UNIVERSITY OF CAMBRIDGE

Formal p-Category Theory and Normalisation for Simple Type Theory

David George Berry

Churchill College

October 2025

This thesis is submitted for the degree of Doctor of Philosophy.

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or, is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

David George Berry
February 2026

Abstract

Formal P-Category Theory and Normalisation for Simple Type Theory

David George Berry

This thesis extends P-category theory, introduced in Čubrić et al. (1998), and develops P-bicategory theory, and thereafter uses them to conduct a P-categorical analysis and synthesis of normalisation by evaluation for simple type theory. P-category theory was introduced as a non-standard categorical framework for phrasing the normalisation by Yoneda embedding result of Čubrić et al. (1998). They provide only a minimal collection of definitions for their result and analysis. We extend their base theory into encompassing more definitions from standard category theory thereby demonstrating the robustness of their base theory as well as its utility in providing a non-standard framework for formalised category theory. Moreover, we augment P-category theory into P-bicategory theory, allowing the incorporation of bicategory theory into this non-standard framework. We use this P-categorical framework to formalise the categorical and universal structure of simple type theory, allowing the reconstruction of the normalisation result of Čubrić et al. (1998). We broaden their result with a fuller analysis, allowing for alternative presentations thereof, and combine their techniques with a new universal property of unquotiented well-typed syntax of simple type theory. This allows for a fully-categorical proof of the basic correctness properties of normalisation by evaluation for simple type theory, without resorting to neutral and normal types, a proof not known to have been done before. Finally, we use our P-categorical framework to formalise the normalisation result of Fiore (2002, 2022), to allow comparisons to be made with our work, and to demonstrate the strengths of P-category theory for type-theoretic formalisation. All the normalisation results, and P-categorical constructions required therefor have been formalised in the proof assistant Rocq.

Acknowledgements

First and foremost, I would like to acknowledge and thank my wife, Eleanor, whom I met just as I started as a Doctoral student, without whom these last years would not have been so enjoyable, and who has softened the many hardships thereof. Most especially I thank her for supporting me whilst I took my time writing this thesis, and believing in me during this process.[1] I would like to thank my parents, Colin and Lynda, and her parents, Douglas and Catherine, for providing support throughout the course of research and writing.

My thanks also extend to my supervisor, Professor Marcelo Fiore, for taking me on as a Doctoral student during the challenging times of late 2021 and beyond, for guiding me during the early stages of research and giving me an interesting subject matter about which to think. Sanjiv, another of Marcelo's students, who started a year after I did, has provided many fun and interesting conversations about Category Theory, Mathematics, and life in general from his neighbouring office. Dima, also (at that time) another of Marcelo's students, proof-read my proposal in my application and gave useful feedback. Meven, during his time as a post-doctoral researcher, occasionally and very usefully advised about Rocq.

I would like to thank Dr John Fawcett, my Director of Studies when I was an undergraduate at Churchill College, for admitting me, and taking me from being a fledgling Computer Scientist to a researcher hopefully worthy of the name. Additionally, he has provided me with plenty of undergraduate students to supervise, allowing me to hone my teaching skills and connect more deeply with the content I claimed to know from my own time as an undergraduate student. Finally, I thank him for entrusting me with joining him, and his colleagues Matthew Ireland and Luana Bulat, in interviewing applicants for Computer Science.

Last, but not least, I thank the people of the churches of S. Botolph and S. Mary the Less, for providing a spiritual home during my years in Cambridge both as an undergraduate student and as a graduate student, and of S. Philip the Apostle, South Tottenham for providing a spiritual home whilst living in Muswell Hill and writing this thesis. I thank Almighty God Who hath set the path of these last years before me, and Who knoweth the path ahead.

Finally, I would like to disacknowledge all sources of funding for declining to fund my research, and the Department of Computer Science-and-Technology for advising "*... please do reapply. It would be a far better scheme than trying to self-fund your PhD. We strongly advise against the self-funding route.*". I would like to thank Churchill College for their rigorous patience with me due to my self-funding.

[1] I also thank her for her thorough proof-reading of this thesis, identifying both missing and excessive commata, and other poor punctuation and grammar.

To my dearest wife.

Contents

List of Rocq Code Listings

Part I

Introduction, Background, and Rocq Formalism

CHAPTER 1

Introduction

In this chapter, we introduce some required preliminary concepts for the subject of this thesis: *viz.*, simple type theory, category theory, and Rocq formalisms. We also briefly overview the structure of the thesis, and advise about the style of presentation throughout the thesis.

Although we rely on some of the content contained in the preliminaries, any reader sufficiently familiar with each of the subjects of the preliminaries can safely jump over each thereof.

1.1 Outline

The body of this thesis is composed of three parts, each of which is comprised of a number of chapters. The first part, Contents, has three chapters offering introductory material and background to the subject of this thesis, and reviewing type-theoretic formalisms in the Rocq proof assistant. The second part, Notations, has two chapters. The first of these chapters introduces p-category theory, and extends concepts from standard category theory to the p-setting. The second introduces p-bicategory theory, building on the definitions and concepts from the preceding chapter. It provides a number of standard bicategorical constructions and definitions within the p-setting. The third part, Comments, has three chapters and a final conclusion. The first of these chapters presents a formalism of simple type theory amenable to type-theoretic formalism. The second gives a p-categorical, and subsequently a p-bicategorical, analysis of the universal structure of simple type theory. The third of these chapters constructs a number of p-categorical normal form algorithms based on the p-categorical analysis of simple type theory from the preceding chapter. Finally, we conclude this thesis with a summary of the work presented and the main contributions thereof, and some suggestions for possible future work.

1.2 Presentation

Throughout this thesis we give many definitions that have both a purely Mathematical presentation, and the form used in the Rocq formalism. We give these side-by-side in the main chapters; the reader unfamiliar with Rocq code, for the most part, should be able to ignore the definitions given thus. Nonetheless, for some of the discussion around the form of the definitions, attention to the Rocq formalism's definitions may prove useful. Moreover, we are a little relaxed around technicalities of the Mathematical definitions, confident that any such technicalities are addressed in the Rocq formalism and the definitions thereof. When a Mathematical theory is formalised many definitions of the theory need to be modified so as to fit the nature of type-theoretic formalism. For us, this is a lesser consideration as the Mathematical theory has been developed alongside a type-theoretic formalism in Rocq. We have used the Rocq formalism to guide how we phrase definitions, and build up and present the Mathematical theories.

$$\frac{x \in \mathbb{V}}{x \in \Lambda} \qquad \frac{t_1 \in \Lambda \qquad t_2 \in \Lambda}{t_1\, t_2 \in \Lambda} \qquad \frac{x \in \mathbb{V} \qquad t \in \Lambda}{\lambda x.t \in \Lambda}$$

Figure 1.1: Term Structure of the λ-Calculus

$$\begin{aligned} \mathrm{fv}(x) &= \{x\} \\ \mathrm{fv}(t_1\, t_2) &= \mathrm{fv}(t_1) \cup \mathrm{fv}(t_2) \\ \mathrm{fv}(\lambda x.t) &= \mathrm{fv}(t) \setminus \{x\} \end{aligned}$$

Figure 1.2: Free Variables for the λ-Calculus

1.3 Preliminaries

This thesis covers two main subjects: category theory, and simple type theory. A third strand is woven into these aspects by the presence of a Rocq formalism accompanying the theoretical presentation. As such it is thought expedient to have a brief introduction to these concepts, with some historical remarks.

1.3.1 *Simple Type Theory*

Simple Type Theory (STT), due to Church (1940), finds its origins in the λ-calculus, and early analysis thereof. It is also known as the *simply typed λ-calculus* (STLC). We include neither product types nor sum types in our simply typed λ-calculi. There are a number of different definitions of these terms, each of which is its own Mathematical theory with its own metatheory. Although these theories are often very similar with some minor differences, for the sake of concreteness we lay out here a definition of simple type theory that suffices for this thesis.

First, we begin by defining the λ-calculus. Thereafter we proceed to giving a basic overview of the types of simple type theory.

There is just one concept in the λ-calculus: the λ-abstraction. It is intended to serve as an abstraction of the concept of function found in everyday Mathematics.

The terms of the λ-calculus are defined by the inductive rules given in figure 1.1, where $\mathbb{V}$ is some collection of variables that supports decidable (non-)equality, and is not finitely exhaustible (*i.e.*, every finite collection of variables can be extended with a new variable not already in the given finite collection).

Present within the term structure of the λ-calculus is a binding operator: the 'x' in '$\lambda x.t$' is bound. This necessitated (within the current presentation) naming 'x' as 'x'; however, it would have served equally well to have used 'y' instead as long as we consistently renamed the 'x's into 'y's within 't'. Here consistent renaming means respecting the binding graph of the term as in the work of Kahl (1998). We consider these two terms, and all other terms generated similarly from any of our countable set of variable to be α-equivalent. All further definitions are thus required to respect α-equivalence. This binding structure induces a trichotomy on the elements of our set of variables: the variables that are bound, the variables that are free, and the variables that do not occur. The first and third of these do not respect α-equivalence (although their union does), and as such we do not consider them any further. Nonetheless, it is useful to consider the second subset of free variables which we define in figure 1.2.

The definition of free variables in figure 1.2 is relatively subtle for two reasons: it does not proceed by recursion on the terms themselves, moreover it requires some choice of variable to substitute into λ-abstractions. First, It fails to be a definition by recursion as the recursion for substituting into λ-abstractions does not use a sub-term of the input: it uses a renamed form of the input. However, this renamed form of the input has exactly the same complexity of the input measured by size of the term as it only changes variables into other variables, therefore the recursion is not ill-formed.

$$
\begin{aligned}
x[y \mapsto t] &= \begin{cases} t & x = y \\ x & x \neq y \end{cases} \\
(t_1\, t_2)[y \mapsto t] &= (t_1[y \mapsto t])\,(t_2[y \mapsto t]) \\
(\lambda x.t')[y \mapsto t] &= \lambda z.(t'[x \mapsto z][y \mapsto t]) \qquad \text{for some } z \neq y \text{ and } z \notin \mathrm{fv}(t)
\end{aligned}
$$

Figure 1.3: Substitution for the λ-Calculus

$$
\frac{}{x \equiv_\alpha x} \qquad \frac{t_1 \equiv_\alpha t_1' \qquad t_2 \equiv_\alpha t_2'}{t_1\, t_2 \equiv_\alpha t_1'\, t_2'}
$$

$$
\frac{t_1[x \mapsto z] \equiv_\alpha t_2[y \mapsto z]}{\lambda x.t_1 \equiv_\alpha \lambda y.t_2} \text{ for some } z \notin \mathrm{fv}(t_1) \cup \mathrm{fv}(t_2)
$$

Figure 1.4: α-Equivalence for the λ-Calculus

Second, it requires some choice for the variable 'z'; we can select this by appealing to the finite inexhaustibility of $\mathbb{V}$. The choice of 'z' is irrelevant as two different choices will result in α-equivalent results.

Furthermore, we also need to know how to substitute terms for (one of) the free variables of a given term. This substitution procedure needs to be careful that it does not modify the binding graph of the terms; this is exemplified in the definition of substitution into an explicit λ-function. The definition of substitution is given in figure 1.3. Substitution satisfies the important substitution lemma for the λ-calculus.

$$
x \neq y \wedge x \notin \mathrm{fv}(s_2) \Rightarrow t[x \mapsto s_1][y \mapsto s_2] = t[y \mapsto s_2][x \mapsto s_1[y \mapsto s_2]]
$$

We can now give a formal presentation of α-equivalence of terms in figure 1.4.

In order to endow our currently static terms (we do not consider α-equivalence to be a dynamic process) with a dynamic computational structure we need to specify how these terms may be simplified or reduced. The first (reduction) rule is β-reduction, and encodes how λ-functions behave as functions. The second (expansion) rule is η-expansion, and encodes how λ-functions are the only terms of the λ-calculus. These rules, with their congruential closure, are given in figure 1.5.

We extend the directed rules of reduction in figure 1.5 to an equivalence relation of βη-conversion, by taking the reflexive, symmetric, transitive, congruential closure of the generating rules of β-reduction and η-expansion. There are a number of methods to define this equivalence relation, each having its own advantages and disadvantages; however, we define it by the rules in figure 1.6.

In the prequel we have considered the terms of the λ-calculus using a named approach to variables. This necessitated considering α-equivalence, and taking some care with some definitions. In the definition of simple type theory in the body of the thesis we use de Bruijn indices which obviates all the pitfalls of named variables arguably at the expense of reading and understanding terms.

$$
\frac{t_1 \rightsquigarrow_\beta t_1'}{t_1\, t_2 \rightsquigarrow_\beta t_1'\, t_2} \qquad \frac{t_2 \rightsquigarrow_\beta t_2'}{t_1\, t_2 \rightsquigarrow_\beta t_1\, t_2'} \qquad \frac{t \rightsquigarrow_\beta t'}{\lambda x.t \rightsquigarrow_\beta \lambda x.t'} \qquad \frac{}{(\lambda x.t_1)\, t_2 \rightsquigarrow_\beta t_1[x \mapsto t_2]}
$$

$$
\frac{t_1 \rightsquigarrow_\eta t_1'}{t_1\, t_2 \rightsquigarrow_\eta t_1'\, t_2} \qquad \frac{t_2 \rightsquigarrow_\eta t_2'}{t_1\, t_2 \rightsquigarrow_\eta t_1\, t_2'} \qquad \frac{t \rightsquigarrow_\eta t'}{\lambda x.t \rightsquigarrow_\eta \lambda x.t'} \qquad \frac{x \notin \mathrm{fv}(t)}{t \rightsquigarrow_\eta \lambda x.t\, x}
$$

Figure 1.5: β-Reduction and η-Expansion Rules for the λ-Calculus

$$\frac{}{x \equiv_{\beta\eta} x} \qquad \frac{t_1 \equiv_{\beta\eta} t_1' \qquad t_2 \equiv_{\beta\eta} t_2'}{t_1\, t_2 \equiv_{\beta\eta} t_1'\, t_2'}$$

$$\frac{t[x \mapsto z] \equiv_{\beta\eta} t'[y \mapsto z]}{\lambda x.t \equiv_{\beta\eta} \lambda y.t'} \text{ for some } z \notin \mathrm{fv}(t) \cup \mathrm{fv}(t')$$

$$\frac{t \rightsquigarrow_\beta t'}{t \equiv_{\beta\eta} t'} \qquad \frac{t \rightsquigarrow_\eta t'}{t \equiv_{\beta\eta} t'} \qquad \frac{t \rightsquigarrow_\beta t'}{t' \equiv_{\beta\eta} t} \qquad \frac{t \rightsquigarrow_\eta t'}{t' \equiv_{\beta\eta} t}$$

$$\frac{t \equiv_{\beta\eta} t' \qquad t' \equiv_{\beta\eta} t''}{t \equiv_{\beta\eta} t''}$$

Figure 1.6: $\beta\eta$-Convertibility Rules for the λ-Calculus

$$\frac{}{\iota \in \mathbb{T}} \qquad \frac{T_1 \in \mathbb{T} \qquad T_2 \in \mathbb{T}}{T_1 \to T_2 \in \mathbb{T}}$$

Figure 1.7: Type Structure of the λ-Calculus

Having laid out the basics of the λ-calculus we proceed to giving a type structure thereto. The type structure is a method of classifying terms based on how they can be expected to behave. STT is, arguably, the simplest non-trivial system of applying types to the terms of the λ-calculus. The typing structure predicates terms on whether they have a given type or not. Henceforth, we consider only terms that have an associated type to be legitimate terms of STT.

There are two types: a nominal base type, and an arrow type for functions, as given in figure 1.7. The type structure is *simple* as the types' well-formation does not have any hypotheses; so long as a type is formed from the structural rules then it is a valid type. By introducing a nominal base type we introduce the possibility that there are terms that are not expected to behave as λ-functions. Therefore some of the β-contraction, η-expansion, and $\beta\eta$-conversion may not always make sense based on the types of the terms involved. Such rules can have their terms be typed such that the rules are type-preserving.

The binding structure in the λ-calculus can be handled by introducing contexts to record the hypotheses of the types of bound variables when typing the body of a λ-function. Contexts are either empty, in which case the term being typed has no free variables, or they are formed by extending a context with a typing judgment. We denote these two contexts thus:

$\bullet$	for the empty context;
$\Gamma, x : T$	for the context Γ extended by the hypothesis that variable x has type T.

We can now state the typing rules of the λ-calculus. Although presented thus, we do not have in mind a two-stage formational process whereby the raw terms are considered first, and the well-typed terms thereafter. We reify the single formational process of only accepting well-typed terms as legitimate terms by considering typing judgment derivations to be the true objects of consideration following the work of Benton et al. (2012). This does change their intrinsic and essential nature from being a predicate over the structure of terms to being a structure in their own right. We do not consider this change to be wanting of too much thought in the case of simply typed terms[1]. We briefly remark that the previous definitions of free-variables, substitution, β-reduction, and η-expansion can all be reformulated to operate with well-typed terms such that they preserve well-typing. Moreover, the free variables of a well-typed term are simply the variables present in the context: in this way well-typing also enforces well-scoping on the terms. This change suggests that the typing rules should be largely similar in shape to the rules of the raw terms, which they are as can be seen in figure 1.8. The relation $(x : T) \in \Gamma$ implements a right-to-left look-up of a variable's type. A variable may occur multiple times in the context in

[1]The switch to intrinsically well-typed syntax from extrinsically well-typed syntax is not as straightforward for dependently typed terms; although, intrinsic well-scoping can still be accommodated in the dependently typed setting by using natural numbers as proto-contexts.

$$\frac{}{(x\,:\,T)\in\Gamma, x\,:\,T} \qquad \frac{(x\,:\,T)\in\Gamma}{(x\,:\,T)\in\Gamma, y\,:\,T'}\ \text{for } x\neq y$$

$$\frac{(x\,:\,T)\in\Gamma}{\Gamma\vdash x\,:\,T} \qquad \frac{\Gamma\vdash t_1\,:\,T\to T' \qquad \Gamma\vdash t_2\,:\,T}{\Gamma\vdash t_1\,t_2\,:\,T'} \qquad \frac{\Gamma, x\,:\,T\vdash t\,:\,T'}{\Gamma\vdash \lambda x\,:\,T.t\,:\,T\to T'}$$

Figure 1.8: Typing Rules for the λ-Calculus

$$\frac{}{x\in\mathcal{U}} \qquad \frac{t_1\in\mathcal{U} \qquad t_2\in\mathcal{N}}{t_1\,t_2\in\mathcal{U}}$$

$$\frac{t\in\mathcal{U}}{t\in\mathcal{N}} \qquad \frac{t\in\mathcal{N}}{\lambda x.t\in\mathcal{N}}$$

Figure 1.9: β-Normal Forms for the λ-Calculus

which case the relation selects the rightmost occurrence and thereby implements variable shadowing.

Whenever there exists a notion of reduction for a computational system, it is necessary to describe the shape of terms that cannot be further reduced. This is seen in a structure as simple as fractions which are often described as being in simplest form when the numerator and denominator are coprime, *i.e.*, the only divisors they share are units.

When it comes to the λ-calculus there are two notions of normal form, dependent on whether one includes η-expansion in one's computational rules. η-Expansion requires some care, as η-expanding a term may introduce the possibility of performing a β-reduction resulting in the same term with which we began. We wish to say that a term is maximally η-expanded precisely when performing any η-expansion would introduce such a possibility of β-reduction; such terms are in η-long normal form. We first describe the β-normal forms in figure 1.9, and thereafter the β-normal–η-long forms in figure 1.10.

The definition of β-normal forms requires a mutually inductive definition of two classes of terms: the neutral, denoted by '$\mathcal{U}$'; and the normal, denoted by '$\mathcal{N}$'. The neutral terms are those terms that when they are used in the function position of an application do not introduce the possibility of a β-reduction, and also any subterms not in head position are in β-normal form. Note that the definition of β-normal forms makes no reference to any types, and so serves equally well to describe normal forms for raw untyped (and even untypable) terms. This is not the case for η-long normal forms, as we require the typing information to describe precisely which terms are maximally η-expanded.

In the definition of β-normal–η-long forms, we index our two mutually-inductively defined classes by the types of terms that they contain: we denote the class of β-normal–η-long forms of type T by '$\mathcal{L}_T$', and the class of atomic terms of type T by '$\mathcal{A}_T$'. These atomic terms play a similar rôle to the neutral terms of β-normal forms. However, the type-indexing allows us to restrict when they embed into the β-normal–η-long forms: *viz.*, only when the term is typed at the base type.

$$\frac{}{x\in\mathcal{A}_T} \qquad \frac{t_1\in\mathcal{A}_{T\to T'} \qquad t_2\in\mathcal{L}_T}{t_1\,t_2\in\mathcal{A}_{T'}}$$

$$\frac{t\in\mathcal{A}_\iota}{t\in\mathcal{L}_\iota} \qquad \frac{t\in\mathcal{L}_{T'}}{\lambda x:T.t\in\mathcal{L}_{T\to T'}}$$

Figure 1.10: β-Normal–η-Long Forms for the Typed λ-Calculus

1.3.2 *Category Theory*

In this subsection we give a brief overview of category theory from a more standard perspective than that which we present in the body of the thesis. We are not overly concerned with giving many examples, as any required examples are covered in the main body of the thesis. Nonetheless, the examples that we present here serve as a basis with which to compare the non-standard presentation of category theory in the main body of the thesis. Category theory was originally invented as a language to describe phenomena in algebraic topology. It has since found uses in many seemingly disparate areas of Mathematics due to its ability to describe structural relations between various Mathematical objects.

Central to much discussion in category theory are foundational issues, and particularly foundational issues that are about size. In this preliminary introduction we are neither too careful nor precise about such issues saving such discussion to later in the thesis, where we are concrete with our presentation and definitions.

Definition 1.3.1 (Category). The central notion in category theory is the *category* – a generic example of which we denote in this introduction by $\mathcal{C}$, $\mathcal{D}$, *&c.* – and is given by the following data:

- a collection of *objects*, which we denote in this introduction by $\mathrm{Obj}_{\mathcal{C}}$ or (by abuse of notation) simply by $\mathcal{C}$;
- for any pair of objects, x and y, a collection of *morphisms* or *arrows*, which we denote in this introduction by $\mathrm{Hom}_{\mathcal{C}}(x, y)$ or $\mathcal{C}(x, y)$, examples of which we may denote by $f : x \to y$;
- for any triple of objects, $x, y,$ and z, and morphisms, $f : x \to y$ and $g : y \to z$, a composition of these two morphisms, $g \circ f : x \to z$; and
- for any object an identity morphism, $\mathrm{id}_x : x \to x$,

such that

- composition of morphisms is associative, *i.e.*, for any triple of composable morphisms, h, g, and f, $(h \circ g) \circ f = h \circ (g \circ f)$; and
- identity morphisms are identities for composition, *i.e.*, $f \circ \mathrm{id} = f = \mathrm{id} \circ f$.

In Mathematics, having defined a structure, an important next step is to define an appropriate notion of structure-preserving mapping between instances of the structure. This is an essential theme of category theory as much of the structure of a category is derived from the morphisms rather than the objects it contains. In the body of this thesis we consider and analyse multiple categories with the same objects, but with different morphism structures.

Definition 1.3.2 (Functor). The appropriate definition of structure-preserving mapping between categories is a *functor*. A *functor*, F, from the category $\mathcal{C}$ to the category $\mathcal{D}$, which we denote by $F : \mathcal{C} \longrightarrow \mathcal{D}$, is given by the following data:

- a mapping of the objects of $\mathcal{C}$ to the objects of $\mathcal{D}$, which we sometimes may denote by F_0 or (by abuse of notation) simply by F; and
- for any two objects of $\mathcal{C}$, x and y, a mapping of the hom $\mathcal{C}(x, y)$ to the hom $\mathcal{D}(F_0 x, F_0 y)$, which we sometimes may denote by F_1 or (by abuse of notation) simply by F,

such that

- it respects composition of morphisms, *i.e.*, $F_1(g \circ f) = F_1(g) \circ F_1(f)$; and
- it respects identities, *i.e.*, $F_1(\mathrm{id}_x) = \mathrm{id}_{F_0(x)}$.

Definition 1.3.3 (Natural Transformation). The process of defining a structure-preserving mapping repeats for functors. The possibility of such mappings existing is given by the presence of both objects and arrows in categories. A *natural transformation*, α, from the functor F to the functor G (both from category $\mathcal{C}$ to category $\mathcal{D}$), which we denote by $\alpha : F \Rightarrow G$, is given by the following data:

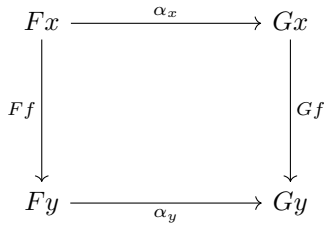


Figure 1.11: Naturality Square

- for each object, x, in $\mathcal{C}$ a morphism, $\alpha_x : F\,x \to G\,x$, which we call *the component of* α *at* x,

such that

- for any morphism, $f : x \to y$, in $\mathcal{C}$ we have $G\,f \circ \alpha_x = \alpha_y \circ F\,f$.

We refer to the restriction on the components as the *naturality condition*, or *naturality square* as it can be rendered in diagrammatical form as seen in figure 1.11.

We conclude this brief overview of category theory by giving two constructions of examples of categories. A category is, put simply, a structured collection of objects with mappings therebetwixt; such a structure exists for functors and natural transformations, which form a category.

Example 1.3.4 (Functor Category). Given any two categories, $\mathcal{C}$ and $\mathcal{D}$, we may form their *functor category*, which we denote by $[\mathcal{C}, \mathcal{D}]$ or $\mathcal{D}^{\mathcal{C}}$. It is given by the following data:

- functors, $\mathcal{C} \longrightarrow \mathcal{D}$, as objects; and
- natural transformations, $F \Rightarrow G$, as arrows.

It is readily verified that identity morphisms induce identity natural transformations, and that morphism composition induces (vertical) composition of natural transformations that is unital and associative. For both, the naturality condition holds.

Example 1.3.5 (Arrow Category). Given a category, $\mathcal{C}$, we may form its *arrow category*. It is given by the following data:

- pairs of objects, and an arrow therebetwixt, $f : x \to y$, as objects; and
- commuting squares as arrows.

Remark 1.3.6. Arrow categories can be expressed as an instance of functor categories.

1.3.3 *Dependent Type Theory and Rocq*

A major component of the research undertaken has been to formalise the Mathematics in the Rocq proof assistant.[2] Rocq is an implementation of a dependent type theory based off the calculus of constructions, albeit with many extensions to this base type theory. There is one aspect of type theory which is often extended: universes. For the sake of expressivity (and consistency), dependent type theories are often extended with an hierarchy of universes; and, for the sake of being closer to standard Mathematics, dependent type theories are often extended with a universe of propositions (although

[2] Rocq used to be known as 'Coq' as a pun on both the Calculus of Constructions (the formalism on which it is built) and the name of the founder Thierry Coquand, and in line with the French tradition of naming software projects after animals.However, the community, finding the name unfortunate in English, decided to rename it to 'Rocq'.Incidentally, this was the author's preferred option for a rename.In this thesis we refer to the proof assistant as 'Rocq', despite (at the time of writing) the renaming not yet being complete, in anticipation of it being complete before the thesis is approved.

recent type theories oriented towards higher Mathematics take a contrary position on this). Furthermore, there is one key difference between the type theory of Rocq, and Mathematics: function extensionality.

The type theory of Rocq (or, more precisely, the subset of Rocq's type theory which we use in this thesis) is an extension of Martin-Löf Type Theory (MLTT) with the extensions of an unbounded hierarchy of universes, an impredicative proof-irrelevant universe of strict propositions, inductively defined types/propositions, record types, and a few surface level presentational extensions of implicit arguments, coercions, and user-defined notations.

Universe Polymorphism

When a definition is made in a type theory that has a hierarchy of universes, this definition has to be anchored at some level in this hierarchy, despite the fact that the definition may make sense at any level in the hierarchy. Now, when this hierarchy is potentially infinite the definition cannot be reproduced at every level, as this would introduce an infinite number of terms into our type-theoretic definitions. However, the number of usages of the definition is finite within any (finite) body of type-theoretic definitions, and therefore the definition need only be reproduced a finite number of times for each universe level instance. However, the combinatorial explosion of this would be cumbersome and result in a large number of essentially repeated definitions.

This problem is reminiscent of the problem of type-generic programming, where definitions would have to be duplicated for each intended type usage. It was for this reason that generic programming was introduced as a way of allowing definitions to be agnostic over the types being used. Such genericity was introduced through prenex polymorphism: allowing quantification of type variables for the scope of a whole definition, but no finer. Such quantifications often has associated constraints when the type system has a notion of subtype.

Rocq has such a feature for universe levels: it offers the ability to quantify universe level variables at the scope of a definition but no finer. Since universe levels produce a partial order, it is necessary to associate to such definitions constraints using this partial order.

Propositions

After the work of Gilbert et al. (2019), Rocq's type theory is extended with a universe of strict propositions, SProp, that are intended to model how propositions behave in standard Mathematics. Proofs of propositions record that some fact is true, but not how it is true: the proof is irrelevant to the knowledge of the truth of the proposition. Since propositions are represented by types (in the universe of propositions), proofs are terms of the type representing the propositions and thus it is necessary to know when two terms are judgmentally equivalent. For propositions it is simple: always. Therefore, since any proof is convertible with any other proof it should not be possible to extract any information out thereof save the truth of the proposition represented. Thus when propositions are defined inductively, they cannot then be eliminated except to produce the proof of another proposition (unless the proposition being eliminated is definitionally empty).

This safeguarding of the strict propositions from the terms of relevant types interacts well with our non-standard developments within this thesis.

Function Extensionality and Canonicity

One of the important aspects of type theories that are intended to serve as a formal system for describing effective construction, is that closed terms are of a canonical form: *e.g.*, booleans in the empty context are judgmentally equivalent either to true or to false. The further property of normalisation requires that any term in any context is judgmentally equivalent with some normal form. Having these properties precludes the use of any axioms[3] within a type-theoretic development.

In MLTT, functions are not necessarily equivalent when they are pointwise equivalent; *i.e.*, function extensionality is independent of the rules of MLTT and thus not provable internally. A common solution is to assert function extensionality as an axiom, but this sacrifices the canonicity property desired for computation.

[3] There are some instances where axioms do not affect the property of canonicity; these are not relevant for this thesis.

CHAPTER 2

Background

In this chapter, we present background material for the body of the thesis. This background material is split into three parts: setoids, type-theoretic formalisms of category theory, and normal-form algorithms for simple type theory. Additionally, we discuss prior work in the areas related to the research presented.

2.1 Setoids

In this section we introduce the concept of a *setoid*, due to Hofmann et al. (1995), as used in type-theoretic formalisms, and the motivation for using them. One of the significant disadvantages of the uniform definition of the identity type in MLTT is that quotients are generally not possible. Moreover, the identity type does not satisfy function extensionality. A workaround to these problems is to use setoids. Using setoids is very much a workaround rather than a solution, as they do not change the behaviour of identity type. Instead, setoids introduce a discipline of using explicit equivalence relations rather than uniformly using the identity type. This particular downside manifests itself in a higher level of effort in producing type-theoretic formalisms compared with using the standard identity type.

Definition 2.1.1 (Setoid). A *setoid* in type theory is a carrier type and an equivalence relation (ER) over that type, where 'equivalence relation' means a family of propositions indexed over pairs of elements of the given type. We denote an arbitrary setoid by $(A; \sim_A)$.[1]

Setoids provide a type-theoretic analogue to "*Bishop sets*" from the constructive foundations of Bishop (1967): to define a structure is to define both the structure of the elements thereof, and also when two elements thereof are to be considered equivalent. This explicit carrying around of the equivalence relation allows the simulation of quotients.

Whenever a function is defined to and from a setoid, it is necessary also to prove that the function respects the equivalence relation of the source setoid. This motivates the definition of the function setoid.

Construction 2.1.2 (Function Setoid). The *function setoid* from $(A;\ \sim_A)$ to $(B;\ \sim_B)$ is given by

$$\left(\sum_{f : A \to B} \forall_{x,y\,:\,A}\ x \sim_A y \Rightarrow f(x) \sim_B f(y) \quad ; \quad (f;\_)\,(g;\_) \mapsto \forall_{a\,:\,A}\ f(a) \sim_B g(a) \right)$$

Thus, the full definition of a function requires simply to exhibit it as an element of the appropriate function setoid.

Every type induces a (discrete) setoid by using the identity type as the equivalence relation; this is often the correct mapping from types into the universe of setoids.

Therefore, if we use setoids instead of raw types in the development of a type-theoretic formalism, we are able to overcome the issues faced by the lack of function extensionality in standard MLTT. Unfortunately, using setoids induces a

[1] Here and throughout the thesis we use semi-cola for the tupling of dependent pairs, and commata for the tupling of non-dependent pairs.

problem known as *setoid hell* where all the extra proofs of preservation of the equivalence relation are required even though they may be trivial, either by construction or by triviality of the equivalence relations. For instance, all term constructions in MLTT respect function extensionality (the only way to "use" a function term is to apply it to some argument term, which evidently respects function extensionality); therefore, the proofs of respecting the setoid relations are often quite mechanical, following closely the structure of the function defined.

The discipline of setoids resulting in only considering functions up to extensionality can be seen as a form of quotienting; we are forcing all functions which produce equivalent outputs for equivalent inputs to be considered equivalent. Quotients, essentially, are the increase in expressive power offered by using setoids.

Although using setoids provides a resolution to the lack of function extensionality in standard MLTT, it does create new problems. It is not clear how to model universes into the setoid approach: what should the equivalence relation of a universal setoid be? Additionally the answer to this problem is likely to depend on the ambient type theory and how setoids are defined. If the ambient type theory affirms the uniqueness of identity proofs then the identity type of the ambient type theory may be sufficient for the choice of equivalence relation. Since equivalences within a universal setoid would have to induce coercions, the equivalence relations may have to be valued in proof-relevant universes of the ambient type theory. In this thesis we do not need a notion of universal setoid so we leave these questions to other work.

2.1.1 *Relevance*

The precise meaning of "equivalence relation" can vary based on the kind of type-theoretic universe in which it is valued. This variation results in a number of different flavours of setoids each with their own characteristic advantages and disadvantage. In definition 2.1.1 we alluded to their valuation in a universe of propositions; such setoids are known as proof-irrelevant setoids. The equivalence relations may also be valued in a universe of types; such setoids are known as proof-relevant setoids.[2] There are advantages and disadvantages to both of these approaches. Proof-relevance entails a predicative quantification, which requires careful analysis of universe levels.[3] Proof-irrelevance allows an impredicative quantification, which obviates the need for careful analysis of universe levels. It also has the advantage that it helps separate the universes used for elements and the universe used for proofs of relation.

2.2 Type-Theoretic Formalism of Category Theory

In this section we give a brief overview of prior work in formalising category theory in various proof assistants. We discuss the various design decisions made in the formalisations as a precursor to discussion on decisions made for our own formalisation. Category theory has seen many formalisms within a variety of proof assistants. We restrict our focus here to those formalisms performed in some variant of dependent type theory.

An important design decision to be made for type-theoretic formalisms of category theory is what kind of structure within the type theory is to be used for the homs. If the homs are raw types then the formalism is rather limited, unless function extensionality is available or asserted as an axiom. If one is only interested in establishing Mathematical truths, rather than performing constructions, then this may be an appropriate setting. If, on the other hand, the homs are setoids, then this implements 1-category theory. This offers a more suitable setting for performing constructions, as no axiomata have to be asserted, so canonicity and normalisation are preserved. Again, there is a choice of whether the setoids are proof-relevant or not. If they are proof relevant then it is not clear whether the notion formalised is a true category, or is a *wild category* where there is deeper/higher structure beyond the morphisms. A further disadvantage of using proof-relevant setoids is that fewer judgmental equations hold of the structures proved. For example, in a proof-relevant setting a category may not be judgmentally equivalent with its double opposite. Additionally, two categories with judgmentally the same categorical structure (*viz.*, objects, homs, compositions, and identities) but different proofs of the categorical laws are

[2] Within HoTT there is a third possibility where the relations are valued in types, but whose elements are provably identical internally.
[3] Within HoTT propositional resizing axioms have been considered that would reduce the impact of proof relevance on universe levels.

not judgmentally equivalent. There is, nonetheless, a categorical equivalence between these judgmentally-inequivalent categories.

The formalism of category theory in Rocq (then called Coq) has its origins in at least the work of Huet et al. (2000). Their formalism of category theory used setoids for the homs, thus implementing E-category theory. They remark that they could have used 'partial setoids' for their formalism, which would have resulted in them formalising the flavour of category theory presented in this thesis. They make no further remark as to why they did not pursue this direction.

Gross et al. (2014) report on their experience formalising category theory in Rocq (again, then called Coq), advising on three design decisions: dependently-typed indexed morphism types, bundled records, and duality-centric design.

More recently, Hu et al. (2021) have developed a formalism of category theory for the Agda proof assistant. They stress three design decisions: universe polymorphism, proof-relevant setoids, and duality-centric design. Their using proof-relevant setoids complicates the exact structures that they have formalised. For instance, they construct a category of categories by allowing the equivalence relation on the setoids to be natural isomorphism, thereby obfuscating the two-dimensional structure of the bicategorical study of categories.

The 1Lab project (The 1Lab Development Team (2024)) break from the prevalence of using bundled records by partially unbundling following a trichotomic separation of the components of the records as *stuff*, *structure*, and *property*. This separation is a refinement of the more common understanding of 'structure' *vs* 'property'.

Recent work in Homotopy Type Theory and Univalent Foundations has opened up the scope of more flavours of category theory formalisms in proof assistants. In Univalent Foundations, categories come in three varieties: strict categories, precategories, and (univalent) categories. Precategories are the essential form of category-like structures in univalent foundations. The hom types of precategories are required to have uniqueness of identity proofs. Strict categories are those precategories whose objects have uniqueness of identity proofs. Univalent categories are those precategories whose equality type for objects is equivalent with the type of isomorphisms between objects. This latter form of category translates the non-categorical concept of working up to equality into the categorical concept of working up to isomorphism. These type-theoretic presentations are often in opposition to the setoid approach for formalising category theory as the problems that setoids are used to overcome (*e.g.*, function extensionality and quotients) are already overcome by univalent foundations.

