}_{\mathrm{SRS}B}$
RSRSB
$\mu^{\mathrm{R}}_{\mathrm{SRS}B}$
RRSRSB
μ^{S^r}
RRSB
$\mathrm{R}\theta$
$\mu^{\mathrm{R}}_{\mathrm{S}B}$
RRSSB
$\mu^{\mathrm{R}}_{\mathrm{SS}B}$
RSSB
$\mu^{\mathrm{S}}_{\mathrm{R}B}$
RSB

5.


η^{S}_{SA}
SA
$\mathrm{S}^r\eta^{\mathrm{S}}$
SSA
$\mathrm{RS}^r\eta^{\mathrm{R}}_{SA}$
$\mathrm{S}^r\eta^{\mathrm{S}}_{A}$
$\eta^{\mathrm{R}}_{\mathrm{SS}A}$
RRSRSA
RSSA
RSA
RRSA
$\mathrm{R}\mu^{\mathrm{S}}_{A}$
$\eta^{\mathrm{R}}_{\mathrm{RS}A}$
$\mathrm{RR}\theta_{\mathrm{S}A}$
η^{R}
$\mathrm{R}\eta^{\mathrm{R}}_{\mathrm{SS}A}$
$u(\mathrm{R})$
$\mathrm{id}_{\mathrm{RSS}A}$
$\mathrm{R}\mu^{\mathrm{S}}_{A}$
$\mathrm{id}_{\mathrm{RS}A}$
$u(\mathrm{R})$
$\mu^{\mathrm{R}}_{\mathrm{S}A}$
RRRSSA
$\mu^{\mathrm{R}}_{\mathrm{SS}A}$
RRSSA
$\mu^{\mathrm{R}}_{\mathrm{SS}A}$
RSSA
RSA

6.



The diagrams for an unnatural distributive law closely mirror those of a standard distribution. The main difference is that an unnatural distributive law does not need to be natural. Instead, it is in a way *partially natural*, where θ is not natural but $\mathrm{S}^r\theta$ is. All distributing monads yield an unnatural distributive law in addition to the standard distributive law. It would not be suitable to call them a more general form of distributions however, due to their lack of connection with Eilenberg-Moore categories. Rather than adding a new dimension to distributing monads, unnatural distributions exist at a sort of metaphorical "right angle" to standard distributions.

Remark 7.1.6.
Aside from the final two diagrams, all components of unnatural distributions hold for any monad $?$ *on a Kleisli category* $\mathbb{C}_{\mathrm{R}}$. *One must merely take* $\mathrm{S}^r A = ?A$ *and* $\mathrm{S}^r f = ?(f; \eta^{\mathrm{R}}_A)$. *The final two diagrams are only true when the monad is lifted from some natural transformations in* $\mathbb{C}$, *i.e.* $\eta^? = \eta^{\mathrm{S}}; \eta^{\mathrm{R}}$ *and* $\mu^? = \mu^{\mathrm{S}}; \eta^{\mathrm{R}}$ *for some natural transformations* $\eta^{\mathrm{S}}, \mu^{\mathrm{S}}$.

7.2 Generalising Non-idempotent Sum

Now that we have defined the notion of an unnatural distribution, we can return to the problem of obtaining semiring based (co)monads on the Kleisli category of non-idempotent semiring monads. In the distribution of semiring monads where $\mathcal{R}$ has an idempotent sum, we only needed to fix the exponential action. For the non-idempotent case we also have a coefficient morphism, and the coefficient and exponential action are interconnected. For simplicities sake, we will continue to impose our standard restrictions on $\mathcal{R}$ and the monad of $\mathcal{S}$ to avoid issues with infinite products and infinite sums ($\mathcal{R}$ is continuous, S utilises finite support).

Definition 7.2.1 (Semiring Interaction Pairs)**.**
Let $\mathcal{R}$ and $\mathcal{S}$ be arbitrary semirings, where $\mathcal{R}$ is commutative and continuous. Let R be the semiring monad of $\mathcal{R}$, and S be the semiring monad of $\mathcal{S}$ restricted to functions of finite support. We say that $\mathcal{R}$ and $\mathcal{S}$ have an interaction pair *if*

- *there exists an exponential action of $\mathcal{S}$ on $\mathcal{R}$: $(-)^{-} : \mathcal{R} \times \mathcal{S} \to \mathcal{R}$;*
- *there exists a function $\frac{-\|-}{-} : \mathrm{S}A \times \mathrm{S}B \times \mathrm{S}(A \times B) \to \mathcal{R}$ that we call the* coefficient*;*

such that the following requirements are satisfied:
(for $a \in A, b \in B, f : A \to B, \alpha, \alpha' \in \mathrm{S}A, \bar{\alpha} \in \mathrm{SS}A, \beta \in \mathrm{S}B, \gamma \in \mathrm{S}C, \sigma \in \mathrm{S}(A \times B)$)

- *The coefficient exhibits the following properties:*

$$\frac{\eta^{\mathcal{S}}(a)\|\eta^{\mathcal{S}}(b)}{\sigma} = \delta_{\eta^{\mathcal{S}}((a,b)),\sigma} \tag{coef1}$$

$$\frac{\eta^{\mathcal{S}}(a)\|\alpha}{\sigma} \neq 0 \Rightarrow \exists a' \in A \text{ such that } \alpha = \eta^{\mathcal{S}}(a') \tag{coef2}$$

$$\sum_{\substack{\sigma_{ab} \in \mathrm{S}(A\times B),\\ \sigma_{bc} \in \mathrm{S}(B\times C)}} \frac{\alpha\|\beta}{\sigma_{ab}} \cdot \frac{\beta\|\gamma}{\sigma_{bc}} = \sum_{\substack{\sigma_{ac} \in \mathrm{S}(A\times C),\\ \sigma_{bac} \in \mathrm{S}(B\times A\times C)}} \frac{\alpha\|\gamma}{\sigma_{ac}} \cdot \frac{\beta\|\sigma_{ac}}{\sigma_{bac}} \tag{coef3}$$

$$\frac{\alpha\|\beta}{\sigma} \neq 0 \Rightarrow \sigma \in \mathcal{W}(\alpha, \beta) \tag{coef4}$$

$$\frac{\alpha\|\alpha'}{\lambda a,a'.\alpha(a)\cdot\delta_{a,a'}} = \delta_{\alpha,\alpha'} \tag{coef5}$$

$$\frac{\alpha\|\mathrm{S}\eta^{\mathrm{S}}_A(\alpha)}{\lambda a,\alpha'.\alpha(a)\cdot\delta_{\eta^{\mathrm{S}}_A(a),\alpha}} = 1 \tag{coef6}$$

$$\frac{\hat{\alpha}\|\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha})}{\lambda\bar{\alpha},\alpha.\hat{\alpha}(\bar{\alpha})\cdot\delta_{\mu^{\mathrm{S}}_A(\bar{\alpha}),\alpha}} = 1 \tag{coef7}$$

$$\frac{\alpha\|\mathrm{S}f(\alpha)}{\lambda(a,b).\delta_{f(a),b}\cdot\alpha(a)} \cdot \frac{\mathrm{S}f(\alpha)\|\gamma}{\sigma} = \mathrm{RS}(f \times \mathrm{id}_C)(\lambda\tau.\tfrac{\alpha\|\gamma}{\tau})(\sigma) \tag{coef8}$$

$$\frac{\alpha\|\mathrm{S}f(\alpha)}{\lambda(a,b).\delta_{f(a),b}\cdot\alpha(a)} \cdot \frac{\gamma\|\alpha}{\sigma} = \frac{\gamma\|\mathrm{S}f(\alpha)}{\mathrm{S}(\mathrm{id}\times f)(\sigma)} \cdot \frac{\alpha\|\mathrm{S}(\mathrm{id}\times f)(\sigma)}{\mathrm{S}(\lambda(c,a).(a,(c,f(a))))(\sigma)} \tag{coef9}$$

- *The coefficient and exponential action interact to satisfy the following equations:*

$$\prod_{b\lhd\beta}\sum_{a\in A} f(a,b) = \sum_{\substack{\alpha\in\mathrm{S}A,\\ \sigma\in\mathrm{S}(A\times B)}} \frac{\alpha\|\beta}{\sigma} \prod_{a,b\lhd\sigma} f(a,b) \tag{coefexp}$$

$$\frac{\mu^{\mathrm{S}}_A(\bar{\alpha})\|\beta}{\sigma} = \sum_{\substack{\bar{\beta}\in\mathrm{SS}B,\\ \bar{\sigma}\in\mathrm{SS}(A\times B)}} \delta_{\mu^{\mathrm{S}}_B(\bar{\beta}),\beta} \cdot \delta_{\mu^{\mathrm{S}}_{A\times B}(\bar{\sigma}),\sigma} \sum_{\substack{\rho\in\mathrm{S}(\mathrm{S}A\times\mathrm{S}B),\\ \tau\in\mathrm{S}(\mathrm{S}(A\times B)\times\mathrm{S}A\times\mathrm{S}B)}} \frac{\bar{\alpha}\|\bar{\beta}}{\rho} \cdot \frac{\bar{\sigma}\|\rho}{\tau} \prod_{\sigma',\alpha',\beta'\lhd\tau} \frac{\alpha'\|\beta'}{\sigma'} \tag{coefsplit}$$

Notation 7.2.2.
To improve readability, we also introduce the syntactic sugaring (for $f : A \to B, \alpha : \mathrm{S}A$) $[\alpha/\mathrm{S}f(\alpha)] = \frac{\alpha\|\mathrm{S}f(\alpha)}{\lambda(a,b).\delta_{f(a),b}\cdot\alpha(a)}$.

Many of the above properties are essentially properties from the idempotent section generalised to fit the coefficient. When discussing Lemma 5.2.13, the necessity of needing to pair

individual elements together was the key justification for the necessity of the lemma. The coefficient morphism quantifies the number of ways a pairing can occur, something that is irrelevant when the sum is idempotent.

The property (coef4) ensures the validity of a collection of pairs with respect to two collections of elements. The properties (coef1) and (coef2) support this by ensuring that no new elements appear in the collection of pairs. To make this aspect a bit more obvious, we have the following lemma:

Lemma 7.2.3.
If $\mathcal{S}$ *is* positive *and* discrete*, then properties* (coef1) *and* (coef2) *are given by* (coef4)*.*

Proof. Proof of Lemma 7.2.3. Note that both properties can be written as one statement, namely $\frac{\alpha \| \eta^{\mathcal{S}}(b)}{\sigma} = \sum_{a \in A} \delta^{\mathcal{R}}_{\eta^{\mathcal{S}}((a,b)),\sigma} \cdot \delta^{\mathcal{R}}_{\eta^{\mathcal{S}}(a),\alpha}$. Assume that $\mathcal{S}$ is positive and discrete. Then $\sum_b \beta(b) = 1$ implies that there exists a unique b such that $\beta(b) = 1$. Thus by (coef4), $\frac{\alpha \| \eta^{\mathcal{S}}(b)}{\sigma} \neq 0$ implies that $\sum_{a \in A} \sigma(a,b) = 1$, so there exists a unique a such that $\sigma = \eta^{\mathcal{S}}((a,b))$, which implies that $\lambda a. \sum_{b' \in A} \eta^{\mathcal{S}}((a,b')) = \alpha$, so $\alpha = \eta^{\mathcal{S}}(a)$. □

Remark 7.2.4.
The converse is not directly true as a result of (coef4) *being a single implication. One could define the coefficient such that for a particular pairing* $(a,b) \in A \times B$, $\frac{\eta^{\mathrm{S}}_A(a) \| \eta^{\mathrm{S}}_B(b)}{\eta^{\mathrm{S}}_{A \times B}((a,b))} = 0$*. We conjecture that such a definition would violate other properties, most likely a combination of* (coefexp) *and the definition of an exponential action, but this has not been investigated in detail.*

The property (coef3) is closely linked to the associativity of witnesses and Lemma 6.1.6. (coefexp) is clearly a generalisation of the coefficientless expansion, and (coefsplit) is actually a combination of generalised multiplicative and additive splitting, something that was hinted at in the previous chapter.