2.3 Normalisation of Simple Type Theory

In this section, we give an overview of normalisation of simple type theory from various perspectives. First we define notions of normalisation, and thereafter discuss how to implement them algorithmically.

Strictly speaking there are two different properties often referred to by 'normalisation': *strong* and *weak*.

Definition 2.3.1 (Strong Normalisation). *Strong normalisation* is the property that for all terms all reduction sequences terminate.

Therefore, to prove strong normalisation, one has to consider all possible ways in which a term may reduce. However, strong normalisation does not establish that there is some algorithm which derives for each term a normal form.[4]

Definition 2.3.2 (Weak Normalisation). *Weak normalisation* is the property that for all terms there is some reduction sequence that terminates.

Therefore, to prove weak normalisation, one only has to establish for each term a single reduction sequence to a normal form. Furthermore, normalisation can either be reductional or reduction-free. We discuss each of these approaches in the following subsections.

[4] If reduction steps are enumerable — and, therefore, normality is decidable — then a brute-force algorithm can be employed, but this is not generally the case for rewrite systems.

2.3.1 *Reductional Normalisation*

Reductional normalisation arises when the sequence of reduction steps is tracked and constructed in the process of normalising an input term into an output normal term. There are a number of advantages and disadvantages associated to this approach. An obvious advantage is that one of the correctness properties is immediate: the output term is equivalent under the rewriting system to the input. However, the building up of this reduction sequence is often quite inefficient. Reductional normalisation algorithms usually operate by one reduction step at a time, which results in a lot of deep recursion into a term to find and subsequently to make only a single reduction step. This is repeated until no more reduction steps can be taken, *i.e.*, the term is in normal form. Moreover, this procedure is not well-justified until some strong normalisation result has been obtained, as otherwise it is not guaranteed to terminate. Strong normalisation is often quite a strong property to establish for a rewriting system, particularly one intended to serve as a model of computation. Subsequently, this approach is undesirable.

Reductional normalisation performs relatively well for some reduction systems, but not for others. For example, reductional normalisation works relatively well for β-reduction of the λ-calculus as this reduction system has a clear notion of direction: should a rewriting rule be a reduction rule where simplification occurs, or should a rewriting rule be an expansion rule where complexification occurs? With β-reduction, it is quite clear that the correct direction is reducing a β-redex. However, η-expansion does not work particularly well with reductional normalisation. Although it may be clear that η-expansion should be expansion (as argued in, *e.g.*, Jay et al. (1995)), this direction has non-trivial interaction with β-reduction, thus complicating reductional normalisation. More abstractly, reductional normalisation does not work well with rewriting systems that incorporate extensionality or unicity rules, such as η-expansion.

2.3.2 *Reduction-Free Normalisation*

Reduction-free normalisation is a family of normalisation algorithms that make no reference to reduction sequences. Because it does not use reduction sequences, reduction-free normalisation is much more well-suited than reductional normalisation to systems with extensionality or unicity rules. Reduction-free normalisation has been successfully employed by a technique known as *normalisation by evaluation*. Normalisation by evaluation, loosely, is a strategy for normalisation where terms are evaluated within some kind of computational model in such a manner that terms can be extracted out of elements of the model. The computational effectivity of the model is, in a sense, what performs the normalisation. Normalisation by evaluation has its origins in the work of Berger et al. (1991). Their work was partially adapted to a categorical setting by Altenkirch et al. (1995), albeit incompletely so, as they still used *ad-hoc* non-categorical techniques. Fiore (2002, 2022) further categorified normalisation by evaluation by inserting the syntactic notions of neutral forms and normal forms into the categorical setting by fully using Artin-Wraith gluing. Čubrić et al. (1998) give a categorical presentation of normalisation by evaluation without using Artin-Wraith gluing and without using the syntactic notions of neutral forms and normal forms. However, they only prove categorically some of the properties given by Fiore, reverting to non-categorical techniques for other proofs.

2.3.3 *Correctness Properties*

Normalisation algorithms are required to satisfy three correctness properties: soundness, completeness, and stability. Each of these properties serve to ensure that the normalisation algorithm can be used to find unique normal forms for the input term. Authors have disagreed on which of soundness and completeness is which; we name them as we do due to our viewing equality of normal forms as sound and complete for $\beta\eta$-conversion of λ-calculus terms. In the sequel, we use 'Λ' to represent the set of well-typed terms of simple type theory in arbitrary context, and of arbitrary type; although, all normal-form algorithms are required to preserve the context[5] and type of a term. We also use '$\mathcal{L}$' to represent the set of β-normal–η-long terms.

[5] A normal form for some given term may not use some variables present in the original term, nonetheless it is still well-typed in the same context as the input.

Definition 2.3.3 (Soundness). A normal-form algorithm for simple type theory, $\mathrm{nf} : \Lambda \to \Lambda$, is *sound* whenever outputs are $\beta\eta$-convertible with their respective inputs; *i.e.*, the following property holds:

$$\mathrm{nf}(t) \equiv_{\beta\eta} t$$

Remark 2.3.4. This definition of soundness entails the more usual phrasing of soundness for normal-form algorithms for simple type theory, $\mathrm{nf} : \Lambda \to \Lambda$:

$$\mathrm{nf}(t) = \mathrm{nf}(t') \Rightarrow t \equiv_{\beta\eta} t'$$

Definition 2.3.5 (Weak Completeness). A normal-form algorithm for simple type theory, $\mathrm{nf} : \Lambda \to \Lambda$, is *weakly complete* whenever outputs of $\beta\eta$-convertible inputs are $\beta\eta$-convertible; *i.e.*, the following property holds:

$$t \equiv_{\beta\eta} t' \Rightarrow \mathrm{nf}(t) \equiv_{\beta\eta} \mathrm{nf}(t')$$

Remark 2.3.6. Weak completeness follows from soundness, and thus is not a very useful property to consider in the presence of soundness. However, in the body of this thesis we construct normal-form algorithms where it is not easily provable that they are sound, but where it is easily provable that they are weakly complete. Arguably, weak completeness is not a correctness property of a normal-form algorithm but a well-definition property.

Definition 2.3.7 (Strong Completeness). A normal-form algorithm for simple type theory, $\mathrm{nf} : \Lambda \to \Lambda$, is *strongly complete* whenever outputs of related inputs are α-equivalent/equal; *i.e.*, the following property holds:

$$t \equiv_{\beta\eta} t' \Rightarrow \mathrm{nf}(t) = \mathrm{nf}(t')$$

Definition 2.3.8 (Stability). A normal-form algorithm for simple type theory, $\mathrm{nf} : \Lambda \to \Lambda$, is *stable* whenever the outputs of normal inputs are α-equivalent/equal with the original normal input; *i.e.*, the following property holds:[6]

$$t \in \mathcal{L}_T \Rightarrow \mathrm{nf}(t) = t.$$

Remark 2.3.9. Strong completeness and soundness together already imply a weaker form of stability: the output of such a normal-form algorithm is stable, *i.e.*, doubly normalising a term is identical with only singly normalising the term.

Stability, as remarked by Fiore (2002, 2022), is an interesting property from the perspective of the reduction-free normal-form algorithms of normalisation by evaluation as it, with strong completeness, entails that $\beta\eta$-convertible normal forms are α-equivalent/equal. Such uniqueness of normal forms is often seen as a rewriting-theoretic result requiring application of a confluence result such as the Church-Rosser theorem, so the ability to prove it without using rewriting-theoretic machinery is not without interest.

[6] Here, $\mathcal{L}_T$ is used as defined in figure 1.10.

CHAPTER 3

Rocq Formalism

All the results in this thesis leading to a normal-form algorithm and the relevant correctness properties thereof have been formalised in the Rocq proof assistant[1]. Most of the results about formal p-category theory have also been formalised due their usage in normal-form algorithms and their relevant correctness properties. In this chapter we discuss the formalism in Rocq, giving attention to particular features of both the proof assistant and of the underlying type theory that we use heavily. Proof assistants, like Rocq, are not simply algorithmic artefacts that type-check their input; they often provide a multitude of further features to ease the user interface with the underlying type theory. For example, Rocq provides tactics for the construction of proof terms, rather than demanding that the user construct proof terms in complete detail in explicit term-level syntax. Indeed, these presentational features are an important advantage of using proof assistants over pen-and-paper Mathematics. Proof assistants allow the user to extract computational content from their type-theoretic constructions, automatically check the validity of type-theoretic constructions, and ease the burden of performing type-theoretic constructions.

3.1 Implicit Arguments

One of the main advantages of the Hindley-Milner type system is that type-level arguments are inferrable for type-correct code; however, more type-theoretically complicated type systems do not admit this property. Nonetheless, arguments to functions in dependent type theory can frequently be inferred by the types of later (and often computationally relevant) arguments. Although this inference cannot be complete, it is often powerful enough. These inferrable arguments can be omitted in Rocq terms by using an underscore: '_'. Although this reduces the workload for the user, it results in textually-long syntactic terms of the constructions internal to the type theory of the proof assistant. An alternative is to declare some arguments as implicit and therefore textually omitted in the raw syntax of the proof assistant. This creates a new problem in the presence of Currying and partial application: should the implicit arguments be maximally inferred and inserted, or should the implicit arguments be minimally inferred and inserted? We use both kinds of implicit arguments in the Rocq development.

The availability of implicit arguments is used to good effect with higher-dimensional categorical constructions. For example, a category has no dependencies, functors depend on the two end-point categories, and natural transformations depend on the two end-point functors. Therefore, the type of natural transformations would have four arguments. However, the two categories can be marked as implicit as they can often be inferred from the end-point functors of the natural transformations. Moreover, this level of detail of specification is more in line with how practitioners of category theory think: we say 'α is a natural transformation from F to G', stopping there, rarely proceeding to say 'which are functors from $\mathbb{C}$ to $\mathbb{D}$' unless we are being particularly explicit.

[1] The formalism may be found at Berry (2026)

Implicit arguments are also useful for parameterised constructions: *ee.g.*, parameterised inductive types and parameterised records. For example, natural transformations are records parameterised over the end-point categories and functors, and therefore the fields of the natural transformation record have these parameters as part of their arguments. Record fields, implicitly, inherit the implicitness of the parameters of the record. Therefore, accessing the naturality proof of a natural transformation would not require specifying the end-point categories; it would require specifying the end-point functors, even though having the natural transformation already provides these. It is therefore useful to set all the parameters of a record to be implicit for the fields of the record, so that only the record instance needs to be specified, instead of all the parameters.

Implicit arguments are often inferred poorly (*e.g.*, '`fst (a, b)`' rather than simply '`a`') or are uninferrable and so have to be manually specified. Where arguments are not inferrable (*e.g.*, during interactive proof construction with tactics) metavariables are introduced; however, these metavariables do not satisfy η-laws (*e.g.*, a metavariable of product type is not expanded to a pair of metavariables when unification so requires). This often results in having to specify explicitly the shape of implicit arguments of product type as '`(_, _)`' to force a pair of metavariables that can then be inferred.

Although the occasional manual specification of implicit arguments is inconvenient, on the whole, implicit arguments have made using the Rocq proof assistant much smoother.

3.2 Records

The basic type-formers of dependent type theory are the dependent-function type and the dependent-pair type. Although dependent functions iterate well through (dependent) Currying, dependent pairs do not iterate well, resulting in awkward and *ad-hoc* nesting. A resolution to this problem is record types, which generalise dependent pairs by allowing named multiple fields, with arbitrary (but, well-founded) dependencies. Furthermore, they allow full η-laws, which we enable through Rocq's `Primitive Projections` mechanism.

Records internalise the notion of the extension of a context by a telescope. This allows them to package together the components of Mathematical structures. But this creates a new problem: bundling. How much of a Mathematical structure should be part of the type of the record by way of parameters, and how much should be part of the data contained within a record by way of fields? For our purposes, we choose to bundle maximally, save for the boundary constraints of categorical structure. Explicitly, we therefore formalise what a category is, rather than what structure and properties some stuff must have and satisfy in order to present a category; we formalise what a functor between two given categories is, rather that what properties the functor maps must satisfy in order to present a functor.

3.2.1 *Coercions*

Frequently, a record is an extension of another record by containing an instance thereof as a field. For example, a pointed category could be modelled as a record containing both a category and an object of that category. However, this results in a pointed category not being a category. Therefore, whenever it needs to be used as a category the category field needs to be projected out from the pointed-category record instance. This inconvenience can be mitigated by declaring the projection of the category field as a coercion. This makes Rocq silently insert such a projection whenever a term of pointed-category record type is used whenever a term of category record type is expected. This facility aligns with standard Mathematical practice of, *ee.g.*, silently treating a group as a monoid, or silently treating a ring as its additive Abelian group.

Such coercions are highly prevalent throughout our Rocq formalism, allowing the silent coercion of, *ee.g.*, p-sets to their underlying carrier type; of p-categories to their type of objects; p-functors to their object mapping; and p-Cartesian categories to p-categories.

Occasionally, a record is an extension of another record in two orthogonal ways. For example, a p-lex-closed category is both a p-lex category and a p-Cartesian-closed category, both of which are p-Cartesian categories. For this we define p-lex-closed category as a record extending p-Cartesian categories with all of the structure of both a p-lex category (*viz.*, p-equalisers), and a p-Cartesian-closed category (*viz.*, p-Cartesian exponentials). We further define manually two functions

constructing both a P-lex category from a P-lex-closed category, and a P-Cartesian-closed category from a P-lex-closed category. We can declare such functions as coercions since they map record types to record types.

3.3 Universe Polymorphism

Rocq has a cumulative hierarchy of proof-relevant universes denoted by `Type@{i}`, with the lowest universe also being denoted by `Set`. Therefore if we are to have constructions apply at different, and arbitrary, levels of this hierarchy, some mechanism of universe polymorphism must be used. Universe polymorphism in Rocq is implemented as a form of prenex polymorphism at the level of definitions binding universe levels, and (in)equality constraints therebetwixt. This basic form of universe polymorphism does not permit algebraic description of universe levels: one cannot reference the next universe after an arbitrary universe but only an higher universe (which contains the intended next universe by cumulativity); one cannot reference the least upper bound of two (or more) universes but only an upper bound of the two universes (again, which contains the intended least upper bound by cumulativity). We use universe polymorphism throughout the Rocq formalism for all definitions.

3.3.1 *Cumulative Record Types*

Categories are, by nature, indexed in two universes: the universe in which the type of objects exists, and the universe in which the type of morphisms exists. Therefore, we index them in two universe levels without any (in)equality constraints therebetwixt. This allows us to use the same definition of category to describe both a large category of small sets as well as a very large category of large sets; or a locally-small category and the large category of presheaves thereover.

If a construction were expecting a large category but only a small category were provided this would be a type error: a small category is not a large category. However, a small category can be coerced trivially into a large category due to cumulativity for the types of objects and morphisms. Rocq permits the description of universe-polymorphic record types as cumulative with three forms of variance: irrelevant (denoted by a '`*`' before the universe), where the universe is only used in the type of the record; covariant (denoted by a '`+`' before the universe), where the universe is only used in strictly positive positions within the types of the fields of the record; and invariant (denoted by a '`=`' before the universe), where the universe is used either in a negative or a non-strictly positive position within the types of the fields of the record. The type of categories is covariant in the levels of its two universes, thereby allowing the type of small categories to be a subtype of the type of large categories.

3.4 Strict Propositions

Rocq's universe of strict propositions, due to the work of Gilbert et al. (2019), allows for a universe whose types are definitionally propositional. All terms of a given strict proposition are judgmentally convertible and are removed upon extraction. Moreover, inductively defined strict propositions have a restricted elimination principle: unless they are judgmentally empty (*i.e.*, they have no constructors) they may only be eliminated to produce terms of strict propositions. Additionally, they are also impredicative: strict propositions quantified over a large domain remain strict propositions. However, they may not be used for the accessibility predicate. The accessibility predicate is an inductive type encoding the ability to perform recursion along a well-founded relation. If it were to be defined in the sort of strict propositions it would result in any two proofs of an element being accessible being identified judgmentally thereby allowing infinite recursion. We do not use the accessibility predicate in our development so this is an irrelevant disadvantage. These properties of strict propositions provide a number of advantages within the Rocq formalism.

Judgmental convertibility of terms of strict propositions Type-checking is significantly sped up by using strict propositions, as two proofs of the same strict proposition do not have to be compared one with the other for structural convertibility; they are simply declared to be judgmentally convertible.

Proof irrelevance Their erasure upon extraction ensures that no computational aspect of the development relies upon a logical property that cannot be deduced simply from the computational content of the constructions. This acts as a secondary source of validation on the separation of computation from logic within P-category theory.

Impredicativity Strict propositions forming a separate impredicative universe means that when Σ-types (or records) are formed (*e.g.*, for P-functors) any logical properties specified by strict propositions play no part in the universe level algebra. This impredicativity eases the development and speeds up type-checking. If strict propositions instead formed a separate hierarchy of predicative universes each contained within a universe of ordinary types then they would partake of the universe level algebra. A fully-P-theoretic form of category theory where the objects have a notion of partiality/well-definition would allow for a complete separation of computation from logic which would ease the burden of a predicative hierarchy of strict propositions by having two separate systems of universe level algebra.

3.5 Differences from On-Paper Definitions

Type-theoretic formalism of Mathematical theories within dependent type theories frequently necessitates differences from the on-paper description of the Mathematical theory. Either this is because of a difference in foundations, or because of an increased level of rigour and precision.

Firstly, for example, dependent type theories, including Rocq, do not have full η-expansion for inductive types (even non-recursive ones) as type-checking in the presence of η-expansion for (recursive) inductive types is undecidable. This prevents desired judgmental conversions from holding where the analogous equality would hold definitionally in ordinary on-paper Mathematics. This can be overcome by using an equality type, although this is not always the best solution.

Secondly, on-paper category theory is frequently ambiguous about notions of size, allowing categories of any size (or, sufficiently large categories), whereas Rocq requires precision about the size of the universe for any given type.

Thirdly, on-paper Mathematics silently coerces between obviously equivalent structures even when such a coercion depends upon some logical property holding. In this thesis, P-natural transformations and self-related morphisms of P-functor categories are equivalent structures. The former bundles together the computational part with the logical parts (somewhat inimically to the P-setting), and so it is simple to forget the logical parts and regard the computational part as a morphism in a P-functor category, and to use the logical parts as a proof of its self-relation. The latter does not bundle the computational part with the logical parts, and so to consider the latter as the former would require a proof search that would often be elided in on-paper Mathematics.

3.6 Notations

Rocq allows for the definition of user-specified notations. Although we have used this somewhat minimally, we have used them for the following cases beyond that which is usual:

- '$\sim$' for the PER associated to a P-type;
- '`s&`' for conjunction of strict propositions;
- '`s|`' for (squashed) disjunction of strict propositions.

These latter two could be eliminated by using sort polymorphism, allowing a single definition of products to be polymorphic between being the product of ordinary types and being the conjunction of strict propositions.

Part II

p-Category and p-Bicategory Theory

CHAPTER 4

P-CATEGORY THEORY

In this chapter, we motivate and introduce P-category theory. Thereafter, we proceed to define many of the key concepts of P-category theory familiar from standard category theory: P-categories, P-functors, P-natural transformations, *&c*. These are, of course, largely similar to their respective definitions in standard category theory. Some intricacies of the P-categorical definitions are "inherited" from E-category; however, there are a few key details in which P-category theory differs therefrom, and requires further adaptation. Arguably, it is in these details that the strength of P-category theory is found. After providing the definitions of key concepts, we develop standard category-theoretic results in the P-setting, paving the way for analysing P-categorically the structure of simple type theory in chapter 7.

Remark 4.0.1. Within this chapter we gloss over issues arising from size, such as largeness of a category, or local-smallness of a category. Such issues are given due attention in the ROCQ formalism.

4.1 P-SETS

Primary motivation for P-category theory can be found in the motivation of P-sets as an appropriate structure in which to "enrich" standard category theory[1], and as an appropriate structure with which to perform constructive Mathematics. It is in this latter perspective of desiring a good setting for constructive Mathematics that P-sets can most well be motivated. E-sets/setoids are already relatively well-motivated for constructive Mathematics, due to their ability to provide a good simulation of quotients within both constructive Mathematics and type theory. This is achieved by combining together a type with a desired equivalence relation into a single Mathematical entity. Arguably, this is taking the Bishop's instructions, found in Bishop (1967), on how to construct a set seriously: *viz.*, firstly describe what the elements of the set are; and secondly describe when two elements should be considered equivalent. Not only is this solution simple, but it is also powerful as it is compatible with many set-theoretic operations. However, the construction of subsets for Bishop sets requires either a philosophical justification for restricting what the elements of a Bishop set are by a predicate, or a structural characterisation of the desired subset (which is not available in general). This philosophical quandary carries forward into the construction of subsets within the world of setoids: we are forced to modify the underlying carrier type by using a Σ-type in its computational part, instead of being able to achieve the desired construction by modifying its logical part. The logical part of setoids, the equivalence relation, was introduced to achieve more logical expressivity in allowing the simulation of quotients. The modification of the underlying carrier type results in the arguably undesirable mixing of computational data with logical property. It is clear that setoids do not provide enough logical power to simulate logical operations purely within their logical component. It is therefore desirable to find an alternative solution for simulating quotients that has better facilities for simulating subsets. P-sets provide such a solution by uniformly approaching the dual operations of quotients and subsets, as highlighted in Rothgang et al. (2025).

[1] Strictly speaking P-category theory is not a form of enrichment in the standard sense; nonetheless, it is similar enough that intuitions from enrichment can carry over.We consider these subtleties further in subsection 4.2.7.

E-sets/setoids and P-sets/subsetoids/partial setoids, and sundry variants thereof, are given a systematic study in Barthe et al. (2003). They argue for the correctness of the definition of P-sets and the appropriate mappings therebetwixt based on which definition results in an appropriately Cartesian-closed category.

Combining equivalence relations (which define quotients) with predicates (which define subsets) necessitates removing the global reflexivity from equivalence relations, since not necessarily all elements will satisfy the predicate. This motivates finding a "solution" to the following problem.[2]

$$\text{ER} - \text{Reflexivity} = ?$$

That is, what kind of Mathematical object arises when global reflexivity is removed from equivalence relations? Possibly the most obvious answer is "partial equivalence relations" which we now define.

Definition 4.1.1 (Partial Equivalence Relation). A *partial equivalence relation* (*PER*) is a binary relation that is both:

- symmetric, and
- transitive.

It is similar to an equivalence relation, but it need not be the case that elements are related to themselves. This definition is rendered in Rocq in listing 4.1.

```
Cumulative Record PER@{*i} (A : Type@{i}) := {
  PER_rel : A -> A -> SProp;
  PER_symm : forall {x y}, PER_rel x y -> PER_rel y x;
  PER_trans : forall {x y z}, PER_rel x y -> PER_rel y z -> PER_rel x z;
}.
```

Listing 4.1: Partial Equivalence Relation Rocq Definition

By combining types with PERs rather than ERs we arrive at the notion of subsetoids/P-sets.

Definition 4.1.2 (Subsetoid/P-Set). A *subsetoid* or *P-set* is a type together with a partial equivalence relation over the type. We denote P-sets as pairs of the underlying carrier type and the PER over the given type. By abuse of notation, occasionally we identify a P-set with its underlying carrier type; when we wish to emphasise that we are referring only to the underlying carrier type, we use vertical bars around the name of the P-set. Moreover, we will denote the PER of a P-set by $\sim$, often with a disambiguating subscript of the name of the P-set. Thus, a general P-set may be denoted by the following.

$$A \triangleq (|A|; \sim_A)$$

P-Sets are a strict generalisation of E-sets, as they encompass strictly more examples. Thus, one might hope that P-sets would provide a good and more general setting in which to perform constructive Mathematics than E-sets provide. In this thesis we aim to convince the reader of this fact. E-sets and the resultant E-category theory have already received some attention — *e.g.*, Lack et al. (1996) gives a formal account of what an E-category is within the setting of standard Mathematics — but P-sets and the resultant P-category theory have received little to no attention since their first introduction by Čubrić et al. (1998). Perhaps this is due to the inconvenience of not having global reflexivity.

[2] Perhaps this "equation" should be rephrased as "? + Reflexivity = ER"; *i.e.*, what class of relations, when restricted to those relations that are reflexive, becomes a subclass of equivalence relations? When rephrased in this way other, solutions are suggested: notably, quasi-partial-equivalence relations/zig-zag relations/difunctional relations.We believe that such relations may be a better solution to this problem.

4.1.1 *P-Set-Theoretic Definitions*

We now turn to providing further definitions and constructions relevant to the (non-categorical) theory of P-sets. Much of the theory of P-sets is similar to that of the theory of E-sets, although there are a few important differences that arise due to the lack of global reflexivity.

Definition 4.1.3 (Domain). The *domain* of a P-set are the elements of the underlying carrier type that are self-related. We may refer to such elements as *P-well-defined* or *P-proper*.

Remark 4.1.4. Understanding the self-related elements as the well-defined elements resembles the ideas of existence in the work of Scott (1979).

By restricting a PER to its diagonal a predicate is induced on elements of a P-set that, in general, captures two notions: hereditary P-well-definition (*i.e.*, (hereditary) P-well-definition of all the appropriate subcomponents of the element); and subset restriction inducing a top-level notion of P-well-definition.

Remark 4.1.5. Informally, we refer to non-self-related elements as *ghost* elements, or *improper* elements. These are precisely the ill-defined elements that fall outside the predicate induced by the diagonal.

Construction 4.1.6 (Domain E-Set). Every P-set induces an E-set by the following mapping.

$$(A;\sim_A) \mapsto \left(\sum_{a\,:\,A} a \sim_A a\,;\,(a;\,\_)\;(a';\,\_) \mapsto a \sim_A a'\right)$$

It can easily be verified that the resultant relation is an equivalence relation. Indeed, this mapping suggests the power of P-sets over E-sets by avoiding Σ-types.

Construction 4.1.7 (Discrete P-Set). Every type, A, has an associated *discrete* P-set, which we denote by $A_=$, given by:

- $|A_=| \triangleq A$; and
- $a \sim_{A_=} a' \triangleq a = a'$.

Construction 4.1.8 (Unit P-Set). The unit P-set has a single unconditionally self-related element.

Construction 4.1.9 (Product P-Set). The Cartesian product of P-sets, A and B, which we denote by $A \times B$, is given by:

- $|A \times B| \triangleq |A| \times |B|$; and
- $(a,b) \sim_{A\times B} (a',b') \triangleq a \sim_A a' \wedge b \sim_B b'$.

Construction 4.1.10 (Function P-Set). The P-set of P-functions between P-sets, A and B, which we denote by $A \to B$, is given by:

- $|A \to B| \triangleq |A| \to |B|$; and
- $f \sim_{A\to B} f' \triangleq \underset{a\,a'\,:\,A}{\forall} a \sim_A a' \Rightarrow f\,a \sim_B f'\,a'$.

Remark 4.1.11. The definition of the PER for function P-sets diverges from the respective definition for E-sets. The presence of partiality, *i.e.*, lack of global reflexivity, requires that we quantify over two elements in the domain P-set requiring them to be related by the domain PER. This logical definition for the PER for function P-sets refines the ER for function E-sets: it must map related inputs to related outputs. The non-logical definition is recovered just in case the domain PER is discrete.

It will be useful to be able to take the dependent product of a discretely-indexed family of P-sets.

Construction 4.1.12 (Dependent Product of P-Sets). Given a family of P-sets, $(B_a)_{a\,:\,A}$, discretely-indexed over a type, A, we can form its dependent product by the following construction.

$$\left(\prod_{a\,:\,A} B_a\,;\,f\,f' \mapsto \underset{a\,:\,A}{\forall} f\,a \sim_{B_a} f'\,a\right)$$

4.1.2 *Sub-P-Sets*

Given that the modelling of subsets was a principal source of motivation for using P-sets instead of E-sets, we now devote the following subsection to discussing the modelling of subsets within P-sets. The modelling of subsets is principally facilitated by the removal of global reflexivity from equivalence relations, and by the embracing of partiality. For the sake of discussion and example, suppose that we wished to model the subset of some P-set, $(A; \sim_A)$, by some predicate, P, on A. Disputably the most obvious solution would be to adopt the following P-set.

$$(A;\ a\, a' \mapsto P\, a \wedge P\, a' \wedge a \sim_A a')$$

Indeed, in general this is the only available solution. However, in certain cases there may be a *better* (for some aesthetical definition of better) construction available. In the case where the predicate, P, respects the pre-existing PER, *i.e.*, $P\, a \wedge a \sim_A a' \Rightarrow P\, a'$, then a simpler construction is available which often reduces the formalisation effort. We refer to such predicates as *strong*.

$$(A;\ a\, a' \mapsto (P\, a \vee P\, a') \wedge a \sim_A a')$$

It could be argued that this is the only case of the two above that should be considered, as all predicates over P-sets should respect the PER of the P-set.

Experience working with P-sets and modelling their subsets suggests that there is in fact a third case, when the predicate, P, can be rephrased as the diagonal of some binary relation, R, such that the following two conditions hold:

- $P\, a \Leftrightarrow R\, a\, a$; and
- $a_1 \sim_A a_2 \wedge R\, a_2\, a_3 \wedge a_3 \sim_A a_4 \Rightarrow R\, a_1\, a_4$

We refer to such predicates as *bistrong*. It is often more convenient to consider only the binary relation, and induce the predicate by the diagonal, than to consider both; we, therefore, also refer to such binary relations as *bistrong*. In this third case the sub-P-set construction is given by the sequel.

$$(A;\ a\, a' \mapsto R\, a\, a' \wedge a \sim_A a')$$

Note that in each of the three solutions the underlying carrier type is left unmodified. The goal of using the logical aspect, rather than the computational aspect, of P-sets to model the logical constructions of subsets and quotients has been achieved. Where each construction exists and is well-justified, all three constructions given hitherto are equivalent one with each other: *i.e.*, there are P-well-defined P-functions mapping from each to the other that are the identity function on the underlying carrier type.

As suggested in the motivation, there may be a fourth solution to modelling sub-P-sets: finding a structural presentation of the desired sub-P-set, *i.e.*, a type equivalent with $\sum_{a\,:\,A} P\, a$ that avoids using the Σ-type, which mixes the computational data with logical property. Interestingly, this solution is not equivalent, in general, with any of the three constructions given above. To see why, consider the following construction. For a given P-set, $A \triangleq (|A|, \sim_A)$, define A^* by the sequel:

- $|A^*| \triangleq 1 + |A|$; and
- $x \sim_{A^*} y$ is inductively generated by:
  - $a \sim_A a' \Rightarrow \mathrm{inr}(a) \sim_{A^*} \mathrm{inr}(a')$[3].

Intuitively, perhaps, it would seem that in general A and A^* should be equivalent as P-sets, as they have exactly the same P-well-defined elements. Although this analysis of P-well-defined elements is true, A and A^* may have different ghost elements. It is quite clear how to define a (well-defined) P-function from A to A^*: it is simply the injection into the

[3] Where inr is the injection into the right summand of $1 + |A|$.

right-hand $|A|$ component of $|A^*|$. This P-function is injective (in an appropriate P-sense), and surjective (in an appropriate P-sense) over the P-well-defined elements, but it does not, in general, have an inverse as there may not be elements of $|A|$ to which to map the left component of $|A^*|$. This construction exemplifies a further point: P-functions must compute on ghost elements, even if they map ghost elements to proper elements.

Construction 4.1.13 (Sub-P-Sets)**.** Given a P-set, A, and a predicate, P, over the elements of its underlying carrier type, the induced *sub-P-set* by restricting the P-well-defined elements only to them that satisfy P is defined by the sequel.

$$A\restriction_P \triangleq (|A|, a\, a' \mapsto P\, a \wedge P\, a' \wedge a \sim_A a')$$

Given a P-set, A, and a strong predicate, P, over the elements of its underlying carrier type, the induced *sub-P-set* by restricting the P-well-defined elements only to them that satisfy P is defined by the sequel.

$$A\restriction_P \triangleq (|A|, a\, a' \mapsto (P\, a \vee P\, a') \wedge a \sim_A a')$$

Given a P-set, A, and a bistrong relation, R, over the elements of its underlying carrier type, the induced *sub-P-set* by restricting the P-well-defined elements only to them that satisfy the diagonal of R is defined by the sequel.

$$A\restriction_R \triangleq (|A|, a\, a' \mapsto R\, a\, a' \wedge a \sim_A a')$$

4.1.3 *Advantages and Disadvantages*

The uniformity of the modelling of both quotients and subsets in P-sets may seem to be merely a philosophical or aesthetic advantage. Nonetheless, this uniformity gives a number of other advantages. It avoids and reduces using Σ-types within the carriers types within type-theoretic formalisms, and, since it is the underlying carrier type that is computational this may lead to more computationally efficient developments. When proof-relevant relations are used, it avoids mixing the universes in which the computational data resides with the universes in which the logical properties reside. Therefore, it would reduce the complexity of how universes are tracked.

However, there are a number of disadvantages to P-sets. The need to establish self-relation of elements of the underlying carrier type to ensure their P-well-definition gives a more formal account of setoid hell. However, this need is worse than setoid hell, and is its own problem of *subsetoid hell*. Subsetoid hell is arguably "worse" than setoid hell as P-well-definition is an hereditary property rather than a top-level property, and because of the non-availability of general reflexivity for sub-proofs. This latter property is exemplified well in the case of binary P-functions: if one wishes to relate $f(a, b)$ with $f(a', b)$ whenever f is P-well-defined, one needs to prove that b is P-well-defined in addition to proving that a is related with a'. Moreover, properties are always phrased in this logical form where properties are not defined pointwise but over two related elements. This logical phrases results in contexts being longer. This is reminiscent of the parametricity translation for type theory; indeed, relations in parametricity translations are precisely PERs. It also favours proofs whose number of steps of transitivity is an odd number; the inelegance of having an even number of steps of transitivity is exemplified well by the proof of transitivity for the PER for function P-sets.[4] As remarked in subsection 4.1.2, P-functions must compute on ill-defined elements, whereas E-functions do not need to do so as E-well-definition is included in the underlying carrier type, and so ill-defined elements can be eliminated when computing.

The avoidance of Σ-types in modelling subsets could prove to be a useful property even in type-theoretic settings where setoids are not traditionally used. For example, univalent foundations do not traditionally use setoids as there is enough strength in the type theory not to require using setoids. However, computational efficiency is an important consideration for some presentations of univalent foundations and Mathematical formalisms therein. By structuring a Mathematical theory within univalent foundations not to use Σ-types, large computational efficiency gains may be possible. Although univalent foundations model propositions by types that are computationally irrelevant, this irrelevance does not necessarily

[4] Such inelegance is not present when quasi-partial-equivalence relations/zig-zag relations/difunctional relations are used.

equate well with (lack of) computational cost. By separating the computationally relevant from the computationally irrelevant, a univalent formalism of a Mathematical theory could be made much more efficient, as the computational model for the type theory would not attempt to compute the computationally irrelevant propositions when computing the computationally relevant structural data.

4.2 P-Categorical Definitions

In this section we provide definitions for all the necessary P-categorical structures for this thesis. Further P-categorical concepts are defined in section 4.4, having discussed the structure of the P-category of P-sets in section 4.3. Having defined the necessary concepts, in subsection 4.2.6 we discuss certain drawbacks and problems with the definition of P-categories and how they manifest themselves within the definitions. We also set our definitions within the greater categorical frameworks of internal category theory and enriched category theory in subsection 4.2.7. Finally, we analyse the general shape of the definition of a P-category, and how it accounts for the differences within the P-setting.

4.2.1 *P-Categories*

The definition of a P-category looks much the same as the definition of an E-category, *e.g.*, in Palmgren (2019), and definitions of categories in standard Mathematical settings, *e.g.*, in Eilenberg et al. (1945). Nevertheless, the P-setting necessitates some important differences and adaptations compared with the more standard settings. These differences are common thread through the definitions of categorical structures in the P-setting.

Definition 4.2.1 (P-Category). A *P-category*, $\mathbb{C}$, is defined by the following stuff and structure:

- a type of objects, $\mathbb{C}_0$;
- a family, indexed over two objects, of P-sets as the homs, $\mathbb{C}_1 \ : \ \mathbb{C}_0 \to \mathbb{C}_0 \to \mathsf{PSet}$;
- a family, indexed over three objects, of composition functions, $- \circ = : \ \mathbb{C}_1(y,z) \to \mathbb{C}_1(x,y) \to \mathbb{C}_1(x,z)$; and
- a family of identity morphisms, $\mathrm{id}_x \ : \ \mathbb{C}_1(x,x)$;

such that the following properties hold:

- the family of composition functions respects the PER in the two arguments simultaneously;
- the family of identity morphisms is self-related;
- the composition is associative; and
- the identity is unital for the composition function.

The first of these conditions is simply that the composition function is P-well-defined. Nonetheless, it has an analogue in the E-setting unlike the self-relation of the identity morphism.

We now give explicitly the form of the latter two conditions.

$$\begin{aligned} \text{Associativity: } & f \sim f' \wedge g \sim g' \ \wedge \ h \sim h' \Rightarrow (f \circ g) \circ h \sim f' \circ (g' \circ h') \\ \text{Left Unit: } & f \sim f' \Rightarrow \mathrm{id} \circ f \sim f' \\ \text{Right Unit: } & f \sim f' \Rightarrow f \circ \mathrm{id} \sim f' \end{aligned}$$

Note that these conditions are given in the logical form commensurate with the partiality of PERs. The associativity condition can thus be phrased as the left-associated triple-composition function being related with the right-associated triple-composition function. The left (respectively, right) unit law can be phrased similarly as the left (respectively, right)

```
Cumulative Record PCat@{+i +j} := {
  PCat_obj :> Type@{i};
  PCat_hom : PCat_obj -> PCat_obj -> PType@{j};

  PCat_id_mor : forall x, PCat_hom x x;
  PCat_comp : forall {x y z}, PCat_hom y z -> PCat_hom x y -> PCat_hom x z;

  PCat_id_rel : forall x, PCat_id_mor x ~ PCat_id_mor x;

  PCat_comp_rel : forall {x y z} {f f' : PCat_hom y z} {g g' : PCat_hom x y},
    f ~ f' -> g ~ g' -> PCat_comp f g ~ PCat_comp f' g';

  PCat_assoc : forall {w x y z}
    {f f' : PCat_hom y z} {g g' : PCat_hom x y} {h h' : PCat_hom w x},
    f ~ f' -> g ~ g' -> h ~ h' ->
      PCat_comp (PCat_comp f g) h ~ PCat_comp f' (PCat_comp g' h');

  PCat_left_id : forall {x y} {f f' : PCat_hom x y},
    f ~ f' -> PCat_comp (PCat_id_mor y) f ~ f';

  PCat_right_id : forall {x y} {f f' : PCat_hom x y},
    f ~ f' -> PCat_comp f (PCat_id_mor x) ~ f';
}.
```

Listing 4.2: P-Category Rocq Definition

composition with identity function being related with the identity function. This definition is rendered in Rocq in listing 4.2.

Remark 4.2.2. By abuse of notation, we often omit the subscripted zero and one when referring to the type of objects, and to the family of hom P-sets of a P-category.

From the definition of a P-category many standard constructions follow without much difficulty.

Construction 4.2.3 (Opposite P-Category). The construction of opposite P-categories follows the standard construction, and proceeds without any difficulty in the P-setting. We denote the opposite of a P-category, $\mathbb{C}$, by $\mathbb{C}^{\mathrm{op}}$.

Construction 4.2.4 (Product P-Category). The construction of product P-categories follows the standard construction, and proceeds without any difficulty in the P-setting. We denote the product of P-categories, $\mathbb{C}$ and $\mathbb{D}$, by $\mathbb{C} \times \mathbb{D}$.

Example 4.2.5 (P-Category of P-Sets). There is a P-category of P-sets, which we denote by **PSet**. It is defined by the following stuff and structure:

- P-sets for the objects;
- the P-set of P-functions for the hom P-sets;
- composition of P-functions for the composition; and
- the identity P-function for the identities.

We develop further the P-categorical structure of **PSet** in section 4.3.

Having defined P-categories, we now define the appropriate notion of isomorphism within the P-setting.

Definition 4.2.6 (P-Isomorphism). A pair of morphisms, $f : x \to y$ and $f^{-1} : y \to x$, in some P-category exhibit a *P-isomorphism* between x and y precisely when the following properties hold:

- $f \sim f$;
- $f^{-1} \sim f^{-1}$;
- $f \circ f^{-1} \sim \mathrm{id}_y$; and
- $f^{-1} \circ f \sim \mathrm{id}_x$.

We say that a morphism $f : x \to y$ is a *P-isomorphism* if there exists a putative f^{-1} such that the above properties hold. This definition is rendered in Rocq in listing 4.3.

```
Cumulative Record IsPIso@{*i *j} {C : PCat@{i j}} {x y : C} (f : PCat_hom x y) := {
  IsPIso_rel : f ~ f;
  IsPIso_inv : PCat_hom y x;
  IsPIso_inv_rel : IsPIso_inv ~ IsPIso_inv;
  IsPIso_left : PCat_comp IsPIso_inv f ~ PCat_id_mor _;
  IsPIso_right : PCat_comp f IsPIso_inv ~ PCat_id_mor _;
}.
```

Listing 4.3: P-Isomorphism Rocq Definition

Remark 4.2.7. In standard category theory inverses to morphisms are unique, and therefore a morphism merely having an inverse is a property of the given morphism rather than structure. However, in P-category theory inverses are only unique up to the PER, and as such there could, in general, be many computationally different (but necessarily PER-equivalent) morphisms that are inverses to a given morphism. Therefore, being a P-isomorphism is not a mere property of a P-morphism, as some choice has to be made about the computationally relevant inverse morphism. We are nonetheless lax about our terminology which may otherwise suggest that it is a property.

We need two further constructions on P-categories, which we now provide for the sake of nomenclature and clarity of definition.

Construction 4.2.8 (Co-Free Dagger P-Category)**.** Given a P-category, $\mathbb{C}$, its *co-free dagger P-category*, $\mathbb{C}^{\dagger}$, is defined by the following stuff and structure:

- the objects of $\mathbb{C}$ for the objects of $\mathbb{C}^{\dagger}$;
- pairs of morphisms in both directions for the homs, $f : x \rightleftarrows y : f^{\dagger}$;
- composition and reverse composition for the composition; and
- pairs of identities for the identities.

Construction 4.2.9 (Co-Free P-Groupoid)**.** Given a P-category, $\mathbb{C}$, its *co-free P-groupoid*, $\mathbb{C}^{\cong}$, is defined by the following stuff and structure:

- the objects of $\mathbb{C}$ for the objects of $\mathbb{C}^{\cong}$;
- pairs of morphisms in both directions for the homs, $f : x \mathrel{\tilde{\rightleftarrows}} y : f^{-1}$, such that they exhibit a P-isomorphism;
- composition and reverse composition for the composition; and
- pairs of identities for the identities.

Remark 4.2.10. The hom-P-set for co-free P-groupoids can be defined as a sub-P-set by a bistrong relation.

4.2.2 *P-Functors*

The definition of P-functor follows straightforwardly from the usual definition of a functor in standard category theory taking into consideration the P-setting. Moreover, we define two concrete instances of multifunctors: binary and ternary P-functors that have separate, but commuting, functorial actions in each of their arguments independently. Such multifunctors allow for stricter specification of left and right actions of binary P-functors than that afforded by P-functors out of product P-categories. Defining ternary P-functors allows us to obviate the need for an associating P-functor from a left-associated triple product of P-categories to a right-associated triple product of P-categories when we come to define P-bicategories.

Definition 4.2.11 (P-Functor). A *P-functor*, F, from the P-category $\mathbb{C}$ to the P-category $\mathbb{D}$, which we denote by $\mathbb{C} \to \mathbb{D}$, is defined by the following structure:

- a mapping of objects, $F_0 : \mathbb{C}_0 \to \mathbb{D}_0$; and
- a family of mappings of morphisms, $F_1 : \mathbb{C}_1(x, y) \to \mathbb{D}_1(F_0\, x, F_0\, y)$;

such that the following properties hold:

- the family of morphism mappings respect the domain PER;
- the family of morphism mappings respect composition; and
- the family of morphism mappings respect identity morphisms.

The first of these conditions is simply that the family of morphism mappings is self-related, and as such has an analogue in the E-setting. The latter two conditions are the usual functoriality conditions, adapted to the P-setting, which we now give explicitly.

$$\text{Respect for composition: } f \sim f' \wedge g \sim g' \Rightarrow F_1(f \circ g) \sim F_1(f') \circ F_1(g')$$
$$\text{Respect for identity: } F_1(\mathrm{id}_x) \sim \mathrm{id}_{F_0\, x}$$

This definition is rendered in Rocq in listing 4.4.

```
Cumulative Record PFun@{*i1 *j1 *i2 *j2} (C : PCat@{i1 j1}) (D : PCat@{i2 j2}) := {
  PFun_obj_of :> C -> D;

  PFun_hom_of : forall {x y},
    PCat_hom x y -> PCat_hom (PFun_obj_of x) (PFun_obj_of y);

  PFun_hom_rel : forall {x y} {f f' : PCat_hom x y},
    f ~ f' -> PFun_hom_of f ~ PFun_hom_of f';

  PFun_comp_of : forall {x y z} {f f' : PCat_hom y z} {g g' : PCat_hom x y},
    f ~ f' -> g ~ g' ->
      PFun_hom_of (PCat_comp f g) ~ PCat_comp (PFun_hom_of f') (PFun_hom_of g');

  PFun_id_of : forall x, PFun_hom_of (PCat_id_mor x) ~ PCat_id_mor (PFun_obj_of x);
}.
```

Listing 4.4: P-Functor Rocq Definition

Remark 4.2.12. By abuse of notation, we often omit the subscripted zero and one when referring to the object mapping, and to the family of morphism mappings of a P-functor.

Definition 4.2.13 (Dominant P-Functor). A P-functor, $F : \mathbb{C} \to \mathbb{D}$, is *dominant* whenever every object in $\mathbb{D}$ is a retract of an object in the image of F. Therefore, for every object, d, in $\mathbb{D}$ there is an object, c, in $\mathbb{C}$, and a pair of morphisms:

$$d \to F\,c \to d \ ;$$

that compose to the identity.

Definition 4.2.14 (P-Bifunctor). A *P-bifunctor*, F, from the P-categories $\mathbb{B}$ and $\mathbb{C}$ to the P-category $\mathbb{D}$, which we denote by $(\mathbb{B};\, \mathbb{C}) \to \mathbb{D}$, is defined by the following structure:

- a mapping of objects, $F_0 : \mathbb{B}_0 \to \mathbb{C}_0 \to \mathbb{D}_0$;
- two families of mappings of morphisms:[5]
  - $F_l : \mathbb{B}_1(x, y) \to \mathbb{D}_1(F_0\,x\,c, F_0\,y\,c)$; and
  - $F_r : \mathbb{C}_1(x, y) \to \mathbb{D}_1(F_0\,b\,x, F_0\,b\,y)$;

such that the following properties hold:

- the two families of mappings of morphisms respect their domain PER;
- the two families of mappings of morphisms respect composition;
- the two families of mappings of morphisms respect identities; and
- the two families of mappings of morphisms commute with each other.

This final condition is given explicitly by the sequel.

$$f \sim f' \wedge g \sim g' \Rightarrow F_l(f) \circ F_r(g) \sim F_r(g') \circ F_l(f')$$

It is this final condition that allows the independent and separate P-functorial actions to be equivalent with a single jointly functorial action.

Remark 4.2.15. A P-bifunctor $(\mathbb{B};\mathbb{C}) \to \mathbb{D}$ is similar to a P-functor $\mathbb{B} \times \mathbb{C} \to \mathbb{D}$. Each can be used to induce the other with each round-trip being naturally isomorphic[6] to the original. The morphism action for the latter kind of P-functor is joint: one has to provide morphisms in both categories even if one is the identity. However, the morphism action for the former kind of P-bifunctor is separate: one can only provide a single morphism (from one of the categories). P-Bifunctors provide stricter P-functorial action for the common case that one of the morphisms is the identity. However, they force one to pick an ordering of actions if one needs to act with morphisms from both categories, although this ordering is irrelevant.

Construction 4.2.16 (Induced P-Functors). Given a P-bifunctor and an object of one of the domain P-categories we may induce an ordinary P-functor. That is, we may induce the following P-functors from the P-bifunctor, $F : (\mathbb{B};\, \mathbb{C}) \to \mathbb{D}$.

- $F(b, -) : \mathbb{C} \to \mathbb{D}$
- $F(-, c) : \mathbb{B} \to \mathbb{D}$

Definition 4.2.17 (P-Trifunctor). We further define a notion of *P-trifunctor*, having three arguments. We denote a P-trifunctor, F from the P-categories $\mathbb{B}$, $\mathbb{C}$, and $\mathbb{D}$ to the P-category $\mathbb{E}$ by $F : (\mathbb{B};\, \mathbb{C};\, \mathbb{D}) \to \mathbb{E}$. The definition largely follows the pattern of the definition of P-bifunctor. However, we now have three P-functorial actions that each need to commute one with each other. We denote these three P-functorial maps by F_l, F_m[7], and F_r.

[5] The subscripts come from 'left' and 'right'.
[6] In an appropriately P-theoretic sense to be defined later.
[7] The subscript here comes from 'middle'.

Remark 4.2.18. Construction 4.2.16 extends to P-trifunctors, allowing us to induce P-bifunctors by fixing any one of the arguments. These P-bifunctors may be further used to induce P-functors according to construction 4.2.16. Thus, we can induce P-functors by fixing any two of the arguments.

Construction 4.2.19 (P-Bifunctor Composition (I))**.** Composition with P-bifunctors is not a single construction, but a number of constructions.

Firstly, we may compose a P-functor after a P-bifunctor according to the sequel.

$$\begin{aligned} F &: \mathbb{C} \to \mathbb{D} \\ G &: (\mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{C} \\ F \circ G &: (\mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{D} \end{aligned}$$

Secondly, we may compose a pair of P-functors before a P-bifunctor according to the sequel.

$$\begin{aligned} F_1 &: \mathbb{B}_1 \to \mathbb{C}_1 \\ F_2 &: \mathbb{B}_2 \to \mathbb{C}_2 \\ G &: (\mathbb{C}_1;\, \mathbb{C}_2) \to \mathbb{D} \\ G \circ (F_1;\, F_2) &: (\mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{D} \end{aligned}$$

Remark 4.2.20. Construction 4.2.19 has analogues for P-trifunctors, where P-functors may be composed before or after a P-trifunctor.

Construction 4.2.21 (P-Bifunctor Composition (II))**.** Having defined the notion of a P-trifunctor, we have two further compositions available to us for P-bifunctors. We can compose two P-bifunctors into a P-trifunctor in two different ways according to the association.

The left-associated composition can be described thus:

$$\begin{aligned} F &: (\mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{C} \\ G &: (\mathbb{C};\, \mathbb{D}) \to \mathbb{E} \\ G \circ_l F &: (\mathbb{B}_1;\, \mathbb{B}_2;\, \mathbb{D}) \to \mathbb{E} \end{aligned}$$

The right-associated composition can be described thus:

$$\begin{aligned} F &: (\mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{D} \\ G &: (\mathbb{C};\, \mathbb{D}) \to \mathbb{E} \\ G \circ_r F &: (\mathbb{C};\, \mathbb{B}_1;\, \mathbb{B}_2) \to \mathbb{E} \end{aligned}$$

Remark 4.2.22. It is the form of P-bifunctor composition in construction 4.2.21 that necessitates the introduction of the notion of P-trifunctor. We do not use higher arity P-multifunctors, and therefore do not consider more general definitions of P-multifunctors.

Construction 4.2.23 (P-Functor Pairing)**.** A pair of P-functors, $F : \mathbb{B} \to \mathbb{C}$ and $G : \mathbb{B} \to \mathbb{D}$, induce a P-functor, $\langle F, G \rangle : \mathbb{B} \to \mathbb{C} \times \mathbb{D}$. Similarly, pairs of P-(bi/tri)functors can be used to induce P-(bi/tri)functors into a product P-category.

Example 4.2.24 (Hom P-Bifunctor)**.** For each P-category, $\mathbb{C}$, we have a P-bifunctor:

$$\mathrm{Hom}_{\mathbb{C}} : (\mathbb{C}^{\mathrm{op}};\, \mathbb{C}) \to \mathbf{PSet}$$

defined by the following mappings:

$$\begin{aligned}\mathrm{Hom}_0(x, y) &\triangleq \mathbb{C}(x, y);\\ \mathrm{Hom}_l(f) &\triangleq (g \mapsto g \circ f); \text{and}\\ \mathrm{Hom}_r(f) &\triangleq (g \mapsto f \circ g).\end{aligned}$$

Remark 4.2.25. Advantages of defining Hom as a P-bifunctor — rather than as a P-functor from the product P-category — include that it is more convenient to have access to separate P-functorial actions, and that we are not forced to pick an association for the triple composition of the functorial action.

4.2.3 *P-Natural Transformations*

The definition of P-natural transformations follows straightforwardly from the usual definition of a natural transformation taking into consideration the P-setting. We also define as a unified concept the notion of a P-natural isomorphism.

Definition 4.2.26 (P-Natural Transformation)**.** A *P-natural transformation*, α, from the P-functor $F : \mathbb{C} \to \mathbb{D}$ to the P-functor $G : \mathbb{C} \to \mathbb{D}$, which we denote by $\alpha : F \Rightarrow G$, is given by the following structure:

- a family of morphisms, $\alpha_x : F\,x \to G\,x$, called the components;

such that the following properties hold:

- the components are self-related, $\alpha_x \sim \alpha_x$; and
- the components satisfy the P-naturality condition.

The components are required to be self-related as otherwise it would not be possible to prove anything about the components of an arbitrary P-natural transformation due to P-categorical properties being phrased in logical rather than pointwise form, nor would the P-naturality condition be meaningful. The P-naturality condition is a rephrasing of the usual naturality condition to account for the partiality of the P-setting. Explicitly it is given by the sequel.

$$\bigforall_{x\,y\,:\,\mathbb{C}_0}\ \bigforall_{f\,f'\,:\,x \to y} f \sim f' \Rightarrow \alpha_y \circ F\,f \sim G\,f' \circ \alpha_x$$

These definitions are rendered in Rocq in listing 4.5.

```
Definition IsPNatural@{i1 j1 i2 j2} {C} {D} {F G : PFun@{i1 j1 i2 j2} C D}
  (alpha : forall x, PCat_hom (F x) (G x)) :=
    forall x y (f f' : PCat_hom x y), f ~ f' ->
      PCat_comp (alpha y) (PFun_hom_of F f) ~ PCat_comp (PFun_hom_of G f') (alpha x).

Cumulative Record PNatTrans@{*i1 *j1 *i2 *j2} {C} {D}
  (F G : PFun@{i1 j1 i2 j2} C D) := {
  PNatTrans_comps_of :> forall x, PCat_hom (PFun_obj_of F x) (PFun_obj_of G x);

  PNatTrans_comps_rel : forall x, PNatTrans_comps_of x ~ PNatTrans_comps_of x;

  PNatTrans_commutes : IsPNatural PNatTrans_comps_of;
}.
```

Listing 4.5: P-Natural Transformation Rocq Definition

Definition 4.2.27 (P-Natural Isomorphism). A *P-natural isomorphism* is simply a P-natural transformation whose components are P-isomorphisms. For simplicity, it is convenient to define this as a single concept, rather than as the intersection of two concepts. As such this allows us to express the P-naturality condition either in terms of the forward components, or in terms of the backwards components.

Definition 4.2.28 (P-(Bi/Tri)Natural Isomorphism). We extend the definition of P-natural isomorphism to that of *P-binatural isomorphism* between P-bifunctors, and *P-trinatural isomorphism* between P-trifunctors. These are defined similarly to P-natural isomorphisms, except that the components are multiply indexed in objects of all the domain P-categories, and that there is a P-naturality requirement for the components for each of the functorial actions.

4.2.4 *P-Functor Categories*

Hitherto, the definitions and constructions in P-category theory have largely followed straightforwardly either from standard category theory or from E-category theory. This is not the case for P-functor categories. The definition of functor categories in E-category theory has the hom E-sets be the *natural* transformations. However, these are an instance of a sub-E-set where there is both a computational part and a logical part: *viz.*, the components, and the E-naturality condition respectively. As such the appropriate definition in P-category theory keeps the computational parts and logical parts separate, by incorporating P-naturality as part of the PER for the hom P-sets. Before defining P-functor categories, we state a fact that we use in the definition thereof.

Fact 4.2.29 (P-Naturality Bistrength). *The P-naturality condition is bistrong. That is, we can rephrase*

$$\alpha \,:\, \prod_{x \,:\, \mathbb{C}_0} F_0\, x \to G_0\, x$$

being P-natural as α being on the diagonal of the following binary relation.

$$\alpha\, \alpha' \mapsto \bigvee_{x\, y \,:\, \mathbb{C}_0} \bigvee_{f\, f' \,:\, x \to y} f \sim f' \Rightarrow \alpha_y \circ F\, f \sim G\, f' \circ \alpha'_x$$

Construction 4.2.30 (P-Set of P-Natural Transformations). For this construction we fix a pair of functors, $F,\ G \,:\, \mathbb{C} \to \mathbb{D}$. Before constructing the P-set of P-natural transformations, we first define the P-set of transformations from F to G to be the discretely-indexed dependent product of the following family of P-sets.

$$(x \,:\, \mathbb{C}_0) \mapsto \mathbb{D}_1(F\, x, G\, x)$$

The *P-set of P-natural transformations* is given by the sub-P-set of transformations by the P-naturality bistrong relation.

Definition 4.2.31 (P-Functor Category). The *P-category of P-functors* from $\mathbb{C}$ to $\mathbb{D}$, which we denote by $[\mathbb{C}, \mathbb{D}]$, is given by the following stuff and structure:

- P-functors, $\mathbb{C} \to \mathbb{D}$, for the objects;
- the P-set of P-natural transformations for the hom P-sets;
- composition of transformations for the composition; and
- identity transformations for the identities.

Remark 4.2.32. The definition of P-functor category has an analogue for P-categories of P-bifunctors.

Remark 4.2.33. A P-natural transformation is simply a self-related morphism in the appropriate P-functor category. A P-natural isomorphism is simply a self-related morphism in the co-free P-groupoid of the appropriate P-functor category.

Example 4.2.34 (P-Category of P-Presheaves)**.** Given a P-category, $\mathbb{C}$, its P-category of P-presheaves is the P-functor category, $[\mathbb{C}^{\mathrm{op}}, \mathbf{PSet}]$. We also denote this P-category by $\widehat{\mathbb{C}}$. We refer to the P-category $\mathbb{C}$ as the base of the P-presheaf category $\widehat{\mathbb{C}}$.

Construction 4.2.35 (Nerve P-Functors)**.** Any P-functor, $F : \mathbb{C} \to \mathbb{D}$, induces a P-functor, $\langle F \rangle : \mathbb{D} \to \widehat{\mathbb{C}}$ defined by the sequel.

$$\langle F \rangle(d)(c) \triangleq \mathrm{Hom}_{\mathbb{D}}(F^{\mathrm{op}}(c), d)$$

Example 4.2.36 (Yoneda P-Functor)**.** The nerve P-functor of the identity P-functor is called the Yoneda P-functor. We denote the Yoneda P-functor by よ.

4.2.5 *P-Comma Categories*

The definition of a P-comma category largely follows straightforwardly from the definitions in standard category theory and E-category theory. First, we incorporate a couple of lessons from P-functor categories to arrive at the appropriate definition. The first of these lessons is perhaps more obvious than the second. Rather pleasantly, these two lessons arise from analysis both of the morphisms and of the objects of P-functor categories. Both seem at first consideration to be pointing out something obvious, but the implications extend beyond simply arriving at the correct definition. The first lesson helps to cement the motivation for P-category theory. The second lesson helps to exemplify one of the significant oddities of P-category theory.

The morphisms of a comma category in standard category theory are the commuting squares. The restriction to commuting squares is an instance of a subset, and so it should be clear and logical by now, that this commutation restriction should be expressed in the PER for the hom P-sets, rather than by Σ-types in the underlying carrier type for the hom P-sets. This is lesson number one.

The second lesson is somewhat more subtle. The objects of P-functor categories are P-functors, and thus incorporate a morphism action. This morphism action is susceptible to a P-well-definition requirement. This P-well-definition requirement is included in the objects of P-functor categories[8]. The objects of a comma category in standard category theory contain morphisms. The presence of these morphisms results in a notion of P-well-definition. Thus, the objects of P-comma categories should contain not simply morphisms from the appropriate hom P-set, but elements from the domain of the appropriate hom P-set; *i.e.*, the morphisms contained in the objects must be self-related.

These two lessons can be arrived at more mechanically by observing that arrow categories are both a special case of comma categories (and the variation to the general case is only slight), as well as an instance of functor categories. We can thus arrive at the correct definition of P-comma categories by appropriately varying the definition of the P-arrow category. The P-arrow category of some P-category, $\mathbb{C}$, is the P-functor category from the walking arrow P-category,[9] I, to $\mathbb{C}$. The walking arrow P-category has two objects, and three P-well-defined morphisms: identities for each of the two objects, and a single morphism from the first object to the second. Since the non-identity morphism is always self-related, its image under some P-functor must also be self-related; thus, an object of the P-functor category, $[I, \mathbb{C}]$, is a choice of two objects in $\mathbb{C}$, and a self-related morphism from the chosen first object to the chosen second object. Moreover, a morphism of the P-functor category, $[I, \mathbb{C}]$, is a P-natural transformation; however, there is only one interesting choice of morphism for naturality:[10] *viz.*, the non-identity morphism from the first object to the second object which results in a commuting square. It is now clear how to define the P-arrow category without reference to P-functor categories, and as such we make the following definition for P-comma categories.

Definition 4.2.37 (P-Comma Category)**.** The *P-comma category* of $F : \mathbb{B} \to \mathbb{D}$ with $G : \mathbb{C} \to \mathbb{D}$, which we denote by $(F \downarrow G)$, is defined by the following stuff and structure:

- pairs of objects, $x : \mathbb{B}$ and $y : \mathbb{C}$, with self-related morphisms, $F\,x \to G\,y$, for the objects;

[8] This inclusion is somewhat inimical to the spirit of P-category theory, and is discussed in subsection 4.2.6.

[9] The walking arrow category, I, consists of two objects, and a single morphism from the first object to the second as well as the identity morphisms of the two objects.

[10] The only other morphisms are identities, which are preserved by P-functors and so always give rise to commuting naturality squares.

- pairs of morphisms between the domain objects in $\mathbb{B}$, and the codomain objects in $\mathbb{C}$ forming commuting squares in $\mathbb{D}$ for the hom P-sets;
- composition in $\mathbb{B}$ and $\mathbb{C}$ for the composition; and
- identities in $\mathbb{B}$ and $\mathbb{C}$ for the identities.

We denote the objects of P-comma categories by the sequel.

$$\left(\begin{array}{c} x \\ \vert \\ f : F\,x \to G\,y \\ \downarrow \\ y \end{array} \right)$$

We denote the morphisms of P-comma categories by the sequel.

$$\left(\begin{array}{c} x \\ \vert \\ f : F\,x \to G\,y \\ \downarrow \\ y \end{array} \right) \begin{array}{c} \cdots\cdots h \cdots\cdots\to \\ \longrightarrow \\ \cdots\cdots k \cdots\cdots\to \end{array} \left(\begin{array}{c} u \\ \vert \\ g : F\,u \to G\,v \\ \downarrow \\ v \end{array} \right)$$

Construction 4.2.38 (Domain and Codomain Projection P-Functors)**.** For the P-comma category, $(F \downarrow G)$ where $F : \mathbb{B} \to \mathbb{D}$ and $G : \mathbb{C} \to \mathbb{D}$, there are two projection P-functors:

- $\mathrm{Dom} : (F \downarrow G) \to \mathbb{B}$; and
- $\mathrm{Cod} : (F \downarrow G) \to \mathbb{C}$.

Moreover, there is a P-natural transformation:

$$F \circ \mathrm{Dom} \Rightarrow G \circ \mathrm{Cod} : (F \downarrow G) \to \mathbb{D}$$

Construction 4.2.39 (Induced P-Functors into P-Comma Categories)**.** We may induce a P-functor from a P-category, $\mathbb{B}$, into a P-comma category, $(F \downarrow G)$ where $F : \mathbb{C} \to \mathbb{E}$ and $G : \mathbb{D} \to \mathbb{E}$, by the data:

- a P-functor, $H : \mathbb{B} \to \mathbb{C}$;
- a P-functor, $K : \mathbb{B} \to \mathbb{D}$; and
- a P-natural transformation, $\alpha : F \circ H \Rightarrow G \circ K$.

We denote the induced P-functor by $\langle H \downarrow_\alpha K \rangle$. We may omit the subscript α if it is canonically evident.

4.2.6 *Drawbacks*

One of the great advantages of P-sets is their ability to model subsets without using Σ-types, thereby avoiding mixing computational data with logical properties. This is used to good effect in P-categories for the homs, as exemplified, *e.g.*, in the definition of P-functor categories. The type of stuff of morphisms in P-functor categories is refined from P-natural transformations to all transformations, but where only the P-natural transformations are (self-)related. This has factored the Σ-type of P-natural transformations into its computational and logical parts. However, the objects in P-functor categories are still Σ-types, as there is no facility to move the logical parts elsewhere. If there were such a mechanism, the objects in functor categories would become some notion of prefunctor (analogous to how all transformations logically precede

the natural transformations) that would, if well-defined according to an appropriate logical notion, become an actual functor. Put simply, P-categories do not allow partiality for objects to be modelled similarly to how partiality is modelled for morphisms. This is a significant drawback for P-categories.

It is probable that introducing a well-definition requirement over objects would complicate the notion of category induced. Despite the infelicities of P-category theory being unable to model subsets for objects well, the relative simplicity of P-category theory suggests that it is a sensible middle-ground in terms of complexity *versus* expressivity for performing constructive category theory. We therefore leave study of how to complete the trajectory started by the switch from E-categories to P-categories to further work.[11]

4.2.7 *Internal and Enriched Perspectives*

Having covered the basic definitions of P-category theory, and considered some of the advantages and disadvantages thereof, we wish now to place the definition of P-category within the wider framework of category theory from the internal and enriched perspectives. We perform this analysis in the setting of E-category theory as a *lingua franca* for type-theoretic formalisms of category theory. We take for granted the definitions of category, functor, and natural transformation/isomorphism within E-category theory, and the existence of pullbacks in the E-category of E-sets. We further take for granted their usage in defining monoidal categories and enriched categories.

Definition 4.2.40 (E-Category of P-Sets). There is an E-category of P-sets given by the following stuff and structure:

- P-sets for the objects;
- the domain of P-functions for the hom E-sets;
- composition of functions for morphism composition; and
- identity functions for the identity morphisms.

Fact 4.2.41. *The E-category of P-sets is a monoidal E-category with monoidal product being the Cartesian product of P-sets, and with the monoidal unit being the unit P-set.*

Fact 4.2.42 (Enriched Definition of a P-Category). *A P-category is the same as an E-category enriched in the E-category of P-sets. The usage of the E-category of P-sets allows us to recover the logical phrasing of the categorical properties of P-categories.*

Fact 4.2.43. *The enriched definition of P-categories extends to the following facts:*

- *a P-functor is the same structure as an enriched functor; and*
- *a P-natural transformation is the same structure as an enriched natural transformation.*

Fact 4.2.44 (Internal Definition of a P-Category[12]). *A (small) P-category is the same as a category internal to the E-category of P-sets where the P-set of objects is discrete,* i.e.*, the PER is logically equivalent to equality.*

Remark 4.2.45. The internal definition has a resemblance to the work of Cottrell et al. (2017).

Remark 4.2.46. The internal definition naturally only defines the concept of a small P-category, as both the collection of objects and the collection of morphisms must be bounded by the same size coming from the same E-category of P-sets. Moreover, the internal definition uses a P-set of all morphisms between any objects. However, many examples would only relate those morphisms that have the same domain and codomain objects.

Remark 4.2.47. The internal definition of a P-category could provide some answers on how to introduce partiality to objects as discussed in subsection 4.2.6. The internal definition being restricted to small P-categories limits the examples that it would encompass.

[11] Subsection 4.2.7 may provide some guidance as to what might serve as a solution to the problem posed in this subsection. Any solution should have a "category" of "sets", functor "categories", and comma "categories".

[12] This fact is not present in the Rocq formalism.

4.2.8 *Structure of Definitions*

In this subsection, we examine the shape of the definition of a P-category more carefully. Similar analysis could be applied to the definitions both of P-functors and of P-natural transformations. This analysis may seem to be a little tedious, and somewhat hair-splitting; nonetheless, it allows the evaluation of the correctness of definitions for any further P-category-theoretic structure, and particularly P-bicategory-theoretic structure.

The shape of the definition of a P-category — as is the case for the definition of categories in standard category theory — is tripartite:

- stuff;
- structure; and
- properties.

Where by 'stuff' we mean any part of the definition valued in some universe, by 'structure' we mean any operation on the elements contained within the stuff, and by 'properties' we mean any logical restriction on the validity of candidate operations for the structures. A P-category has the following stuff:

- a type of objects;
- a doubly-indexed family of types of morphisms; and
- an endo-relation for each hom type.

This stuff has the following structure:

- a composition mapping for morphisms; and
- identity morphisms.

When we analyse the properties required of the stuff and structure for them to form a P-category we find that there are three parts to the properties: namely, P-well-definition properties, 0-structural properties, and 1-structural properties. The P-well-definition properties are:

- the composition mapping respecting the PER in both arguments; and
- the identity morphisms being self-related.

Both of these properties are, in essence, self-relation requirements. The only 0-structural property is:

- the endo-relation for the hom types being symmetric and transitive, *i.e.*, being a PER.

The 1-structural properties are:

- the composition mapping being associative up to the PER; and
- the identity morphisms being both left and right units for composition up to the PER.

We can therefore refine our analysis to state that P-categorical definitions are quadripartite:

- stuff;
- structure;
- P-well-definition properties; and
- structural properties.

This quadripartition motivates a second tripartition as follows:

- stuff;
- P-well-defined structure; and
- structural properties.

Where by 'well-defined structure' we mean that the structure lies in the relevant domain. For convenience we also bundle together the endo-relation and the 0-structural property of being a PER into a single concept of PER; this enables the further bundling together of the family of types of morphisms with their PERs into a single concept of family of P-sets. All the other structure and properties are left unbundled. It is this tripartition that allows the formulation of a convenient definition of P-bicategory in chapter 5.

4.3 P-Category of P-Sets

In this section, we investigate some P-categorical structure of the P-category of P-sets. We demonstrate how to construct (co)ends of P-bifunctors $(\mathbb{C}^{\mathrm{op}};\, \mathbb{C}) \to \mathbf{PSet}$. These constructions entail the expected completeness and cocompleteness properties of the P-category of P-sets. Additionally, we can prove the expected cocontinuity and continuity properties of the hom P-bifunctor.

4.3.1 *P-Ends*

In this subsection, we give the construction of P-ends in the P-category of P-sets. The construction of P-ends, being a general form of limit in the P-category of P-sets, provides insight into the P-limit theory of the P-category of P-sets, and how this is established in the P-setting.

Construction 4.3.1 (P-End)**.** The P-end of a P-bifunctor, $F : (\mathbb{C}^{\mathrm{op}};\, \mathbb{C}) \to \mathbf{PSet}$, is given by the following P-set.

- $\left|\int_{c\,:\,\mathbb{C}} F_0\, c\, c\right| \triangleq \prod_{c\,:\,\mathbb{C}} F_0\, c\, c$
- $w \sim w' \triangleq$
  - $\bigforall_{x\,y\,:\,\mathbb{C}}\, \bigforall_{f\,f'\,:\,x\to y} f \sim f' \Rightarrow F_l(f)(w\, y) \sim F_r(f')(w'\, x) \quad \wedge$
  - $\bigforall_{z\,:\,\mathbb{C}} w\, z \sim w'\, z$

Remark 4.3.2. Note that the structure of the PER for P-ends is that of a bistrong sub-P-set of a discretely indexed dependent product of P-sets.

Fact 4.3.3 (P-End Functor)**.** *The definition of P-ends extends to a P-functor from the P-bifunctor category,* $[(\mathbb{C}^{\mathrm{op}};\, \mathbb{C}), \mathbf{PSet}]$.

$$\mathrm{End} : [(\mathbb{C}^{\mathrm{op}};\, \mathbb{C}), \mathbf{PSet}] \to \mathbf{PSet}$$

Example 4.3.4 (P-Functor Category Hom)**.** The definition of hom P-sets in P-functor categories is a P-end.

$$[\mathbb{C}, \mathbb{D}]_1(F,\, G) \cong \int_{c\,:\,\mathbb{C}} \mathbb{D}_1(F\, c, G\, c)$$

Remark 4.3.5. This characterisation of the definition of the P-functor category hom P-set could be used as the definition of the hom P-sets for P-(bi)functor categories if one is careful about the order of definitions so as to avoid a chicken-and-egg problem.[13] The order of definitions would be something akin to the sequel:

[13] It is not used as the definition in the Rocq formalism due to universe issues. Once algebraic universes are fully available in Rocq, the universe issues will no longer be present, and it is expected that this definition will be adopted.

1. define the construction of P-ends non-functorially; thereafter
2. define the hom P-sets for P-(bi)functor categories; and finally thereafter
3. extend the construction of P-ends to be a P-functor.

Fact 4.3.6 (Hom P-Bifunctor Continuity). *The hom P-bifunctor is continuous in its second argument.*

$$\mathrm{Hom}_{\mathbf{PSet}}(X, \int_{c:C} F_0\, c\, c) \cong \int_{c:C} \mathrm{Hom}_{\mathbf{PSet}}(X, F_0\, c\, c)$$

Moreover, this P-isomorphism is P-natural in both X and F.

Remark 4.3.7. The existence of P-ends extends the enriched perspective of subsection 4.2.7 to allow for a fully enriched-theoretic explanation of the two dimensional structure of P-categories, P-functors, and P-natural transformations.

4.3.2 *P-Coends*

In this subsection, we give the construction of P-coends in the P-category of P-sets. The construction of P-coends, being a general form of colimit in the P-category of P-sets, provides insight into the P-colimit theory of the P-category of P-sets, and how this is established in the P-setting.

Construction 4.3.8 (P-Coend). The P-coend of a P-bifunctor, $F : (\mathbb{C}^{\mathrm{op}}; \mathbb{C}) \to \mathbf{PSet}$, is given by the following P-set.

- $\left|\int^{c:\mathbb{C}} F_0\, c\, c\right| \triangleq \sum_{c:\mathbb{C}} F_0\, c\, c$
- $w \sim w'$ is inductively generated by the sequel:
  - $\bigforall_{z:\mathbb{C}}\ \bigforall_{s\, s' : F_0\, z\, z}\ s \sim s' \Rightarrow (z; s) \sim (z; s')$
  - $\bigforall_{x\, y:\mathbb{C}}\ \bigforall_{f\, f' : y \to x}\ \bigforall_{s\, s' : F_0\, x\, y}\ f \sim f' \wedge s \sim s' \Rightarrow (y; F_l(f)(s)) \sim (x; F_r(f')(s'))$
  - $\bigforall_{x\, y:\mathbb{C}}\ \bigforall_{f\, f' : y \to x}\ \bigforall_{s\, s' : F_0\, x\, y}\ f \sim f' \wedge s \sim s' \Rightarrow (x; F_r(f)(s)) \sim (y; F_l(f')(s'))$
  - $w_1 \sim w_2 \wedge w_2 \sim w_3 \Rightarrow w_1 \sim w_3$

Fact 4.3.9 (P-Coend Functor). *The definition of P-coends extends to a P-functor from the P-category of P-bifunctors,* $[(\mathbb{C}^{\mathrm{op}}; \mathbb{C}), \mathbf{PSet}]$.

$$\mathrm{Coend} : [(\mathbb{C}^{\mathrm{op}}; \mathbb{C}), \mathbf{PSet}] \to \mathbf{PSet}$$

Fact 4.3.10 (Hom P-Bifunctor Cocontinuity). *The hom P-bifunctor is contravariantly cocontinuous in its first argument.*

$$\mathrm{Hom}_{\mathbf{PSet}}(\int^{c:C} F_0\, c\, c,\ Y) \cong \int_{c:C} \mathrm{Hom}_{\mathbf{PSet}}(F_0\, c\, c,\ Y)$$

Moreover, this P-isomorphism is P-natural in both F and Y.