The "new" properties are (coef5)-(coef9). The first of these acts as a sort of "identity" – if every element is paired with itself, then there are no "choices" to make, and so the coefficient can only equal 1 or 0 depending on whether the pairing is possible or not. Strictly speaking, one could also write said property as $\frac{\alpha \| \alpha}{\lambda a, a'. \alpha(a) \cdot \chi^{\mathcal{S}}_{\{a\}}(a')} = 1$, relying on (coef4) for the case where $\alpha \neq \alpha'$. The following two are essentially nullifying the effects of η^{S}_A and μ^{S}_A on the coefficient – we conjecture that these three properties would always result from one larger property encompassing all three, but we present them separately in case this is false.

The final two new properties exist to "realign" the coefficient, allowing us to form a morphism that is natural from one which is not.

Before continuing, we prove some technical lemmas.

Lemma 7.2.5.
Given any $\alpha \in \mathrm{S}A$, $[\alpha / \mathrm{S}\mathrm{id}_A(\alpha)] = 1^{\mathcal{R}}$.

Proof. As $\mathrm{S}\mathrm{id}_A(\alpha) = \alpha$, we have $\left[\alpha/\mathrm{S}\mathrm{id}_A(\alpha)\right] = \frac{\alpha \| \alpha}{\lambda a,a'.\delta_{a,a'}\cdot\alpha(a)}$. We conclude by (coef5). □

Lemma 7.2.6.
For all $f : A \to B, g : B \to C, \alpha \in \mathrm{S}A$, $\left[\alpha/\mathrm{S}f(\alpha)\right] \cdot \left[\mathrm{S}f(\alpha)/\mathrm{S}g(\mathrm{S}f(\alpha))\right] = \left[\alpha/\mathrm{S}(g \circ f)(\alpha)\right]$.

Proof. Let $\beta \in \mathrm{S}D, \sigma \in \mathrm{S}(C \times D)$ such that $\frac{\mathrm{S}g(\mathrm{S}f(\alpha)) \| \beta}{\sigma} \neq 0$. We have:

$$\begin{aligned}
\left[\mathrm{S}f(\alpha)/\mathrm{S}g(\mathrm{S}f(\alpha))\right] \cdot \left[\alpha/\mathrm{S}f(\alpha)\right] &= \frac{\alpha \| \mathrm{S}f(\alpha)}{\lambda a,b.\delta_{f(a),b}\cdot\alpha(a)} \cdot \frac{\mathrm{S}f(\alpha) \| \mathrm{S}g(\mathrm{S}f(\alpha))}{\lambda b,c.\delta_{g(b),c}\cdot\mathrm{S}f(\alpha)(b)} \\
&= \mathrm{RS}(f \times \mathrm{id}_C)(\lambda\tau.\frac{\alpha \| \mathrm{S}g(\mathrm{S}f(\alpha))}{\tau})(\lambda b, c.\delta_{g(b),c} \cdot \mathrm{S}f(\alpha)(b)) \\
&= \sum_{\tau \in \mathrm{S}(A\times C)} \frac{\alpha \| \mathrm{S}g(\mathrm{S}f(\alpha))}{\tau} \cdot \delta_{\mathrm{S}(f\times\mathrm{id})(\tau),\lambda b,c.\delta_{g(b),c}\cdot\mathrm{S}f(\alpha)(b)} \\
&= \frac{\alpha \| \mathrm{S}g(\mathrm{S}f(\alpha))}{\lambda a,c.\delta_{g(f(a)),c}\cdot\alpha(a)} \\
&= \frac{\alpha \| \mathrm{S}(g\circ f)(\alpha))}{\lambda a,c.\delta_{(g\circ f)(a),c}\cdot\alpha(a)} \\
&= \left[\alpha/\mathrm{S}(g \circ f)(\alpha)\right]
\end{aligned}$$

□

Lemma 7.2.7 (Unnatural to Natural)**.**
Given $\tau : A \to \mathrm{R}B, \alpha : \mathrm{S}A, \beta : \mathrm{S}B$*, we have:*

$$\left[\alpha/\mathrm{S}\tau(\alpha)\right] \cdot \sum_{\sigma\in\mathrm{S}(\mathrm{R}B\times B)} \frac{\mathrm{S}\tau(\alpha) \| \beta}{\sigma} \prod_{g,b\lhd\sigma} g(b) = \sum_{\sigma\in\mathrm{S}(A\times B)} \frac{\alpha \| \beta}{\sigma} \prod_{a,b\lhd\sigma} \tau(a)(b)$$

Proof. The lemma follows naturally from (coef8).

$$\begin{aligned}
\left[\alpha/\mathrm{S}\tau(\alpha)\right] \cdot \sum_{\sigma\in\mathrm{S}(\mathrm{R}B\times B)} \frac{\mathrm{S}\tau(\alpha) \| \beta}{\sigma} \prod_{g,b\lhd\sigma} g(b) &= \sum_{\varsigma\in\mathrm{S}(A\times B)} \frac{\alpha \| \beta}{\varsigma} \prod_{g,b\lhd\mathrm{S}(\tau\times\mathrm{id}_B)(\varsigma)} g(b) && \text{(coef8)} \\
&= \sum_{\varsigma\in\mathrm{S}(A\times B)} \frac{\alpha \| \beta}{\varsigma} \prod_{g,b} (g(b))^{\sum_{a\in A}\varsigma(a,b)\cdot\delta_{\tau(a),g}} && \text{Definition} \\
&= \sum_{\varsigma\in\mathrm{S}(A\times B)} \frac{\alpha \| \beta}{\varsigma} \prod_{a,b\lhd\varsigma} \tau(a)(b) && \text{Definition 6.1.1}
\end{aligned}$$

□

We believe that the above properties suffice to form an unnatural distribution.

Conjecture 7.2.8.
Given semiring monads R, S *and morphisms* $(-)^-, \frac{-\|-}{-}$ *forming an interactive pair between them, an unnatural distribution is obtained of* R *over* S *with* S^r *defined as:*

- *Given* $A \in \mathrm{Ob}(\mathbf{Set})$, $\mathrm{S}^r A = \mathrm{S}A$*;*
- *Given* $f : A \to B, \alpha : \mathrm{S}A$, $\mathrm{S}^r f(\alpha) = \eta^{\mathrm{R}}_B(\mathrm{S}f(\alpha)) \cdot \left[\alpha/\mathrm{S}f(\alpha)\right]$*;*

and θ_A defined as

$$\theta_A(\varphi)(\alpha) = \sum_{\sigma \in \mathrm{S}(\mathrm{R}A \times A)} \frac{\varphi \| \alpha}{\sigma} \prod_{f, a \triangleleft \sigma} f(a)$$

The proof for this is a work in progress, and is missing two diagrams. Hence, it remains as conjecture for now.

Proof. We begin by proving that S^r is a functor.

- S^r preserves the unit; $\mathrm{S}^r \mathrm{id}_A = \eta^{\mathrm{R}}_{\mathrm{S}A}$: Let $\alpha, \alpha' \in \mathrm{S}A$.

$$\begin{aligned}
\mathrm{S}^r \mathrm{id}_A(\alpha)(\alpha') &= \eta^{\mathrm{R}}_{\mathrm{S}A}(\mathrm{S}\mathrm{id}_A(\alpha))(\alpha') \cdot [\alpha / \mathrm{S}\mathrm{id}_A(\alpha)] && \text{Definition} \\
&= \eta^{\mathrm{R}}_{\mathrm{S}A}(\mathrm{S}\mathrm{id}_A(\alpha))(\alpha') && \text{Lemma 7.2.5} \\
&= \eta^{\mathrm{R}}_{\mathrm{S}A}(\alpha)(\alpha') && \text{Definition}
\end{aligned}$$

- S^r preserves composition; given $f : A \to B, g : B \to C$, $\mathrm{S}^r g \circ \mathrm{S}^r f = \mathrm{S}^r(g \circ f)$: Let $\alpha \in \mathrm{S}A, \gamma \in \mathrm{S}C$.

$$\begin{aligned}
&(\mathrm{S}^r g \circ \mathrm{S}^r f)(\alpha)(\gamma) \\
&\quad = \sum_{\beta \in B} \eta^{\mathrm{R}}_{\mathrm{S}B}(\mathrm{S}f(\alpha))(\beta) \cdot \eta^{\mathrm{R}}_{\mathrm{S}C}(\mathrm{S}g(\beta))(\gamma) \cdot [\alpha / \mathrm{S}f(\alpha)] \cdot [\beta / \mathrm{S}g(\beta)] && \text{Definition} \\
&\quad = \eta^{\mathrm{R}}_{\mathrm{S}C}(\mathrm{S}g(\mathrm{S}f(\alpha)))(\gamma) \cdot [\alpha / \mathrm{S}f(\alpha)] \cdot [\mathrm{S}f(\alpha) / \mathrm{S}g(\mathrm{S}f(\alpha))] && \beta = \mathrm{S}f(\alpha) \\
&\quad = \eta^{\mathrm{R}}(\mathrm{S}g(\mathrm{S}f(\alpha)))(\gamma) \cdot [\alpha / \mathrm{S}g \circ f(\alpha)] && \text{Lemma 7.2.6} \\
&\quad = \mathrm{S}^r(g \circ f)(\alpha)(\gamma) && \text{Definition}
\end{aligned}$$

For all objects $A \in \mathrm{Ob}(\mathbf{Set})$, $\mathrm{S}^r A = \mathrm{S}A$ is true by definition. We will first show the naturality of θ_A before proving the remaining diagrams:

- Naturality of θ_A: Given $\varphi \in \mathrm{SR}A, \beta \in \mathrm{S}B, f \in \mathbf{Set}(A,B)$,

$$\begin{aligned}
(\theta_B \circ \mathrm{S}^r\mathrm{R}f)(\varphi)(\beta) &= \sum_{\psi\in\mathrm{SR}B} \delta_{\psi,\mathrm{SR}f(\varphi)} \cdot [\varphi/_{\mathrm{SR}f(\varphi)}] \cdot \sum_{\sigma\in\mathrm{S}(\mathrm{R}B\times B)} \frac{\psi\|\beta}{\sigma} \prod_{g,b\lhd\sigma} g(b) && \text{Definition}\\
&= [\varphi/_{\mathrm{SR}f(\varphi)}] \cdot \sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times B)} \frac{\mathrm{SR}f(\varphi)\|\beta}{\sigma} \prod_{g,b\lhd\sigma} g(b) && \psi = \mathrm{SR}f(\varphi)\\
&= \sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times B)} \frac{\varphi\|\beta}{\sigma} \prod_{g,b\lhd\sigma} \mathrm{R}f(g)(b) && \text{Lemma 7.2.7}\\
&= \sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times B)} \frac{\varphi\|\beta}{\sigma} \prod_{g,b\lhd\sigma} \sum_{a\in A} g(a)\cdot\delta_{f(a),b} && \text{Definition}\\
&= \sum_{\substack{\sigma\in\mathrm{S}(\mathrm{R}A\times B),\\ \rho\in\mathrm{S}(A\times(\mathrm{R}A\times B)),\\ \alpha\in\mathrm{S}A}} \frac{\varphi\|\beta}{\sigma}\frac{\alpha\|\sigma}{\rho} \prod_{a,(g,b)\lhd\rho} g(a)\cdot\delta_{f(a),b} && \text{(coefexp)}\\
&= \sum_{\left\{\sigma,\alpha,\rho \;\middle|\; \begin{array}{l}\sigma\in\mathrm{S}(\mathrm{R}A\times B),\\ \alpha\in\mathrm{S}A,\\ \rho\in\mathrm{S}(A\times\mathrm{R}A\times B),\\ \rho(a,(g,b))\neq 0\\ \Rightarrow f(a)=b\end{array}\right\}} \frac{\varphi\|\beta}{\sigma}\frac{\alpha\|\sigma}{\rho} \prod_{g,a\in\mathrm{R}A\times A} (g(a))^{\sum_{b\in B}\rho(a,(g,b)))} && \text{Restricting } \rho\\
&= \sum_{\substack{\alpha\in\mathrm{S}A,\\ \sigma\in\mathrm{S}(\mathrm{R}A\times A)}} [\alpha/_{\mathrm{S}f(\alpha)}]\cdot\delta_{\beta,\mathrm{S}f(\alpha)}\cdot\frac{\varphi\|\alpha}{\sigma}\prod_{g,a\lhd\sigma} g(a) && \text{(coef9)}\\
&= (\mathrm{S}^r f\circ\theta_A)(\varphi)(\beta) && \text{Definition}
\end{aligned}$$