4.3.3 *Sub-P-Presheaves*

In subsection 4.1.2 we considered how to form sub-P-sets given a P-set and a predicate on this P-set. Thereafter, we gave a construction for sub-P-sets for bistrong binary relations. In this subsection, we extend this construction to forming sub-P-presheaves by a compatible family of bistrong binary relations.

Construction 4.3.11 (Sub-P-Presheaf). Given the following data we can form a sub-P-presheaf.

- a P-presheaf, $F : \mathbb{C}^{\mathrm{op}} \to \mathbf{PSet}$; and

- a family, indexed over objects of $\mathbb{C}$, of endo-relations, R_c, over elements of $F\,c$;

such that the following properties hold:

- the family of relations is bistrong,

$$a_1 \sim_{F\,c} a_2 \wedge R_c\,a_2\,a_3 \wedge a_3 \sim_{F\,c} a_4 \Rightarrow R_c\,a_1\,a_4 \ \text{; and}$$

- the family of relations respects the morphism mapping of F,

$$f \sim f' \wedge a \sim_{F\,y} a' \wedge R_y\,a\,a' \Rightarrow R_x\,(F_1\,f\,a)\,(F_1\,f'\,a') \ .$$

The object mapping of the sub-P-presheaf, $F{\restriction_R}$, is defined by the sequel.

$$(F{\restriction_R})_0\ c \triangleq (F_0\,c){\restriction_{R_c}}$$

The morphism mapping of F, and the P-functoriality properties thereof, lift to $F{\restriction_R}$.

4.4 P-Categorical Structure

In this section, we define and analyse various universal constructions within P-category theory. Our guiding principle is to structure our definitions in a robustly category-theoretic style, using pre-existing P-categorical definitions, to ensure the correctness of the definitions within the P-setting. This produces the following strategy for describing these definitions:

1. firstly, define an object-forming operation;
2. secondly, set it within an adjointness property, phrased using hom P-set adjunctions; and
3. finally, induce an equational presentation therefrom.

We prefer this framework as it often reduces the conceptual complexity of the definition, and allows for using P-category-theoretic machinery when instantiating such structure for certain P-categories.

In our constructive and computational setting, we need to be able to construct and compute with the objects and morphisms of the universal structures that we use. We therefore require the objects and morphisms of the universal structures to be chosen. Of course, as universal structures, the objects are unique up to unique isomorphism, and the morphisms are unique up to the appropriate PER of the hom P-sets; thus, although from a P-categorical perspective there is no choice, from a computational perspective there is choice and this choice is necessary for computational effectivity of the definitions of universal structure within P-category theory.

4.4.1 *Terminal Objects*

Possibly the simplest form of universal construction is that of terminal objects. In our constructive and computational setting we define what it means for a chosen object to be a terminal object.

Definition 4.4.1 (P-Terminal Object)**.** An object, $\top$, of some P-category, $\mathbb{C}$, is a *P-terminal object* precisely when there exists a P-natural isomorphism as follows:

$$\mathrm{Hom}_{\mathbb{C}}(-, \top) \cong \Delta_1$$

where $\Delta_1 : \mathbb{C}^{\mathrm{op}} \to \mathbf{PSet}$ is the P-functor that is constantly the unit P-set. This definition is rendered in Rocq in listing 4.6.

```
Definition IsPTermObj {C : PCat} (term : C) :=
  PNatIso (PStrYoFun term) (PCompFun (PConstFun PUnit) PTermFun)
```

Listing 4.6: P-Terminal Object Rocq Definition

Construction 4.4.2 (P-Terminal Object Equations)**.** From this definition of P-terminal objects, we are able to induce for a P-terminal object, $\top$, a P-well-defined morphism from any object, $c : \mathbb{C}$, to $\top$, which we denote by $!_c : c \to \top$, such that $!_c$ is unique (up to the relevant PER) within $\mathrm{Hom}_{\mathbb{C}}(c, \top)$. This equational presentation is equivalent: it can be used to inhabit definition 4.4.1.

4.4.2 *Cartesian Products*

Possibly the next simplest form of universal construction is that of Cartesian products. In our constructive and computational setting we define what it means for a chosen type-theoretic mapping to construct Cartesian products.

Definition 4.4.3 (P-Cartesian Products)**.** A P-category, $\mathbb{C}$, has *P-Cartesian products* when it has an ordinary type-theoretic function:

$$- \times = : \mathbb{C}_0 \to \mathbb{C}_0 \to \mathbb{C}_0$$

such that for all pairs of objects, $a,\ b : \mathbb{C}$, there exists a P-natural isomorphism as follows:

$$\mathrm{Hom}_{\mathbb{C}}(-, a \times b) \cong \mathrm{Hom}_{\mathbb{C}\times\mathbb{C}}((-,-),(a,b))$$

This definition is rendered in Rocq in listing 4.7.

```
Definition IsPCartProd {C : PCat} (prod : C -> C -> C) :=
  forall a b, PNatIso
    (PStrYoFun (prod a b))
    (PCompFun (PStrYoFun (a, b)) (POppFun (PPairFun PIdFun PIdFun))).
```

Listing 4.7: P-Cartesian Products Rocq Definition

Construction 4.4.4 (P-Cartesian Product Equations)**.** From this definition of P-Cartesian products, we are able to induce the following structure for P-Cartesian products:

- P-well-defined projection morphisms, $\pi_1 : a \times b \to a$ and $\pi_2 : a \times b \to b$; and
- a P-well-defined pairing mapping, $\langle -, = \rangle : \mathrm{Hom}_{\mathbb{C}}(c, a) \to \mathrm{Hom}_{\mathbb{C}}(c, b) \to \mathrm{Hom}_{\mathbb{C}}(c, a \times b)$;

such that the following properties hold:

- $f \sim f' \wedge g \sim g' \Rightarrow \pi_1 \circ \langle f, g \rangle \sim f'$;
- $f \sim f' \wedge g \sim g' \Rightarrow \pi_2 \circ \langle f, g \rangle \sim g'$; and
- $h \sim h' \Rightarrow \langle \pi_1 \circ h, \pi_2 \circ h \rangle \sim h'$.

From these one can derive the following convenient property.

$$f \sim f' \wedge g \sim g' \wedge h \sim h' \Rightarrow \langle f, g \rangle \circ h \sim \langle f' \circ h', g' \circ h' \rangle$$

This equational presentation is equivalent: it can be used to inhabit definition 4.4.3.

Construction 4.4.5 (P-Cartesian Product P-Bifunctor)**.** The existence of the P-natural isomorphism extends the type-theoretic product function on objects into a P-bifunctor.

$$- \times = : (\mathbb{C}; \mathbb{C}) \to \mathbb{C}$$

4.4.3 *Cartesian P-Categories and P-Functors*

P-Cartesian structure is essential for the P-categorical analysis of simple type theory in chapter 7; we therefore find it useful to collect together P-Cartesian structure into single Mathematical entities (and, therefore, single entities within our Rocq formalism).

Definition 4.4.6 (P-Cartesian Category)**.** A *P-Cartesian category* is a P-category that has both a P-terminal object, and P-Cartesian products. This definition is rendered in Rocq in listing 4.8.

```
Record PCartCat := {
  PCartCat_cat :> PCat;

  PCC_term : PCartCat_cat;
  PCC_prod : PCartCat_cat -> PCartCat_cat -> PCartCat_cat;

  PCC_term_prop : IsPTermObj PCC_term;
  PCC_prod_prop : IsPCartProd PCC_prod;
}.
```

Listing 4.8: P-Cartesian Category Rocq Definition

Example 4.4.7. The P-category of P-sets, **PSet**, is P-Cartesian with P-terminal object the unit P-set, and P-Cartesian product the product of P-sets.

Example 4.4.8. P-Functor categories are P-Cartesian whenever the codomain P-category is P-Cartesian. The P-terminal object is the functor that is constantly the P-terminal object of the codomain P-category, and the P-Cartesian product is defined objectwise as the P-Cartesian product in the codomain P-category. In particular, P-categories of P-presheaves are P-Cartesian.

Definition 4.4.9 (P-Cartesian Functor)**.** A P-functor, F, from the P-Cartesian category $\mathbb{C}$ to the P-Cartesian category $\mathbb{D}$, is a *P-Cartesian functor* when it preserves the P-Cartesian structure up to P-isomorphism. That is, the following canonical morphisms are P-isomorphisms:

- $!_{F_0(\top_{\mathbb{C}})} : F_0(\top_{\mathbb{C}}) \to \top_{\mathbb{D}}$; and
- $\langle F_1\, \pi_1, F_1\, \pi_2 \rangle : F_0(a \times b) \to F_0(a) \times F_0(b)$ (for all $a\, b : \mathbb{C}$).

We denote the inverse to the former morphism by t, and the inverse to the latter morphism by p. We denote P-Cartesian functors from the P-Cartesian category $\mathbb{C}$ to the P-Cartesian category $\mathbb{D}$ by $\mathbb{C} \to_{\text{Cart}} \mathbb{D}$. This definition is rendered in Rocq in listing 4.9.

Definition 4.4.10 (P-Cartesian Functor Categories)**.** The *P-category of P-Cartesian functors* from the P-Cartesian category $\mathbb{C}$ to the P-Cartesian category $\mathbb{D}$, which we denote by $[\mathbb{C}, \mathbb{D}]_{\text{Cart}}$, is given by the following stuff and structure:

- P-Cartesian functors, $\mathbb{C} \to_{\text{Cart}} \mathbb{D}$, for the objects;
- the P-set of P-natural transformations for the hom P-sets;
- composition of transformations for the composition; and

```
Record PCartFun (C D : PCartCat) := {
  PCartFun_fun :> PFun C D;

  PCartFun_term_of_iso : IsPIso (PCC_zero_mor (PCartFun_fun (PCC_term _)));
  PCartFun_prod_of_iso : forall a b,
    IsPIso (PCC_pair
        (PFun_hom_of PCartFun_fun (PCC_pi_left a b))
        (PFun_hom_of PCartFun_fun (PCC_pi_right a b))
    );
}.
```

Listing 4.9: P-Cartesian Functor Rocq Definition

- identity transformations for the identities.

Remark 4.4.11. All P-natural transformations between P-Cartesian functors respect P-Cartesian structure; thus, there is no need to define any further structure.

Fact 4.4.12 (Identity P-Cartesian Functor). *The identity P-functor is a P-Cartesian functor.*

Fact 4.4.13 (P-Cartesian Functor Composition). *The composition of two P-Cartesian functors is a P-Cartesian functor.*

Example 4.4.14. Nerve P-functors are P-Cartesian functors.

Example 4.4.15. P-Comma categories, $(F \downarrow G)$, where $F : \mathbb{B} \to \mathbb{D}$ and $G : \mathbb{C} \to \mathbb{D}$, are P-Cartesian whenever $\mathbb{B}$, $\mathbb{C}$, and $\mathbb{D}$ are P-Cartesian categories, and G is a P-Cartesian functor.

Example 4.4.16. The domain and codomain projection P-functors are P-Cartesian functors.

Example 4.4.17. The P-functor, $\langle H \downarrow_\alpha K \rangle$, from $\mathbb{B}$ to $(F \downarrow G)$, where $F : \mathbb{C} \to \mathbb{E}$ and $G : \mathbb{D} \to \mathbb{E}$, induced by $H : \mathbb{B} \to \mathbb{C}$, $K : \mathbb{B} \to \mathbb{D}$, and $\alpha : F \circ H \Rightarrow G \circ K$ is P-Cartesian whenever both H and K are P-Cartesian.

4.4.4 *Cartesian Exponentials*

The Cartesian exponential fits within our principle of defining universal constructions using hom P-set adjunctions as an example of an adjunction. In our constructive and computational setting, we define what it means for a chosen type-theoretic mapping to construct Cartesian exponentials.

Definition 4.4.18 (P-Cartesian Exponential). A P-Cartesian category, $\mathbb{C}$, has *P-Cartesian exponentials* when it has an ordinary type-theoretic function:

$$(-)^{(=)} : \mathbb{C}_0 \to \mathbb{C}_0 \to \mathbb{C}_0$$

such that for all pairs of objects, $a\, b : \mathbb{C}$, there exists a P-natural isomorphism as follows:

$$\mathrm{Hom}_{\mathbb{C}}(-, b^a) \cong \mathrm{Hom}_{\mathbb{C}}(- \times a, b)$$

This definition is rendered in Rocq in listing 4.10. We may also denote the exponential, b^a, by $a \Rightarrow b$.

Construction 4.4.19 (P-Cartesian Exponential Equations). From this definition of P-Cartesian exponentials, we are able to induce the following structure for P-Cartesian exponentials:

- P-well-defined evaluation morphism, $\varepsilon : b^a \times a \to b$; and
- a P-well-defined Currying mapping, $(-)^* : \mathrm{Hom}_{\mathbb{C}}(c \times a, b) \to \mathrm{Hom}_{\mathbb{C}}(c, a \Rightarrow b)$.

```
Definition IsPCartExp {C : PCartCat} (exp : C -> C -> C) :=
  forall a b, PNatIso
    (PStrYoFun (exp b a))
    (PCompFun (PStrYoFun b) (POppFun (PBiFunIndRight PCartProdBiFun a))).
```

Listing 4.10: P-Cartesian Exponentials Rocq Definition

such that the following properties hold:

- $h \sim h' \Rightarrow (\varepsilon \circ \langle h \circ \pi_1, \pi_2 \rangle) \sim h'$; and
- $h \sim h' \Rightarrow \varepsilon \circ \langle h^* \circ \pi_1, \pi_2 \rangle \sim h'$.

From these one can derive the following convenient properties:

- $f \sim f' \wedge g \sim g' \wedge h \sim h' \Rightarrow \varepsilon \circ \langle f^* \circ g, h \rangle \sim f' \circ \langle g', h' \rangle$; and
- $f \sim f' \wedge g \sim g' \Rightarrow f^* \circ g \sim (f' \circ \langle g' \circ \pi_1, \pi_2 \rangle)^*$.

This equational presentation is equivalent: it can be used to inhabit definition 4.4.18.

Construction 4.4.20 (P-Cartesian Exponential P-Bifunctor)**.** The existence of the P-natural isomorphism extends the type-theoretic exponential function on objects into a P-bifunctor.

$$(-)^{(=)} \; : \; (\mathbb{C}; \mathbb{C}^{\mathrm{op}}) \to \mathbb{C}$$

4.4.5 *Cartesian-Closed P-Categories and P-Functors*

P-Cartesian-closed structure is essential for the P-categorical analysis of simple type theory in chapter 7; we therefore find it useful to collect together P-Cartesian-closed structure into single Mathematical entities (and, therefore, single entities within our Rocq formalism).

Definition 4.4.21 (P-Cartesian-Closed Category)**.** A *P-Cartesian-closed category* is a P-Cartesian category that has P-Cartesian exponentials. This definition is rendered in Rocq in listing 4.11.

```
Record PCartClosCat := {
  PCartClosCat_cat :> PCartCat;

  PCCC_exp : PCartClosCat_cat -> PCartClosCat_cat -> PCartClosCat_cat;
  PCCC_exp_prop : IsPCartExp PCCC_exp;
}.
```

Listing 4.11: P-Cartesian-Closed Category Rocq Definition

Example 4.4.22. The P-Cartesian category of P-sets, **PSet**, is P-Cartesian-closed with P-Cartesian exponential the P-set of functions.

Example 4.4.23. The P-Cartesian categories of P-presheaves are P-Cartesian-closed. The exponential of two P-presheaves $F, G \; : \; \widehat{\mathbb{C}}$ is given by the following P-end.

$$(F \Rightarrow G)(c) \triangleq \int_{z \,:\, \mathbb{C}} \mathrm{Hom}(z, c) \to F\, z \to G\, z$$

Definition 4.4.24 (P-Cartesian-Closed Functor). A P-Cartesian functor, F, from the P-Cartesian-closed category $\mathbb{C}$ to the P-Cartesian-closed category $\mathbb{D}$, is a *P-Cartesian-closed functor* when it preserves the P-Cartesian exponential structure up to P-isomorphism. That is, the following canonical morphism is a P-isomorphism:

$$(F_1(\varepsilon) \circ p)^* \ : \ F_0(a \Rightarrow b) \to F_0(a) \Rightarrow F_0(b)$$

where $p \ : \ F_0(a \Rightarrow b) \times F_0(a) \to F_0(a \Rightarrow b \times a)$ is the inverse to the canonical morphism establishing that F preserves P-Cartesian products. We denote the inverse to the canonical morphism by e. We denote P-Cartesian-closed functors from the P-Cartesian-closed category $\mathbb{C}$ to the P-Cartesian-closed category $\mathbb{D}$ by $\mathbb{C} \to_{\text{CartClos}} \mathbb{D}$. This definition is rendered in Rocq in listing 4.12.

```
Record PCartClosFun (C D : PCartClosCat) := {
  PCartClosFun_fun :> PCartFun C D;

  PCartClosFun_exp_of_iso : forall a b,
    IsPIso (PCCC_transp (PCat_comp
        (PFun_hom_of PCartClosFun_fun (PCCC_eval a b))
        (PCartFun_prod_of _ _ PCartClosFun_fun _ a)
    ));
}.
```

Listing 4.12: P-Cartesian-Closed Functor Rocq Definition

Definition 4.4.25 (P-Cartesian-Closed Functor Groupoids). The *P-groupoid of P-Cartesian-closed functors* from the P-Cartesian-closed category $\mathbb{C}$ to the P-Cartesian-closed category $\mathbb{D}$, which we denote by $[\mathbb{C}, \mathbb{D}]^{\cong}_{\text{CartClos}}$, is given by the co-free P-groupoid of the P-category defined by the following stuff and structure:

- P-Cartesian-closed functors, $\mathbb{C} \to_{\text{CartClos}} \mathbb{D}$, for the objects;
- the P-set of P-natural transformations for the hom P-sets;
- composition of transformations for the composition; and
- identity transformations for the identities.

Remark 4.4.26. All P-natural isomorphisms between P-Cartesian-closed functors respect P-Cartesian exponential structure; thus, there is no need to define any further structure.

Fact 4.4.27 (Identity P-Cartesian-Closed Functor). *The identity P-Cartesian functor is a P-Cartesian-closed functor.*

Fact 4.4.28 (P-Cartesian-Closed Functor Composition). *The composition of two P-Cartesian-closed functors is a P-Cartesian-closed functor.*

Example 4.4.29. For any P-Cartesian category, $\mathbb{C}$, any P-Cartesian-closed category, $\mathbb{D}$, and any dominant, full, and P-Cartesian functor, $F \ : \ \mathbb{C} \to \mathbb{D}$, the nerve of F, $\langle F \rangle \ : \ \mathbb{D} \to \widehat{\mathbb{C}}$, is a P-Cartesian-closed functor. As a corollary, the Yoneda P-Cartesian functor is P-Cartesian-closed as it is the nerve of the identity functor, which is dominant, full, and P-Cartesian.

Proof Sketch. To establish that $\langle F \rangle \ : \ \mathbb{D} \to \widehat{\mathbb{C}}$ is a P-Cartesian-closed functor we need to give a P-presheaf morphism as follows:

$$\left(\int_{z \,:\, \mathbb{C}} \text{Hom}_{\mathbb{C}}(z, -) \to \text{Hom}_{\mathbb{D}}(F\, z, a) \to \text{Hom}_{\mathbb{D}}(F\, z, b) \right) \to \text{Hom}_{\mathbb{D}}(F -, a \Rightarrow b) \ .$$

We give components of this morphism at c, *i.e.* morphisms:

$$\left(\int_{z\,:\,\mathbb{C}} \mathrm{Hom}_{\mathbb{C}}(z,c) \to \mathrm{Hom}_{\mathbb{D}}(F\,z,a) \to \mathrm{Hom}_{\mathbb{D}}(F\,z,b)\right) \to \mathrm{Hom}_{\mathbb{D}}(F\,c, a \Rightarrow b) \ .$$

Suppose that $f\ :\ a \to F\,x$ and $g\ :\ F\,x \to a$ witness the dominance of F at a, then the components are defined by:

$$\alpha \mapsto (\alpha_{c\times x}(\pi_1)(g \circ F\,\pi_2) \circ p \circ \langle \pi_1, f \circ \pi_2\rangle)^*$$

where $p\ :\ F\,c \times F\,x \to F(c \times x)$ is the inverse to the canonical morphism establishing that F preserves P-Cartesian products. □

4.4.6 *Cartesian Pre-Exponential*

In this subsection, we introduce a notion of Cartesian pre-exponentials that we need in the P-categorical analysis of simple type theory in chapter 7. It is a weaker notion than that of Cartesian exponentials, but it still exhibits many useful properties. It is not a universal construction; nonetheless, it is sufficiently similar with Cartesian exponentials that we place its definition and discussion here. In our constructive and computational setting, we define what it means for a chosen type-theoretic mapping to construct Cartesian pre-exponentials.

Definition 4.4.30 (P-Cartesian Pre-Exponential)**.** A P-category, $\mathbb{C}$, has *P-Cartesian pre-exponentials* when it has an ordinary type-theoretic function:

$$(-)^{(=)}\ :\ \mathbb{C}_0 \to \mathbb{C}_0 \to \mathbb{C}_0$$

such that for all pairs of objects, $a\,b\ :\ \mathbb{C}$, there exists a pair of P-natural transformations as follows:

$$\mathrm{Hom}_{\mathbb{C}}(-, b^a) \rightleftarrows \mathrm{Hom}_{\mathbb{C}}(- \times a, b)$$

This definition is rendered in Rocq in listing 4.13. We may also denote the pre-exponential, b^a, by $a \Rightarrow b$.

```
Definition IsPCartPreExp {C : PCartCat} (exp : C -> C -> C) :=
  forall a b, PNatTrepo
    (PStrYoFun (exp b a))
    (PCompFun (PStrYoFun b) (POppFun (PBiFunIndRight PCartProdBiFun a))).
```

Listing 4.13: P-Cartesian Pre-Exponentials Rocq Definition

Remark 4.4.31. The required P-natural transformations for P-Cartesian pre-exponentials are required to be chosen so that we can effectively construct and compute P-morphisms therefrom. The choice of P-natural transformations has an impact on how the morphisms induced therefrom compute. Since these P-natural transformations are not characterised by a universal property different choices may result in different structure. For instance, we could iterate the round-trip for a (potentially) different structure.

Construction 4.4.32 (P-Cartesian Exponential Equations)**.** From this definition of P-Cartesian pre-exponentials, we are able to induce the following structure for P-Cartesian pre-exponentials:

- P-well-defined pre-evaluation morphism, $\tilde{\varepsilon}\ :\ b^a \times a \to b$; and
- a P-well-defined pre-Currying mapping, $(-)^*$: $\mathrm{Hom}_{\mathbb{C}}(c \times a, b) \to \mathrm{Hom}_{\mathbb{C}}(c, a \Rightarrow b)$.

such that the following property holds of the pre-Currying mapping:

$$f \sim f' \wedge g \sim g' \Rightarrow f^* \circ g \sim (f' \circ \langle g' \circ \pi_1, \pi_2\rangle)^*$$

4.4.7 *Cartesian-Pre-Closed P-Categories and P-Functors*

P-Cartesian-pre-closed structure will prove useful for the P-categorical analysis of simple type theory in chapter 7; we therefore find it useful to collect together P-Cartesian-pre-closed structure into single Mathematical entities (and, therefore, single entities within our Rocq formalism).

Definition 4.4.33 (P-Cartesian-Pre-Closed Category)**.** A *P-Cartesian-pre-closed category* is a P-Cartesian category that has P-Cartesian pre-exponentials. This definition is rendered in Rocq in listing 4.14.

```
Record PCartPreClosCat := {
  PCartPreClosCat_cat :> PCartCat;

  PCPCC_exp : PCartPreClosCat_cat -> PCartPreClosCat_cat -> PCartPreClosCat_cat;
  PCPCC_exp_prop : IsPCartPreExp PCPCC_exp;
}.
```

Listing 4.14: P-Cartesian-Pre-Closed Category Rocq Definition

Definition 4.4.34 (P-Cartesian-Pre-Closed Functor)**.** A P-Cartesian functor, F, from the P-Cartesian-pre-closed category $\mathbb{C}$ to the P-Cartesian-closed category $\mathbb{D}$, is a *P-Cartesian-pre-closed functor* when it interacts well with the P-Cartesian pre-exponential structure of $\mathbb{C}$ and the P-Cartesian exponential structure of $\mathbb{D}$. That is, there is a P-well-defined morphism:

$$\tilde{e} \,:\, F_0(a) \Rightarrow F_0(b) \to F_0(a \Rightarrow b)$$

such that the following property holds:

$$f \sim f' \Rightarrow \tilde{e} \circ \big(F_1(f) \circ p\big)^* \sim F_1\big(f'^*\big)$$

where $p \,:\, F_0(a \Rightarrow b) \times F_0(a) \to F_0(a \Rightarrow b \times a)$ is the inverse to the canonical morphism establishing that F preserves P-Cartesian products. Note that this property is the same as a derivable property for P-Cartesian-closed functors. In the special case that f is the pre-evaluation morphism, it expresses that $\tilde{e}$ is almost a left-inverse to the canonical morphism. This definition is rendered in Rocq in listing 4.15.

Remark 4.4.35. It may seem odd to define P-Cartesian-pre-closed functors in an heterogeneous fashion between P-Cartesian-pre-closed categories and P-Cartesian-closed categories; but, this is what we need to analyse certain P-categorical structures of simple type theory in chapter 7.

Fact 4.4.36 (P-Cartesian-Pre-Closed Functor Composition)**.** *The composition of a P-Cartesian-closed functor after a P-Cartesian-pre-closed functor is a P-Cartesian-pre-closed functor.*

Example 4.4.37. For any P-Cartesian category, $\mathbb{C}$, any P-Cartesian-pre-closed category, $\mathbb{D}$, and any dominant P-Cartesian functor, $F \,:\, \mathbb{C} \to \mathbb{D}$, the nerve of F, $\langle F \rangle \,:\, \mathbb{D} \to \widehat{\mathbb{C}}$, is a P-Cartesian-pre-closed functor. As a corollary, the Yoneda P-Cartesian functor is P-Cartesian-pre-closed as it is the nerve of the identity functor, which is dominant and P-Cartesian. The proof of this is similar to the proof of example 4.4.29; as P-Cartesian-pre-closure of P-functors has weaker conditions we can drop the fullness requirement.

4.4.8 *Equalisers*

Since we have already defined P-terminal objects, and P-Cartesian products, defining P-equalisers allows us to define finitely-complete P-categories as those categories that have all three structures. Moreover, we use equalisers (and Cartesian products) to define pullbacks in the next subsection.

```
Record PCartPreClosFun (C : PCartPreClosCat) (D : PCartClosCat) := {
  PCartPreClosFun_fun :> PCartFun C D;

  PCartPreClosFun_exp_of : forall a b, PCat_hom
    (PCCC_exp (PCartPreClosFun_fun b) (PCartPreClosFun_fun a))
    (PCartPreClosFun_fun (PCPCC_exp b a));

  PCartPreClosFun_exp_of_rel : forall a b,
    PCartPreClosFun_exp_of a b ~ PCartPreClosFun_exp_of a b;

  PCartPreClosFun_exp_of_transp : forall a b c (f f' : PCat_hom (PCC_prod c b) a),
    f ~ f' ->
      PCat_comp
        (PCartPreClosFun_exp_of b a)
        (PCCC_transp
          (PCat_comp (PFun_hom_of PCartPreClosFun_fun f) (PCartFun_prod_of _ _ _ _ _))
        )
    ~
      PFun_hom_of PCartPreClosFun_fun (PCPCC_transp f');
}.
```

Listing 4.15: P-Cartesian-Pre-Closed Functor Rocq Definition

Definition 4.4.38 (P-Equalisers). A P-category, $\mathbb{C}$, has *P-equalisers* when it has an ordinary type-theoretic function:

$$\mathrm{eq}_{a,b}\{-,=\} : \prod_{f\ g : \mathbb{C}_1(a,b)} f \sim f \to g \sim g \to \mathbb{C}_0$$

such that for all pairs of P-well-defined morphisms, $f,\ g\ :\ a \to b$, there exists a P-natural isomorphism as follows:

$$\mathrm{Hom}_{\mathbb{C}}(-, \mathrm{eq}\{f,g\}) \cong \{h\ :\ \mathrm{Hom}_{\mathbb{C}}(-,a) \mid f \circ h \sim g \circ h\}$$

The P-functor defined on the right is a sub-P-presheaf of the P-presheaf $\yo\,(a)$ whose family of subset predicates is bistrong. We therefore use the machinery of construction 4.3.11 for defining sub-P-presheaves to define this P-functor. This definition is rendered in Rocq in listing 4.16.

```
Definition IsPEqualizer {C : PCat}
  (eq : forall (a b : C) (f g : PCat_hom a b), f ~ f -> g ~ g -> C) :=
    forall (a b : C) (f g : PCat_hom a b) (X1 : f ~ f) (X2 : g ~ g), PNatIso
      (PStrYoFun (eq a b f g X1 X2))
      (PBiStrongSubSetFun
        (PStrYoFun a)
        (fun (c : C) (h h' : PCat_hom c a) => PCat_comp f h ~ PCat_comp g h')
        ( ... ) (* Proof that the subset predicate is bistrong. *)
        ( ... ) (* Proof that the subset predicate respects the functorial action. *)
      ).
```

Listing 4.16: P-Equalisers Rocq Definition

Construction 4.4.39 (P-Equaliser Equations). From this definition of P-equalisers, we are able to induce the following structure for P-equalisers:

- P-well-defined inclusion morphism, $i\ :\ \mathrm{eq}\{f,g\} \to a$; and

- a mapping, $\{-\} : \mathrm{Hom}_{\mathbb{C}}(c, a) \to \mathrm{Hom}_{\mathbb{C}}(c, \mathrm{eq}\{f, g\})$, whose output is P-well-defined only when the P-well-defined input equalises f and g, *i.e.*, $f \circ h \sim g \circ h' \wedge h \sim h' \Rightarrow \{h\} \sim \{h'\}$;

such that the following properties hold:

- $f \circ i \sim g \circ i$;
- $f \circ h \sim g \circ h' \wedge h \sim h' \Rightarrow i \circ \{h\} \sim h'$; and
- $k \sim k' \Rightarrow \{i \circ k\} \sim k'$.

From these one can derive the following convenient property.

$$f \circ h \sim g \circ h' \wedge h \sim h' \wedge k \sim k' \Rightarrow \{h \circ k\} \sim \{h'\} \circ k'$$

Construction 4.4.40 (P-Equaliser P-Functor)**.** The existence of the P-natural isomorphism extends the type-theoretic equalising function into a P-functor.

$$\mathrm{Eq} : (\langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle \downarrow \langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle) \to \mathbb{C}$$

Remark 4.4.41. The presence of the P-well-definition requirement of f and g as inputs to eq is due to its being the object mapping of a P-functor from a P-comma category, whose objects contain P-well-defined morphisms.

4.4.9 *Lex P-Categories*

Within the Rocq formalism it is useful to be able to refer both to P-categories that are finitely P-complete, and to P-categories that are both finitely P-complete and P-Cartesian-closed. We refer to such P-categories as P-lex and P-lex-closed respectively.

Definition 4.4.42 (P-Lex Category)**.** A *P-lex category* is a P-Cartesian category that has equalisers. This definition is rendered in Rocq in listing 4.17.

```
Record PLexCat := Build_PLexCat {
  PLexCat_cat :> PCartCat;

  PLC_eq : forall {x y : PLexCat_cat} (f g : PCat_hom x y),
    f ~ f -> g ~ g -> PLexCat_cat;

  PLC_eq_prop : IsPEqualizer (@PLC_eq);
}.
```

Listing 4.17: P-Lex Category Rocq Definition

Example 4.4.43. The P-Cartesian category of P-sets is P-lex with equalisers given by appropriate sub-P-sets.

Example 4.4.44. P-Functor categories are P-lex whenever the codomain P-category is P-lex. P-Equalisers are defined objectwise as the P-equaliser in the codomain P-category.

Definition 4.4.45 (P-Lex-Closed Category)**.** A *P-lex-closed category* is a P-Cartesian category that has equalisers and P-Cartesian exponentials. This definition is rendered in Rocq in listing 4.18.

```
Record PLexClosCat := Build_PLexClosCat {
  PLexClosCat_cat :> PCartCat;

  PLCC_eq : forall {x y : PLexCat_cat} (f g : PCat_hom x y),
    f ~ f -> g ~ g -> PLexClosCat_cat;
  PLCC_eq_prop : IsPEqualizer PLCC_eq;

  PLCC_exp : PLexClosCat_cat -> PLexClosCat_cat -> PLexClosCat_cat;
  PLCC_exp_prop : IsPCartExp PLCC_exp;
}.
```

Listing 4.18: P-Lex-Closed Category Rocq Definition

4.4.10 *Pullbacks*

We now proceed to define pullbacks for P-categories. The definition is much simpler than all the P-categorical structures defined hitherto, as it is simply an appropriate combination of P-Cartesian products with P-equalisers.

Construction 4.4.46 (P-Pullback P-Functor)**.** Any P-lex category has P-pullbacks defined by a P-functor:

$$\mathrm{Pb} \ : \ (\mathrm{Id}_{\mathbb{C}\times\mathbb{C}} \downarrow \langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle) \to \mathbb{C}$$

This can be defined by precomposing the P-equaliser P-functor, Eq, with a P-functor

$$\langle (- \times =) \circ \mathrm{Dom} \downarrow \mathrm{Cod} \rangle \ : \ (\mathrm{Id}_{\mathbb{C}\times\mathbb{C}} \downarrow \langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle) \to (\langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle \downarrow \langle \mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}} \rangle)$$

defined by the P-Cartesian product on the domain objects, and the identity on the codomain objects of the P-comma categories.

Example 4.4.47. P-Comma categories, $(\mathrm{Id}_{\mathbb{D}} \downarrow F)$ where $F \ : \ \mathbb{C} \to \mathbb{D}$, are P-Cartesian-closed whenever $\mathbb{C}$ is a P-Cartesian-closed category, $\mathbb{D}$ is a P-lex-closed category, and F is a P-Cartesian functor.

Example 4.4.48. The codomain projection P-Cartesian functor is a P-Cartesian-closed functor.

Example 4.4.49. The P-Cartesian functor, $\langle H \ \downarrow_\alpha \ K \rangle$, from $\mathbb{B}$ to $(\mathrm{Id}_{\mathbb{D}} \downarrow F)$ where $F \ : \ \mathbb{C} \to \mathbb{D}$, induced by $H \ : \ \mathbb{B} \to \mathbb{D}$, $K \ : \ \mathbb{B} \to \mathbb{C}$, and $\alpha \ : \ H \Rightarrow F \circ K$ is P-Cartesian-pre-closed whenever both H and K are P-Cartesian-pre-closed and α lifts the P-Cartesian-pre-closure of H and K as follows.

$$\begin{pmatrix} H\, b_1 \\ \Big\downarrow \\ \alpha_{b_1} : H\, b_1 \to FK\, b_1 \\ \Big\downarrow \\ K\, b_1 \end{pmatrix} \Rightarrow \begin{pmatrix} H\, b_2 \\ \Big\downarrow \\ \alpha_{b_2} : H\, b_2 \to FK\, b_2 \\ \Big\downarrow \\ K\, b_2 \end{pmatrix} \begin{matrix} \cdots\cdots \tilde{e}_H \circ \mathrm{Dom}(\varepsilon)^* \cdots\cdots\rangle \\ \longrightarrow \\ \cdots\cdots\cdots\cdots \tilde{e}_K \cdots\cdots\cdots\cdots\rangle \end{matrix} \begin{pmatrix} H\,(b_1 \Rightarrow b_2) \\ \Big\downarrow \\ \alpha_{b_1 \Rightarrow b_2} : H\,(b_1 \Rightarrow b_2) \to FK\,(b_1 \Rightarrow b_2) \\ \Big\downarrow \\ K\,(b_1 \Rightarrow b_2) \end{pmatrix}$$

Fact 4.4.50. *For any P-Cartesian category, $\mathbb{B}$, any P-Cartesian-pre-closed category, $\mathbb{C}$, any P-Cartesian-closed category, $\mathbb{D}$, any dominant P-Cartesian functor, $G \ : \ \mathbb{B} \to \mathbb{C}$, and any P-Cartesian-pre-closed functor, $H \ : \ \mathbb{C} \to \mathbb{D}$, the canonical P-natural transformation mapping P-hom-set into P-hom-set by H_1*

$$\langle G \rangle \Rightarrow \langle H \circ G \rangle \circ H$$

satisfies the P-Cartesian-pre-closure-lifting condition in example 4.4.49. The proof of this fact uses the almost-inverse property of $\tilde{e}$ for P-Cartesian-pre-closed functors.

CHAPTER 5

P-BICATEGORY THEORY

In this chapter, we motivate and introduce P-bicategory theory. Thereafter, we proceed to define many of the key concepts of P-bicategory theory familiar from ordinary bicategory theory: P-bicategories, P-pseudofunctors, P-pseudonatural transformations, &c. These are, of course, largely similar to their definitions in ordinary bicategory theory. After providing the definitions of key concepts, we develop standard bicategory-theoretic results in the P-setting.

Remark 5.0.1. Within this chapter we gloss over issues arising from size, such as largeness of a bicategory, local-smallness of a bicategory, or local-local-smallness of a bicategory. Such issues are given due attention in the ROCQ formalism.

Remark 5.0.2. We use the descriptor "pseudo" to refer to a bicategorical construction where commutation of 1-morphism diagrams is up to coherent invertible 2-morphisms, *i.e.*, coherent P-iso-2-morphisms. This is intended to be in opposition to "lax", where commutation of 1-morphism diagrams would be up to coherent not-necessarily-invertible 2-morphisms.

5.1 P-BICATEGORICAL DEFINITIONS

In this section, we provide definitions for P-bicategorical structures. Thereafter, we analyse the structure of our definitions in subsection 5.1.6. From this analysis, we argue for the correctness of our definitions within the P-setting of P-category theory and P-bicategory theory.

5.1.1 *P-Bicategories*

Unlike categories, bicategories have a much wider range of presentations. Moreover, there is the added complexity of strictness of two-dimensional categorical structure resulting in 2-categories as well as bicategories. The definition of a P-bicategory looks similar to definitions of bicategories in standard Mathematical settings, *e.g.*, in Bénabou (1967). Interestingly, the P-setting does not require too many differences from more standard settings, in contrast with the differences required for P-categories, as the P-setting is largely abstracted away in how the definition is structured.

Definition 5.1.1 (P-Bicategory). A *P-bicategory*, $\mathscr{B}$, is defined by the following stuff and structure:

- a type of objects, $\mathscr{B}_0$;
- a family, indexed over two objects, of P-categories as the homs, $\mathscr{B}_1 \ : \ \mathscr{B}_0 \to \mathscr{B}_0 \to \mathsf{PCat}$;
- a family, indexed over three objects, of composition P-bifunctors, $- \bullet = : \ (\mathscr{B}_1(y,z);\ \mathscr{B}_1(x,y)) \to \mathscr{B}_1(x,z)$;
- a family of identity 1-morphisms, $\mathrm{id}_x \ : \ \mathscr{B}_1(x,x)$;
- a composition associator P-trinatural isomorphism, $\mathbf{assoc} \ : \ ((- \bullet =) \bullet \equiv) \xRightarrow{\sim} (- \bullet (= \bullet \equiv))$;
- a composition left unitor P-natural isomorphism, $\lambda \ : \ (\mathrm{id} \bullet -) \xRightarrow{\sim} \mathrm{Id}$; and

- a composition right unitor P-natural isomorphism, $\rho : (- \bullet \mathrm{id}) \xRightarrow{\simeq} \mathrm{Id}$;

such that the following properties hold:

- the associator P-trinatural transformation satisfies the pentagon law; and
- the associator, left unitor, and right unitor satisfy the triangle law.

This definition is rendered in Rocq in listing 5.1.

Remark 5.1.2. By abuse of notation, we often omit the subscripted zero and one when referring to the type of objects, and to the family of hom P-categories of a P-bicategory.

From the definition of a P-bicategory standard basic constructions follow.

Construction 5.1.3 (Opposite P-Bicategory). The construction of opposite P-bicategories follows the standard construction. We denote the opposite of a P-bicategory, $\mathscr{B}$, by $\mathscr{B}^{\mathrm{op}}$.

Construction 5.1.4 (Co P-Bicategory). The construction of co P-bicategories follows the standard construction. We denote the co of a P-bicategory, $\mathscr{B}$, by $\mathscr{B}^{\mathrm{co}}$.

Construction 5.1.5 (Product P-Bicategory). The construction of product P-bicategories follows the standard construction. We denote the product of P-bicategories, $\mathscr{B}$ and $\mathscr{C}$, by $\mathscr{B} \times \mathscr{C}$.

Example 5.1.6 (P-Bicategory of P-Categories). There is a P-bicategory of P-categories, which we denote by **PCat**. It is defined by the following stuff and structure:

- P-categories for the objects;
- P-functor categories for the hom P-categories;
- composition of P-functors for the composition; and
- identity P-functors for the identities.

We develop further the P-bicategorical structure of **PCat** in section 5.5.

Example 5.1.7 (P-Bicategory of P-Cartesian Categories). There is a P-bicategory of P-Cartesian categories, which we denote by **PCartCat**. It is defined by the following stuff and structure:

- P-Cartesian categories for the objects;
- P-Cartesian functor categories for the hom P-categories;
- composition of P-Cartesian functors for the composition; and
- identity P-Cartesian functors for the identities.

Example 5.1.8 (P-Bicategory of P-Cartesian-Closed Categories). There is a locally groupoidal P-bicategory of P-Cartesian-closed categories, which we denote by **PCartClosCat**. It is defined by the following stuff and structure:

- P-Cartesian-closed categories for the objects;
- P-Cartesian-closed functor groupoids for the hom P-categories;
- composition of P-Cartesian-closed functors for the composition; and
- identity P-Cartesian-closed functors for the identities.

```
Cumulative Record PBiCat@{+i +j +k} := {
  PBiCat_obj :> Type@{i};
  PBiCat_hom : PBiCat_obj -> PBiCat_obj -> PCat@{j k};
  PBiCat_id : forall x, PBiCat_hom x x;
  PBiCat_comp : forall {x y z},
    PBiFun (PBiCat_hom y z) (PBiCat_hom x y) (PBiCat_hom x z);

  PBiCat_associator : forall {w x y z},
    PTriNatIso
      (PBiFunCompLeftAssoc (@PBiCat_comp w x z) (@PBiCat_comp x y z))
      (PBiFunCompRightAssoc (@PBiCat_comp w y z) (@PBiCat_comp w x y));

  PBiCat_left_unitor : forall {x y},
    PNatIso (PBiFunIndLeft (@PBiCat_comp x y y) (PBiCat_id y)) PIdFun;
  PBiCat_right_unitor : forall {x y},
    PNatIso (PBiFunIndRight (@PBiCat_comp x x y) (PBiCat_id x)) PIdFun;

  PBiCat_triangle : forall [x y z] (f : PBiCat_hom y z) (g : PBiCat_hom x y),
      (fst (PTriNatIso_comps_of PBiCat_associator f (PBiCat_id y) g))
    ~
      PCat_comp
        (PBiFun_rhom_of PBiCat_comp (snd (PNatIso_comps_of PBiCat_left_unitor _)))
        (PBiFun_lhom_of PBiCat_comp (fst (PNatIso_comps_of PBiCat_right_unitor _)));

  PBiCat_pentagon : forall [v w x y z]
    (f : PBiCat_hom y z) (g : PBiCat_hom x y)
    (h : PBiCat_hom w x) (k : PBiCat_hom v w),
      (PCat_comp
        (fst (PTriNatIso_comps_of PBiCat_associator f g _))
        (fst (PTriNatIso_comps_of PBiCat_associator _ h k))
      )
    ~
      (PCat_comp
        (PCat_comp
          (PBiFun_rhom_of PBiCat_comp (fst (PTriNatIso_comps_of PBiCat_associator g h
          ↪  k)))
          (fst (PTriNatIso_comps_of PBiCat_associator f _ k))
        )
        (PBiFun_lhom_of PBiCat_comp (fst (PTriNatIso_comps_of PBiCat_associator f g
        ↪  h)))
      );
}.
```

Listing 5.1: p-Bicategory Rocq Definition

```
Cumulative Record IsPAdjEqv@{*i *j *k} {B : PBiCat@{i j k}} {x y : B}
  (f : PBiCat_hom x y) := {
    IsPAdjEqv_adj : PBiCat_hom y x;
    IsPAdjEqv_unit : PCat_hom (PBiCat_id _) (PBiCat_comp IsPAdjEqv_adj f);
    IsPAdjEqv_unit_iso : IsPIso IsPAdjEqv_unit;
    IsPAdjEqv_counit : PCat_hom (PBiCat_comp f IsPAdjEqv_adj) (PBiCat_id _);
    IsPAdjEqv_counit_iso : IsPIso IsPAdjEqv_counit;
    IsPAdjEqv_triangle_left :
        PCat_comp
          (PCat_comp
            (fst (PNatIso_comps_of PBiCat_right_unitor _))
            (PBiFun_rhom_of PBiCat_comp IsPAdjEqv_counit)
          )
          (PCat_comp
            (fst (PTriNatIso_comps_of PBiCat_associator _ _ _))
            (PBiFun_lhom_of PBiCat_comp IsPAdjEqv_unit)
          )
      ~
        fst (PNatIso_comps_of PBiCat_left_unitor IsPAdjEqv_adj);
    IsPAdjEqv_triangle_right :
        PCat_comp
          (PCat_comp
            (fst (PNatIso_comps_of PBiCat_left_unitor _))
            (PBiFun_lhom_of PBiCat_comp IsPAdjEqv_counit)
          )
          (PCat_comp
            (snd (PTriNatIso_comps_of PBiCat_associator _ _ _))
            (PBiFun_rhom_of PBiCat_comp IsPAdjEqv_unit)
          )
      ~
        fst (PNatIso_comps_of PBiCat_right_unitor f);
}.
```

Listing 5.2: P-Adjoint Equivalence Rocq Definition

Definition 5.1.9 (P-Adjoint Equivalence)**.** The 1-morphisms, $f : x \to y$ and $f^\dagger : y \to x$, with the 2-morphisms, $\eta : \mathrm{id} \Rightarrow f^\dagger \bullet f, \eta^{-1} : f^\dagger \bullet f \Rightarrow \mathrm{id}, \epsilon : f \bullet f^\dagger \Rightarrow \mathrm{id}$, and $\epsilon^{-1} : \mathrm{id} \Rightarrow f \bullet f^\dagger$, in some P-bicategory exhibit a *P-adjoint equivalence* between x and y precisely when the following properties hold:

- η and η^{-1} exhibit a P-isomorphism;
- ϵ and ϵ^{-1} exhibit a P-isomorphism;
- $\rho_{f^\dagger} \circ (\mathrm{id} \bullet \epsilon) \circ \mathbf{assoc}_{f^\dagger; f; f^\dagger} \circ (\eta \bullet \mathrm{id}) \sim \lambda_{f^\dagger}$; and
- $\lambda_f \circ (\epsilon \bullet \mathrm{id}) \circ \mathbf{assoc}^{-1}_{f; f^\dagger; f} \circ (\mathrm{id} \bullet \eta) \sim \rho_f$.

We say that a 1-morphism, $f : x \to y$, is a *P-adjoint equivalence* if there exists putative $f^\dagger$, η, η^{-1}, ϵ, and ϵ^{-1} such that the above properties hold. This definition is rendered in Rocq in listing 5.2.

5.1.2 *P-Pseudofunctors*

Much like ordinary bicategories, P-bicategories allow for a number of different notions of "functor" therebetwixt, although we only define and formalise the notion of P-pseudofunctor. Moreover, we define one concrete instance of multipseudofunctors: binary P-pseudofunctors that have separate, but commuting, pseudofunctorial actions in each of

their arguments independently. Such multipseudofunctors allow for stricter specification of left and right actions of binary P-pseudofunctors than that afforded by P-pseudofunctors out of product P-bicategories.

Definition 5.1.10 (P-Pseudofunctor). A *P-pseudofunctor*, F, from $\mathscr{B}$ to $\mathscr{C}$, which we denote by $\mathscr{B} \to_{\text{ps}} \mathscr{C}$, is defined by the following:

- a mapping of objects, $F_0 \ : \ \mathscr{B}_0 \to \mathscr{C}_0$;
- a family, indexed over two objects, of P-functors between hom P-categories, $F_1 \ : \ \mathscr{B}_1(x, y) \to \mathscr{C}_1(F_0\, x, F_0\, y)$;
- a family, indexed over three objects, of composition-preservation P-binatural isomorphisms,

$$F_2 \ : \ F_1 \circ (- \bullet_{\mathscr{B}} =) \mathrel{\tilde{\Rightarrow}} (- \bullet_{\mathscr{C}} =) \circ (F_1;\ F_1) \ ;$$

- a family of identity-preservation P-isomorphisms, $F_{\text{id}} \ : \ F_1(\text{id}_x) \mathrel{\tilde{\Rightarrow}} \text{id}_{F_0\, x}$;

such that the following properties hold:

- the composition preservation and identity preservation are coherent with the left unitors;
- the composition preservation and identity preservation are coherent with the right unitors; and
- the composition preservation is coherent with the associators.

This definition is rendered in Rocq in listing 5.3.

Remark 5.1.11. By abuse of notation, we often omit the subscripted zero and one when referring to the object mapping, and to the family of morphism maps of a P-pseudofunctor.

Definition 5.1.12 (P-Pseudobifunctor). A *P-pseudobifunctor*, F, from $\mathscr{B}$ and $\mathscr{C}$ to $\mathscr{D}$, which we denote by $(\mathscr{B};\ \mathscr{C}) \to_{\text{ps}} \mathscr{D}$, is defined by the following structure:

- a mapping of objects, $F_0 : \mathscr{B}_0 \to \mathscr{C}_0 \to \mathscr{D}_0$;
- two families, each indexed over three objects, of P-functors between hom P-categories:[1]
  - $F_{1l} : \mathscr{B}_1(x, y) \to \mathscr{D}_1(F_0\, x\, c, F_0\, y\, c)$; and
  - $F_{1r} : \mathscr{C}_1(x, y) \to \mathscr{D}_1(F_0\, b\, x, F_0\, b\, y)$;
- two families, each indexed over four objects, of composition-preservation P-binatural isomorphisms:
  - $F_{2l} \ : \ F_{1l} \circ (- \bullet_{\mathscr{B}} =) \mathrel{\tilde{\Rightarrow}} (- \bullet_{\mathscr{C}} =) \circ (F_{1l};\ F_{1l})$; and
  - $F_{2r} \ : \ F_{1r} \circ (- \bullet_{\mathscr{B}} =) \mathrel{\tilde{\Rightarrow}} (- \bullet_{\mathscr{C}} =) \circ (F_{1r};\ F_{1r})$;
- two families, each indexed over two objects, of identity-preservation P-isomorphisms:
  - $F_{\text{id}l} \ : \ F_{1l}(\text{id}_x) \mathrel{\tilde{\to}} \text{id}_{F_0\, x\, y}$; and
  - $F_{\text{id}r} \ : \ F_{1r}(\text{id}_y) \mathrel{\tilde{\to}} \text{id}_{F_0\, x\, y}$; and
- a family, indexed over four objects, of P-functorial action commutation P-binatural isomorphisms,

$$F_{lr} \ : \ (- \bullet_{\mathscr{D}} =) \circ (F_{1l};\ F_{1r}) \mathrel{\tilde{\Rightarrow}} (= \bullet_{\mathscr{D}} -) \circ (F_{1l};\ F_{1r}) \ ;$$

such that the following properties hold:

[1] The subscripts come from 'left' and 'right'.

```
Cumulative Record PPsdFun@{*i1 *j1 *k1 *i2 *j2 *k2}
  (B : PBiCat@{i1 j1 k1}) (C : PBiCat@{i2 j2 k2}) := {
    PPsdFun_obj_of :> B -> C;

    PPsdFun_hom_of : forall {x y},
      PFun (PBiCat_hom x y) (PBiCat_hom (PPsdFun_obj_of x) (PPsdFun_obj_of y));

    PPsdFun_comp_of : forall {x y z},
      PBiNatIso
        (PBiFunPostComp (@PPsdFun_hom_of x z) PBiCat_comp)
        (PBiFunPreComp PBiCat_comp C (@PPsdFun_hom_of y z) (@PPsdFun_hom_of x y));

    PPsdFun_id_of : forall x,
      PCat_hom (PPsdFun_hom_of (PBiCat_id x)) (PBiCat_id (PPsdFun_obj_of x));
    PPsdFun_id_of_iso : forall x, IsPIso (PPsdFun_id_of x);

    PPsdFun_left_unitor_of : forall {x y : B} (f : PBiCat_hom x y),
        PCat_comp
          (snd (PNatIso_comps_of PBiCat_left_unitor _))
          (PFun_hom_of PPsdFun_hom_of (fst (PNatIso_comps_of PBiCat_left_unitor f)))
      ~
        PCat_comp
          (PBiFun_lhom_of PBiCat_comp (PPsdFun_id_of y))
          (fst (PBiNatIso_comps_of PPsdFun_comp_of _ f));

    PPsdFun_right_unitor_of : forall {x y : B} (f : PBiCat_hom x y),
        PCat_comp
          (snd (PNatIso_comps_of PBiCat_right_unitor _))
          (PFun_hom_of PPsdFun_hom_of (fst (PNatIso_comps_of PBiCat_right_unitor f)))
      ~
        PCat_comp
          (PBiFun_rhom_of PBiCat_comp (PPsdFun_id_of x))
          (fst (PBiNatIso_comps_of PPsdFun_comp_of f _));

    PPsdFun_associator_of : forall {w x y z}
      (f : PBiCat_hom y z) (g : PBiCat_hom x y) (h : PBiCat_hom w x),
          PCat_comp
            (PCat_comp
              (fst (PTriNatIso_comps_of PBiCat_associator _ _ _))
              (PBiFun_lhom_of PBiCat_comp (fst (PBiNatIso_comps_of PPsdFun_comp_of f
              ↪  g)))
            )
            (fst (PBiNatIso_comps_of PPsdFun_comp_of _ h))
        ~
          PCat_comp
            (PCat_comp
              (PBiFun_rhom_of PBiCat_comp (fst (PBiNatIso_comps_of PPsdFun_comp_of g
              ↪  h)))
              (fst (PBiNatIso_comps_of PPsdFun_comp_of f _))
            )
            (PFun_hom_of PPsdFun_hom_of (fst (PTriNatIso_comps_of PBiCat_associator f
            ↪  g h)));
}.
```

Listing 5.3: p-Pseudofunctor Rocq Definition

- the composition preservations and identity preservations are coherent with the left unitors;
- the composition preservations and identity preservations are coherent with the right unitors;
- the composition preservations are coherent with the associators;
- the action commutation is coherent with the composition preservations; and
- the action commutation is coherent with the identity preservations.

Construction 5.1.13 (Induced P-Pseudofunctors)**.** Given a P-pseudobifunctor and an object of one of the domain P-bicategories we may induce an ordinary P-pseudofunctor. That is, we may induce the following P-pseudofunctors from the P-pseudobifunctor, $F : (\mathscr{B}; \mathscr{C}) \to_{\text{ps}} \mathscr{D}$.

- $F(b, -) : \mathscr{C} \to \mathscr{D}$
- $F(-, c) : \mathscr{B} \to \mathscr{D}$

Construction 5.1.14 (P-Pseudofunctor Pairing)**.** A pair of P-pseudofunctors, $F : \mathscr{B} \to \mathscr{C}$ and $G : \mathscr{B} \to \mathscr{D}$, induce a P-pseudofunctor, $\langle F, G \rangle : \mathscr{B} \to \mathscr{C} \times \mathscr{D}$.

Example 5.1.15 (Hom P-Pseudobifunctor)**.** For each P-bicategory, $\mathscr{B}$, we have a P-pseudobifunctor:

$$\text{Hom}_{\mathscr{B}} : (\mathscr{B}^{\text{op}}; \mathscr{B}) \to \textbf{PCat}$$

defined by the following mappings:

$$\begin{aligned} \text{Hom}_0(x, y) &\triangleq \mathscr{B}(x, y); \\ \text{Hom}_{1l}(f) &\triangleq (g \mapsto g \bullet f); \text{and} \\ \text{Hom}_{1r}(f) &\triangleq (g \mapsto f \bullet g). \end{aligned}$$

Remark 5.1.16. Advantages of defining Hom as a P-pseudobifunctor — rather than as a P-pseudofunctor from the product P-bicategory — include that it is more convenient to have access to separate P-functorial actions, and that we are not forced to pick an association for the triple composition of the functorial action.

Example 5.1.17 (P-Pseudofunctors between P-Bicategories of P-Categories)**.** There are P-pseudofunctors:

- **PCartCat** $\to_{\text{ps}}$ **PCat**; and
- **PCartClosCat** $\to_{\text{ps}}$ **PCartCat**.

5.1.3 *P-Pseudonatural Transformations*

Definition 5.1.18 (P-Pseudonatural Transformation)**.** A *P-pseudonatural transformation*, α, from the P-pseudofunctor $F : \mathscr{B} \to_{\text{ps}} \mathscr{C}$ to the P-pseudofunctor $G : \mathscr{B} \to_{\text{ps}} \mathscr{C}$, which we denote by $\alpha : F \Rightarrow_{\text{ps}} G$, is given by the following structure:

- a family of 1-morphisms, $\alpha_x : F\,x \to G\,x$, which we call the components;
- a natural family of P-iso-2-morphisms, $\alpha_{x;y} : (\alpha_y \bullet -) \circ F_1 \mathrel{\tilde{\Rightarrow}} (- \bullet \alpha_x) \circ G_1$

such that the following properties hold:

- the components are coherent with the composition preservation of F and G; and

```
Cumulative Record PPsdNatTrans@{*i1 *j1 *k1 *i2 *j2 *k2}
  {B : PBiCat@{i1 j1 k1}} {C : PBiCat@{i2 j2 k2}} (F G : PPsdFun B C) := {
    PPsdNatTrans_comps_of :> forall b, PBiCat_hom (F b) (G b);
    PPsdNatTrans_commutes : forall {x y},
      PNatIso
        (PCompFun
          (PBiFunIndLeft PBiCat_comp (PPsdNatTrans_comps_of y))
          (PPsdFun_hom_of F)
        )
        (PCompFun
          (PBiFunIndRight PBiCat_comp (PPsdNatTrans_comps_of x))
          (PPsdFun_hom_of G)
        );

    PPsdNatTrans_comp_of : forall {x y z} (f : PBiCat_hom y z) (g : PBiCat_hom x y),
        PCat_comp
          (PCat_comp
            (PBiFun_lhom_of PBiCat_comp (fst (PBiNatIso_comps_of (PPsdFun_comp_of G) f
            ↪  g)))
            (PPsdNatTrans_commutes _)
          )
          (PBiFun_rhom_of PBiCat_comp (snd (PBiNatIso_comps_of (PPsdFun_comp_of F) f
          ↪  g)))
      ~
        PCat_comp
          (PCat_comp
            (PCat_comp
              (snd (PTriNatIso_comps_of PBiCat_associator _ _ _))
              (PBiFun_rhom_of PBiCat_comp (PPsdNatTrans_commutes _))
            )
            (fst (PTriNatIso_comps_of PBiCat_associator _ _ _))
          )
          (PCat_comp
            (PBiFun_lhom_of PBiCat_comp (PPsdNatTrans_commutes _))
            (snd (PTriNatIso_comps_of PBiCat_associator _ _ _))
          );

    PPsdNatTrans_id_of : forall x,
        PPsdNatTrans_commutes (PBiCat_id x)
      ~
        PCat_comp
          (PCat_comp
            (PBiFun_lhom_of PBiCat_comp (PPsdFun_id_of_inv G x))
            (snd (PNatIso_comps_of PBiCat_left_unitor _))
          )
          (PCat_comp
            (fst (PNatIso_comps_of PBiCat_right_unitor _))
            (PBiFun_rhom_of PBiCat_comp (PPsdFun_id_of F x))
          );
}.
```

Listing 5.4: p-Pseudonatural Transformation Rocq Definition

- the components are coherent with the identity preservation of F and G.

This definition is rendered in Rocq in listing 5.4.

Definition 5.1.19 (p-Pseudonatural Adjoint Equivalence). A *p-pseudonatural adjoint equivalence* is simply a p-pseudonatural transformation whose components are p-adjoint equivalences. For simplicity, it is convenient to define this as a single concept, rather than as the intersection of two concepts.

5.1.4 *p-Modifications*

The two-dimensional structure of p-bicategory theory allows for one further level of structure compared with p-categories. This final level of structure is given by p-modifications.

Definition 5.1.20 (p-Modification). A *p-modification*, $\mathfrak{a}$, from the p-pseudonatural transformation, $\alpha : F \Rightarrow_{\text{ps}} G$, to the p-pseudonatural transformation, $\beta : F \Rightarrow_{\text{ps}} G$, (where α and β are from the p-pseudofunctor $F : \mathscr{B} \to_{\text{ps}} \mathscr{C}$ to the p-pseudofunctor $G : \mathscr{B} \to_{\text{ps}} \mathscr{C}$,) which we denote by $\mathfrak{a} : \alpha \Rrightarrow_{\text{ps}} \beta$, is given by the following structure:

- a family of 2-morphisms, $\mathfrak{a}_x : \alpha_x \Rightarrow \beta_x$, called the components;

such that the following properties hold:

- the components are self-related, $\mathfrak{a}_x \sim \mathfrak{a}_x$; and
- the components commute with the components of α and β.

For the sake of clarity we give the commutation condition explicitly:

$$\forall_{x\ y : \mathscr{B}_0} \forall_{f : x \to y} \beta_f \circ (\mathfrak{a}_y \bullet \mathrm{id}) \sim (\mathrm{id} \bullet \mathfrak{a}_x) \circ \alpha_f$$

This definition is rendered in Rocq in listing 5.5.

```
Cumulative Record PModif@{*i1 *j1 *k1 *i2 *j2 *k2}
  {B : PBiCat@{i1 j1 k1}} {C : PBiCat@{i2 j2 k2}} {F G : PPsdFun B C}
  (alpha beta : PPsdNatTrans F G) := {
    PModif_comps_of :> forall b,
      PCat_hom (PPsdNatTrans_comps_of F G alpha b) (PPsdNatTrans_comps_of F G beta b);

    PModif_comps_rel : forall b, PModif_comps_of b ~ PModif_comps_of b;

    PModif_commutes : forall {x y} (f : PBiCat_hom x y),
        PCat_comp
          (PPsdNatTrans_commutes F G beta f)
          (PBiFun_lhom_of PBiCat_comp (PModif_comps_of _))
      ~
        PCat_comp
          (PBiFun_rhom_of PBiCat_comp (PModif_comps_of _))
          (PPsdNatTrans_commutes F G alpha f);
}.
```

Listing 5.5: p-Modification Rocq Definition

5.1.5 *Drawbacks*

As remarked in subsection 4.2.6, one of the main advantages of P-sets is their ability to model subsets without using Σ-types, and thereby they avoid mixing computational data with logical properties. We argued there that this was well-manifested for the hom P-sets of P-functor categories, but not for the objects thereof. This problem cascades into P-bicategory theory, and arguably is magnified, as there are now more levels of structure in which this drawback is exemplified. For example, P-pseudofunctor bicategories have Σ-types for both their objects as well as their 1-morphisms, *i.e.*, the objects of their hom P-categories. A more fully P-theoretic definition of categories and functors, would provide insight into a general solution that would allow for fully P-theoretic bicategories.

5.1.6 *Structure of Definitions*

In this subsection, we examine the shape of the definition of a P-bicategory more carefully. Similar analysis could be applied to the definitions of P-pseudofunctors, of P-pseudonatural transformations, and of P-modifications. This analysis follows on from the analysis of the definition of P-category in subsection 4.2.8, and is similarly tedious and hair-splitting; however, it allows the evaluation of the correctness of definitions for any further P-bicategory-theoretic structure.

The shape of the definition of a P-bicategory — as is the case for the definitions of bicategories in ordinary bicategory theory — is tripartite:

- stuff;
- structure; and
- properties.

Where 'stuff', 'structure', and 'properties' have the same meaning as in subsection 4.2.8. This is hardly surprising: if this analysis applied to P-categories then it ought to apply equally well to P-bicategories. We make this explicit: a P-bicategory has the following stuff:

- a type of objects;
- a doubly-indexed family of types of 1-morphisms;
- a quadruply-indexed family of types of 2-morphisms; and
- an endo-relation for each 2-morphism type.

This stuff has the following structure:

- a composition mapping for 1-morphisms;
- identity 1-morphisms;
- a composition mapping for 2-morphisms;
- identity 2-morphisms;
- a left whiskering of 2-morphisms by 1-morphisms;
- a right whiskering of 2-morphisms by 1-morphisms;
- left-to-right association 2-morphisms for 1-morphism composition;
- right-to-left association 2-morphisms for 1-morphism composition;
- left identity cancellation 2-morphisms for 1-morphism composition with the identity 1-morphism on the left;

- left identity insertion 2-morphisms for 1-morphism composition with the identity 1-morphism on the left;
- right identity cancellation 2-morphisms for 1-morphism composition with the identity 1-morphism on the right; and
- right identity insertion 2-morphisms for 1-morphism composition with the identity 1-morphism on the right.

When we analyse the properties required of the stuff and structure for them to form a P-bicategory we find that there are four parts to the properties: namely, P-well-definition properties, 0-structural properties, 1-structural properties, and 2-structural properties. The P-well-definition properties are:

- the composition mapping for 2-morphisms respecting the PER in both arguments;
- the identity 2-morphisms being self-related;
- the left whiskering respecting the PER for the 2-morphism argument;
- the right whiskering respecting the PER for the 2-morphism argument;
- the left-to-right association 2-morphisms being self-related;
- the right-to-left association 2-morphisms being self-related;
- the left identity cancellation 2-morphisms being self-related;
- the left identity insertion 2-morphisms being self-related;
- the right identity cancellation 2-morphisms being self-related; and
- the right identity insertion 2-morphisms being self-related.

All of these properties are, in essence, self-relation requirements. The only 0-structural property is:

- the relation over the 2-morphism types being symmetric and transitive, *i.e.*, is a PER.

The 1-structural properties are:

- the composition and whiskering mappings being P-functorial;
- the association 2-morphisms being P-trinatural isomorphisms;
- the left identity cancellation/insertion 2-morphisms being P-natural isomorphisms; and
- the right identity cancellation/insertion 2-morphisms being P-natural isomorphisms.

The 2-structural properties are:

- the association 2-morphisms satisfying the pentagon law; and
- the left-to-right association, left identity cancellation, and right identity cancellation 2-morphisms satisfying the triangle law.

We can therefore refine our analysis to state that P-bicategorical definitions are quadripartite:

- stuff;
- structure;
- P-well-definitions properties; and

- structural properties.

This quadripartition motivates a second tripartition:

- stuff;
- P-well-defined structure; and
- structural properties.

Where by 'well-defined structure' we mean that the structure lies in the relevant domain. Similarly to the definition of P-categories, for convenience we also bundle together the endo-relation and the 0-structural property of being a PER into a single concept of PER; this enables the further bundling together of the family of types of 2-morphisms with their PERs into a single concept of family of P-sets. This bundling together motivates further bundling together of the stuff, structure, and properties into relevant P-categorical concepts: *ee.g.*, we can bundle some of the stuff, structure, and properties together into hom P-categories, and we can bundle some of the structure and properties together into a composition P-bifunctor. We perform maximal bundling of the putative expanded definition of P-bicategory into appropriate P-categorical concepts. This leaves only the coherence laws unbundled as the 2-structural properties, much like the unbundled 1-structural properties of associativity and unitality for P-categories. We can therefore be confident that, despite the seeming lack of partiality in the definition of P-bicategory, the definition of P-bicategory is correct within the framework of partiality set up by PERs, P-sets, and P-categories as much of the partiality is bundled up in P-categorical structures contained within the P-bicategorical definitions. Although such bundling obscures the correctness of definitions, it allows using P-categorical results to manipulate the P-categorical concepts within the P-bicategorical definitions.

5.2 P-Category of P-Pseudonatural Transformations

Hitherto, the definitions and constructions have largely followed straightforwardly from ordinary bicategory theory or E-bicategory theory. When formulating the definitions of P-category theory, we encountered the first difference from ordinary category theory or from E-category theory when we defined P-functor categories. A similar situation arises in P-bicategory theory when we want to define the P-category of P-pseudonatural transformations. The category of pseudonatural transformations has pseudonatural transformations as objects, and has modifications as morphisms. Therefore, the definition of pseudonatural transformation categories in E-bicategory theory has the hom E-sets be the modifications. However, these are an instance of a sub-E-set where there is both a computational part and a logical part: *viz.*, the components and the commutation condition respectively. We can therefore use all that we learned in the case of P-functor categories to be able to construct an appropriate definition for P-pseudonatural transformation categories. The appropriate definition in P-bicategory theory keeps the computational parts and logical parts separate by incorporating the commutation condition as part of the PER for the hom P-sets, rather than including it in the underlying carrier type for the hom P-sets. First, we state a fact that we use in the definition of P-pseudonatural transformation category.

Fact 5.2.1. *The commutation condition for P-modifications is bistrong. That is, we can rephrase* $\mathfrak{a} : \prod_{x : \mathscr{B}_0} \alpha_x \to \beta_x$ *commuting as* $\mathfrak{a}$ *being on the diagonal of the following relation.*

$$\mathfrak{a}\,\mathfrak{a}' \mapsto \bigvee_{x\,y : \mathscr{B}_0} \bigvee_{f : x \to y} \beta_f \circ (\mathfrak{a}_y \bullet \mathrm{id}) \sim (\mathrm{id} \bullet \mathfrak{a}'_x) \circ \alpha_f$$

Construction 5.2.2 (P-Set of P-Modifications)**.** For this construction we fix a pair of P-pseudonatural transformations, $\alpha,\ \beta : F \Rightarrow_{\mathrm{ps}} G$. Before constructing the P-set of P-modifications, we first define the P-set of pre-modifications from α to β to be the discretely indexed dependent product of the following family of P-sets.

$$(x : \mathscr{B}_0) \mapsto \mathscr{C}_1(F_0\,x, G_0\,x)(\alpha_x, \beta_x)$$

The *p-set of p-modifications* is given by the sub-p-set of pre-modifications by the commutation bistrong relation.

Definition 5.2.3 (p-Pseudonatural Transformation Category)**.** The *p-category of p-pseudonatural transformations* from the p-pseudofunctor $F : \mathscr{B} \to_{\mathrm{ps}} \mathscr{C}$ to the p-pseudofunctor $G : \mathscr{B} \to \mathscr{C}$, which we denote by $[F, G]_{\mathrm{ps}}$, is given by the following stuff and structure:

- p-pseudonatural transformations, $F \Rightarrow_{\mathrm{ps}} G$, for the objects;
- the p-set of p-modifications for the hom p-sets;
- composition of pre-modifications for the composition; and
- identity pre-modifications for the identities.

Remark 5.2.4. A p-modification is simply a self-related morphism in the the appropriate p-pseudonatural transformation category.

5.3 p-Bicategory of p-Pseudofunctors

Having defined the p-category of p-pseudonatural transformations, we can now proceed to the definition of the p-bicategory of p-pseudofunctors.

Definition 5.3.1 (p-Pseudofunctor Bicategory)**.** The *p-bicategory of p-pseudofunctors* from the p-bicategory $\mathscr{B}$ to the p-bicategory $\mathscr{C}$, which we denote by $[\mathscr{B}, \mathscr{C}]_{\mathrm{ps}}$, is given by the following stuff and structure:

- p-pseudofunctors, $\mathscr{B} \to_{\mathrm{ps}} \mathscr{C}$, for the objects;
- the p-category of p-pseudonatural transformations for the hom p-categories;
- composition of p-pseudonatural transformations for the composition; and
- identity p-pseudonatural transformations for the identities.

Example 5.3.2 (p-Bicategory of p-Pseudopresheaves)**.** Given a p-bicategory, $\mathscr{B}$, its p-bicategory of p-pseudopresheaves is the p-pseudofunctor bicategory, $[\mathscr{B}^{\mathrm{op}}, \mathbf{PCat}]_{\mathrm{ps}}$. We also denote this p-bicategory by $\widehat{\mathscr{B}}$. We may refer to the p-bicategory $\mathscr{B}$ as the base of the p-pseudopresheaf bicategory $\widehat{\mathscr{B}}$.

Construction 5.3.3 (Nerve p-Pseudofunctors)**.** Any p-pseudofunctor, $F : \mathscr{B} \to \mathscr{C}$, induces a p-pseudofunctor, $\langle F \rangle : \mathscr{C} \to \widehat{\mathscr{B}}$ defined by the following formula.

$$\langle F \rangle(d)(c) \triangleq \mathrm{Hom}_{\mathscr{C}}(F^{\mathrm{op}}(c), d)$$

Example 5.3.4 (Yoneda p-Pseudofunctor)**.** The nerve p-pseudofunctor of the identity p-pseudofunctor is called the Yoneda p-pseudofunctor. We denote the Yoneda p-pseudofunctor by よ.

5.4 p-Pseudo-Comma Bicategories

The construction of p-comma categories extends to the p-bicategorical setting, using p-bicategories and p-pseudofunctors as the generating data for such p-pseudo-comma bicategories. We use the descriptor "pseudo" in front of "comma" to denote that the squares of 1-morphisms formed by the 1-morphisms of the p-pseudo-comma bicategories commute up to (coherent) invertible 2-morphisms.

The definition of p-pseudo-comma bicategories proceeds in a number of stages; thus, in this section, for brevity and to avoid repetition, we fix the following data:

- three P-bicategories, $\mathscr{B}, \mathscr{C}, \mathscr{D}$; and
- two P-pseudofunctors, $F : \mathscr{B} \to_{\text{ps}} \mathscr{D}$ and $G : \mathscr{C} \to_{\text{ps}} \mathscr{D}$.

We denote the resultant P-pseudo-comma bicategory by $(F \downarrow_{\text{ps}} G)$.

Definition 5.4.1 (Objects of P-Pseudo-Comma Bicategories). The *objects* of $(F \downarrow_{\text{ps}} G)$ are given by the following data:

- an object, $z_{\text{src}} : \mathscr{B}$;
- an object, $z_{\text{tgt}} : \mathscr{C}$; and
- a 1-morphism in $\mathscr{D}$, $z_{\to} : F\, z_{\text{src}} \to G\, z_{\text{tgt}}$.

When we refer to an object, z, of a P-pseudo-comma bicategory, we may refer to these three components as the source, target, and arrow component.

We denote the objects of P-pseudo-comma bicategories as follows.

$$\begin{pmatrix} z_{\text{src}} \\ \vert \\ z_{\to} : F\, z_{\text{src}} \to G\, z_{\text{tgt}} \\ \downarrow \\ z_{\text{tgt}} \end{pmatrix}$$

Definition 5.4.2 (1-Morphisms of P-Pseudo-Comma Bicategories). In this definition we fix two objects of $(F \downarrow_{\text{ps}} G)$, x and y. A 1-*morphism* from x to y is given by the following data:

- a 1-morphism in $\mathscr{B}$, $f_{\text{top}} : x_{\text{src}} \to y_{\text{src}}$;
- a 1-morphism in $\mathscr{C}$, $f_{\text{bot}} : x_{\text{tgt}} \to y_{\text{tgt}}$; and
- a P-iso-2-morphism in $\mathscr{D}$, $f_{\square} : (y_{\to} \bullet F\, f_{\text{top}}) \xRightarrow{\sim} (G\, f_{\text{bot}} \bullet x_{\to})$.

When we refer to a 1-morphism, $f : x \to y$, of a P-pseudo-comma bicategory, we may refer to these three components as the top, bottom, and commutation component.