- Proving the diagram η^{R}: Given $\alpha,\alpha' \in \mathrm{S}A$,

$$\begin{aligned}
& (\theta_A \circ \mathrm{S}^r\eta^{\mathrm{R}}_A)(\alpha)(\alpha')\\
=\ & \sum_{\varphi\in\mathrm{SR}A} \mathrm{S}^r\eta^{\mathrm{R}}_A(\alpha)(\varphi)\cdot\theta_A(\varphi)(\alpha') && \text{Definition}\\
=\ & \sum_{\varphi\in\mathrm{SR}A} \eta^{\mathrm{R}}_{\mathrm{SR}A}(\mathrm{S}\eta^{\mathrm{R}}_A(\alpha))(\varphi)\cdot[\alpha/_{\mathrm{S}\eta^{\mathrm{R}}_A(\alpha)}]\cdot\theta_A(\varphi)(\alpha') && \text{Definition}\\
=\ & [\alpha/_{\mathrm{S}\eta^{\mathrm{R}}_A(\alpha)}]\cdot\theta_A(\mathrm{S}\eta^{\mathrm{R}}_A(\alpha))(\alpha') && \varphi = \mathrm{S}\eta^{\mathrm{R}}_A(\alpha)\\
=\ & [\alpha/_{\mathrm{S}\eta^{\mathrm{R}}_A(\alpha)}]\cdot\sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times A)} \frac{\mathrm{S}\eta^{\mathrm{R}}_A(\alpha)\|\alpha'}{\sigma}\prod_{f,a'\lhd\sigma} f(a') && \text{Definition}\\
=\ & \sum_{\sigma\in\mathrm{S}(A\times A)} \frac{\alpha\|\alpha'}{\sigma}\prod_{a,a'\lhd\sigma}\eta^{\mathrm{R}}_A(a)(a') && \text{Lemma 7.2.7}\\
=\ & \sum_{\sigma\in\mathrm{S}(A\times A)} \frac{\alpha\|\alpha'}{\sigma}\cdot\delta_{\sigma,\lambda(a,a').\alpha(a)\cdot\delta_{a,a'}} && \eta^{\mathrm{R}}_A(a)(a'), \text{(coef4)}\\
=\ & \frac{\alpha\|\alpha'}{\lambda(a,a').\alpha(a)\cdot\delta_{a,a'}} && \sigma = \lambda(a,a').\alpha(a)\cdot\delta_{a,a'}\\
=\ & \delta_{\alpha,\alpha'} && \text{(coef5)}\\
=\ & \eta^{\mathrm{R}}_{\mathrm{S}A}(\alpha)(\alpha') && \text{Definition}
\end{aligned}$$

- Proving the diagram $\eta^{\mathrm{S}r}$: Given $h \in \mathrm{R}A, \alpha \in \mathrm{S}A$,

$$\begin{aligned}
\theta_A(\eta^{\mathrm{S}}_{\mathrm{R}A}(h))(\alpha) &= \sum_{\sigma \in \mathrm{S}(\mathrm{R}A\times A)} \frac{\eta^{\mathrm{S}}_{\mathrm{R}A}(h) \| \alpha}{\sigma} \prod_{f,a\lhd\sigma} f(a) && \text{Definition}\\
&= \sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times A), a\in A} \frac{\eta^{\mathrm{S}}_{\mathrm{R}A}(h) \| \eta^{\mathrm{S}}_A(a)}{\sigma} \delta_{\alpha,\chi^{\mathrm{S}}_{\{a\}}} \prod_{f,a\lhd\sigma} f(a) && \text{(coef2)}\\
&= \sum_{a\in A} \delta_{\alpha,\chi^{\mathrm{S}}{}_{\{a\}}} \prod_{f,a\lhd\eta^{\mathrm{S}}_{\mathrm{R}A\times A}((h,a))} f(a) && \text{(coef1)}\\
&= \sum_{a\in A} h(a)\cdot \delta_{\alpha,\chi^{\mathrm{S}}{}_{\{a\}}} && \mathsf{supp}(\eta^{\mathrm{S}}_{\mathrm{R}A\times A}((h,a)))\\
&= \mathrm{R}\eta^{\mathrm{S}}_A(h)(\alpha) && \text{Definition}
\end{aligned}$$

- Proving the diagram μ^{R}: Given $\varphi \in \mathrm{SRR}A, \alpha \in \mathrm{S}A$,

$$\begin{aligned}
&(\theta_A\circ\theta_{\mathrm{R}A})(\varphi)(\alpha)\\
&= \sum_{\psi\in\mathrm{SR}A} \theta_{\mathrm{R}A}(\varphi)(\psi)\cdot\theta_A(\psi)(\alpha) && \text{Definition}\\
&= \sum_{\substack{\psi\in\mathrm{SR}A,\\ \sigma_{gf}\in\mathrm{S}(\mathrm{SR}A\times\mathrm{R}A),\\ \sigma_{fa}\in\mathrm{S}(\mathrm{R}A\times A)}} \frac{\varphi\|\psi}{\sigma_{gf}}\cdot\frac{\psi\|\alpha}{\sigma_{fa}}\cdot\Big(\prod_{g,f\lhd\sigma_{gf}} g(f)\Big)\cdot\Big(\prod_{f,a\lhd\sigma_{fa}} f(a)\Big) && \text{Definition}\\
&= \sum_{\substack{\psi\in\mathrm{SR}A,\\ \sigma_{ga}\in\mathrm{S}(\mathrm{SR}A\times A),\\ \sigma_{fga}\in\mathrm{S}(\mathrm{R}A\times\mathrm{SR}A\times A)}} \frac{\varphi\|\alpha}{\sigma_{ga}}\frac{\psi\|\sigma_{ga}}{\sigma_{fga}}\left[\prod_{g,f}(g(f))^{\sum_a \sigma_{fga}(f,g,a)}\right]\left[\prod_{f,a}(f(a))^{\sum_g\sigma_{fga}(f,g,a)}\right] && \text{(coef3)}\\
&= \sum_{\sigma_{ga}\in\mathrm{S}(\mathrm{SR}A\times A)} \frac{\varphi\|\alpha}{\sigma_{ga}} \sum_{\substack{\psi\in\mathrm{SR}A,\\ \sigma_{fga}\in\mathrm{S}(\mathrm{R}A\times\mathrm{SR}A\times A)}} \frac{\psi\|\sigma_{ga}}{\sigma_{fga}} \prod_{f,(g,a)\lhd\sigma_{fga}} g(f)\cdot f(a) && \text{Def. 6.1.1}\\
&= \sum_{\sigma_{ga}\in\mathrm{S}(\mathrm{SR}A\times A)} \frac{\varphi\|\alpha}{\sigma_{ga}} \prod_{g,a\lhd\sigma_{ga}} \sum_f g(f)\cdot f(a) && \text{(coefexp)}\\
&= \sum_{\sigma_{ga}\in\mathrm{S}(\mathrm{SR}A\times A)} \frac{\varphi\|\alpha}{\sigma_{ga}} \prod_{g,a\lhd\sigma_{ga}} \mu^{\mathrm{R}}_A(g)(a) && \text{Definition}\\
&= [\varphi/\mathrm{S}\mu^{\mathrm{R}}_A(\varphi)] \sum_{\sigma\in\mathrm{S}(\mathrm{R}A\times A)} \frac{\mathrm{S}\mu^{\mathrm{R}}_A(\varphi)\|\alpha}{\sigma} \prod_{f,a\lhd\sigma} f(a) && \text{Lemma 7.2.7}\\
&= [\varphi/\mathrm{S}\mu^{\mathrm{R}}{}_A(\varphi)]\cdot\theta_A(\mathrm{S}\mu^{\mathrm{R}}_A(\varphi))(\alpha) && \text{Definition}\\
&= \sum_{\psi'\in\mathrm{SR}A} \eta^{\mathrm{R}}_{\mathrm{SR}A}(\mathrm{S}\mu^{\mathrm{R}}_A(\varphi))(\psi')\cdot[\varphi/\mathrm{S}\mu^{\mathrm{R}}_A(\varphi)]\cdot\theta_A(\psi')(\alpha) && \psi' = \mathrm{S}\mu^{\mathrm{R}}_A(\varphi)\\
&= \sum_{\psi'\in\mathrm{SR}A} \mathrm{S}^r\mu^{\mathrm{R}}_A(\varphi)(\psi')\cdot\theta_A(\psi')(\alpha) && \text{Definition}\\
&= (\theta_A\circ\mathrm{S}^r\mu^{\mathrm{R}}_A)(\varphi)(\alpha) && \text{Definition}
\end{aligned}$$

- Proving the diagram $\mu^{\mathrm{S}r}$: Given $\varphi \in \mathrm{SSR}A, \alpha \in \mathrm{S}A$,

$$\begin{aligned}
&\mathrm{R}\mu^{\mathrm{S}}_A((\theta_{\mathrm{S}A}\circ\mathrm{S}^r\theta_A)(\varphi))(\alpha)\\
&= \sum_{\substack{\xi\in\mathrm{SRS}A,\\ \bar\alpha\in\mathrm{SS}A}} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot\theta_{\mathrm{S}A}(\xi)(\bar\alpha)\cdot\mathrm{S}^r\theta_A(\varphi)(\xi) && \text{Definition}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\substack{\xi\in \mathrm{SRS}A,\\ \bar\alpha\in \mathrm{SS}A}} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot \theta_{\mathrm{S}A}(\xi)(\bar\alpha)\cdot \eta^{\mathrm{R}}_{\mathrm{SRS}A}(\mathrm{S}\theta_A(\varphi))(\xi)\cdot [\varphi/_{\mathrm{SS}\theta_A(\varphi)}] && \text{Definition}\\
&= \sum_{\bar\alpha\in \mathrm{SS}A} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot [\varphi/_{\mathrm{SS}\theta_A(\varphi)}]\cdot \theta_{\mathrm{S}A}(\mathrm{S}\theta_A(\varphi))(\bar\alpha) && \xi=\mathrm{S}\theta_A(\varphi)\\
&= \sum_{\bar\alpha\in \mathrm{SS}A} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot [\varphi/_{\mathrm{SS}\theta_A(\varphi)}] \sum_{\rho\in \mathrm{S}(\mathrm{RS}A\times \mathrm{S}A)} \frac{\mathrm{S}\theta_A(\varphi)\|\bar\alpha}{\rho} \prod_{\zeta,\gamma\lhd\rho}\zeta(\gamma) && \text{Definition}\\
&= \sum_{\substack{\bar\alpha\in \mathrm{SS}A,\\ \rho\in \mathrm{S}(\mathrm{SR}A\times \mathrm{S}A)}} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot \frac{\varphi\|\bar\alpha}{\rho} \prod_{\psi,\gamma\lhd\rho}\theta_A(\psi)(\gamma) && \text{Lemma 7.2.7}\\
&= \sum_{\substack{\bar\alpha\in \mathrm{SS}A,\\ \rho\in \mathrm{S}(\mathrm{SR}A\times \mathrm{S}A)}} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot \frac{\varphi\|\bar\alpha}{\rho} \prod_{\psi,\gamma\lhd\rho} \sum_{\sigma\in \mathrm{S}(\mathrm{R}A\times A)} \frac{\psi\|\gamma}{\sigma} \prod_{f,a\lhd\sigma} f(a) && \text{Definition}\\
&= \sum_{\substack{\bar\alpha\in \mathrm{SS}A,\\ \rho\in \mathrm{S}(\mathrm{SR}A\times \mathrm{S}A),\\ \bar\varsigma\in \mathrm{SS}(\mathrm{R}A\times A),\\ \tau\in \mathrm{S}(\mathrm{S}(\mathrm{R}A\times A)\times(\mathrm{SR}A\times \mathrm{S}A))}} \delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot \frac{\varphi\|\bar\alpha}{\rho}\cdot \frac{\bar\varsigma\|\rho}{\tau} \prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma} \prod_{f,a\lhd\sigma} f(a) && \text{(coefexp)}
\end{aligned}
$$