We denote the 1-morphisms of P-pseudo-comma bicategories as follows.

$$\begin{pmatrix} x_{\text{src}} \\ \vert \\ x_{\to} : F\, x_{\text{src}} \to G\, x_{\text{tgt}} \\ \downarrow \\ x_{\text{tgt}} \end{pmatrix} \quad \begin{array}{c} \cdots\cdots f_{\text{top}} \cdots\cdots\to \\ \text{———} f_{\square} \text{———}\to \\ \cdots\cdots f_{\text{bot}} \cdots\cdots\to \end{array} \quad \begin{pmatrix} y_{\text{src}} \\ \vert \\ y_{\to} : F\, y_{\text{src}} \to G\, y_{\text{tgt}} \\ \downarrow \\ y_{\text{tgt}} \end{pmatrix}$$

Definition 5.4.3 (P-Set of 2-Morphisms of P-Pseudo-Comma Bicategories). In this definition we fix two objects of $(F \downarrow_{\text{ps}} G)$, x and y, and two 1-morphisms, $f : x \to y$ and $g : x \to y$. The *P-set of* 2*-morphisms* from f to g is the sub-P-set of pairs of 2-morphisms:

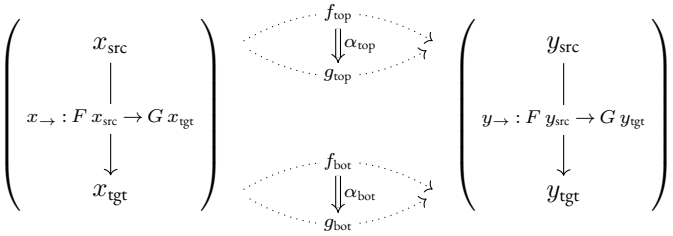
- $\alpha_{\text{top}} : f_{\text{top}} \Rightarrow g_{\text{top}}$; and
- $\alpha_{\text{bot}} : f_{\text{bot}} \Rightarrow g_{\text{bot}}$;

by the following bistrong relation.

$$\alpha\, \alpha' \mapsto g_{\square} \circ (y_{\to} \bullet F\, \alpha_{\text{top}}) \sim (G\, \alpha'_{\text{bot}} \bullet x_{\to}) \circ f_{\square}$$

When we refer to a 2-morphism, $\alpha : f \Rightarrow g$, of a P-pseudo-comma bicategory, we may refer to the two components thereof as the top and bottom components.

We denote the 2-morphisms of P-pseudo-comma bicategories as follows.



Thus, the shape of the 2-morphisms of P-pseudo-comma bicategories is cylindrical, due to the cubical nature of the boundary 1-morphisms and the globular nature of the 2-morphisms.

Definition 5.4.4 (Hom P-Categories of P-Pseudo-Comma Bicategories)**.** In this definition we fix two objects, x and y, of $(F \downarrow_{\text{ps}} G)$, and two 1-morphisms, $f : x \to y$ and $g : x \to y$. The *hom P-category*, $(F \downarrow_{\text{ps}} G)(x, y)$, is defined by the following stuff and structure:

- 1-morphisms of $(F \downarrow_{\text{ps}} G)$ for the objects;
- the P-set of 2-morphisms of $(F \downarrow_{\text{ps}} G)$ for the hom P-sets;
- composition in the hom P-categories of $\mathscr{B}$ and $\mathscr{C}$ for the composition; and
- identities in the hom P-categories of $\mathscr{B}$ and $\mathscr{C}$ for the identities.

Construction 5.4.5 (Top and Bottom Projection P-Functors)**.** For the hom P-category, $(F \downarrow_{\text{ps}} G)(x, y)$, there are two projection P-functors:

- $\text{Top} : (F \downarrow_{\text{ps}} G)(x, y) \to \mathscr{B}(x_{\text{src}}, y_{\text{src}})$; and
- $\text{Bot} : (F \downarrow_{\text{ps}} G)(x, y) \to \mathscr{B}(x_{\text{tgt}}, y_{\text{tgt}})$.

Moreover, there is a P-natural isomorphism:

$$(y_{\rightarrow} \bullet -) \circ F_1 \circ \text{Top} \xRightarrow{\sim} (- \bullet x_{\rightarrow}) \circ G_1 \circ \text{Bot}$$

Construction 5.4.6 (Induced P-Functors into Hom P-Categories of P-Pseudo-Comma Bicategories)**.** We may induce a P-functor from a P-category, $\mathbb{C}$, into a hom P-category, $(F \downarrow_{\text{ps}} G)(x, y)$, by the following data:

- a P-functor, $H : \mathbb{C} \to \mathscr{B}(x_{\text{src}}, y_{\text{src}})$;
- a P-functor, $K : \mathbb{C} \to \mathscr{B}(x_{\text{tgt}}, y_{\text{tgt}})$; and
- a P-natural isomorphism, $\alpha : (y_{\rightarrow} \bullet -) \circ F_1 \circ H \xRightarrow{\sim} (- \bullet x_{\rightarrow}) \circ G_1 \circ K$.

We denote the induced P-functor by $\langle H \downarrow_{\alpha} K \rangle_2$.

Construction 5.4.7 (P-Natural Isomorphisms between Induced P-Functors into Hom P-Categories of P-Pseudo-Comma Bicategories)**.** We may induce a P-natural isomorphism

$$\langle H_1 \downarrow_{\alpha_1} K_1 \rangle_2 \xRightarrow{\sim} \langle H_2 \downarrow_{\alpha_1} K_2 \rangle_2$$

where

- $H_1 : \mathbb{C} \to \mathscr{B}(x_{\text{src}}, y_{\text{src}})$,
- $H_2 : \mathbb{C} \to \mathscr{B}(x_{\text{src}}, y_{\text{src}})$,
- $K_1 : \mathbb{C} \to \mathscr{B}(x_{\text{tgt}}, y_{\text{tgt}})$,
- $K_2 : \mathbb{C} \to \mathscr{B}(x_{\text{tgt}}, y_{\text{tgt}})$,
- $\alpha_1 : (y_{\to} \bullet -) \circ F_1 \circ H \xRightarrow{\sim} (- \bullet x_{\to}) \circ G_1 \circ K$, and
- $\alpha_2 : (y_{\to} \bullet -) \circ F_1 \circ H \xRightarrow{\sim} (- \bullet x_{\to}) \circ G_1 \circ K$;

by the following data:

- a P-natural isomorphism, $\beta : H_1 \xRightarrow{\sim} H_2$;
- a P-natural isomorphism, $\gamma : K_1 \xRightarrow{\sim} K_2$; and
- a coherence, $\bigvee_{c : \mathbb{C}} (G\,\gamma_c \bullet x_{\to}) \circ \alpha_{1,c} \sim \alpha_{2,c} \circ (y_{\to} \bullet F\,\beta_c)$.

Remark 5.4.8. Constructions 5.4.6 and 5.4.7 extend to inducing P-bifunctors and P-binatural isomorphisms, and P-trifunctors and P-trinatural isomorphisms.

Definition 5.4.9 (Identity 1-Morphisms of P-Pseudo-Comma Bicategories). In this definition, we fix an object, z, of $(F \downarrow_{\text{ps}} G)$. The *identity* 1*-morphism* for z is given by the following data:

- the identity 1-morphism, $\text{id} : z_{\text{src}} \to z_{\text{src}}$, for the top component;
- the identity 1-morphism, $\text{id} : z_{\text{tgt}} \to z_{\text{tgt}}$, for the bottom component; and
- the P-isomorphism, $(z_{\to} \bullet G^{-1}_{\text{id}, z_{\text{tgt}}}) \circ \lambda^{-1} \circ \rho \circ (F_{\text{id}, z_{\text{src}}} \bullet z_{\to})$, for the commutation component.

Definition 5.4.10 (Composition of 1-Morphisms of P-Pseudo-Comma Bicategories). In this definition, we fix three objects of $(F \downarrow_{\text{ps}} G)$, x, y, and z. The *composition P-bifunctor of* 1*-morphisms* from x to y to z is induced by the following data:

- the P-bifunctor, $(- \bullet_{\mathscr{B}} =) \circ (\text{Top}_{y,z}; \text{Top}_{x,y})$;
- the P-bifunctor, $(- \bullet_{\mathscr{C}} =) \circ (\text{Bot}_{y,z}; \text{Bot}_{x,y})$; and
- a certain canonical P-binatural isomorphism.

Definition 5.4.11 (P-Pseudo-Comma Bicategory). The *P-pseudo-comma bicategory* of $F : \mathscr{B} \to_{\text{ps}} \mathscr{D}$ with $G : \mathscr{C} \to_{\text{ps}} \mathscr{D}$, which we denote by $(F \downarrow_{\text{ps}} G)$, is defined by the following stuff and structure:

- objects of P-pseudo-comma bicategories for the objects;
- the hom P-categories of P-pseudo-comma bicategories for the hom P-categories;
- the composition of 1-morphisms of P-pseudo-comma bicategories; and
- identity 1-morphisms of P-pseudo-comma bicategories for the identities.

The associator is induced by the associators of the P-bicategories $\mathscr{B}$ and $\mathscr{C}$. The left and right unitors are induced, respectively, by the left and right unitors of the P-bicategories $\mathscr{B}$ and $\mathscr{C}$.

Example 5.4.12 (P-Bicategory of Pointed P-Categories, Pointed P-Functors, and Pointed P-Natural Transformations). The *P-bicategory of pointed P-categories, pointed P-functors, and pointed P-natural transformations*, which we denote by $\mathbf{PCat}_\bullet$, is the P-pseudo-comma bicategory

$$(\Delta_1 \downarrow_{\text{ps}} \text{Id})$$

where

- $\Delta_1 : 1 \to_{\text{ps}} \mathbf{PCat}$ is the P-pseudofunctor that is constantly the terminal P-category, and
- $\text{Id} : \mathbf{PCat} \to_{\text{ps}} \mathbf{PCat}$ is the identity P-pseudofunctor on the P-bicategory of P-categories.

Example 5.4.13 (P-Bicategory of Pointed P-Cartesian Categories, Pointed P-Cartesian Functors, and Pointed P-Natural Transformations). The *P-bicategory of pointed P-Cartesian categories, pointed P-Cartesian functors, and pointed P-natural transformations*, which we denote by $\mathbf{PCartCat}_\bullet$, is the P-pseudo-comma bicategory

$$(\Delta_1 \downarrow_{\text{ps}} I)$$

where

- $\Delta_1 : 1 \to_{\text{ps}} \mathbf{PCat}$ is the P-pseudofunctor that is constantly the terminal P-category, and
- $I : \mathbf{PCartCat} \to_{\text{ps}} \mathbf{PCat}$ is the canonical P-pseudofunctor from the P-bicategory of P-Cartesian categories to the P-bicategory of P-categories.

Example 5.4.14 (P-Bicategory of Pointed P-Cartesian-closed Categories, Pointed P-Cartesian-closed Functors, and Pointed P-Natural Transformations). The *P-bicategory of pointed P-Cartesian-closed categories, pointed P-Cartesian-closed functors, and pointed P-natural transformations*, which we denote by $\mathbf{PCartClosCat}_\bullet$, is the P-pseudo-comma bicategory

$$(\Delta_1 \downarrow_{\text{ps}} I)$$

where

- $\Delta_1 : 1 \to_{\text{ps}} \mathbf{PCat}$ is the P-pseudofunctor that is constantly the terminal P-category, and
- $I : \mathbf{PCartClosCat} \to_{\text{ps}} \mathbf{PCat}$ is the canonical P-pseudofunctor from the P-bicategory of P-Cartesian-closed categories to the P-bicategory of P-categories.

Construction 5.4.15 (Domain and Codomain Projection P-Pseudofunctors). For the P-pseudo-comma bicategory, $(F \downarrow_{\text{ps}} G)$, where $F : \mathscr{B} \to_{\text{ps}} \mathscr{D}$ and $G : \mathscr{C} \to_{\text{ps}} \mathscr{D}$, there are two projection P-pseudofunctors:

- $\text{Dom} : (F \downarrow_{\text{ps}} G) \to_{\text{ps}} \mathscr{B}$; and
- $\text{Cod} : (F \downarrow_{\text{ps}} G) \to_{\text{ps}} \mathscr{C}$.

Moreover, there is a P-pseudonatural transformation:

$$F \circ \text{Dom} \Rightarrow_{\text{ps}} G \circ \text{Cod} \quad .$$

5.5 P-Bicategory of P-Categories

In this section, we investigate some P-bicategorical structure of the P-bicategory of P-categories. We further demonstrate how to construct pseudoends of P-pseudobifunctors $(\mathscr{B}^{\text{op}}; \mathscr{B}) \to_{\text{ps}} \mathbf{PCat}$. This entails the expected pseudo-completeness properties of the P-bicategory of P-categories.

5.5.1 *P-Pseudoends*

In this subsection, we give the construction of P-pseudoends in the P-bicategory of P-categories. The construction of P-pseudoends, being a general form of pseudolimit in the the P-bicategory of P-categories, provides insight into the P-pseudolimit theory of the P-bicategory of P-categories, and how this is established in the P-setting.

Construction 5.5.1 (P-Pseudoend)**.** The P-pseudoend of a P-pseudobifunctor, $F : (\mathscr{B}^{\mathrm{op}}; \mathscr{B}) \to_{\mathrm{ps}} \mathbf{PCat}$, is given by the following P-category. The objects, $\left(\int_{b : \mathscr{B}} F_0\, b\, b\right)_0$, of the P-pseudoend, $\int_{b : \mathscr{B}} F_0\, b\, b$, contain the following stuff and structure:

- a family of objects in the diagonal of F, $w_\Delta : \prod_{b : \mathscr{B}} F_0\, b\, b$; and
- a family of P-isomorphisms commuting the left and right actions of F, $w_{lr} : F_{1l}(f)(w_\Delta\, y) \xrightarrow{\sim} F_{1r}(f)(w_\Delta\, x)$;

such that the following properties hold:

- w_{lr} is coherent with the composition preservation of F; and
- w_{lr} is coherent with the identity preservation of F.

The hom P-set, $\left(\int_{b : \mathscr{B}} F_0\, b\, b\right)_1 (w, w')$, of the P-pseudoend, $\int_{b : \mathscr{B}} F_0\, b\, b$, is given by the following P-set.

- $\left|\left(\int_{b : \mathscr{B}} F_0\, b\, b\right)_1 (w, w')\right| \triangleq \prod_{b : \mathscr{B}} w_\Delta\, b \to w'_\Delta\, b$
- $\mathfrak{a} \sim \mathfrak{a}' \triangleq$
  - $\bigwedge_{x\, y : \mathscr{B}} \bigwedge_{f : x \to y} F_{1r}(f)(\mathfrak{a}_x) \circ w_{lr}(f) \sim w'_{lr}(f) \circ F_{1l}(f)(\mathfrak{a}'_y) \quad \wedge$
  - $\bigwedge_{z : \mathscr{B}} \mathfrak{a}_z \sim \mathfrak{a}'_z$

Remark 5.5.2. Note that the structure of the PER for the hom P-set for P-pseudoends is the structure of a bistrong sub-P-set of a discretely indexed dependent product of P-sets.

Example 5.5.3 (P-Pseudofunctor Bicategory Hom)**.** The definition of hom P-categories in P-pseudofunctor bicategories is a P-pseudoend.

$$[\mathscr{B}, \mathscr{C}]_{\mathrm{ps}}(F, G) \cong \int_{b : \mathscr{B}} \mathscr{C}_1(F\, b, G\, b)$$

Remark 5.5.4. This characterisation of the definition of the hom P-categories for P-pseudofunctor bicategories could be used as the definition of the hom P-categories for P-pseudo(bi)functor categories if one is careful about the order of definitions so as to avoid a chicken-and-egg problem.[2] The order of definitions would be something akin to the sequel:

1. define the construction of P-pseudoends non-functorially; thereafter
2. define the hom P-sets for P-pseudo(bi)functor categories; and finally thereafter
3. extend the construction of P-pseudoends to be a P-pseudofunctor.

[2] It is not used as the definition in the Rocq formalisation due to universe issues. Once algebraic universes are fully available in Rocq, the universe issues will no longer be present, and it is expected that this definition will be adopted.

5.6 P-Bicategorical Structure

In this section we define and analyse various biuniversal constructions within P-bicategory theory as a demonstration of the robustness of our P-bicategorical definitions. We continue with the guiding principle we followed for P-categorical universal constructions in structuring our definitions in a robustly (bi)category-theoretic style, using pre-existing P-(bi)categorical definitions, to ensure the correctness of the definitions both within the P-setting and within the two-dimensional setting of P-bicategory theory. This produces the following strategy for describing these definitions:

1. firstly, define an object forming operation;
2. secondly, set it within a biadjointness property, phrased using hom P-category biadjunctions; and
3. finally, induce an equational presentation therefrom.

We prefer this conceptual framework for much the same reasons as our preference therefor within the P-categorical setting: it reduces the complexity of the definitions, and allows for using P-(bi)category-theoretic machinery when instantiating such structures for certain P-bicategories. It also allows us to be confident that any equations deduced about any biuniversal construction are sound for the desired biuniversal constructions. Moreover, it gives us a standard by which to determine the completeness of any equational presentation with respect to the desired biuniversal construction.

5.6.1 *Terminal Objects*

Much like the P-categorical setting, the simplest form of biuniversal construction is pseudoterminal objects.

Definition 5.6.1 (P-Pseudoterminal Object)**.** An object, $\top$, of some P-bicategory, $\mathscr{B}$, is a *P-pseudoterminal object* precisely when there exists a P-pseudonatural adjoint equivalence as follows:

$$\mathrm{Hom}_{\mathscr{B}}(-, \top) \cong_{\mathrm{ps}} \Delta_1$$

where $\Delta_1 : \mathscr{B}^{\mathrm{op}} \to_{\mathrm{ps}} \mathbf{PCat}$ is the P-pseudofunctor that is constantly the terminal P-category. This definition is rendered in Rocq in listing 5.6.

```
Definition IsPPsdTermObj {B : PBiCat} (term : B) :=
  PPsdNatAdjEqv
    (PPsdBiFunIndRight PHomPsdBiFun term)
    (PCompPsdFun (PConstPsdFun PTermCat) PTermPsdFun).
```

Listing 5.6: P-Pseudoterminal Object Rocq Definition

Remark 5.6.2. Our definition of P-pseudoterminal object bears much semblance to our definition for P-terminal object: we consider this to be strong evidence for the advantages of this style of definition. As long as the P-bicategorical framework set-up in order to phrase this definition is correct, we can be confident that we are formalising the appropriate P-bicategorification of the P-categorical concept.

Construction 5.6.3 (P-Pseudoterminal Object Equations)**.** From this definition of P-pseudoterminal objects we are able to induce for a P-pseudoterminal object, $\top$, a 1-morphism from any object, $b : \mathscr{B}$, to $\top$, which we denote $!_b : b \to \top$, such that there is a P-iso-2-morphism, from any 1-morphism, $f : b \to \top$, to $!_b$, which we denote $!_f : f \to !_b$, such that $!_f$ is unique (up to the relevant PER) within $\mathrm{Hom}_{\mathscr{B}}(b, \top)(f, !_b)$. These data are equivalent with the definition given above.

Definition 5.6.4 (P-Pseudoinitial Object)**.** An object, $\bot$, of some P-bicategory, $\mathscr{B}$, is a *P-pseudoinitial object* precisely when it is a P-pseudoterminal object in the opposite P-bicategory, $\mathscr{B}^{\mathrm{op}}$.

5.6.2 *Cartesian Products*

Much like the P-categorical setting, the next simplest form of biuniversal construction is pseudo-Cartesian products.

Definition 5.6.5 (P-Pseudo-Cartesian Products)**.** A P-bicategory, $\mathscr{B}$, has *P-pseudo-Cartesian products* when it has an ordinary type-theoretic function:

$$- \times = : \mathscr{B}_0 \to \mathscr{B}_0 \to \mathscr{B}_0$$

such that for all pairs of objects, $a\, b : \mathscr{B}$, there exists a P-pseudonatural adjoint equivalence as follows:

$$\mathrm{Hom}_{\mathscr{B}}(-, a \times b) \cong_{\mathrm{ps}} \mathrm{Hom}_{\mathscr{B}\times\mathscr{B}}((-,-),(a,b))$$

This definition is rendered in Rocq in listing 5.7.

```
Definition IsPPsdCartProd {B : PBiCat} (prod : B -> B -> B) := forall a b,
  PPsdNatAdjEqv
    (PPsdBiFunIndRight PHomPsdBiFun (prod a b))
    (PCompPsdFun
      (PPsdBiFunIndRight (@PHomPsdBiFun (PProdBiCat B B)) (a, b))
      (POppPsdFun (PPairPsdFun PIdPsdFun PIdPsdFun))
    ).
```

Listing 5.7: P-Pseudo-Cartesian Product Rocq Definition

Construction 5.6.6 (P-Pseudo-Cartesian Product Equations)**.** From this definition of P-pseudo-Cartesian products we are able to induce the following structure for P-pseudo-Cartesian products:

- projection 1-morphisms, $\pi_1 : a \times b \to a$ and $\pi_2 : a \times b \to b$;
- pairing mapping for 1-morphisms, $\langle -, = \rangle : \mathrm{Hom}_{\mathscr{B}}(c,a) \to \mathrm{Hom}_{\mathscr{B}}(c,b) \to \mathrm{Hom}_{\mathscr{B}}(c, a \times b)$;
- a P-iso-2-morphism, $\pi_1 \bullet \langle f, g \rangle \to f$;
- a P-iso-2-morphism, $\pi_2 \bullet \langle f, g \rangle \to g$; and
- a P-well-defined pairing mapping on 2-morphisms,

$$\langle -, = \rangle_2 : \mathrm{Hom}_{\mathscr{B}(c,a)}(\pi_1 \bullet h, \pi_1 \bullet k) \to \mathrm{Hom}_{\mathscr{B}(c,b)}(\pi_2 \bullet h, \pi_2 \bullet k) \to \mathrm{Hom}_{\mathscr{B}(c,a\times b)}(h,k) \ ;$$

such that the following properties hold:

- $\alpha \sim \alpha' \wedge \beta \sim \beta' \Rightarrow \pi_1 \bullet \langle \alpha, \beta \rangle_2 \sim \alpha'$;
- $\alpha \sim \alpha' \wedge \beta \sim \beta' \Rightarrow \pi_2 \bullet \langle \alpha, \beta \rangle_2 \sim \beta'$; and
- $\gamma \sim \gamma' \Rightarrow \langle \pi_1 \bullet \gamma, \pi_2 \bullet \gamma \rangle_2 \sim \gamma'$;

From these one can derive the following useful properties:

- $\alpha \sim \alpha' \wedge \beta \sim \beta' \Rightarrow \langle \alpha, \beta \rangle_2 \bullet l \sim \langle \mathbf{assoc} \circ (\alpha' \bullet l) \circ \mathbf{assoc}^{-1}, \mathbf{assoc} \circ (\alpha' \bullet l) \circ \mathbf{assoc}^{-1} \rangle_2$;
- $\alpha \sim \alpha' \wedge \beta \sim \beta' \wedge \gamma \sim \gamma' \Rightarrow \langle \alpha, \beta \rangle_2 \circ \gamma \sim \langle \alpha' \circ (\pi_1 \bullet \gamma'), \beta' \circ (\pi_2 \bullet \gamma') \rangle_2$; and
- $\alpha \sim \alpha' \wedge \beta \sim \beta' \wedge \gamma \sim \gamma' \Rightarrow \gamma \circ \langle \alpha, \beta \rangle_2 \sim \langle (\pi_1 \bullet \gamma') \circ \alpha', (\pi_2 \bullet \gamma') \circ \beta' \rangle_2$;

and the following useful structure:

- a P-iso-2-morphism, $\langle \pi_1 \bullet k, \pi_2 \bullet k \rangle \to k$; and
- a P-iso-2-morphism, $\langle f, g \rangle \bullet h \to \langle f \bullet h, g \bullet h \rangle$.

5.6.3 *Comments*

In the two preceding sections, we have demonstrated the robustness of P-bicategory theory to support standard definitions from ordinary bicategory theory, with only mild and predictable translation into the P-setting. This further supports the correctness of our proposed definitions. Further development of P-bicategory theory should therefore follow ineluctably from the definitions laid down and proposed in this thesis.

Part III

Normalisation of Simple Type Theory

CHAPTER 6

Simple Type Theory

In this chapter, we give a presentation of simple type theory readily amenable to both type-theoretic formalism and to categorical analysis. We review the types and syntax of simple type theory from subsection 1.3.1 in a more principled and formal fashion. The Mathematical presentation is combined with relevant definitions from the Rocq formalism. This chapter, therefore, completes the presentation of all the necessary material for the p-categorical analysis of simple type theory in chapter 7.

Remark 6.0.1. In this thesis, we follow Sterling et al. (2021) in our use of *subjective* and *objective*. Explicitly we use *subjective* to refer to particular concrete presentations of a type theory with respect to a chosen judgmental and term structure, and *objective* to refer to universal characterisations of a type theory that is presentation independent. In short, subjective theorems are about a chosen presentation, and objective theorems are about the underlying theory presented by a chosen presentation.

We give a more subjective than objective presentation of simple type theory, as we are interested in computing with a subjective presentation rather than obtaining abstract results about the objective presentation of type theory. This results in a large degree of effort for the Rocq formalism; nonetheless, the principled approach we take guides the development of the formalism relatively smoothly. We connect our subjective presentation with a more objective understanding of simple type theory in chapter 7.

6.1 Types and Syntax

In this section, we introduce the types and syntax of simple type theory, as formalised in our Rocq formalism. In our Rocq formalism, we use intrinsically well-scoped and well-typed terms by using dependent types. We achieve well-scoping of terms by way of well-scoped de Bruijn indices for variables. We achieve well-typing of terms by unifying the raw terms with their well-typing derivations, producing a type of structurally well-typed terms. We choose to consider only a single base type, rather than some arbitrary set of base types for simplicity, although considering multiple base types would not be a difficult change. We use well-scoping and well-typing as this allows the construction of their elimination as used in the universal property in chapter 7.

Definition 6.1.1 (Simple Types). A *simple type*, the set of which we denote by $\mathbb{T}$, is inductively generated by the following rules.

$$\frac{}{\iota \in \mathbb{T}} \qquad \frac{T_1 \in \mathbb{T} \qquad T_2 \in \mathbb{T}}{T_1 \to T_2 \in \mathbb{T}}$$

This definition is rendered in Rocq in listing 6.1.

```
Inductive Ty : Set :=
  | Iota : Ty
  | Arr : Ty -> Ty -> Ty.
```

Listing 6.1: Simple Types Rocq Definition

Remark 6.1.2. We could introduce multiple base types by indexing the base type, ι/the `Iota` constructor, with some other set, such as the natural numbers for a countably infinite family of base types.

Definition 6.1.3 (Contexts). A *context* is a sequence of types that serve to record the type of each free variable in scope. As such, contexts are inductively generated by the following rules.

- $\bullet$ for the empty context; and
- Γ, T for the context Γ extended by the type T.

This definition is rendered in Rocq in listing 6.2.

```
Inductive Ctxt : Set :=
  | Nil : Ctxt
  | Snoc : Ctxt -> Ty -> Ctxt.
```

Listing 6.2: Contexts Rocq Definition

Remark 6.1.4. The introduction of a specific Rocq type for sequences of types to serve as contexts for our terms of simple type theory allows both having appropriately and specifically named constructors for empty and extended contexts, and also extending the sequence of types on the right, rather than the left as would be usual for lists.

We use de Bruijn indices for variables as they offer a unique representation for each term thus obviating the need to consider α-equivalence. They are also a more natural choice over de Bruijn levels as they number variables from the right of the context rather than the left, resulting in uniqueness of closed terms.

Definition 6.1.5 (de Bruijn Indices). A *de Bruijn index* is a natural number less than the length of the context, which identifies the variable of the context, counting from the right and starting with zero for the rightmost variable in the context. We can encode this by way of an inductive definition. Due to our intrinsically well-scoped and well-typed syntax de Bruijn indices do not exist outwith a specific context and type.

$$\frac{}{\Gamma, T \vdash i_0 \,:\, T} \qquad \frac{\Gamma \vdash i_n \,:\, T}{\Gamma, T' \vdash i_{n+1} \,:\, T}$$

This definition is rendered in Rocq in listing 6.3.

```
Inductive Idx : Ctxt -> Ty -> Set :=
  | IdxZero {Gamma T} : Idx (Snoc Gamma T) T
  | IdxSucc {Gamma T T'} : Idx Gamma T -> Idx (Snoc Gamma T') T.
```

Listing 6.3: de Bruijn Indices Rocq Definition

Remark 6.1.6. By indexing the type of de Bruijn indices in contexts, we are able to enforce intrinsic well-scoping for indices at the type level; by indexing the type of de Bruijn indices in types, we are able to enforce at the type level the type of the variable represented by the de Bruijn index.

The definition of the terms of simple type theory now follows straightforwardly from the above definitions. Again we index the type of terms in contexts and types to enforce intrinsic well-scoping and well-typing at the type level within our Rocq formalism.

Definition 6.1.7 (Simply Typed Terms). A *term* in simple type theory is either a variable, an application of one term to another, or an abstraction of a term over a variable. To ensure well-scoping variables have the same context as the term containing them, the two parts of an application are in the same context, and the abstraction removes the rightmost variable from the context. To ensure well-typing variables have the same type as the the term containing them, the two parts of an application must have appropriate types, and the abstraction has arrow type. This can be encoded by way of an inductive definition. Due to our intrinsically well-scoped and well-typed syntax de Bruijn indices do not exist outwith a specific context and type.

$$\frac{\Gamma \vdash i_n \,:\, T}{\Gamma \vdash \underline{i_n} \,:\, T} \qquad \frac{\Gamma \vdash t_1 \,:\, T \to T' \qquad \Gamma \vdash t_2 \,:\, T}{\Gamma \vdash t_1\, t_2 \,:\, T'} \qquad \frac{\Gamma, T \vdash t \,:\, T'}{\Gamma \vdash \lambda_T.t \,:\, T \to T'}$$

This definition is rendered in Rocq in listing 6.4.

```
Inductive Tm (Gamma : Ctxt) : Ty -> Set :=
  | Var {T} : Idx Gamma T -> Tm Gamma T
  | App {T T'} : Tm Gamma (Arr T T') -> Tm Gamma T -> Tm Gamma T'
  | Abs {T T'} : Tm (Snoc Gamma T) T' -> Tm Gamma (Arr T T').
```

Listing 6.4: Simply-Typed Terms Rocq Definition

Remark 6.1.8. Note that since variables are represented by de Bruijn indices, and are thus nameless, we do not mention names in either contexts or terms. Therefore, referencing a variable in context is simply using its respective de Bruijn index, and abstracting a term is simply reducing the context by its rightmost type and recording this type in the abstraction. This nameless representation allows the identity type of Rocq to serve as a *bona fide* alternative to α-equivalence.

6.2 Renaming and Substitution

In this section, we introduce two structures between contexts; *viz.*, renamings and substitutions. These provide the basis for three p-categories, that have contexts as objects, that are used in chapter 7. Although the two structures are used in establishing categorical structures, using both is also a good strategy for formalising the structure of substitutions following Benton et al. (2012).

6.2.1 *Renaming*

In this subsection, we introduce the structure of renamings and detail their compositional structure.

Definition 6.2.1 (Context Renaming). A *context renaming* from Γ to Δ, which we denote by $\Gamma \to_{\text{ren}} \Delta$ or $\Gamma \vdash_{\text{ren}} \Delta$, is a sequence of de Bruijn indices in context Γ whose types form the context Δ. This is described by the following inductive definition. By abuse of notation we reuse '$\bullet$' and ',' for renamings.

$$\frac{}{\Gamma \vdash_{\text{ren}} \bullet \,:\, \bullet} \qquad \frac{\Gamma \vdash_{\text{ren}} \rho \,:\, \Delta \qquad \Gamma \vdash i_n \,:\, T}{\Gamma \vdash_{\text{ren}} \rho,\, i_n \,:\, \Delta, T}$$

This definition is rendered in Rocq in listing 6.5.

```
Inductive CtxtRnm Gamma : Ctxt -> Set :=
  | CtxtRnmNil : CtxtRnm Gamma Nil
  | CtxtRnmSnoc {Delta T} :
    CtxtRnm Gamma Delta -> Idx Gamma T -> CtxtRnm Gamma (Snoc Delta T).
```

Listing 6.5: Context Renamings Rocq Definition

Construction 6.2.2 (Context Renaming Weakening). Any context renaming, $\rho \ : \ \Gamma \to_{\text{ren}} \Delta$, may be *weakened* into a context renaming, $\rho_T \ : \ \Gamma, \ T \to_{\text{ren}} \Delta$.

Construction 6.2.3 (Index Renaming). Any de Bruijn index, $\Delta \vdash i_n \ : \ T$, may be renamed by $\Gamma \vdash_{\text{ren}} \rho \ : \ \Delta$ resulting in

$$\Gamma \vdash i_n[\rho] \ : \ T$$

Construction 6.2.4 (Term Renaming). Any term, $\Delta \vdash t \ : \ T$, may be renamed by $\Gamma \vdash_{\text{ren}} \rho \ : \ \Delta$ resulting in

$$\Gamma \vdash t[\rho] \ : \ T$$

It is computed by the sequel.

$$\begin{aligned}
\Gamma \vdash \underline{i_n}[\rho] \ : \ T &\triangleq \Gamma \vdash \underline{i_n[\rho]} \ : \ T \\
\Gamma \vdash (t_1 \, t_2)[\rho] \ : \ T &\triangleq \Gamma \vdash t_1[\rho] \, t_2[\rho] \ : \ T \\
\Gamma \vdash (\lambda_T.t)[\rho] \ : \ T &\triangleq \Gamma \vdash \lambda_T.t[\rho_T, \ i_0] \ : \ T
\end{aligned}$$

Construction 6.2.5 (Identity Renaming). For any context, Γ, there is an *identity renaming*.

$$\text{id}_\Gamma \ : \ \Gamma \to_{\text{ren}} \Gamma$$

It is computed by the sequel.

$$\begin{aligned}
\text{id}_\bullet &\triangleq \bullet \\
\text{id}_{\Gamma, \, T} &\triangleq (\text{id}_\Gamma)_T, \ i_0
\end{aligned}$$

Construction 6.2.6 (Renaming Composition). For any two renamings, $\rho \ : \ \Delta \to_{\text{ren}} \Theta$ and $\varphi \ : \ \Gamma \to_{\text{ren}} \Delta$, there is a *composition*.

$$\rho \circ \varphi \ : \ \Gamma \to_{\text{ren}} \Theta$$

It is computed by the sequel.

$$\begin{aligned}
\bullet \circ \varphi &\triangleq \bullet \\
(\rho, i_n) \circ \varphi &\triangleq (\rho \circ \varphi), i_n[\varphi]
\end{aligned}$$

Fact 6.2.7. *The identity renaming and the composition of renamings are rightly called such, as they satisfy the following properties:*

- $i_n[\text{id}] = i_n$*;*
- $t[\text{id}] = t$*;*
- $i_n[\rho \circ \varphi] = i_n[\rho][\varphi]$*; and*

- $t[\rho \circ \varphi] = t[\rho][\varphi]$.

From these facts about index renaming, the following facts follow:

- $\mathrm{id} \circ \varphi = \varphi$;
- $\rho \circ \mathrm{id} = \rho$; *and*
- $(\rho \circ \varphi) \circ \chi = \rho \circ (\varphi \circ \chi)$.

6.2.2 *Substitution*

In this subsection, we introduce the structure of substitutions and detail their compositional structure.

Definition 6.2.8 (Context Substitution). A *context substitution* from Γ to Δ, which we denote by $\Gamma \to_{\mathrm{sub}} \Delta$ or $\Gamma \vdash_{\mathrm{sub}} \Delta$, is a sequence of terms in context Γ whose types form the context Δ. This is described by the following inductive definition. By abuse of notation we reuse '$\bullet$' and ',' for substitutions.

$$\frac{}{\Gamma \vdash_{\mathrm{sub}} \bullet \,:\, \bullet} \qquad \frac{\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta \qquad \Gamma \vdash t \,:\, T}{\Gamma \vdash_{\mathrm{sub}} \sigma,\, t \,:\, \Delta,\, T}$$

This definition is rendered in Rocq in listing 6.6.

```
Inductive CtxtSubst Gamma : Ctxt -> Set :=
  | CtxtSubstNil : CtxtSubst Gamma Nil
  | CtxtSubstSnoc {Delta T} :
    CtxtSubst Gamma Delta -> Tm Gamma T -> CtxtSubst Gamma (Snoc Delta T).
```

Listing 6.6: Context Substitutions Rocq Definition

Construction 6.2.9 (Context Substitution Weakening). Any context substitution, $\sigma \,:\, \Gamma \to_{\mathrm{ren}} \Delta$, may be *weakened* into a context substitution $\sigma_T \,:\, \Gamma,\, T \to_{\mathrm{ren}} \Delta$.

Construction 6.2.10 (Index Substitution). Any de Bruijn index, $\Delta \vdash i_n \,:\, T$, may be substituted by $\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta$ into a term resulting in

$$\Gamma \vdash i_n[\sigma] \,:\, T$$

Construction 6.2.11 (Term Substitution). Any term, $\Delta \vdash t \,:\, T$, may be substituted by $\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta$ resulting in

$$\Gamma \vdash t[\sigma] \,:\, T$$

It is computed by the sequel.

$$\begin{aligned}
\Gamma \vdash \underline{i_n}[\sigma] \,:\, T &\triangleq \Gamma \vdash \underline{i_n[\sigma]} \,:\, T \\
\Gamma \vdash (t_1\, t_2)[\sigma] \,:\, T &\triangleq \Gamma \vdash t_1[\rho]\, t_2[\sigma] \,:\, T \\
\Gamma \vdash (\lambda_T.t)[\sigma] \,:\, T &\triangleq \Gamma \vdash \lambda_T.t[\sigma_T,\, i_0] \,:\, T
\end{aligned}$$

Construction 6.2.12 (Identity Substitution). For any context, Γ, there is an *identity substitution*.

$$\mathrm{id}_\Gamma \,:\, \Gamma \to_{\mathrm{sub}} \Gamma$$

It is computed by the sequel.

$$\mathrm{id}_\bullet \triangleq \bullet$$
$$\mathrm{id}_{\Gamma, T} \triangleq (\mathrm{id}_\Gamma)_T,\ \underline{i_0}$$

Construction 6.2.13 (Substitution Composition). For any two substitutions, $\sigma : \Delta \to_{\mathrm{sub}} \Theta$ and $\psi : \Gamma \to_{\mathrm{sub}} \Delta$, there is a *composition*.

$$\sigma \circ \psi : \Gamma \to_{\mathrm{sub}} \Theta$$

It is computed by the sequel.

$$\bullet \circ \psi \triangleq \bullet$$
$$(\sigma, t) \circ \psi \triangleq (\sigma \circ \psi), t[\psi]$$

Fact 6.2.14. *The identity substitution and the composition of substitutions are rightly called such, as they satisfy the following properties:*

- $i_n[\mathrm{id}] = \underline{i_n}$*;*
- $t[\mathrm{id}] = t$*;*
- $i_n[\sigma \circ \psi] = i_n[\sigma][\psi]$*; and*
- $t[\sigma \circ \psi] = t[\sigma][\psi]$.