Due to the sheer size of the equations at this point, to aid readability we will write $\mathcal{P}(\cdots)$ as short for $\delta_{\mu^{\mathrm{S}}_A(\bar\alpha),\alpha}\cdot\delta_{\mu^{\mathrm{S}}_{\mathrm{R}A\times A}(\bar\varsigma),\varsigma}\cdot\frac{\varphi\|\bar\alpha}{\rho}\cdot\frac{\bar\varsigma\|\rho}{\tau}$ during the rest of this proof. For the same reason, we define the set

$$\mathscr{X} = \mathrm{SS}A\times \mathrm{S}(\mathrm{SR}A\times \mathrm{S}A)\times \mathrm{SS}(\mathrm{R}A\times A)\times \mathrm{S}(\mathrm{S}(\mathrm{R}A\times A)\times \mathrm{SR}A\times \mathrm{S}A)$$

This may cause some disconnect with the current step of the proof. The next step is essentially introducing $\varsigma=\mu^{\mathrm{S}}_{\mathrm{R}A\times A}(\bar\varsigma)$, as it will be necessary later.

$$
\begin{aligned}
&= \sum_{\substack{\varsigma\in \mathrm{S}(\mathrm{R}A\times A),\\ (\bar\alpha,\rho,\bar\varsigma,\tau)\in\mathscr{X}}} \mathcal{P}(\cdots) \prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma} \prod_{f,a\lhd\sigma} f(a) && \text{Rearranging}\\
&= \sum_{\substack{\varsigma\in \mathrm{S}(\mathrm{R}A\times A),\\ (\bar\alpha,\rho,\bar\varsigma,\tau)\in\mathscr{X}}} \mathcal{P}(\cdots) \left(\prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma}\right)\left(\prod_{\sigma,(\psi,\gamma)\lhd\tau}\prod_{f,a\lhd\sigma} f(a)\right) && \text{Rearranging}\\
&= \sum_{\substack{\varsigma\in \mathrm{S}(\mathrm{R}A\times A),\\ (\bar\alpha,\rho,\bar\varsigma,\tau)\in\mathscr{X}}} \mathcal{P}(\cdots) \left(\prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma}\right)\left(\prod_{\sigma\lhd\bar\varsigma}\prod_{f,a\lhd\sigma} f(a)\right) && \text{Definition 6.1.1}\\
&= \sum_{\substack{\varsigma\in \mathrm{S}(\mathrm{R}A\times A),\\ (\bar\alpha,\rho,\bar\varsigma,\tau)\in\mathscr{X}}} \mathcal{P}(\cdots) \left(\prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma}\right)\left(\prod_{f,a\lhd\mu^{\mathrm{S}}_{\mathrm{R}A\times A}(\bar\varsigma)} f(a)\right) && \text{Definition 6.1.1}\\
&= \sum_{\varsigma\in \mathrm{S}(\mathrm{R}A\times A)} \left(\sum_{(\bar\alpha,\rho,\bar\varsigma,\tau)\in\mathscr{X}} \mathcal{P}(\cdots)\prod_{\sigma,(\psi,\gamma)\lhd\tau}\frac{\psi\|\gamma}{\sigma}\right)\left(\prod_{f,a\lhd\varsigma} f(a)\right) && \varsigma=\mu^{\mathrm{S}}_{\mathrm{R}A\times A}(\bar\varsigma)\\
&= \sum_{\varsigma\in \mathrm{S}(\mathrm{R}A\times A)} \frac{\mu^{\mathrm{S}}_{\mathrm{R}A}(\varphi)\|\alpha}{\varsigma} \prod_{f,a\lhd\varsigma} f(a) && \text{(coefsplit)}\\
&= \theta_A(\mu^{\mathrm{S}}_{\mathrm{R}A}(\varphi))(\alpha) && \text{Definition}
\end{aligned}
$$

- Proving the diagram $\mathrm{S}^r\eta^{\mathrm{S}}$: Given $\alpha \in \mathrm{S}A, \hat{\alpha} \in \mathrm{SS}A$, we have

$$\begin{array}{rcll} \mathrm{S}^r\eta^{\mathrm{S}}_A(\alpha)(\hat{\alpha}) & = & \eta^{\mathrm{R}}_{\mathrm{SS}A}(\mathrm{S}\eta^{\mathrm{S}}_A(\alpha))(\hat{\alpha}) \cdot [\alpha/_{\mathrm{S}\eta^{\mathrm{S}}_A(\alpha)}] & \text{Definition} \\ & = & \eta^{\mathrm{R}}_{\mathrm{SS}A}(\eta^{\mathrm{S}}_{\mathrm{S}A}(\alpha))(\hat{\alpha}) & \text{(coef6)} \end{array}$$

- Proving the diagram $\mathrm{S}^r\mu^{\mathrm{S}}$: Given $\hat{\alpha} \in \mathrm{SSS}A, \alpha \in \mathrm{S}A$, we have

$$\begin{array}{rcll} \mathrm{R}\mu^{\mathrm{S}}_A(\mathrm{S}^r\mu^{\mathrm{S}}_A(\hat{\alpha}))(\alpha) & = & \sum_{\bar{\alpha}\in\mathrm{SS}A} \mathrm{S}^r\mu^{\mathrm{S}}_A(\hat{\alpha})(\bar{\alpha}) \cdot \delta_{\mu^{\mathrm{S}}_A(\bar{\alpha}),\alpha} & \text{Definition} \\ & = & \sum_{\bar{\alpha}\in\mathrm{SS}A} \eta^{\mathrm{R}}_{\mathrm{SS}A}(\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha}))(\bar{\alpha}) \cdot [\hat{\alpha}/_{\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha})}] \cdot \delta_{\mu^{\mathrm{S}}_A(\bar{\alpha}),\alpha} & \text{Definition} \\ & = & \sum_{\bar{\alpha}\in\mathrm{SS}A} \delta_{\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha}),\bar{\alpha}} \cdot [\hat{\alpha}/_{\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha})}] \cdot \delta_{\mu^{\mathrm{S}}_A(\bar{\alpha}),\alpha} & \text{Definition} \\ & = & \delta_{\mu^{\mathrm{S}}_A(\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha})),\alpha} \cdot [\hat{\alpha}/_{\mathrm{S}\mu^{\mathrm{S}}_A(\hat{\alpha})}] & \bar{\alpha} = \mu^{\mathrm{S}}_{\mathrm{S}A}(\hat{\alpha}) \\ & = & \eta^{\mathrm{R}}_{\mathrm{S}A}(\mu^{\mathrm{S}}_A(\mu^{\mathrm{S}}_A(\hat{\alpha})))(\alpha) & \text{(coef7)} \end{array}$$

Conclusions, Musings, and Further Work

We have in this thesis done what we have set out to do: Present a fully abstract model of PCF based on Extended Addressing Machines, and discovered reliable methods to combine semiring monads.That is not to say that this work is "complete" – indeed, just as life keeps on turning, there are always new avenues to explore and discover. Likewise, there are avenues which were partially explored or mused about, yet set aside for the time being in the interest of completing the goals at hand. To conclude this work, I wish to spend some time briefly discussing some of these avenues, idle musings, and interesting curiosities which did not fit into the content or the flow of the main body. These have not been formally proven or fully explored, but are simply based on my own intuition and understanding.

Double Categories

There is a connection between double categories and unnatural distributions. In fact, the results presented in Section 7.1 were discovered by using double categories as a tool. Some information about double categories and how they link to monads are present in the appendix. Double categories as a whole are a bit underexplored in the literature, but the results found in this thesis provide a strong motivation for further investigation into them.

Mathematicians often prefer to utilise Eilenberg-Moore categories when working with monads, whereas Kleisli categories are much simpler to reason about and come more naturally to computer scientists. In the main body, it was mentioned that unnatural distributions sit at a sort of right angle to the standard form of distributions. It would be interesting to see what other sorts of structures could be found at this "right angle" – perhaps a similar structure exists for Eilenberg-Moore categories, or perhaps there exists a different notion of distribution that subsumes unnatural distributions. If Eilenberg-Moore categories were to lack this sort of structure, then unnatural distributions would provide a motivation to investigate Kleisli categories independently from Eilenberg-Moore categories. Regardless, I think there are interesting results to be found from the intersection between monads, double categories, Kleisli categories, and unnatural distributions.

As an aside, I also wish to comment on the naming of "unnatural distributions" – finding a suitable name for this version of a distribution was more annoying than expected. The distribution is sort of "relative to" one of the semirings, yet a relative distributive law is already a concept in the field, describing a distribution between a relative monad and a monad – a completely different concept. Various words amounting to "almost" (Quasi, Pseudo, ...) have also been used or are unsuitable. unnatural distribution was settled on after encountering unnatural transformations, but the name is still somewhat unsatisfactory as the distribution is natural, just not in the category that one would initially expect. This is more a general comment on the issue of namespace pollution in mathematics. If a more suitable name was proposed for unnatural distributions, or a more suitable concept for the name "Unnatural Distribution" were to arise, I would have no opposition to it.

Linear Logic

Another link which must obviously be discussed is that to Linear Logic. As mentioned in the introduction, one would expect that the distributions presented here can always be used to form models of classical linear logic. When the subject of distributing semirings was first presented to me, it was as a building block along the (endless?) road to finding every exponential in Linear Logic. I believe that the additional separation of unnatural distributions is certainly of use to said end goal. I do believe that the work encompasses all models of Linear Logic arising from semirings, though I could easily be wrong about this – I merely do not see how a counterexample could arise.

Something more specific and applied regarding Linear Logic is that models of classical linear logic include a morphism commonly denoted with m with the type $!A \otimes !B \to !(A \otimes B)$ which causes $!$ to be a symmetric monoidal functor. By currying and morphism duality, the type of this morphism can be seen to be the same as that of the coefficient $\frac{=|=}{-}$, so some time was spent trying to see whether $\frac{=|=}{-}$ is a possible candidate for m, or whether the trivial choice of m – inclusion in $\mathcal{W}$ – is the only possible choice. It turns out that $\frac{=|=}{-}$ is not natural, so it cannot be used as m. However, a lot of the diagrams that m requires have a direct relationship with the properties of $\frac{=|=}{-}$ – in particular, the property (coefsplit) is a direct algebraic expression of one of said diagrams. In addition if we treat $\frac{=|=}{-}$ as having the type $\mathrm{S}A \otimes \mathrm{S}B \to \mathrm{RS}(A \otimes B)$, then I am under the belief that the following diagram holds:

$$\begin{array}{ccc} \mathrm{S}A \times \mathrm{S}B & \xrightarrow{\frac{=|=}{-}} & \mathrm{S}(A \times B) \\ \downarrow{\scriptstyle \mathrm{S}^r f \times \mathrm{S}^r g} & & \downarrow{\scriptstyle \mathrm{S}(f \times g)} \\ \mathrm{S}C \times \mathrm{S}D & \xrightarrow{\frac{=|=}{-}} & \mathrm{S}(C \times D) \end{array}$$

There is certainly something interesting that is not fully understood yet here.

Semirings

There is also more to explore down the avenue of semirings. For one, new pairings of semirings whose monads distribute or have unnatural distributions between them were not presented. In particular, I would expect $\bar{\mathbb{N}}$ with finite support to be a suitable choice of $\mathcal{S}$ for all continuous semirings. An interesting avenue of research would be to investigate how many choices of exponential action one has – clearly, if one has multiple choices, then one obtains multiple (unnatural) distributions, but whether there exist any pairs of semirings with multiple choices of exponential action is not known.

Likewise, what requirements semirings have to allow there to exist at least one exponential action per pairing would be an interesting topic. I would expect that instead of multiplicative splitting and additive splitting, the semirings used for $\mathcal{S}$ would require a "multiplicative-additive splitting" which combines the two, but is more general than both individually. I believe there is also a possibility that the requirements of the semiring being positive and/or discrete could be dropped in the idempotent case – after all, the requirements of the coefficient in the non-idempotent case still apply to the idempotent case, but in the idempotent case they were a consequence of multiplicity semirings while in the non-idempotent case they were presented independent from the choice of semiring. The idempotent case merely simplifys the coefficient properties by requiring that the coefficient always equals 0 or 1. It is my belief that introducing this requirement will also force $\mathcal{R}$ to be idempotent.

Finally, some investigation of semirings with "infinite products" could be interesting. Infinite sums and complete semirings are well-understood and not too difficult to reason about, but I do not know or have any intuitive understanding of what it would mean for a semiring to have "infinite products" – if such a thing could exist though, then the exponentials resulting from dropping the finite support requirement would be very interesting indeed.

Models using Semirings

We should also discuss the topic through which semiring based models were presented: Models of PCF. The semiring $\mathcal{S}$ was set to $\mathbb{N}$ when working with weighted relations previously, but now it could be a variety of other semirings. One of the consequences of accessing further exponentials (to borrow the term from Linear Logic) is the ability to quantify additional aspects regarding function application. The most intuitive vague ideas would be to track degrees of parallelism and security. Unfortunately, the most intuitive candidate for parallelism, some form of $\mathbb{N} \times \mathbb{N}$, is not a multiplicity semiring due to not being discrete. It remains to be seen whether a valid

exponential action and coefficient can be found despite this. Regarding security, TwoN or a similar semiring seems like it could potentially have some applicability there - using the second $\mathbb{N}$ to handle permissions.

Addressing Machines

Finally, we should return to the subject of the first half of the thesis: Addressing Machines. Addressing Machines have not been explored much in the literature, but I feel that having a fully abstract model of PCF is a strong justifier for their usefulness and applicability going forward. It is not known to me how useful they will be to discover new properties of PCF, but one potential use case would be to identify what form common and well-known data structures and algorithms take when translated into PCF.

Rather than merely as a tool to help analyse PCF, I believe that addressing machines have intrinsic value on their own. Addressing Machines occupy a unique space between traditional abstract machines and lambda calculus which to my knowledge is currently unoccupied. This positioning allows them to potentially be a very useful bridge between the two. I could see a scenario where addressing machines are the primary bridge between the fields of complexity theory, program semantics, and compiler theory – I believe that they are uniquely easily accessible from all three areas. Though addressing machines were not designed for complexity theory in mind, one could recover the missing parts of complexity theory by encoding the ”cost” of a function call in the address of a machine – the cost would be equal to the number of registers it has, plus a fixed cost handling the moving of tape and instruction set.

The link between addressing machines and game semantics certainly needs exploration as well. All fully abstract models of PCF being isomorphic means that there is certainly a link between the two, and it would be interesting and almost mandatory to see if there are any low hanging fruit to be obtained from the intersection. Just as interesting to me is the potential link between the two halves of this thesis – using weighted relations to quantify aspects of addressing machines. The earlier mention of allowing the address of a machine to include a quantitative ”cost” is directly related to that. The connection to linear logic could also be interesting to investigate, as every machine instruction has a hidden duplication intrinsic to it, yet a single addressing machine lacks the need for a full-strength !-modality as it only has a limited number of instructions and thus a fixed number of ”duplications”.

Bibliography

[ABM14] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. *SIGPLAN Not.*, 49(9):363–376, aug 2014.

[Acc18] Beniamino Accattoli. Proof nets and the linear substitution calculus. In Bernd Fischer and Tarmo Uustalu, editors, *Theoretical Aspects of Computing - ICTAC 2018 - 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, volume 11187 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2018.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

[AK10] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, pages 381–395, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[AMJ94] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full abstraction for PCF. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1994.

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

[BBLRD95] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, and Jocelyne Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. Research Report RR-2477, INRIA, 1995. Projet EURECA.

[BDS13] Henk Barendregt, Wil Dekkers, and Richard Statman. *The Simply Typed Lambda Calculus*, page 5–54. Perspectives in Logic. Cambridge University Press, 2013.

[BMM23] Intrigila Benedetto, Giulio Manzonetto, and Nicolas Munnich. Extended addressing machines for pcf, with explicit substitutions. *Electronic Notes in Theoretical Informatics and Computer Science*, Volume 1 - Proceedings of..., 02 2023.

[BP15] Flavien Breuvart and Michele Pagani. Modelling coeffects in the relational semantics of linear logic. In *Computer Science Logic*, 01 2015.

[Bre15] Flavien Breuvart. *Dissecting Denotational Demantics: From the well-established H* to the more recent Quantitative Coeffects*. PhD thesis, Paris VII - Sorbonne Paris Cité, 2015.

[Bru72] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[Bru12] R.A. Brualdi. *Introductory Combinatorics*. Pearson Education. Pearson Prentice Hall, 2012.

[BW95] M. Barr and C. Wells. *Category Theory for Computing Science*. Number Bd. 1 in Prentice-Hall international series in computer science. Prentice Hall, 1995.

[Can01] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[CES10] Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. Exponentials with infinite multiplicities. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, pages 170–184, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CHL96] Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM*, 43(2):362–397, mar 1996.

[CHU41] ALONZO CHURCH. *The Calculi of Lambda Conversion. (AM-6)*. Princeton University Press, 1941.

[CLM10] Martin Churchill, James Laird, and Guy McCusker. A concrete representation of observational equivalence for PCF. *CoRR*, abs/1003.0107, 2010.

[Com] Category Theory Community. Ncatlab. https://ncatlab.org. Accessed: 2024-01-31.

[Cre91] Pierre Cregut. *Machines a environnement pour la reduction symbolique et l'evaluation partielle*. PhD thesis, Université Paris-Diderot (Paris VII), 1991. In French.

[Cur30] H. B. Curry. Grundlagen der kombinatorischen logik. *American Journal of Mathematics*, 52(3):509–536, 1930.

[Cur07] Pierre-Louis Curien. Definability and full abstraction. *Electron. Notes Theor. Comput. Sci.*, 172:301–310, 2007.

[Dan05] Olivier Danvy. A rational deconstruction of landin's secd machine. In Clemens Grelck, Frank Huch, Greg J. Michaelson, and Phil Trinder, editors, *Implementation and Application of Functional Languages*, pages 52–71, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DCK97] Roberto Di Cosmo and Delia Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 35–46, 01 1997.

[Del97] Giuseppe Della Penna. Una semantica operazionale per il network computing: le macchine di Turing virtuali. Master's thesis, Università degli Studi di L'Aquila, 1996-97. In Italian.

[DHS00] Stephan Diehl, Pieter Hartel, and Peter Sestoft. Abstract machines for programming language implementation. *Future Generation Computer Systems*, 16(7):739–751, 2000.

[DIM22] Giuseppe Della Penna, Benedetto Intrigila, and Giulio Manzonetto. Addressing machines as models of λ-calculus. *Log. Methods Comput. Sci.*, 18(3), 2022.

[DK09] Manfred Droste and Werner Kuich. *Semirings and Formal Power Series*, pages 3–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[DM82] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, page 207–212, New York, NY, USA, 1982. Association for Computing Machinery.

[EGRS04] Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott. *Linear Logic in Computer Science*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004.

[ER64] Calvin C. Elgot and Abraham Robinson. Random-access stored-program machines, an approach to programming languages. *J. ACM*, 11(4):365–399, oct 1964.

[FW87] Jon Fairbairn and Stuart Wray. Tim: A simple, lazy abstract machine to execute supercombinators. In Gilles Kahn, editor, *Functional Programming Languages and Computer Architecture*, pages 34–45, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[HO00] J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[How80] William Alvin Howard. The formulae-as-types notion of construction. In Haskell Curry, Hindley B., Seldin J. Roger, and P. Jonathan, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[Jay19] Barry Jay. A simpler lambda calculus. In *Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM 2019, page 1–9, New York, NY, USA, 2019. Association for Computing Machinery.

[JS93] Achim Jung and Alley Stoughton. Studying the fully abstract model of pcf within its continuous function model. In *International Conference on Typed Lambda Calculus and Applications*, 1993.

[Kri07] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher Order Symbol. Comput.*, 20(3):199–207, sep 2007.

[Lan64] Peter J. Landin. The Mechanical Evaluation of Expressions. *The Computer Journal*, 6(4):308–320, 01 1964.

[Lan07] Frederic Lang. Explaining the lazy Krivine machine using explicit substitution and addresses. *Higher-Order and Symbolic Computation*, 2007.

[Law63] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872, 1963.

[Ler90] Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.

[LM99] Jean-Jacques Lévy and Luc Maranget. Explicit substitutions and programming languages. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, volume 1738 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 1999.

[LMMP13] Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, page 301–310, USA, 2013. IEEE Computer Society.

[Loa01] Ralph Loader. Finitary PCF is not decidable. *Theor. Comput. Sci.*, 266(1-2):341–364, 2001.

[Man08] Giulio Manzonetto. *Models and theories of lambda calculus*. PhD thesis, Univ. Ca'Foscari (Venice) and Univ. Paris Diderot (Paris 7), 2008.

[Mel95] Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*, TLCA '95, page 328–334, Berlin, Heidelberg, 1995. Springer-Verlag.

[Mil77] Robin Milner. Fully abstract models of typed λ-calculi. *Theor. Comput. Sci.*, 4(1):1–22, 1977.

[Mil99] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[Mit96] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[ML78] Saunders Mac Lane. *Categories, Functors, and Natural Transformations*, pages 7–30. Springer New York, New York, NY, 1978.

[MM07] Ernest Manes and Philip Mulry. Monad compositions i: general constructions and recursive distributive laws. *Theory and Applications of Categories*, 18:172–208, 04 2007.

[Mor69] James Hiram Morris. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, Alfred P. Sloan School of Management, 1969.

[MP21] Damiano Mazza and Michele Pagani. Automatic differentiation in PCF. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–27, jan 2021.