From these facts about substitution, the following facts follow:

- $\mathrm{id} \circ \psi = \psi$*;*
- $\sigma \circ \mathrm{id} = \rho$*; and*
- $(\sigma \circ \psi) \circ \xi = \sigma \circ (\psi \circ \xi)$.

Remark 6.2.15 (Hetero-compositions). We also define mixed hetero-compositions between renamings and substitutions (in both orders), resulting in a substitution. They satisfy the expected unitality properties with respect to the identity renamings and substitutions. They also satisfy various mixed-associativity properties where they are well-defined.

6.3 βη-Equational Theory

In this section we introduce the formal presentation of the βη-equational theory of simple type theory. This endows the structure of terms with a notion of computation and normal form. We define βη-conversion first for terms, and thereafter for substitutions.

Definition 6.3.1 (Term βη-Conversion). Two terms of simple type theory, in the same context with the same type, are *βη-convertible* whenever the first can be transformed into the second via a series of β-contractions, β-expansions, η-expansions,

and η-contractions. This can be expressed by the following inductive definition.

$$\frac{\Gamma \vdash i_n \,:\, T}{\Gamma \vdash \underline{i_n} \equiv_{\beta\eta} \underline{i_n} \,:\, T} \qquad \frac{\Gamma \vdash t_1 \equiv_{\beta\eta} t_1' \,:\, T_1 \to T_2 \qquad \Gamma \vdash t_2 \equiv_{\beta\eta} t_2' \,:\, T_1}{\Gamma \vdash t_1\, t_2 \equiv_{\beta\eta} t_1'\, t_2' \,:\, T_2} \qquad \frac{\Gamma, T \vdash t_1 \equiv_{\beta\eta} t_2 \,:\, T'}{\Gamma \vdash \lambda_T.t_1 \equiv_{\beta\eta} \lambda_T.t_2 \,:\, T \to T'}$$

$$\frac{\Gamma, T_1 \vdash t_1 \,:\, T_2 \qquad \Gamma \vdash t_2 \,:\, T_1}{\Gamma \vdash (\lambda_{T_1}.t_1)\, t_2 \equiv_{\beta\eta} t_1[\mathrm{id}_\Gamma, t_2] \,:\, T_2} \qquad \frac{\Gamma \vdash t \,:\, T_1 \to T_2}{\Gamma \vdash t \equiv_{\beta\eta} \lambda_{T_1}.(t[(\mathrm{id}_\Gamma)_{T_1}]\ \underline{i_0}) \,:\, T_1 \to T_2}$$

$$\frac{\Gamma \vdash t_1 \equiv_{\beta\eta} t_2 \,:\, T}{\Gamma \vdash t_2 \equiv_{\beta\eta} t_1 \,:\, T} \qquad \frac{\Gamma \vdash t_1 \equiv_{\beta\eta} t_2 \,:\, T \qquad \Gamma \vdash t_2 \equiv_{\beta\eta} t_3 \,:\, T}{\Gamma \vdash t_1 \equiv_{\beta\eta} t_3 \,:\, T}$$

This definition is rendered in Rocq in listing 6.7.

```
Inductive BetaEtaConv {Gamma} : forall {T}, Tm Gamma T -> Tm Gamma T -> SProp :=
  | BetaEtaConvVar {T} (i : Idx Gamma T) : BetaEtaConv (Var i) (Var i)
  | BetaEtaConvApp {T1 T2} {t1 t1' : Tm Gamma (Arr T1 T2)} {t2 t2' : Tm Gamma T1} :
    BetaEtaConv t1 t1' ->
    BetaEtaConv t2 t2' ->
    BetaEtaConv (App t1 t2) (App t1' t2')
  | BetaEtaConvAbs {T T'} {t1 t2 : Tm (Snoc Gamma T) T'} :
    BetaEtaConv t1 t2 ->
    BetaEtaConv (Abs t1) (Abs t2)
  | BetaEtaConvBeta {T1 T2} (t1 : Tm (Snoc Gamma T1) T2) (t2 : Tm Gamma T1) :
    BetaEtaConv (App (Abs t1) t2) (BetaSubst t1 t2)
  | BetaEtaConvEta {T1 T2} (t : Tm Gamma (Arr T1 T2)) :
    BetaEtaConv t (Abs (App (Shift t) (Var IdxZero)))
  | BetaEtaConvSymm {T} {t1 t2 : Tm Gamma T} : BetaEtaConv t1 t2 -> BetaEtaConv t2 t1
  | BetaEtaConvTrans {T} {t1 t2 t3 : Tm Gamma T} :
    BetaEtaConv t1 t2 ->
    BetaEtaConv t2 t3 ->
    BetaEtaConv t1 t3.
```

Listing 6.7: Term $\beta\eta$-Conversion Rocq Definition

Remark 6.3.2. There are a number of different ways to express $\beta\eta$-conversion. We choose the congruential, symmetric, transitive closure over β-contraction and η-expansion, as it is immediately a (partial) equivalence relation without any proof. We do not need to characterise our definition in terms of other definitions, and so this is the most convenient presentation for our work.

Fact 6.3.3 (Term $\beta\eta$-Conversion Reflexivity). *Term $\beta\eta$-conversion is reflexive; this follows from induction over the structure of a term. Note that the base case of variables is reflexive, and the congruence rules propagate reflexivity.*

Fact 6.3.4 (Term $\beta\eta$-Conversion Renaming/Substitution Invariance). *Two terms that are $\beta\eta$-convertible remain $\beta\eta$-convertible when renamed by the same renaming, or substituted by the same substitution. That is, the following two properties hold:*

- $t_1 \equiv_{\beta\eta} t_2 \Rightarrow t_1[\rho] \equiv_{\beta\eta} t_2[\rho]$*; and*
- $t_1 \equiv_{\beta\eta} t_2 \Rightarrow t_1[\sigma] \equiv_{\beta\eta} t_2[\sigma]$.

We extend the definition of $\beta\eta$-conversion to substitutions, in a pointwise fashion.

Definition 6.3.5 (Substitution $\beta\eta$-Conversion). Two substitutions from the same context to the same context are *$\beta\eta$-convertible* when each pair of terms from the two substitutions are $\beta\eta$-convertible. This can be expressed by the following

inductive definition.

$$\frac{}{\Gamma \vdash_{\text{sub}} \bullet \equiv_{\beta\eta} \bullet \,:\, \bullet} \qquad \frac{\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \qquad \Gamma \vdash t_1 \equiv_{\beta\eta} t_2 \,:\, T}{\Gamma \vdash_{\text{sub}} \sigma_1,\, t_1 \equiv_{\beta\eta} \sigma_2,\, t_2 \,:\, \Delta, T}$$

This definition is rendered in Rocq in listing 6.8.

```
Inductive CtxtSubstConv {Gamma} : forall {Delta},
  CtxtSubst Gamma Delta -> CtxtSubst Gamma Delta -> SProp :=
  | CtxtSubstConvNil : CtxtSubstConv (CtxtSubstNil Gamma) (CtxtSubstNil Gamma)
  | CtxtSubstConvSnoc {Delta sigma1 sigma2 T t1 t2} :
    CtxtSubstConv sigma1 sigma2 ->
    BetaEtaConv t1 t2 ->
    CtxtSubstConv (CtxtSubstSnoc sigma1 t1) (CtxtSubstSnoc sigma2 t2).
```

Listing 6.8: Context Substitution βη-Conversion Rocq Definition

Fact 6.3.6 (Substitution βη-Conversion Reflexivity). *Substitution βη-conversion is reflexive; this follows from induction over the structure of the substitution, and from term βη-conversion being reflexive.*

Fact 6.3.7 (Substitution βη-Conversion Symmetry). *Substitution βη-conversion is symmetric; this follows from induction over the rules of substitution βη-conversion, and from term βη-conversion being symmetric.*

Lemma 6.3.8. *Two substitutions are βη-convertible precisely when for all de Bruijn indices the pair of terms induced by substitution are βη-convertible;* i.e.*, the following property holds.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta$$

$$\Leftrightarrow$$

$$\bigvee_{T \in \mathbb{T}} \bigvee_{\Delta \vdash i_n \,:\, T} \Gamma \vdash i_n[\sigma_1] \equiv_{\beta\eta} i_n[\sigma_2] \,:\, T$$

Fact 6.3.9 (Substitution βη-Conversion Transitivity). *Substitution βη-conversion is transitive; this follows from term βη-conversion being transitive, and from the characterisation of substitution βη-conversion in lemma 6.3.8.*

Fact 6.3.10 (Substitution βη-Convertibility Weakening). *Two βη-convertible substitutions remain βη-convertible when weakened;* i.e.*, the following property holds.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \Rightarrow \Gamma, T \vdash_{\text{sub}} (\sigma_1)_T \equiv_{\beta\eta} (\sigma_2)_T \,:\, \Delta$$

Fact 6.3.11 (Substitution βη-Convertibility on Terms). *Substitution by βη-convertible substitutions creates βη-convertible terms;* i.e.*, the following property holds.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \Rightarrow \Gamma \vdash t[\sigma_1] \equiv_{\beta\eta} t[\sigma_2] \,:\, T$$

Fact 6.3.12. *Composition of context substitutions respects βη-conversion;* i.e.*, the following property holds.*

$$\Delta \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Theta \wedge \Gamma \vdash_{\text{sub}} \psi_1 \equiv_{\beta\eta} \psi_2 \,:\, \Delta \Rightarrow \Gamma \vdash \sigma_1 \circ \psi_1 \equiv_{\beta\eta} \sigma_2 \circ \psi_2 \,:\, \Theta$$

6.3.1 *Neutral and Normal Terms*

In this subsection, we introduce the concepts both of neutral terms and of normal terms, and then extend these concepts to context substitutions. Both neutral and normal forms are a instance of the fourth kind of sub-p-set from subsection 4.1.2:

they can be expressed either as a predicate on terms, or defined as a separate structure with coercion maps into all terms. We opt for the former approach, as it allows the inclusion both of neutral terms and of normal terms into all terms to be the identity mapping computationally, which is evidently much more efficient than a recursive deep coercion mapping. The same analysis applies both to neutral context substitutions and normal context substitutions; thus, the advantages of our choice for how to model both neutral terms and normal terms extend to advantages for how we model both neutral context substitutions and normal context substitutions. We therefore make the following definitions.

Definition 6.3.13 (Neutral and Normal Terms). *Neutral terms and normal terms* can be picked out from all terms by the following mutually-inductive predicates, where we denote a term, $\Gamma \vdash t \,:\, T$, being neutral by $\Gamma \vdash_{\mathcal{M}} t \,:\, T$ and being normal by $\Gamma \vdash_{\mathcal{N}} t \,:\, T$.

$$\frac{}{\Gamma \vdash_{\mathcal{M}} \underline{i_n} \,:\, T} \qquad \frac{\Gamma \vdash_{\mathcal{M}} t_1 \,:\, T_1 \to T_2 \qquad \Gamma \vdash_{\mathcal{N}} t_2 \,:\, T_1}{\Gamma \vdash_{\mathcal{M}} t_1\, t_2 \,:\, T_2}$$

$$\frac{\Gamma \vdash_{\mathcal{M}} t \,:\, \iota}{\Gamma \vdash_{\mathcal{N}} t \,:\, \iota} \qquad \frac{\Gamma, T \vdash_{\mathcal{N}} t \,:\, T'}{\Gamma \vdash_{\mathcal{N}} \lambda_T.t \,:\, T \to T'}$$

These rules should not be understood as defining new structures, but merely predicates on the existing structure of terms; thus the rules each presuppose that the terms involved are already terms as in definition 6.1.7. This definition is rendered in Rocq in listing 6.9.

```
Inductive IsNeutral {Gamma} : forall {T}, Tm Gamma T -> SProp :=
  | IsNeutralVar {T} (i : Idx Gamma T) : IsNeutral (Var i)
  | IsNeutralApp {T T'} {t0 : Tm Gamma (Arr T T')} {t1 : Tm Gamma T} :
    IsNeutral t0 -> IsNormal t1 -> IsNeutral (App t0 t1)
with IsNormal {Gamma} : forall {T}, Tm Gamma T -> SProp :=
  | IsNormalNeutral {t : Tm Gamma Iota} : IsNeutral t -> IsNormal t
  | IsNormalAbs {T T'} {t : Tm (Snoc Gamma T) T'} : IsNormal t -> IsNormal (Abs t).
```

Listing 6.9: Neutral and Normal Terms Predicates Rocq Definition

Fact 6.3.14 (Term Neutrality/Normality Renaming Invariance). *A neutral/normal term remains neutral/normal when renamed by a context renaming;* i.e.*, the following properties hold:*

- $\Delta \vdash_{\mathcal{M}} t \,:\, T \wedge \Gamma \vdash_{\text{ren}} \rho \,:\, \Delta \Rightarrow \Gamma \vdash_{\mathcal{M}} t[\rho] \,:\, T$*; and*
- $\Delta \vdash_{\mathcal{N}} t \,:\, T \wedge \Gamma \vdash_{\text{ren}} \rho \,:\, \Delta \Rightarrow \Gamma \vdash_{\mathcal{N}} t[\rho] \,:\, T$.

Definition 6.3.15 (Neutral and Normal Substitutions). *Neutral substitutions* are precisely those substitutions all of whose component terms are neutral. *Normal substitutions* are precisely those substitutions all of whose component terms are normal. These predicates can be defined by the following rules; thus, they presuppose that the terms and context substitutions involved are already terms as in definition 6.1.7, and context substitutions as in definition 6.2.8.

$$\frac{}{\Gamma \vdash_{\mathcal{M}\,\text{sub}} \bullet \,:\, \bullet} \qquad \frac{\Gamma \vdash_{\mathcal{M}\,\text{sub}} \sigma \,:\, \Delta \qquad \Gamma \vdash_{\mathcal{M}} t \,:\, T}{\Gamma \vdash_{\mathcal{M}\,\text{sub}} \sigma, t \,:\, \Delta, T}$$

$$\frac{}{\Gamma \vdash_{\mathcal{N}\,\text{sub}} \bullet \,:\, \bullet} \qquad \frac{\Gamma \vdash_{\mathcal{N}\,\text{sub}} \sigma \,:\, \Delta \qquad \Gamma \vdash_{\mathcal{N}} t \,:\, T}{\Gamma \vdash_{\mathcal{N}\,\text{sub}} \sigma, t \,:\, \Delta, T}$$

This definition is rendered in Rocq in listing 6.10.

```
Inductive IsNeutralCtxtSubst {Gamma} :
  forall {Delta}, CtxtSubst Gamma Delta -> SProp :=
  | IsNeutralCtxtSubstNil : IsNeutralCtxtSubst (CtxtSubstNil Gamma)
  | IsNeutralCtxtSubstSnoc {Delta} {T} {sigma} {t : Tm Gamma T} :
    @IsNeutralCtxtSubst Gamma Delta sigma ->
    IsNeutral t ->
    IsNeutralCtxtSubst (CtxtSubstSnoc sigma t).

Inductive IsNormalCtxtSubst {Gamma} :
  forall {Delta}, CtxtSubst Gamma Delta -> SProp :=
  | IsNormalCtxtSubstNil : IsNormalCtxtSubst (CtxtSubstNil Gamma)
  | IsNormalCtxtSubstSnoc {Delta} {T} {sigma} {t : Tm Gamma T} :
    @IsNormalCtxtSubst Gamma Delta sigma ->
    IsNormal t ->
    IsNormalCtxtSubst (CtxtSubstSnoc sigma t).
```

Listing 6.10: Neutral and Normal Context Substitutions Predicates Rocq Definition

Fact 6.3.16 (Term Neutrality/Normality Neutral Substitution Invariance). *A neutral/normal term remains neutral/normal when substituted by a neutral context substitution;* i.e., *the following properties hold:*

- $\Delta \vdash_{\mathcal{M}} t : T \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma : \Delta \Rightarrow \Gamma \vdash_{\mathcal{M}} t[\sigma] : T$*; and*
- $\Delta \vdash_{\mathcal{N}} t : T \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma : \Delta \Rightarrow \Gamma \vdash_{\mathcal{N}} t[\sigma] : T$.

Fact 6.3.17 (Context Substitution Neutrality/Normality Renaming Invariance). *A neutral/normal context substitution remains neutral/normal when renamed by a context renaming on either side;* i.e., *the following properties hold:*

- $\Delta \vdash_{\mathrm{ren}} \rho : \Theta \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma : \Delta \Rightarrow \Gamma \vdash_{\mathcal{M}} \rho \circ \sigma : \Theta$*; and*
- $\Delta \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma : \Theta \wedge \Gamma \vdash_{\mathrm{ren}} \rho : \Delta \Rightarrow \Gamma \vdash_{\mathcal{M}} \sigma \circ \rho : \Theta$*; and*
- $\Delta \vdash_{\mathrm{ren}} \rho : \Theta \wedge \Gamma \vdash_{\mathcal{N}\,\mathrm{sub}} \sigma : \Delta \Rightarrow \Gamma \vdash_{\mathcal{N}} \rho \circ \sigma : \Theta$*; and*
- $\Delta \vdash_{\mathcal{N}\,\mathrm{sub}} \sigma : \Theta \wedge \Gamma \vdash_{\mathrm{ren}} \rho : \Delta \Rightarrow \Gamma \vdash_{\mathcal{N}} \sigma \circ \rho : \Theta$.

Fact 6.3.18 (Susbtitution Neutrality/Normality Neutral Substitution Invariance). *A neutral/normal context substitution remains neutral/normal when substituted by a neutral context substitution;* i.e., *the following properties hold:*

- $\Delta \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma : \Theta \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \psi : \Delta \Rightarrow \Gamma \vdash_{\mathcal{M}} \sigma \circ \psi : \Theta$*; and*
- $\Delta \vdash_{\mathcal{N}\,\mathrm{sub}} \sigma : \Theta \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \psi : \Delta \Rightarrow \Gamma \vdash_{\mathcal{N}} \sigma \circ \psi : \Theta$.

CHAPTER 7

P-CATEGORICAL STRUCTURE OF SIMPLY TYPED SYNTAX

In this chapter, we introduce the P-categorical structures induced by the types and syntax of simple type theory. This gives rise to four separate P-categories: a P-category of renamings, a P-category of substitutions under equality, a P-category of neutral substitutions under equality, and a P-category of substitutions under $\beta\eta$-conversion; each of which we use in the construction of normal-form algorithms in chapter 8.

By formalising a category-theoretic perspective on our subjective presentation of simple type theory, we provide an objective perspective on the formalised presentation. Nonetheless, our approach skirts the boundary between the objective and subjective perspectives, borrowing from each when useful. Such an approach allows us to combine the robustness of the objective with the convenience of the subjective.

7.1 UNIVERSAL STRUCTURE

The structures of renamings and substitutions induce four P-categories; in this section we define each of these and their P-categorical structure. Each of the four P-categories have contexts for their objects, but have different P-sets for their hom P-sets. The difference in their hom-P-sets gives rise to their different P-categorical structure and behaviour. In P-category theory we have easy access to fine control over difference in hom-P-sets: we may have two P-sets with the same underlying type, but different PERs. Having access to this finer control of difference motivates constructing more P-categories and extracting more structures and behaviours out therefrom than is ordinarily done. First, we define some preliminary constructions that we use in the P-categorical structure of the four P-categories.

Construction 7.1.1 (Context Concatenation). Two contexts, Γ and Δ, can be concatenated into a third, which we denote by $\Gamma +\!\!+ \Delta$. It can be computed thus:

$$\Gamma +\!\!+ \bullet \triangleq \Gamma$$
$$\Gamma +\!\!+ (\Delta, T) \triangleq (\Gamma +\!\!+ \Delta), T$$

Construction 7.1.2 (Context Exponential). A context, Γ, can be exponentiated by a type, T, to form a new context, which we denote by Γ^T. It can be computed thus:

$$(\bullet)^T \triangleq (\bullet)$$
$$(\Gamma, T')^T \triangleq \Gamma^T, T \to T'$$

Two contexts, Γ and Δ, can be exponentiated into a third, which we denote by Γ^{Δ}. It can be computed thus:

$$\Gamma^{(\bullet)} \triangleq \Gamma$$
$$\Gamma^{\Delta, T} \triangleq (\Gamma^{T})^{\Delta}$$

7.1.1 *P-Category of Renamings, $\mathcal{R}$*

The structure of contexts and context renamings gives rise to a P-category of renamings.

Definition 7.1.3 (P-Category of Renamings). The *P-category of context renamings*, which we denote by $\mathcal{R}$, has the following stuff and structure:

- contexts for the objects;
- the discrete P-set of context renamings for the hom P-sets;
- composition of context renamings for the composition of morphisms; and
- identity context renamings for the identity morphisms.

Construction 7.1.4 (Context Concatenation Projection and Pairing for Renamings). There are two projection renamings from the concatenation of contexts:

- $\pi_1 \,:\, \Gamma +\!\!+ \Delta \to_{\text{ren}} \Gamma$; and
- $\pi_2 \,:\, \Gamma +\!\!+ \Delta \to_{\text{ren}} \Delta$.

These renamings are straightforwardly defined by recursion on the structure of Δ. Moreover, there is a pairing mapping for two renamings from the same context:

$$\frac{\rho \,:\, \Gamma \to_{\text{ren}} \Delta \qquad \varphi \,:\, \Gamma \to_{\text{ren}} \Theta}{\langle \rho, \varphi \rangle \,:\, \Gamma \to_{\text{ren}} \Delta +\!\!+ \Theta}\ .$$

This renaming is straightforwardly defined by recursion on the structure of φ.

Fact 7.1.5. *The projection and pairing renamings satisfy the following properties:*

- $\pi_1 \circ \langle \rho, \varphi \rangle = \rho$;
- $\pi_2 \circ \langle \rho, \varphi \rangle = \varphi$; *and*
- $\langle \pi_1 \circ \rho, \pi_2 \circ \rho \rangle = \rho$.

Construction 7.1.6 (P-Cartesian Category of Renamings). The P-category of context renamings is a P-Cartesian category. The P-terminal object is given by the empty context; the P-Cartesian product is given by the concatenation of contexts, and the associated constructions given in construction 7.1.4.

Remark 7.1.7. The P-category of context renamings is the free P-Cartesian category over the set of simple types, $\mathbb{T}$.[1]

Remark 7.1.8. Construction 7.1.4 is sufficient to give a P-Cartesian product structure on the P-category of context renamings; however, it does not have particularly good computational properties when used in the framework of our definition of P-Cartesian product structure. We therefore define an alternative structure for projections that are not renamings but operations over renamings:

[1] This result is not formalised in the Rocq formalism, as a universal characterisation of the P-category of context renamings is not necessary for the results of this thesis.

- $\mathcal{R}(\Gamma, \Delta +\!\!+ \Theta) \to \mathcal{R}(\Gamma, \Delta)$; and
- $\mathcal{R}(\Gamma, \Delta +\!\!+ \Theta) \to \mathcal{R}(\Gamma, \Theta)$.

These operations are equivalent, respectively, with:

- $\rho \mapsto \pi_1 \circ \rho$; and
- $\rho \mapsto \pi_2 \circ \rho$.

7.1.2 *P-Category of Substitutions, $\mathcal{A}$*

The structure of contexts and context substitutions gives rise to a P-category of substitutions.

Definition 7.1.9 (P-Category of Substitutions). The *P-category of context substitutions*, which we denote by $\mathcal{A}$, has the following stuff and structure:

- contexts for the objects;
- the discrete P-set of context substitutions for the hom P-sets;
- composition of context substitutions for the composition of morphisms; and
- identity context substitutions for the identity morphisms.

Construction 7.1.10 (Context Concatenation Projection and Pairing for Substitutions). There are two projection substitutions from the concatenation of contexts:

- $\pi_1 \ : \ \Gamma +\!\!+ \Delta \to_{\text{sub}} \Gamma$; and
- $\pi_2 \ : \ \Gamma +\!\!+ \Delta \to_{\text{sub}} \Delta$.

These substitutions are straightforwardly defined by recursion on the structure of Δ. Moreover, there is a pairing mapping for two substitutions from the same context:

$$\frac{\sigma \ : \ \Gamma \to_{\text{sub}} \Delta \qquad \psi \ : \ \Gamma \to_{\text{sub}} \Theta}{\langle \sigma, \psi \rangle \ : \ \Gamma \to_{\text{sub}} \Delta +\!\!+ \Theta} \ .$$

This substitution is straightforwardly defined by recursion on the structure of ψ.

Fact 7.1.11. *The projection and pairing substitutions satisfy the following properties:*

- $\pi_1 \circ \langle \sigma, \psi \rangle = \sigma$*;*
- $\pi_2 \circ \langle \sigma, \psi \rangle = \psi$*; and*
- $\langle \pi_1 \circ \sigma, \pi_2 \circ \sigma \rangle = \sigma$.

Construction 7.1.12 (P-Cartesian Category of Substitutions). The P-category of context substitutions is a P-Cartesian category. The P-terminal object is given by the empty context; the P-Cartesian product is given by the concatenation of contexts, and the associated constructions given in construction 7.1.10.

Remark 7.1.13. Similarly with the situation with renamings, construction 7.1.4 is sufficient to give a P-Cartesian product structure on the P-category of context substitutions; however, it does not have particularly good computational properties when used in the framework of our definition of P-Cartesian product structure. We therefore define an alternative structure for projections that are not substitutions but operations over substitutions:

- $\mathcal{A}(\Gamma, \Delta +\!\!+ \Theta) \to \mathcal{A}(\Gamma, \Delta)$; and
- $\mathcal{A}(\Gamma, \Delta +\!\!+ \Theta) \to \mathcal{A}(\Gamma, \Theta)$.

These operations are equivalent, respectively, with:

- $\sigma \mapsto \pi_1 \circ \sigma$; and
- $\sigma \mapsto \pi_2 \circ \sigma$.

Construction 7.1.14 (Context Pre-Exponential Transpose and Evaluation for Substitutions). By abuse of notation we treat the constructions connected with pre-exponentiation by types as not essentially different from the constructions connected with pre-exponentiation by contexts — the former being a special case of the latter — even though the definition of the latter computationally uses the definition of the former. There are pre-evaluation substitutions for context pre-exponentiations both by a type and by a context:

- $\varepsilon \,:\, (\Gamma^T,\, T) \to_{\text{sub}} \Gamma$; and
- $\varepsilon \,:\, \Gamma^\Delta +\!\!+ \Delta \to_{\text{sub}} \Gamma$.

These substitutions are straightforwardly defined by induction on the structure of Γ and on the structure of Δ, respectively. Moreover, there are pre-Currying maps for context pre-exponentiation both by a type, and by a context:

$$\frac{\sigma \,:\, (\Gamma,\, T) \to_{\text{sub}} \Delta}{\sigma^* \,:\, \Gamma \to_{\text{sub}} \Delta^T} \qquad \frac{\sigma \,:\, \Gamma +\!\!+ \Delta \to_{\text{sub}} \Theta}{\sigma^* \,:\, \Gamma \to_{\text{sub}} \Theta^\Delta} \quad .$$

These substitutions are both straightforwardly defined by induction on the structure of σ.

Remark 7.1.15. Similarly with the situation with projections, construction 7.1.14 is sufficient to give a P-Cartesian pre-exponential structure on the P-category of context substitutions; however, it does not have particularly good computational properties when used in the framework of our definition of P-Cartesian pre-exponential structure. We therefore define an alternative structure for evaluations that are not substitutions but operations over substitutions:

$$\mathcal{F}(\Gamma, \Delta^\Theta) \to \mathcal{F}(\Gamma +\!\!+ \Theta, \Delta)$$

This operation is equivalent with the following.

$$\sigma \mapsto \varepsilon \circ \langle \sigma \circ \pi_1, \pi_2 \rangle$$

Construction 7.1.16 (P-Cartesian-Pre-Closed Category of Substitutions). The P-Cartesian category of context substitutions is a P-Cartesian-pre-closed category. The P-Cartesian pre-exponential is given by the exponentiation of contexts, and the associated constructions given in construction 7.1.14.

7.1.3 *P-Category of Neutral Substitutions, $\mathcal{U}$*

The facts that the composition of neutral substitutions remains neutral, and that the identity substitution is neutral, allow us to construct a P-category of substitutions that are neutral in the following definition.

Definition 7.1.17 (P-Category of Neutral Substitutions). The *P-category of neutral context substitutions*, which we denote by $\mathcal{U}$, has the following stuff and structure:

- contexts for the objects;
- the discrete sub-P-set of neutral context substitutions for the hom P-sets;

- composition of context substitutions for the composition of morphisms; and
- identity context substitutions for the identity morphisms.

The facts that the projection substitutions are neutral, and that pairing of substitutions preserves neutrality, allow us to perform the following construction.

Construction 7.1.18 (P-Cartesian Category of Neutral Substitutions)**.** The P-category of neutral context substitutions is a P-Cartesian category. The P-terminal object is given by the empty context; the P-Cartesian product is given by the concatenation of contexts, and the associated constructions given in construction 7.1.10.

7.1.4 *P-Category of Substitutions with βη-Conversion, $\mathcal{F}$*

As well as considering context substitutions discretely, we may also consider them under the equivalence relation of βη-conversion thereby simulating the quotienting of the hom-P-sets. Much of the structure from the discrete case carries forward; however, some new structure arises.

Definition 7.1.19 (P-Category of Substitutions with βη-Convertibility)**.** The *P-category of context substitutions*, which we denote by $\mathcal{F}$, has the following stuff and structure:

- contexts for the objects;
- the P-set of context substitutions under βη-conversion for the hom P-sets;
- composition of context substitutions for the composition of morphisms; and
- identity context substitutions for the identity morphisms.

Fact 7.1.20. *Pairing of context substitutions respects βη-conversion;* i.e.*, the following property holds.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \wedge \Gamma \vdash_{\text{sub}} \psi_1 \equiv_{\beta\eta} \psi_2 \,:\, \Theta \Rightarrow \Gamma \vdash_{\text{sub}} \langle \sigma_1, \psi_1 \rangle \equiv_{\beta\eta} \langle \sigma_2, \psi_2 \rangle \,:\, \Delta +\!\!+ \Theta$$

Remark 7.1.21. The alternative operations described in remark 7.1.13 also respect βη-conversion.

Construction 7.1.22 (P-Cartesian Category of Substitutions with βη-Convertibility)**.** The P-category of context substitutions with βη-conversion is a P-Cartesian category. The P-terminal object is given by the empty context; the P-Cartesian product is given by the concatenation of contexts, and the associated constructions given in construction 7.1.10.

Fact 7.1.23. *Pre-Currying of context substitutions respects βη-conversion;* i.e.*, the following property holds.*

$$\Gamma +\!\!+ \Delta \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Theta \Rightarrow \Gamma \vdash_{\text{sub}} \sigma_1^* \equiv_{\beta\eta} \sigma_2^* \,:\, \Theta^{\Delta}$$

Remark 7.1.24. The alternative operation described in remark 7.1.15 also respects βη-conversion.

Fact 7.1.25. *The pre-evaluation substitutions and pre-Currying of substitutions satisfy the following properties:*

- $\Gamma, T \vdash_{\text{sub}} \varepsilon \circ ((\sigma^*)_T, \underline{i_0}) \equiv_{\beta\eta} \sigma \,:\, \Delta$*;*
- $\Gamma +\!\!+ \Theta \vdash_{\text{sub}} \varepsilon \circ \langle \sigma^* \circ \pi_1, \pi_2 \rangle \equiv_{\beta\eta} \sigma \,:\, \Delta$*;*
- $\Gamma \vdash_{\text{sub}} (\varepsilon \circ (\sigma_T, \underline{i_0}))^* \equiv_{\beta\eta} \sigma \,:\, \Delta^T$*; and*
- $\Gamma \vdash_{\text{sub}} (\varepsilon \circ \langle \sigma \circ \pi_1, \pi_2 \rangle)^* \equiv_{\beta\eta} \sigma \,:\, \Delta^{\Theta}$.

Remark 7.1.26. Note that the properties in fact 7.1.25 hold only for $\beta\eta$-conversion, and not equality of substitutions, as they require the β-reduction laws and η-expansion laws for terms. Moreover, these properties make the pre-evaluation substitutions and pre-Currying of substitutions into *bona fide* evaluation substitutions and Currying of substitutions when considered with respect to the P-category of context substitutions with $\beta\eta$-conversion.

Construction 7.1.27 (P-Cartesian-Closed Category of Substitutions with $\beta\eta$-Conversion)**.** The P-Cartesian category of context substitutions with $\beta\eta$-conversion is a P-Cartesian-closed category. The P-Cartesian exponential is given by the exponentiation of contexts, and the associated constructions given in construction 7.1.14.

Construction 7.1.28 (Inclusion and Quotient P-Functors)**.** There are identity-on-objects P-functors:

- $i_1 : \mathcal{R} \to \mathcal{U}$;
- $i_2 : \mathcal{U} \to \mathcal{A}$; and
- $j : \mathcal{A} \to \mathcal{F}$.

The first, i_1, includes renamings into neutral substitutions by embedding de Bruijn indices as variable references and as such is faithful; the second, i_2, includes neutral substitutions into all substitutions forgetting the neutrality of the substitutions and as such is faithful; the third, j, is a quotient mapping and is computationally identity on morphisms and as such is full. All three P-functors are P-Cartesian functors. Moreover, the third P-functor, j, is a P-Cartesian-pre-closed functor.

We denote by i_3 the composition $i_2 \circ i_1$, by j_2 the composition $j \circ i_2$, and by j_3 the composition $j \circ i_3 = j \circ i_2 \circ i_1$. These P-functors are also P-Cartesian, as the composition of two P-Cartesian functors is a P-Cartesian functor.

7.2 Freeness of $\mathcal{F}$

In this section we establish that the P-category of context substitutions with $\beta\eta$-conversion, $\mathcal{F}$, is the free P-Cartesian-closed category over the generating base type, ι. Establishing the freeness requires constructing and establishing the following:

- a strictly pointed P-Cartesian-closed interpretation functor into any pointed P-Cartesian-closed category;
- a lifting of P-isomorphisms in any P-Cartesian-closed category to a P-natural isomorphism between interpretation P-functors;
- a strictly pointed P-natural isomorphism between the interpretation P-functor and any strictly pointed P-Cartesian-closed functor; and
- the uniqueness of the P-natural isomorphism.

The construction of an interpretation P-functor requires interpreting all the structures used to build the P-category of context substitutions with $\beta\eta$-conversion.

Remark 7.2.1. In the sequel, we fix a P-Cartesian-closed category, $\mathbb{C}$, with a chosen object, c. Furthermore, we denote interpretation by $[\![-]\!]_c$, although often we omit the subscript c. If it is evident what the chosen object is then we may also denote interpretation into $\mathbb{C}$ by $\mathbb{C}[\![-]\!]$.

Moreover, we fix a second chosen object c', and a pair of P-well-defined morphisms, $f : c \to c'$ and $g : c' \to c$, such that $f \circ g \sim \mathrm{id}_{c'}$ and $g \circ f \sim \mathrm{id}_c$. We denote the lifting of interpretation by this pair of morphisms by $[\![-]\!]_{f,g}$. Furthermore, where necessary, we fix a third chosen object c'', and a pair of P-well-defined morphisms, $h : c' \to c''$ and $k : c'' \to c'$, such that $h \circ k \sim \mathrm{id}_{c''}$ and $k \circ h \sim \mathrm{id}_{c'}$. The lifting by $f : c \to c'$ and $g : c' \to c$ is a morphism:

$$[\![-]\!]_{f,g} : [\![-]\!]_c \to [\![-]\!]_{c'} \ .$$

We first need to interpret types as the first structure not dependent on any other structure.

Construction 7.2.2 (Interpretation of Types). The set of types, $\mathbb{T}$, can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}[\![\iota]\!] &\triangleq c \\ [\![T_1 \to T_2]\!] &\triangleq [\![T_1]\!] \Rightarrow [\![T_2]\!]\end{aligned}$$

Construction 7.2.3 (Lifting of the Interpretation of Types). The interpretation of types lifts by the following recursive definition.

$$\begin{aligned}[\![\iota]\!]_{f,g} &\triangleq f \\ [\![T_1 \to T_2]\!]_{f,g} &\triangleq [\![T_1]\!]_{g,f} \Rightarrow [\![T_2]\!]_{f,g}\end{aligned}$$

These morphisms are P-well-defined. Note that contravariance requires that we swap the morphisms f and g.

Fact 7.2.4. *The following properties hold, for any type T, of the lifting of the interpretation of types:*

- $[\![T]\!]_{f,g} \circ [\![T]\!]_{g,f} \sim \mathrm{id}$*;*
- $[\![T]\!]_{\mathrm{id},\mathrm{id}} \sim \mathrm{id}$*; and*
- $[\![T]\!]_{h \circ f, g \circ k} \sim [\![T]\!]_{h,k} \circ [\![T]\!]_{f,g}$.

The interpretation of contexts (as finite sequences of types) would seem to follow directly using the P-Cartesian product structure of the interpretation P-category; however, we complicate the interpretation of contexts to improve the computational strictness and reduce the need for isomorphisms.