[MRS99] Michael Marz, Alexander Rohr, and Thomas Streicher. Full abstraction and universality via realisability. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 174–182. IEEE Computer Society, 1999.

[Nic94] Hanno Nickau. Hereditarily sequential functionals. In Anil Nerode and Yuri V. Matiyasevich, editors, *Logical Foundations of Computer Science, Third International Symposium, LFCS'94, St. Petersburg, Russia, July 11-14, 1994, Proceedings*, volume 813 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994.

[Ong95] C.-H. Luke Ong. Correspondence between operational and denotational semantics of PCF. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 269–356. Oxford University Press, 1995.

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

[Plo73] G. D. Plotkin. Lambda-definability and logical relations. Technical report, University of Edinburgh, October 1973.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, jan 1965.

[Sco70] Dana Scott. Outline of a mathematical theory of computation. Technical Report PRG02, OUCL, November 1970.

[Sta02] Richard Statman. On the lambda y calculus. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, page 159–166, USA, 2002. IEEE Computer Society.

[Tur37] A. M. Turing. Computability and λ-definability. *The Journal of Symbolic Logic*, 2(4):153–163, 1937.

[ZM22] Maaike Zwart and Dan Marsden. No-Go Theorems for Distributive Laws. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022.

Glossary

λ-Term – A λ-calculus term. 11–14, 18, 20, 41, 43, 45

Abstraction – A term such that applying an argument and reducing results in a substitution; i.e. a function. 11–13, 15, 16, 18, 21, 26, 29–31, 42, 43, 66, 102

Application – A term where an argument is applied to a function; the act of applying an argument to a function. 11, 13, 16, 26, 36, 41–43, 46, 49, 50, 64, 67, 79, 85, 102, 128, 157

Closed – A term is called closed if it has no free variables. 12, 16, 27–32, 35–37, 48, 71, 72

Combinator – A closed λ-term. 12, 64

Normal form – A term which cannot reduce. 13–15, 19, 35, 40

Head normal form – A term which does not have a redex at head position. 13

Weak head normal form – A term is in weak head normal form if there is no redex at the leftmost outermost part of the term. 14, 15

π-Calculus – A calculus designed to model concurrency. 45

PCF term – A PCF term. 16, 17

PCF program – A PCF term which is closed. 16

Abstract machine – A model which allows for sequential execution of programs similar to a computer system. 7, 8, 25, 28, 41, 44, 45, 158

Krivine Abstract Machine – A simple abstract machine performing weak head reduction on λ-terms. 43, 44

Milner Abstract Machine – A variation of the KAM where all variables are global via α equivalence. 44, 45

Register machine – An abstract machine manipulating registers containing integers according to a set of rules. 45

SECD Machine – The first abstract machine designed to interpret λ-terms.. 41

Turing machine – An abstract machine manipulating symbols on an infinite tape according to a set of rules. 44, 45

Abstract programming language – A formal system which resembles a programming language and is equally capable, but abstracts away some of the nuances required of a programming language. 7, 14, 17, 21

Address table map – A bijective function between a set of addresses and machines over said set. 50, 51, 57, 64

Addressing Machine (AM) – An abstract machine which exists in a network with other addressing machines, whose only data type is the addresses of machines. 7, 8, 44–46, 50, 55, 158

Address – A value which is assigned to a particular machine in a bijective manner. 45–51, 55–57, 64, 69, 74, 77, 158

Extended Addressing Machine – An addressing machine extended with operators and predefined machines for numeral computations and fixed point. 8, 9, 45, 47–53, 55–57, 63–70, 72–74, 85–87, 89, 155

Valid program – A program which does not read uninitialised registers. 48, 49, 54, 96, 157

Adjoint functors (adjoint) – A pair of functors which morally act as inverse to one another. 99, 100, 104, 117, 141

Counit – A particular natural transformation formed by adjoint functors. 100

Unit – A particular natural transformation formed by adjoint functors. 96, 97, 99, 100, 103, 133, 141, 150

Cartesian closed category (CCC) – A category which is cartesian closed, one where for all objects in the category the arrows between them are also arrows in the category in a well-behaved manner. 22, 23, 104–107, 118

Category – A formalism which exists to study the connections between objects. 7, 9, 95–98, 100–106, 108, 118, 141, 142, 156

Arrow – A morphism in a category. 95–100, 106, 107

Categorification – The act of lifting a notion from another branch of mathematics into category theory. 100, 104

Diagram – A visual representation of commuting arrows. 97, 99–103, 106, 107, 116, 117, 129, 136, 139, 140, 142, 143, 146, 156

Circular definition – A definition is circular if it references itself in its definition. 42

Closure – A construct which pairs instructions with local variable assignments to enable functions as a data type. 42–45

Coefficientless expansion – A method to shift from the exponent of a sum to a sum of exponents without introducing an additional coefficient. 124, 129, 148

Complete partial order – A type of partial order which has additional requirements regarding subsets and supremums. 105, 111

Complete semiring (complete parent) – A semiring with a well-defined infinite sum. 109, 111, 157

Confluent – Confluence is a property of a formal system; In such a formal system, the order of reductions does not matter when considering the outcome. 13

Context – A term which has one unique hole in it where a subterm could be located. 20, 21, 27, 30, 46, 55

Denotational semantics – A mathematical interpretation of the underlying meaning of a language. 21

Distributive law (distribution) – A morphism causing the composite of two monads to be a monad. 9, 10, 109, 117, 119, 128, 129, 133, 136, 139, 141, 146, 147, 150, 155–157

Unnatural distributive law – A distributive law between monads which is not natural. 141, 142, 146, 147, 155–157

Equivalence – Two programs are referred to as equivalent if they have the same meaning. Also used to refer to an equivalence relation. 12, 19, 20, 22, 27, 74, 82, 84

α-Equivalence – Two terms are alpha equivalent if they only differ in the names of their bound variables. 12, 13, 17, 26, 27, 29, 31, 44

Applicative equivalence – Two PCF programs are referred to as applicatively equivalent if they always reduce to the same number when given an appropriate number of valid arguments. 21

Observational equivalence – Two programs are referred to as observationally equivalent if they have the same meaning, i.e. they can be substituted for one another in all circumstances without changing the outcome. 20, 21, 23, 31

Exponential action – A function allowing one semiring to act on another in a manner similar to an exponent. 123, 124, 129, 134, 147, 148, 157

Finite products – An attribute a category can have; related to the cartesian product. 102, 104, 157

Finite support – A function has finite support if there are only a finite number of elements in its domain for which it does not output 0. 118, 122, 125, 129, 133, 147, 157

Formal system – A calculus in mathematical logic, given by the rules of formation of expressions and of constructing derivations in that calculus. 7, 9, 11, 20, 22, 23

Functor – A structure preserving arrow between categories. 98–101, 103, 104, 115, 117, 133, 141, 143, 144, 150, 156

Endofunctor – A functor from a category to itself. 100, 104, 141

Interaction pair – A pair of morphisms between semiring monads allowing an (unnatural) distribution to arise.. 147

Coefficient – A morphism forming an interaction pair together with an exponential action.. 148, 149, 156, 157

Isomorphism – An arrow which has a perfect inverse; the categorification of bijection. 100, 103, 104

λ-Calculus – A model of computation where everything is a function. 7, 11, 14, 16, 17, 20, 25, 26, 28, 32, 42–44, 104

$\lambda\sigma$-Calculus – A λ-calculus endowed with explicit substitutions. 25, 27

Simply typed λ-calculus – The λ-calculus adorned with simple types. 14–16, 19, 22, 102, 105, 108

Untyped λ-calculus – A model of computation where everything is a function, without an included type system. 11, 13–15, 22, 27

Linear Logic – A logic with restrictions on weakening and contraction; the logic of resources. 7, 9, 156, 157

Linear Substitution Calculus (LSC) – A calculus with explicit substitutions based on Linear Logic. 27, 31, 44

Logic of Computable Functions (LCF) – A logical calculus for computable function. 7, 16

Model – A model of a programming language is a mathematical object with a mapping from terms to elements of the object. 7–9, 21–23, 41, 44, 63, 74, 84, 85, 94, 95, 102, 104, 105, 107, 108, 117, 155–158

Adequate model – A model is adequate if it coincides with its language on the meaning of terms. 22, 23, 85

Complete model – A model is complete if it does not differentiate between terms which have the same meaning. 22, 23, 85, 94, 109, 111, 114, 115, 155, 156

Fully abstract model – A model is fully abstract if it is both adequate and complete. 7–9, 22, 23, 94

Monad – A construction induced by an adjunction. Morally, a sort of "box" that objects can be put into, where a box of a box can be flattened to be just a single box. 9, 10, 100–102, 114–118, 120, 122, 125, 129, 133, 136, 138, 139, 141–143, 146, 147, 155–157

Kleisli category – The free category induced by a monad. 101, 117, 118, 133, 141, 146, 147

Semiring monad – A monad on the category of sets and functions induced by a semiring. 117, 121, 147, 150, 155

Monoid – A set with an associative binary operation that has an identity element. 96, 97, 100, 104, 108, 156

Natural transformation (natural) – A structure preserving arrow between functors. 99, 100, 102–104, 108, 115, 120, 123, 128, 129, 133–137, 139, 141, 142, 146, 149, 155–157

Unnatural transformation – An arrow between functors. 139, 141, 156

Operational semantics – A formal description of how a program is interpreted through a series of computational steps. 17, 21, 23, 30, 37, 52, 63, 73

Big-step – A type of operational semantics which only describes terms and the normal form which can be obtained from it, if one exists. 8, 19, 21, 33

Small-step – A type of operational semantics which describes every reduction a term can make. 8, 17, 19, 21, 30, 52

Weak head reduction – A head reduction which is 'weak' - does not reduce inside abstractions. 17, 30, 32

Partial evaluation (Currying) – Allowing arguments to be passed to a function one at a time rather than all at once. 42

Programming Computable Functions (PCF) – An abstract programming language which can be seen as λ-calculus + types + numerals + fixed point + weak head reduction. iv, 3, 6–9, 11, 16–24, 28–31, 33, 35–40, 47, 50, 55, 63, 67, 73–76, 78, 82, 85, 86, 93–95, 102, 105, 107, 108, 117, 155, 157, 158

Programming Computable Functions with Explicit Substitutions (EPCF) – Programming Computable Functions extended with a limited form of explicit substitutions designed for closed terms. 8, 9, 28–33, 35–37, 39, 40, 55, 56, 63, 66–68, 70, 73

Projection – A morphism extracting individual elements from a product (tuple). 64, 67

Redex – A subterm within a term which can be reduced. 13, 15, 27, 44

Head position – A redex is at head position if it is the leftmost outermost redex, and is not applied to another subterm. 13, 34

Reduction – A directional relation between terms, generally describing a computational step from one term to the next. 13–15, 17, 18, 26–28, 30–33, 36, 39, 40, 44, 52, 53, 61, 68, 69, 73, 74, 95

β**-Reduction** – A particular reduction which substitutes a term for a bound variable. 13, 44

Big step reduction – A reduction which does not consider a particular path to an outcome, only the end result. 14, 19

Reduction strategy – A strategy describing which redex to reduce whenever multiple are available. 52

Call-by-name – An evaluation strategy where arguments are evaluated only when it becomes necessary after being substituted. 17, 30, 43

Call-by-need – An evaluation strategy where arguments are evaluated only when used in a function, but the resulting output is then saved to avoid repeated evaluation of the argument. 42

Call-by-value – An evaluation strategy where arguments are evaluated before being substituted. 17, 42

Head reduction – A reduction strategy which always reduces the redex at head position, if it exists. 13

Weak head reduction – A reduction strategy which reduces to a weak head normal form (when possible) and then terminates. 13–15

Register – A storage location for a single value. 45–49, 51–53, 55–57, 60, 64, 66, 74, 77, 85, 86, 89, 158

Relation – A subset of the cartesian product of two sets, describing whether or not elements have a particular relation to one another. 96, 100, 105, 111, 124, 134, 156–158

Scott-continuous functions – A function which preserves complete partial orders. 105

Semiring – A pair of monads with the same underlying set, where one monoid is commutative, with particular interactions between the monoids. 9, 108, 109, 111, 112, 114, 115, 117–121, 123, 124, 129, 133, 134, 147, 156, 157

Continuous semiring – A semiring with a well-defined infinite sum. 109, 111, 118, 157

Multiplicity semiring – A semiring which contains the semiring of natural numbers, and whose addition and multiplication are that of the natural numbers. 112, 118, 122, 124, 125, 129, 133, 157

Strict total order – An order which is not reflexive, where any two elements can be compared. 54

Strongly normalising – A formal system is strongly normalising if all its terms reduce to a normal form regardless of reduction strategy. 15, 27, 39

Substitution – The process of substituting a term N for variables in a different term M. 20, 25–29, 31, 44, 58, 59

Capture free substitution – The process of substituting a term N for variables in a different term M, without changing the overall meaning of N. 12, 13, 17, 26, 27, 29

Explicit substitution – A substitution which is part of the syntax of a language. 7, 8, 25–32, 35, 37, 39, 44, 67

Symmetric monoidal category – A category equipped with a strongly symmetric tensor product that acts similar to the monoidal product. 103, 104

Symmetric monoidal closed category – A symmetric monoidal category that is closed with respect to its monoidal structure. 104, 117

Thunk – A function created to delay the execution of an expression until a later time – often used to enable call-by-name or call-by-need evaluation in a system which is natively call-by-value. 42

Transitive-reflexive closure – The transitive-reflexive closure of a relation acting on a set is the smallest reflexive and transitive relation which acts on said set and contains the original relation. 13, 17, 30, 52

Translation – A function between formal languages. 65–68, 85, 86

Turing complete – A formal system is Turing complete if it can simulate any Turing machine. 11, 15, 16

Type – An element of a set of types that can be assigned to terms. 8, 14–16, 19–22, 28, 31, 35–37, 40, 41, 43, 45, 47, 55–59, 65, 67, 70, 73–75, 79–81, 84–87, 93, 136, 156

Typing context – An allocation of types to variable names. 14, 20, 21, 55, 86

Typing derivation – A series of typing judgements linked together via derivation rules to prove that a term is typable with a particular type. 15, 57

Typing judgement – A statement which says that using a particular context, a term can be given a particular type. 14, 15, 55–58

Variable – A symbol acting as a placeholder or representative of something. 11, 12, 18, 25, 27, 28, 42–45, 53, 58, 59, 67

Bound variable – A variable is bound when it has been "captured" by an abstraction. 12, 13, 42

Free variable – A variable is free when it has not been "captured" by any abstraction. 8, 12, 16, 26, 28–31, 55, 66, 67

Fresh variable – A variable is fresh with respect to a term if it does not appear in said term. 12, 29, 50, 58, 59

Global variable – A variable set to be permanently accessible during the runtime of a program. 44, 45

Local variable – A variable only accessible by a particular "block" of a program. 42, 44, 45

Variable convention – Bound variables in terms always have maximally distinguishable names. 13

Weakly normalising – A term or reduction sequence is weakly normalising if there exists at least one reduction sequence to reduce the term to a normal form. A rewriting system is weakly normalising if all terms are weakly normalising. 15, 27

Witness – A morphism that "witnesses" a particular way to pair two collections of elements together. 120, 121, 133, 148

Appendix

There are three points that warrant mentioning which are not discussed in the main body of the thesis. Two of these are quite standard discussions when the subject arises: ZFC set theory's connections to category theory, and modelling the simply typed λ calculus with cartesian closed categories. Both of these were promised in the main body, and the first in particular must be given a brief discussion. The other point to discuss is double categories – these were used as a tool when working on Chapter 7, but are not helpful in understanding the content, so their definitions and some brief discussion on them was moved to the appendix. Most of this content was taken from the nlab [Com].

Set Theory and Category Theory

Category theory is defined by using sets. Thus, to define categories, we require some understanding of what a set is. Ideally, we could use the naive approach: a set is a potentially infinite collection of elements that respect some defined property. However, this approach was disproven in 1901 by what is now known as "Russell's Paradox": If one takes the naive approach, then one can define a "set of all sets". Then one could define the contradictory set

$$A = \{X | X \notin X\}$$

The question of whether or not $A \in A$ leads to a paradox, thus the naive approach cannot be used. The typical approach to resolving the issue is Zermelo–Fraenkel set theory, also known as ZFC set theory when the axiom of choice is included. There are also extensions to ZFC, in particular von Neumann–Bernays–Gödel (NBG) set theory.

The resolution used is intuitively described as follows: Instead of defining sets the naive way, we define sets as what is formally known as *small sets*. We then forbid the existence of a "small set of all small sets", but instead use *class* to describe all collections which are too "large" to be called a "small set". Thus all small sets are classes, but not all classes are small sets. We run into a similar issue here, though, in that one cannot speak of the "class of all classes". Thus we include in our definition a sort of "levelling up" ability – when we need to work with objects which are classes but not sets in a "dangerous" manner, we consider a type of collection "class+1" defined as all collections which are too large to be called classes, and then rearrange our definitions – we use "small set" to refer to what was previously called a class, and "class" to refer to what was previously called "class+1".

Simply Typed λ Calculus and Cartesian Closed Categories

Any cartesian closed category can be used to construct a sound model of the simply typed λ calculus. To explain what is meant by this, we need to first define some "missing" parts from our definitions of the λ-calculus:

Definition .0.9.

- *Define* η-reduction *as follows:* $\lambda x.Mx \to_\eta M$. *Essentially, removing a "useless" abstraction.*
- *Define* $\beta\eta$*-reduction, written* $\to_{\beta\eta}$ *as the union of* β*-reduction and* η*-reduction.*
- *Define* $\twoheadrightarrow_{\beta\eta}$ *as the transitive-reflexive closure of* $\to_{\beta\eta}$.
- *Define* $\beta\eta$*-equivalence, written* $=_{\beta\eta}$*, where for any two terms* M, N, $M =_{\beta\eta} N$ *if and only if at least one of the following statements hold:*
  - $M \twoheadrightarrow_{\beta\eta} N$
  - $N \twoheadrightarrow_{\beta\eta} M$
- *Define* $\alpha\beta\eta$*-equivalence, written* $=_{\alpha\beta\eta}$*, as the union of* α*-equivalence and* $\beta\eta$*-equivalence.*

By a sound model of the simply typed λ calculus, one means that there is the presence of some mathematical construct C containing distinct elements that can be compared to see if they are equivalent and some interpretation function $[\![1]\!]$ which has the following behaviour

- For every element $\alpha \in \mathbb{T}^{\mathbb{B}}$, i.e. every type that a term can be assigned, $[\![\alpha]\!]$ is an element of C.
- For every typed term $\Gamma \vdash M : \alpha$, $[\![\Gamma \vdash M : \alpha]\!]$ is an element of C.
- For every pair of typed terms $\Gamma_1 \vdash M_1 : \alpha_1$, $\Gamma_1 \vdash M_1 : \alpha_1$, if $M_1 =_{\alpha\beta\eta} M_2$, then $[\![\Gamma_1 \vdash M_1 : \alpha_1]\!]$ and $[\![\Gamma_2 \vdash M_2 : \alpha_2]\!]$ are equivalent in the mathematical construct.

When the mathematical construct C is a monoidal category $\mathbb{C}$ (symmetric monoidal category without the braiding), then the above definition is equivalent to the following:

- For every element $\alpha \in \mathbb{T}^{\mathbb{B}}$, i.e. every type that a term can be assigned, $[\![\alpha]\!] \in \mathrm{Ob}(\mathbb{C})$.
- For every typed term $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \alpha$, $[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash M : \alpha]\!] \in \mathbb{C}([\![\alpha_1]\!] \otimes \cdots \otimes [\![\alpha_n]\!], [\![\alpha]\!])$
- For every pair of typed terms $\Gamma_1 \vdash M_1 : \alpha_1$, $\Gamma_1 \vdash M_1 : \alpha_1$, if $M_1 =_{\alpha\beta\eta} M_2$, then there is a natural isomorphism in $\mathbb{C}$ between $[\![\Gamma_1 \vdash M_1 : \alpha_1]\!]$ and $[\![\Gamma_2 \vdash M_2 : \alpha_2]\!]$.

If $\mathbb{C}$ is a cartesian closed category (reminder that then $\otimes = \times$), then we can use the properties of cartesian closure to inductively define such a model without any additional requirements:

- We assign to each base type $o \in \mathbb{B}$ an element $[\![o]\!] \in \mathrm{Ob}(\mathbb{C})$.

- Given types $\alpha, \beta \in \mathbb{T}^{\mathbb{B}}$, we define $[\![\alpha \to \beta]\!]$ as $[\![\alpha]\!] \multimap [\![\beta]\!]$, using the closed structure of $\mathbb{C}$.
- Given the typed term $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash x_i : \alpha_i$, we define

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash x_i : \alpha_i]\!] = \pi_i \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\alpha_i]\!])$$

  This is using the finite products of $\mathbb{C}$.
- Given the typed term $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash \lambda y.M : \beta \to \gamma$, we obtain the morphism

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash \lambda y.M : \beta \to \gamma]\!] \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\beta]\!] \multimap [\![\gamma]\!])$$

  by applying the closed structure of $\mathbb{C}$ to the morphism given by

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n, y : \beta \vdash M : \gamma]\!] \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!] \times [\![\beta]\!], [\![\gamma]\!])$$

- Given the typed term $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash MN : \gamma$, we obtain the morphism

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash MN : \gamma]\!] \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\gamma]\!])$$

  by using the finite product structure of $\mathbb{C}$ on the morphisms given by

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash \lambda y.M : \beta \to \gamma]\!] \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\beta]\!] \multimap [\![\gamma]\!])$$

  $$[\![x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash N : \beta]\!] \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\beta]\!])$$

  This gives us a morphism

  $$\varphi \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\beta]\!] \multimap [\![\gamma]\!] \times [\![\beta]\!])$$

  We then compose this morphism with the counit $\epsilon_A : A \multimap B \times A \to B$ of the adjunction providing the closed structure to obtain the desired morphism

  $$\varphi; \epsilon_A \in \mathbb{C}([\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!], [\![\gamma]\!])$$

Double Categories

Intuitively, a double category is a type of category with two related types of morphisms. Morphisms of double categories are typically presented as squares, such as the following square:

$$\begin{array}{ccc} A & \xrightarrow{u} & B \\ {\scriptstyle f}\downarrow & \Downarrow{\scriptstyle \alpha} & \downarrow{\scriptstyle g} \\ C & \xrightarrow[v]{} & D \end{array}$$

Here A, B, C, D are the objects of the double category, f, g are its vertical morphisms, u, v are its horizontal morphisms, and α is called the "square". Double categories can be restricted to a standard category in every direction.

Definition .0.10 (Double Categories)**.**
A double category $\mathbb{C}$ is a category internal to $\mathbf{Cat}$*, the category of small categories. In other words, a double category consists of:*

- *A (vertical) category* $\mathbb{C}_v$*;*
- *An arrow category* $\mathbb{C}_a$*;*
- *A* source *functor* $s : \mathbb{C}_a \to \mathbb{C}_v$ *taking a morphism* $u : x \to y \in \mathrm{Ob}(\mathbb{C}_a)$ *to* $x \in \mathrm{Ob}(\mathbb{C}_v)$ *and a square* $\alpha \in \mathbb{C}_a(A \multimap B, C \multimap D)$ *to a morphism* $f \in \mathbb{C}_v(A, C)$*;*
- *A* target *functor* $t : \mathbb{C}_a \to \mathbb{C}_v$ *taking a morphism* $u : x \to y \in \mathrm{Ob}(\mathbb{C}_a)$ *to* $y \in \mathrm{Ob}(\mathbb{C}_v)$ *and a square* $\alpha \in \mathbb{C}_a(A \multimap B, C \multimap D)$ *to a morphism* $g \in \mathbb{C}_v(B, D)$*;*
- *An* identity-assigning *functor* $e : \mathbb{C}_v \to \mathbb{C}_a$ *which takes an object* $x \in \mathrm{Ob}(\mathbb{C})_v$ *to the identity morphism* $\mathrm{id}_x : x \to x \in \mathrm{Ob}(\mathbb{C})_a$*;*
- *A* composition *functor* $c : \mathbb{C}_a \times \mathbb{C}_a \to \mathbb{C}_a$ *acting as horizontal composition;*

such that the following diagrams commute:

- *Source and target of the identity morphism:*

$$\begin{array}{ccc} \mathbb{C}_v & \xrightarrow{e} & \mathbb{C}_a \\ & \searrow & \downarrow{\scriptstyle s} \\ & & \mathbb{C}_v \end{array} \qquad \begin{array}{ccc} \mathbb{C}_v & \xrightarrow{e} & \mathbb{C}_a \\ & \searrow & \downarrow{\scriptstyle t} \\ & & \mathbb{C}_v \end{array}$$

- *Source and target of the composition morphism:*

$$\begin{array}{ccc} \mathbb{C}_a \times \mathbb{C}_a & \xrightarrow{c} & \mathbb{C}_a \\ \downarrow{\scriptstyle \pi_1} & & \downarrow{\scriptstyle s} \\ \mathbb{C}_a & \xrightarrow[s]{} & \mathbb{C}_v \end{array} \qquad \begin{array}{ccc} \mathbb{C}_a \times \mathbb{C}_a & \xrightarrow{c} & \mathbb{C}_a \\ \downarrow{\scriptstyle \pi_2} & & \downarrow{\scriptstyle t} \\ \mathbb{C}_a & \xrightarrow[t]{} & \mathbb{C}_v \end{array}$$

- *The composition morphism is associative:*

$$\begin{array}{ccc} (\mathbb{C}_a \times \mathbb{C}_a) \times \mathbb{C}_a & \xrightarrow{c \times \mathrm{id}_{\mathbb{C}_a}} & \mathbb{C}_a \times \mathbb{C}_a \\ \downarrow{\scriptstyle \langle \pi_1 \circ \pi_1, \pi_2 \circ \pi_1 \times \mathrm{id}_{\mathbb{C}_a} \rangle} & & \\ \mathbb{C}_a \times (\mathbb{C}_a \times \mathbb{C}_a) & & \downarrow{\scriptstyle c} \\ \downarrow{\scriptstyle \mathrm{id}_{\mathbb{C}_a} \times c} & & \\ \mathbb{C}_a \times \mathbb{C}_a & \xrightarrow[c]{} & \mathbb{C}_a \end{array}$$

- *Units for composition:*

$$\begin{array}{ccccc} \mathbb{C}_a \times \mathbb{C}_v & \xrightarrow{\mathrm{id} \times e} & \mathbb{C}_a \times \mathbb{C}_a & \xleftarrow{e \times \mathrm{id}} & \mathbb{C}_v \times \mathbb{C}_a \\ & {\scriptstyle \pi_1} \searrow & \downarrow{\scriptstyle c} & \swarrow {\scriptstyle \pi_2} & \\ & & \mathbb{C}_a & & \end{array}$$

Regarding the "restriction" of a double category to various standard categories, the method of obtaining the categories in the vertical direction is obvious from the definition. The horizontal direction relies on the morphism c for composition, and e for the identities.

Lemma .0.11.
Let R *be a monad on a category* $\mathbb{C}$*. We can form a double category* $\mathbb{C} \boxtimes \mathbb{C}_{\mathrm{R}}$ *by selecting* $\mathbb{C}$ *as the vertical category, with the arrow category is defined using the arrows of* $\mathbb{C}_{\mathrm{R}}$ *as objects and equality as morphisms.*

Proof. $\mathbb{C}$ is a category as given. Proving that the arrow category as described is indeed a category is trivial. For the source and target, we have (for all $f \in \mathbb{C}_{\mathrm{R}}(A, B)$)

$$s(f) = A, t(f) = B$$

The identity-assigning morphism e sends $X \in \mathrm{Ob}(\mathbb{C}_v)$ to $\mathrm{id}_X \in \mathbb{C}_{\mathrm{R}}(X, X)$. The composition morphism c sends a pair of morphisms $f : A \to \mathrm{R}B$, $g : B \to \mathrm{R}C$ to $f; \mathrm{R}g; \mu^{\mathrm{R}}_C : A \to \mathrm{R}C$.

All diagrams are then trivial to check. □

Double categories allow us to redefine monads in a double-categorical setting.

Definition .0.12 (Double Functor)**.**
A double functor is a functor between double categories. Let $\mathbb{C}$ *and* $\mathbb{D}$ *be double categories, and denote with* $\mathbb{C}_v, \mathbb{D}_v$ *the corresponding vertical categories and* $\mathbb{C}_a, \mathbb{D}_a$ *the corresponding arrow categories. A double functor* $\mathrm{F} : \mathbb{C} \to \mathbb{D}$ *consists of:*

- *A functor* $F_v : \mathbb{C}_v \to \mathbb{D}_v$*;*
- *A functor* $F_a : \mathbb{C}_a \to \mathbb{D}_a$*;*

such that for all $A \in \mathbb{C}_v, X, Y \in \mathbb{C}_a$*,*

- $s(F_a X) = F_v(s(X))$*;*
- $t(F_a X) = F_v(t(X))$*;*
- $c(F_a X \times F_a Y) = F_a(c(X \times Y))$*;*
- $e(F_v A) = F_a(e(A))$.

We often write (F_v, F_a) *for the double functor* F.

From the definition of a double functor we can define the notion of a monad in a double category.

Definition .0.13.
A double monad M *in a double category* $\mathbb{C}$ *consists of:*

- *A monad* M_v *in* $\mathbb{C}_v$*;*
- *A monad* M_a *in* $\mathbb{C}_a$*;*

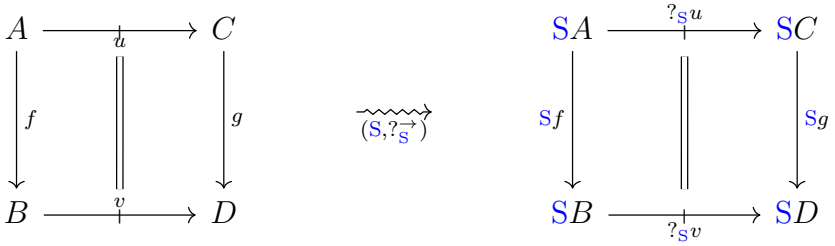
such that

- $(\mathrm{M}_v, \mathrm{M}_a)$ *is a double functor;*
- $s(\eta^{\mathrm{M}_a}) = \eta^{\mathrm{M}_v}$*;*
- $s(\mu^{\mathrm{M}_a}) = \mu^{\mathrm{M}_v}$*.*

We often write $(\mathrm{M}_v, \mathrm{M}_a)$ *for the double monad* M*.*

This double-categorical framework allows for a different perspective to the distribution of monads. We will later use this new perspective to help understand how the monads $\bar{\mathrm{N}}$ and $\mathrm{N_f}$ interact.

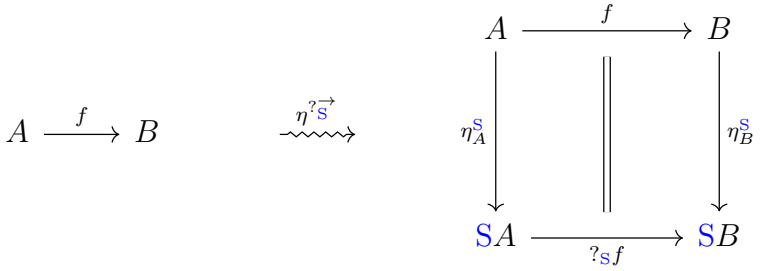
Proposition .0.14.
Given monads R, S *on* $\mathbb{C}$*,* R *distributing over* S *(there is a distributive law* θ*) implies that there exists a monad* $?_{\mathrm{S}}$ *on* $\mathbb{C}_{\mathrm{R}}$ *such that we can form a composite double monad* $(\mathrm{S}, ?_{\mathrm{S}}^{\rightarrow})$ *on* $\mathbb{C} \boxtimes \mathbb{C}_{\mathrm{R}}$ *acting as follows:*



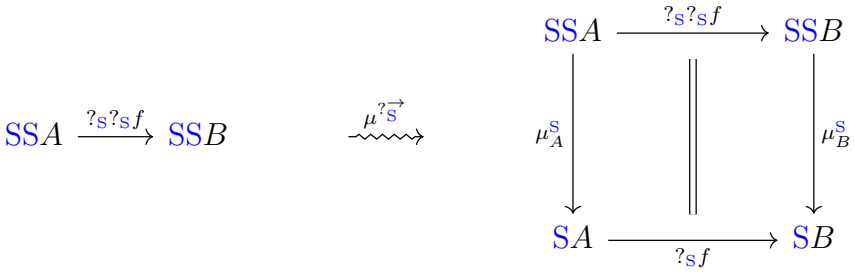
Proof. We set $?_{\mathrm{S}}$ to be the monad $\theta \circ \mathrm{S}$. We define the monad $?_{\mathrm{S}}^{\rightarrow}$ acting on $(\mathbb{C}_{\mathrm{R}})_a$ as follows:

- Let $u \in \mathbb{C}_{\mathrm{R}}(A, B) \subset \mathrm{Ob}((\mathbb{C} \boxtimes \mathbb{C}_{\mathrm{R}})_a)$. We set $?_{\mathrm{S}}^{\rightarrow}(u) = ?_{\mathrm{S}}(u)$.
- $?_{\mathrm{S}}^{\rightarrow}$ takes commuting squares $(v \circ f = g \circ u) \in (\mathbb{C} \boxtimes \mathbb{C}_{\mathrm{R}})_a(A \multimap \mathrm{R}C, B \multimap \mathrm{R}D)$ to the square $(?_{\mathrm{S}}v \circ \mathrm{S}f = \mathrm{S}g \circ ?_{\mathrm{S}}u) \in (\mathbb{C} \boxtimes \mathbb{C}_{\mathrm{R}})_a(\mathrm{S}A \multimap \mathrm{RS}C, \mathrm{S}B \multimap \mathrm{RS}D)$.

Functoriality is inherited from the functoriality of S and $?_{\mathrm{S}}$ (composition of commuting diagrams). The unit $\eta^{?_{\mathrm{S}}^{\rightarrow}}$ acts on functions $f : A \to B$ as follows:



The multiplication $\mu^{?_{\mathrm{S}}^{\rightarrow}}$ acts on functions $f : A \to B$ as:



The remaining checks (naturality of $\eta^{?\overrightarrow{s}}$ and $\mu^{?\overrightarrow{s}}$, the diagram for $\eta^{?\overrightarrow{s}}$, and the diagram for $\mu^{?\overrightarrow{s}}$) are trivial, albeit a bit annoying to display due to the graphical representation of said diagrams being cubes. □