Construction 7.2.5 (Interpretation of Contexts). Contexts can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}[\![\bullet]\!] &\triangleq \top \\ [\![\bullet, T]\!] &\triangleq [\![T]\!] \\ [\![\Gamma, T', T]\!] &\triangleq [\![\Gamma, T']\!] \times [\![T]\!]\end{aligned}$$

Remark 7.2.6. Note that, following Čubrić et al. (1998), we give a three-clause definition in construction 7.2.5 that results in stricter computation for singleton contexts, and also does not result in a trailing product with the terminal object when interpreting a non-empty context. We thus only have the following as an isomorphism.

$$[\![\Gamma, T]\!] \cong [\![\Gamma]\!] \times [\![T]\!]$$

Construction 7.2.7 (Context Interpretation Projection and Pairing). To alleviate the complexity of the three-clause definition in construction 7.2.5 and the resultant isomorphism between $[\![\Gamma, T]\!]$ and $[\![\Gamma]\!] \times [\![T]\!]$, we define the following P-well-defined projection and pairing maps:

- $\varpi_1 : [\![\Gamma, T]\!] \to [\![\Gamma]\!]$;
- $\varpi_2 : [\![\Gamma, T]\!] \to [\![T]\!]$; and
- $\langle\!\langle -, = \rangle\!\rangle : \mathbb{C}(x, [\![\Gamma]\!]) \to \mathbb{C}(x, [\![T]\!]) \to \mathbb{C}(x, [\![\Gamma, T]\!])$.

These projection and pairing maps satisfy the expected equations.

Construction 7.2.8 (Lifting of the Interpretation of Contexts). The interpretation of contexts lifts by the following recursive definition.

$$\begin{aligned}\llbracket \bullet \rrbracket_{f,g} &\triangleq \mathrm{id}_{\top} \\ \llbracket \Gamma,\, T \rrbracket_{f,g} &\triangleq \langle\!\langle \llbracket \Gamma \rrbracket_{f,g} \circ \varpi_1, \llbracket T \rrbracket_{f,g} \circ \varpi_2 \rangle\!\rangle\end{aligned}$$

These morphisms are P-well-defined.

Fact 7.2.9. *The following properties hold, for any context* Γ*, of the lifting of the interpretation of contexts:*

- $\llbracket \Gamma \rrbracket_{f,g} \circ \llbracket \Gamma \rrbracket_{g,f} \sim \mathrm{id}$*;*
- $\llbracket \Gamma \rrbracket_{\mathrm{id},\mathrm{id}} \sim \mathrm{id}$*; and*
- $\llbracket \Gamma \rrbracket_{h \circ f, g \circ k} \sim \llbracket \Gamma \rrbracket_{h,k} \circ \llbracket \Gamma \rrbracket_{f,g}$*.*

Having interpreted contexts as objects within our interpretation P-category (and types as the building blocks for the interpretation of contexts), we now proceed to interpreting type-theoretic structure as morphisms within our interpretation P-category. Just as types are the building blocks for contexts, so are de Bruijn indices and terms the building blocks for context renamings and context substitutions respectively. Thus, all these type-theoretic structures are interpreted as morphisms. Within the setting of P-category theory, thought should be given as to when any given morphism should be P-well-defined; since the source of the interpretation morphisms is well-defined type-theoretic structure, the morphisms should always be P-well-defined. It is crucial that we have used well-defined, *i.e.* well-scoped and well-typed, type-theoretic structure as this allows us to inhabit the underlying carrier type of the hom P-sets of the P-category in which we are interpreting the type-theoretic structure.

Construction 7.2.10 (Interpretation of de Bruijn Indices). de Bruijn indices can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}\llbracket \Gamma,\, T \vdash i_0 \,:\, T \rrbracket &\triangleq \varpi_2 && :\, \llbracket \Gamma,\, T \rrbracket \to \llbracket T \rrbracket \\ \llbracket \Gamma,\, T' \vdash i_{n+1} \,:\, T \rrbracket &\triangleq \llbracket \Gamma \vdash i_n \,:\, T \rrbracket \circ \varpi_1 && :\, \llbracket \Gamma,\, T' \rrbracket \to \llbracket T \rrbracket\end{aligned}$$

These morphisms are P-well-defined.

Fact 7.2.11. *The following naturality property holds of the interpretation of de Bruijn indices.*

$$\llbracket T \rrbracket_{f,g} \circ \llbracket \Gamma \vdash i_n \,:\, T \rrbracket_c \sim \llbracket \Gamma \vdash i_n \,:\, T \rrbracket_{c'} \circ \llbracket \Gamma \rrbracket_{f,g}$$

Construction 7.2.12 (Interpretation of Context Renamings). Context renamings can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}\llbracket \Gamma \vdash_{\mathrm{ren}} \bullet \,:\, \bullet \rrbracket &\triangleq !_{\llbracket \Gamma \rrbracket} && :\, \llbracket \Gamma \rrbracket \to \llbracket \bullet \rrbracket \\ \llbracket \Gamma \vdash_{\mathrm{ren}} \rho,\, i_n \,:\, \Delta,\, T \rrbracket &\triangleq \langle\!\langle \llbracket \Gamma \vdash_{\mathrm{ren}} \rho \,:\, \Delta \rrbracket, \llbracket \Gamma \vdash i_n \,:\, T \rrbracket \rangle\!\rangle && :\, \llbracket \Gamma \rrbracket \to \llbracket \Delta,\, T \rrbracket\end{aligned}$$

These morphisms are P-well-defined.

Fact 7.2.13. *The following properties hold of the interpretation of context renamings:*

- $\llbracket \Gamma,\, T \vdash_{\mathrm{ren}} \rho_T \,:\, \Delta \rrbracket \sim \llbracket \Gamma \vdash_{\mathrm{ren}} \rho \,:\, \Delta \rrbracket \circ \varpi_1$*; and*
- $\llbracket \Gamma \vdash_{\mathrm{ren}} \mathrm{id}_\Gamma \,:\, \Gamma \rrbracket \sim \mathrm{id}_{\llbracket \Gamma \rrbracket}$*.*

Fact 7.2.14. *The following naturality property holds of the interpretation of context renamings.*

$$[\![\Delta]\!]_{f,g} \circ [\![\Gamma \vdash \rho \,:\, \Delta]\!]_c \sim [\![\Gamma \vdash \rho \,:\, \Delta]\!]_{c'} \circ [\![\Gamma]\!]_{f,g}$$

Construction 7.2.15 (Interpretation of Terms)**.** Terms can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}
[\![\Gamma \vdash \underline{i_n} \,:\, T]\!] &\triangleq [\![\Gamma \vdash i_n \,:\, T]\!] && : [\![\Gamma]\!] \to [\![T]\!] \\
[\![\Gamma \vdash t_1\, t_2 \,:\, T_2]\!] &\triangleq \varepsilon \circ \langle [\![\Gamma \vdash t_1 \,:\, T_1 \to T_2]\!], [\![\Gamma \vdash t_2 \,:\, T_1]\!] \rangle && : [\![\Gamma]\!] \to [\![T_2]\!] \\
[\![\Gamma \vdash \lambda_T.t \,:\, T \to T']\!] &\triangleq ([\![\Gamma, T \vdash t \,:\, T']\!] \circ \langle\!\langle \pi_1, \pi_2 \rangle\!\rangle)^* && : [\![\Gamma]\!] \to [\![T \to T']\!]
\end{aligned}$$

These morphisms are P-well-defined.

Fact 7.2.16. *The following naturality property holds of the interpretation of terms.*

$$[\![T]\!]_{f,g} \circ [\![\Gamma \vdash t \,:\, T]\!]_c \sim [\![\Gamma \vdash t \,:\, T]\!]_{c'} \circ [\![\Gamma]\!]_{f,g}$$

Construction 7.2.17 (Interpretation of Context Substitutions)**.** Context substitutions can be interpreted in $\mathbb{C}$ by the following recursive definition.

$$\begin{aligned}
[\![\Gamma \vdash_{\text{sub}} \bullet \,:\, \bullet]\!] &\triangleq !_{[\![\Gamma]\!]} && : [\![\Gamma]\!] \to [\![\bullet]\!] \\
[\![\Gamma \vdash_{\text{sub}} \sigma,\, t \,:\, \Delta,\, T]\!] &\triangleq \langle\!\langle [\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!], [\![\Gamma \vdash t \,:\, T]\!] \rangle\!\rangle && : [\![\Gamma]\!] \to [\![\Delta,\, T]\!]
\end{aligned}$$

These morphisms are P-well-defined.

Fact 7.2.18. *The following properties hold of the interpretation of context substitutions:*

- $[\![\Gamma,\, T \vdash_{\text{sub}} \sigma_T \,:\, \Delta]\!] \sim [\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!] \circ \varpi_1$*; and*
- $[\![\Gamma \vdash_{\text{sub}} \text{id}_\Gamma \,:\, \Gamma]\!] \sim \text{id}_{[\![\Gamma]\!]}$.

Fact 7.2.19. *The following naturality property holds of the interpretation of context substitutions.*

$$[\![\Delta]\!]_{f,g} \circ [\![\Gamma \vdash \sigma \,:\, \Delta]\!]_c \sim [\![\Gamma \vdash \sigma \,:\, \Delta]\!]_{c'} \circ [\![\Gamma]\!]_{f,g}$$

Fact 7.2.20. *The following properties hold of the action of renamings both on de Bruijn indices and on terms, and of composition of context renamings:*

- $[\![\Gamma \vdash i_n[\rho] \,:\, T]\!] \sim [\![\Delta \vdash i_n \,:\, T]\!] \circ [\![\Gamma \vdash_{\text{ren}} \rho \,:\, \Delta]\!]$*;*
- $[\![\Gamma \vdash t[\rho] \,:\, T]\!] \sim [\![\Delta \vdash t \,:\, T]\!] \circ [\![\Gamma \vdash_{\text{ren}} \rho \,:\, \Delta]\!]$*; and*
- $[\![\Gamma \vdash_{\text{ren}} \rho \circ \varphi \,:\, \Theta]\!] \sim [\![\Delta \vdash_{\text{ren}} \rho \,:\, \Theta]\!] \circ [\![\Gamma \vdash_{\text{ren}} \varphi \,:\, \Delta]\!]$.

Fact 7.2.21. *The following properties hold of the action of substitution both on de Bruijn indices and on terms:*

- $[\![\Gamma \vdash i_n[\sigma] \,:\, T]\!] \sim [\![\Delta \vdash i_n \,:\, T]\!] \circ [\![\Gamma \vdash_{\text{sub}} \rho \,:\, \Delta]\!]$*; and*
- $[\![\Gamma \vdash t[\sigma] \,:\, T]\!] \sim [\![\Delta \vdash t \,:\, T]\!] \circ [\![\Gamma \vdash_{\text{sub}} \rho \,:\, \Delta]\!]$.

Fact 7.2.22 (Soundness of Term Interpretation)**.** *The following property holds of the interpretation of terms.*

$$\Gamma \vdash t \equiv_{\beta\eta} t' \,:\, T \Rightarrow [\![\Gamma \vdash t \,:\, T]\!] \sim [\![\Gamma \vdash t' \,:\, T]\!]$$

Fact 7.2.23 (Soundness of Context Substitution Interpretation). *The following property holds of the interpretation of context substitutions.*

$$\Gamma \vdash_{\text{sub}} \sigma \equiv_{\beta\eta} \sigma' \,:\, \Delta \Rightarrow [\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!] \sim [\![\Gamma \vdash_{\text{sub}} \sigma' \,:\, \Delta]\!]$$

Fact 7.2.24. *The following property holds of composition of context substitutions.*

$$[\![\Gamma \vdash_{\text{sub}} \sigma \circ \psi \,:\, \Theta]\!] \sim [\![\Delta \vdash_{\text{sub}} \sigma \,:\, \Theta]\!] \circ [\![\Gamma \vdash_{\text{sub}} \psi \,:\, \Delta]\!]$$

Remark 7.2.25. The interpretations both of context renamings and of context substitutions respect the hetero-compositions of remark 6.2.15.

We have now defined the interpretation of all the structures of simple type theory, and stated appropriate properties to establish that the interpretation construction assembles into a P-functor.

Construction 7.2.26 (Interpretation P-Functor). For any pointed P-Cartesian-closed category, $\mathbb{C}$, with chosen object c, we have an interpretation P-functor.

$$[\![-]\!]_c \,:\, \mathcal{F} \to \mathbb{C}$$

Construction 7.2.27 (Lifting of the Interpretation P-Functor). For any P-Cartesian-closed category, $\mathbb{C}$, with a pair of chosen objects c and c', and a pair of P-well-defined morphisms $f \,:\, c \to c'$ and $g \,:\, c' \to c$, such that $f \circ g \sim \mathrm{id}$ and $g \circ f \sim \mathrm{id}$, we have a P-natural isomorphism whose components are given by the lifting of the interpretation of contexts.

$$[\![-]\!]_{f,g} \,:\, [\![-]\!]_c \cong [\![-]\!]_{c'}$$

Construction 7.2.28. The P-well-defined canonical map,

$$\langle [\![\Gamma +\!\!+ \Delta \vdash_{\text{sub}} \pi_1 \,:\, \Gamma]\!], [\![\Gamma +\!\!+ \Delta \vdash_{\text{sub}} \pi_2 \,:\, \Delta]\!] \rangle \,:\, [\![\Gamma +\!\!+ \Delta]\!] \to [\![\Gamma]\!] \times [\![\Delta]\!] \ ,$$

has a P-well-defined inverse, $p_{\Gamma;\Delta}$, recursively constructed by the sequel.

$$\begin{aligned} p_{\Gamma;\bullet} &\triangleq \pi_1 && : [\![\Gamma]\!] \times [\![\bullet]\!] \to [\![\Gamma +\!\!+ \bullet]\!] \\ p_{\Gamma;(\Delta,\, T)} &\triangleq \langle\!\langle p_{\Gamma;\Delta} \circ \langle \pi_1, \varpi_1 \circ \pi_2 \rangle, \varpi_2 \circ \pi_2 \rangle\!\rangle && : [\![\Gamma]\!] \times [\![\Delta,\, T]\!] \to [\![\Gamma +\!\!+ \Delta, T]\!] \end{aligned}$$

Remark 7.2.29. Due to the three-clause definition in construction 7.2.5, we also define an inverse that has stricter computational properties than that afforded by using construction 7.2.7. We use this alternative inverse in our Rocq formalism. This exemplifies well the observation in remark 4.2.7.

Construction 7.2.30 (Interpretation P-Cartesian Functor). The interpretation P-functor is a P-Cartesian functor. This follows from its strict preservation of the P-terminal object, and from the P-isomorphism defined in construction 7.2.28.

Construction 7.2.31. The P-well-defined canonical map, $([\![\Gamma^T,\, T \vdash_{\text{sub}} \varepsilon \,:\, \Gamma]\!] \circ \langle\!\langle \pi_1, \pi_2 \rangle\!\rangle)^* \,:\, [\![\Gamma^T]\!] \to ([\![T]\!] \Rightarrow [\![\Gamma]\!])$, has a P-well-defined inverse, $e_{\Gamma;T}$, recursively constructed by the sequel.

$$\begin{aligned} e_{\bullet;T} &\triangleq ! && : ([\![T]\!] \Rightarrow [\![\bullet]\!]) \to [\![\bullet]\!] \\ e_{\Gamma,\, T';T} &\triangleq \langle\!\langle e_{\Gamma;T} \circ (\varpi_1 \circ \varepsilon)^*, (\varpi_2 \circ \varepsilon)^* \rangle\!\rangle && : ([\![T]\!] \Rightarrow [\![\Gamma,\, T']\!]) \to [\![(\Gamma,\, T')^T]\!] \end{aligned}$$

Construction 7.2.32. The P-well-defined canonical map, $([\![\Gamma^\Delta +\!\!+ \Delta \vdash_{\text{sub}} \varepsilon \,:\, \Gamma]\!] \circ p)^* \,:\, [\![\Gamma^\Delta]\!] \to ([\![\Delta]\!] \Rightarrow [\![\Gamma]\!])$, has a P-well-defined inverse, $e_{\Gamma;\Delta}$, recursively constructed by the sequel.

$$\begin{aligned} e_{\Gamma;\bullet} &\triangleq \varepsilon \circ \langle \mathrm{id}, ! \rangle && : ([\![\bullet]\!] \Rightarrow [\![\Gamma]\!]) \to [\![\Gamma^{(\bullet)}]\!] \\ e_{\Gamma;\Delta,\, T} &\triangleq e_{\Gamma;\Delta} \circ (e_{\Gamma;T} \circ (\varepsilon \circ \langle \pi_1 \circ \pi_1, \langle\!\langle \pi_2 \circ \pi_1, \pi_2 \rangle\!\rangle)^*)^* && : ([\![\Delta,\, T]\!] \Rightarrow [\![\Gamma]\!]) \to [\![\Gamma^{\Delta,\, T}]\!] \end{aligned}$$

Remark 7.2.33. Due to the three-clause definition in construction 7.2.5, we also define inverses that have stricter computational properties than that afforded by using construction 7.2.7. We use this alternative inverse in our Rocq formalism.

Construction 7.2.34 (Interpretation p-Cartesian-Closed Functor)**.** The interpretation p-Cartesian functor is a p-Cartesian-closed functor. This follows from the p-isomorphism defined in construction 7.2.32.

Remark 7.2.35. In the sequel, we fix a p-Cartesian-closed functor, $F : \mathcal{F} \to \mathbb{C}$. This induces a canonical choice of an object of $\mathbb{C}$ to serve as c: *viz.*, $F_0(\bullet, \iota)$.

Construction 7.2.36 (Quote and Unquote on Types)**.** We have a pair of type-indexed families of p-well-defined morphisms in $\mathbb{C}$:

- $q_T : [\![\bullet, T]\!] \to F_0(\bullet, T)$; and
- $u_T : F_0(\bullet, T) \to [\![\bullet, T]\!]$.

They are constructed mutually recursively by the sequel.

$$\begin{aligned} q_\iota &\triangleq \mathsf{id} && : [\![\bullet, \iota]\!] \to F_0(\bullet, \iota) \\ q_{T_1 \to T_2} &\triangleq e_{F;\bullet,\, T_1;\bullet,\, T_2} \circ (u_{T_1} \Rightarrow q_{T_2}) && : [\![\bullet, T_1 \to T_2]\!] \to F_0(\bullet, T_1 \to T_2) \end{aligned}$$

$$\begin{aligned} u_\iota &\triangleq \mathsf{id} && : F_0(\bullet, \iota) \to [\![\bullet, \iota]\!] \\ u_{T_1 \to T_2} &\triangleq (q_{T_1} \Rightarrow u_{T_2}) \circ (F_1(\varepsilon) \circ p_{F;\bullet,\, T_1 \to T_2;\bullet,\, T_1})^* && : [\![\bullet, T_1 \to T_2]\!] \to F_0(\bullet, T_1 \to T_2) \end{aligned}$$

The construction uses the following canonical maps: p witnessing that F is p-Cartesian; ε witnessing (one-half of) $\mathbb{C}$ being p-Cartesian-closed; and e witnessing that F is p-Cartesian-closed.

Remark 7.2.37. We denote the morphisms q and u as such, as they are precisely the 'quote' and 'unquote' maps familiar from normalisation by evaluation. Although standard accounts of normalisation by evaluation require definitions of normal and neutral terms, we avoid needing such definitions here.

Fact 7.2.38. *The two families of morphisms q_T and u_T are inverses with each other. This follows straightforwardly from induction over the structure of T.*

Construction 7.2.39 (Quote and Unquote on Contexts)**.** We have a pair of context-indexed families of p-well-defined morphisms in $\mathbb{C}$:

- $q_\Gamma : [\![\Gamma]\!] \to F_0(\Gamma)$; and
- $u_\Gamma : F_0(\Gamma) \to [\![\Gamma]\!]$.

They are constructed recursively by the sequel.

$$\begin{aligned} q_\bullet &\triangleq t_F && : [\![\bullet]\!] \to F_0(\bullet) \\ q_{\Gamma,\, T} &\triangleq p_{F;\Gamma;\bullet,\, T} \circ \langle q_\Gamma \circ \varpi_1, q_T \circ \varpi_2 \rangle && : [\![\Gamma, T]\!] \to F_0(\Gamma, T) \end{aligned}$$

$$\begin{aligned} u_\bullet &\triangleq ! && : F_0(\bullet) \to [\![\bullet]\!] \\ u_{\Gamma,\, T} &\triangleq \langle\!\langle u_\Gamma \circ F_1((\mathrm{id})_T), u_T \circ F_1(\bullet, \underline{i_0}) \rangle\!\rangle && : F_0(\Gamma, T) \to [\![\Gamma, T]\!] \end{aligned}$$

Fact 7.2.40. *The two families of morphisms q_Γ and u_Γ are inverses with each other. This follows straightforwardly from induction over the structure of Γ.*

Fact 7.2.41. *The two families of morphisms u_T and u_Γ satisfy the following mutual naturality properties:*

- $[\![\Gamma \vdash i_n \,:\, T]\!] \circ u_\Gamma \sim u_T \circ F_1(\Gamma \vdash_{\text{sub}} \bullet,\, \underline{i_n} \,:\, \bullet,\, T)$*; and*
- $[\![\Gamma \vdash t \,:\, T]\!] \circ u_\Gamma \sim u_T \circ F_1(\Gamma \vdash_{\text{sub}} \bullet,\, t \,:\, \bullet,\, T)$.

From this we can conclude that the morphism family u_Γ satisfies the following naturality property.

$$[\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!] \circ u_\Gamma \sim u_\Delta \circ F_1(\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta)$$

Remark 7.2.42. We only need to establish naturality for one of q and u, appealing to their being inverses to conclude that the other must also be natural. We found it much easier to derive naturality for u than for q, hence our choice only to formalise explicitly naturality for u.

Construction 7.2.43 (Quote and Unquote P-Natural Isomorphism). The two families of morphisms q_Γ and u_Γ form a P-natural isomorphism.

$$q \,:\, [\![-]\!]_{F_0(\bullet,\, \iota)} \mathrel{\tilde{\Leftrightarrow}} F \,:\, u$$

Fact 7.2.44. *The two families of morphisms q_T and u_T are uniquely defined amongst all strictly-pointed P-natural isomorphisms;* i.e.*, they satisfy the following property.*

$$\bigforall_{\alpha \,:\, [\![-]\!]_{F_0(\bullet,\, \iota)} \tilde{\Leftrightarrow} F} \alpha_{\bullet,\, \iota} \sim \mathrm{id} \Rightarrow \bigforall_{T \,:\, \mathbb{T}} \alpha_{\bullet,\, T} \sim q_T \wedge \alpha^{-1}_{\bullet,\, T} \sim u_T$$

Fact 7.2.45. *The family of morphisms u_Γ is uniquely defined amongst all strictly-pointed P-natural isomorphisms;* i.e.*, it satisfies the following property.*

$$\bigforall_{\alpha \,:\, [\![-]\!]_{F_0(\bullet,\, \iota)} \Leftrightarrow F} \alpha_{\bullet,\, \iota} \sim \mathrm{id} \Rightarrow \bigforall_{\Gamma \,:\, \mathcal{F}_0} \alpha^{-1}_\Gamma \sim u_\Gamma$$

Remark 7.2.46. We only need to establish uniqueness for one of q and u, appealing to their being inverses to conclude that the other must also be uniquely defined. We found it much easier to derive uniqueness for u than for q, hence our choice only to formalise explicitly uniqueness for u.

Theorem 7.2.47 (Freeness of $\mathcal{F}$). *The P-category of context substitutions with $\beta\eta$-convertibility, $\mathcal{F}$, is the free P-Cartesian-closed category over the generating base type, ι. In the language of P-bicategory theory, the pointed P-Cartesian-closed category, $\mathcal{F}$ with point $(\bullet,\, \iota)$, is P-pseudoinitial in the P-bicategory of pointed P-Cartesian-closed categories, pointed P-Cartesian-closed functors, and pointed P-natural isomorphisms.*

7.3 Universal Property of $j : \mathcal{A} \to \mathcal{F}$

In this section, we describe a universal property that the P-category of context substitutions satisfies. We choose this universal property, rather than the (perhaps) obvious universal property of $\mathcal{A}$ being initial in some P-bicategory of P-Cartesian-pre-closed categories, as it allows us to relate the structures of the two P-categories of context substitutions.

Remark 7.3.1. In the sequel, we fix a P-Cartesian-pre-closed functor, $F \,:\, \mathcal{A} \to \mathbb{C}$. This induces a canonical choice of an object of $\mathbb{C}$ to serve as c: namely, $F_0(\bullet,\, \iota)$.

Construction 7.3.2 (Quote and Unquote on Types). We have a pair of type-indexed families of P-well-defined morphisms in $\mathbb{C}$:

- $\tilde{q}_T \,:\, [\![\bullet,\, T]\!] \to F_0(\bullet,\, T)$; and
- $\tilde{u}_T \,:\, F_0(\bullet,\, T) \to [\![\bullet,\, T]\!]$.

They are constructed mutually recursively by the sequel.

$$\begin{aligned}
\tilde{q}_{\iota} &\triangleq \mathrm{id} && : [\![\bullet, \iota]\!] \to F_0(\bullet, \iota) \\
\tilde{q}_{T_1 \to T_2} &\triangleq \tilde{e}_{F;\bullet, T_1;\bullet, T_2} \circ (\tilde{u}_{T_1} \Rightarrow \tilde{q}_{T_2}) && : [\![\bullet, T_1 \to T_2]\!] \to F_0(\bullet, T_1 \to T_2)
\end{aligned}$$

$$\begin{aligned}
\tilde{u}_{\iota} &\triangleq \mathrm{id} && : F_0(\bullet, \iota) \to [\![\bullet, \iota]\!] \\
\tilde{u}_{T_1 \to T_2} &\triangleq (\tilde{q}_{T_1} \Rightarrow \tilde{u}_{T_2}) \circ (F_1(\tilde{\varepsilon}) \circ p_{F;\bullet, T_1 \to T_2;\bullet, T_1})^* && : [\![\bullet, T_1 \to T_2]\!] \to F_0(\bullet, T_1 \to T_2)
\end{aligned}$$

The construction uses the following canonical maps: p witnessing that F is P-Cartesian; $\tilde{\varepsilon}$ witnessing (one-half of) $\mathbb{C}$ being P-Cartesian-pre-closed; and $\tilde{e}$ witnessing that F is P-Cartesian-pre-closed.

Remark 7.3.3. We denote the morphisms $\tilde{q}$ and $\tilde{u}$ as such, as they are precisely the quote and unquote maps familiar from normalisation by evaluation. Although standard accounts of normalisation by evaluation require definitions of normal and neutral terms, we avoid needing such definitions here.

Remark 7.3.4. The two families of morphisms $\tilde{q}_T$ and $\tilde{u}_T$ are not necessarily inverses one with each other, as $\tilde{e}$ is not necessarily an inverse to the canonical morphism.

Fact 7.3.5. *For any two P-Cartesian-closed categories, $\mathbb{C}$ and $\mathbb{D}$, any P-Cartesian-closed functor, $F : \mathbb{C} \to \mathbb{D}$, and any P-Cartesian-pre-closed functor, $G : \mathcal{A} \to \mathbb{C}$, we have that q, $\tilde{q}$, u, and $\tilde{u}$ satisfy the following properties for any type, T:*

- $\tilde{q}_{G \circ F;T} \sim F_1(\tilde{q}_{G;T}) \circ q_{F \circ [\![-]\!];T}$, *and*
- $\tilde{u}_{G \circ F;T} \sim u_{F \circ [\![-]\!];T} \circ F_1(\tilde{u}_{G;T})$;

where the interpretation functor $[\![-]\!]$ is induced by $G_0(\bullet, \iota)$. This follows from fact 7.3.15.

Fact 7.3.6. *For any P-Cartesian-closed category, $\mathbb{C}$, and any P-Cartesian-closed functor, $F : \mathcal{F} \to \mathbb{C}$, we have that q, $\tilde{q}$, u, and $\tilde{u}$ satisfy the following properties for any type, T:*

- $\tilde{q}_{F \circ j;T} \sim q_{F;T}$;
- $\tilde{u}_{F \circ j;T} \sim u_{F;T}$.

This follows from fact 7.3.15.

Remark 7.3.7. The two preceding facts allow us informally to deduce that the two families of morphisms q_T and $\tilde{q}_T$ perform the same computation, and that the two families of morphisms u_T and $\tilde{u}_T$ perform the same computation.

Construction 7.3.8 (Quote and Unquote on Contexts)**.** We have a pair of context-indexed families of P-well-defined morphisms in $\mathbb{C}$:

- $\tilde{q}_{\Gamma} : [\![\Gamma]\!] \to F_0(\Gamma)$; and
- $\tilde{u}_{\Gamma} : F_0(\Gamma) \to [\![\Gamma]\!]$.

They are constructed recursively by the sequel.

$$\begin{aligned}
\tilde{q}_{\bullet} &\triangleq t_F && : [\![\bullet]\!] \to F_0(\bullet) \\
\tilde{q}_{\Gamma, T} &\triangleq p_{F;\Gamma;\bullet, T} \circ \langle \tilde{q}_{\Gamma} \circ \varpi_1, \tilde{q}_T \circ \varpi_2 \rangle && : [\![\Gamma, T]\!] \to F_0(\Gamma, T)
\end{aligned}$$

$$\begin{aligned}
\tilde{u}_{\bullet} &\triangleq ! && : F_0(\bullet) \to [\![\bullet]\!] \\
\tilde{u}_{\Gamma, T} &\triangleq \langle\!\langle \tilde{u}_{\Gamma} \circ F_1((\mathrm{id})_T), \tilde{u}_T \circ F_1(\bullet, \underline{i_0}) \rangle\!\rangle && : F_0(\Gamma, T) \to [\![\Gamma, T]\!]
\end{aligned}$$

Fact 7.3.9. *For any two P-Cartesian-closed categories, $\mathbb{C}$ and $\mathbb{D}$, any P-Cartesian-closed functor, $F : \mathbb{C} \to \mathbb{D}$, and any P-Cartesian-pre-closed functor, $G : \mathcal{A} \to \mathbb{C}$, we have that q, $\tilde{q}$, u, and $\tilde{u}$ satisfy the following properties for any context, Γ:*

- $\tilde{q}_{G \circ F;\Gamma} \sim F_1(\tilde{q}_{G;\Gamma}) \circ q_{F \circ [\![-]\!];\Gamma}$, *and*
- $\tilde{u}_{G \circ F;\Gamma} \sim u_{F \circ [\![-]\!];\Gamma} \circ F_1(\tilde{u}_{G;\Gamma})$;

where the interpretation functor $[\![-]\!]$ is induced by $G_0(\bullet, \iota)$. This follows from fact 7.3.16.

Fact 7.3.10. *For any P-Cartesian-closed category, $\mathbb{C}$, and any P-Cartesian-closed functor, $F : \mathcal{F} \to \mathbb{C}$, we have that q, $\tilde{q}$, u, and $\tilde{u}$ satisfy the following properties for any context, Γ:*

- $\tilde{q}_{F \circ j;\Gamma} \sim q_{F;\Gamma}$;
- $\tilde{u}_{F \circ j;\Gamma} \sim u_{F;\Gamma}$.

This follows from fact 7.3.16.

Remark 7.3.11. The two preceding facts allow us informally to deduce that the two families of morphisms q_Γ and $\tilde{q}_\Gamma$ perform the same computation, and that the two families of morphisms u_Γ and $\tilde{u}_\Gamma$ perform the same computation.

Fact 7.3.12. *The two families of morphisms $\tilde{q}_T$ and $\tilde{q}_\Gamma$ satisfy the following mutual naturality property:*

$$\tilde{q}_T \circ [\![\Gamma \vdash i_n : T]\!] \sim F_1(\Gamma \vdash_{\text{sub}} \bullet, \underline{i_n} : \bullet, T) \circ \tilde{q}_\Gamma$$

From this we can conclude that the morphism family $\tilde{q}_\Gamma$ satisfies the following naturality property.

$$\tilde{q}_\Delta \circ [\![\Gamma \vdash_{\text{ren}} \rho : \Delta]\!] \sim F_1(\Gamma \vdash_{\text{ren}} \rho : \Delta) \circ \tilde{q}_\Gamma$$

Fact 7.3.13. *The two families of morphisms $\tilde{u}_T$ and $\tilde{u}_\Gamma$ satisfy the following mutual naturality property:*

$$[\![\Gamma \vdash i_n : T]\!] \circ \tilde{u}_\Gamma \sim \tilde{u}_T \circ F_1(\Gamma \vdash_{\text{sub}} \bullet, \underline{i_n} : \bullet, T)$$

From this we can conclude that the morphism family $\tilde{u}_\Gamma$ satisfies the following naturality property.

$$[\![\Gamma \vdash_{\text{ren}} \rho : \Delta]\!] \circ u_\Gamma \sim u_\Delta \circ F_1(\Gamma \vdash_{\text{ren}} \rho : \Delta)$$

Construction 7.3.14 (Quote and Unquote P-Natural Transformations). The two families of morphisms $\tilde{q}_\Gamma$ and $\tilde{u}_\Gamma$ form two P-natural transformations.

$$\tilde{q} : [\![-]\!]_{F_0(\bullet, \iota)} \circ j \circ i_3 \Rightarrow F \circ i_3 : \mathcal{R} \to \mathbb{C}$$
$$\tilde{u} : F \circ i_3 \Rightarrow [\![-]\!]_{F_0(\bullet, \iota)} \circ j \circ i_3 : \mathcal{R} \to \mathbb{C}$$

Fact 7.3.15. *The two families of morphisms $\tilde{q}_T$ and $\tilde{u}_T$ are uniquely defined amongst all pairs of strictly pointed P-natural*

transformations that satisfy a technical property for interaction with e and $\tilde{e}$; i.e.*, they satisfy the following property.*

$$\bigforall\ \alpha\ :\ [\![-]\!]_{F_0(\bullet,\,\iota)} \circ j \circ i_3 \Rightarrow F \circ i_3.$$
$$\bigforall\ \beta\ :\ F \circ i_3 \Rightarrow [\![-]\!]_{F_0(\bullet,\,\iota)} \circ j \circ i_3.$$
$$\alpha_{\bullet,\,\iota} \sim \mathrm{id} \wedge \beta_{\bullet,\,\iota} \sim \mathrm{id} \Rightarrow$$
$$\left(\bigforall_{\Delta\,\Theta\,:\,\mathcal{R}} \alpha_{\Delta \Rightarrow \Theta} \circ e \sim \tilde{e} \circ (\beta_\Delta \Rightarrow \alpha_\Theta) \right) \Rightarrow$$
$$\left(\bigforall_{\Delta\,\Theta\,:\,\mathcal{R}} ([\![\varepsilon]\!] \circ p)^* \circ \beta_{\Delta \Rightarrow \Theta} \sim (\alpha_\Delta \Rightarrow \beta_\Theta) \circ (F\varepsilon' \circ p)^* \right) \Rightarrow$$
$$\bigforall_{T\,:\,\mathbb{T}} \alpha_{\bullet,\,T} \sim \tilde{q}_T \wedge \beta_{\bullet,\,T} \sim \tilde{u}_T$$

Fact 7.3.16. *The two families of morphisms $\tilde{q}_\Gamma$ and $\tilde{u}_\Gamma$ are uniquely defined amongst all pairs of strictly pointed* P*-natural transformations that satisfy a technical property for interaction with e and $\tilde{e}$;* i.e.*, they satisfy the following property.*

$$\bigforall\ \alpha\ :\ [\![-]\!]_{F_0(\bullet,\,\iota)} \circ j \circ i_3 \Rightarrow F \circ i_3.$$
$$\bigforall\ \beta\ :\ F \circ i_3 \Rightarrow [\![-]\!]_{F_0(\bullet,\,\iota)} \circ j \circ i_3.$$
$$\alpha_{\bullet,\,\iota} \sim \mathrm{id} \wedge \beta_{\bullet,\,\iota} \sim \mathrm{id} \Rightarrow$$
$$\left(\bigforall_{\Delta\,\Theta\,:\,\mathcal{R}} \alpha_{\Delta \Rightarrow \Theta} \circ e \sim \tilde{e} \circ (\beta_\Delta \Rightarrow \alpha_\Theta) \right) \Rightarrow$$
$$\left(\bigforall_{\Delta\,\Theta\,:\,\mathcal{R}} ([\![\varepsilon]\!] \circ p)^* \circ \beta_{\Delta \Rightarrow \Theta} \sim (\alpha_\Delta \Rightarrow \beta_\Theta) \circ (F\varepsilon' \circ p)^* \right) \Rightarrow$$
$$\bigforall_{\Gamma\,:\,\mathcal{R}} \alpha_\Gamma \sim \tilde{q}_\Gamma \wedge \beta_\Gamma \sim \tilde{u}_\Gamma$$

7.3.1 *P-Bicategorical Pseudo-Initiality*

The results in the former part of this section have demonstrated informally a bicategorical initiality property; in this subsection we define precisely the P-bicategory in which the initiality property holds, and state the resultant P-pseudoinitiality property. We denote the P-bicategory in question by $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$, as the objects are P-Cartesian-pre-closed coslices under $\mathcal{A}$.

Definition 7.3.17 (Objects of $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$)**.** An *object* of $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$ is a dependent pair of a P-Cartesian category, and a P-Cartesian-pre-closed functor thereinto from $\mathcal{A}$; *i.e.*, they are terms of the following type.

$$\sum_{\mathbb{C}\,:\,\mathbf{PCartClosCat}} \mathcal{A} \to_{\mathrm{CartPreClos}} \mathbb{C}$$

Definition 7.3.18 (1-Morphisms of $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$)**.** In this definition we fix two objects, $(\mathbb{C}; F)$ and $(\mathbb{D}; G)$. A 1-*morphism* from $(\mathbb{C}; F)$ to $(\mathbb{D}; G)$ is given by the following structure (where $i_3 : \mathcal{R} \to \mathcal{A}$ is the inclusion P-functor):

- a P-Cartesian-closed functor, $H : \mathbb{C} \to \mathbb{D}$;
- a P-natural transformation, $\alpha : H \circ F \circ i_3 \Rightarrow G \circ i_3$; and
- a P-natural transformation, $\alpha^\dagger : G \circ i_3 \Rightarrow H \circ F \circ i_3$;

such that the following properties hold:

- $\alpha_{(\bullet,\, \iota)} \circ \alpha^{\dagger}_{(\bullet,\, \iota)} \sim \mathrm{id}$;
- $\alpha^{\dagger}_{(\bullet,\, \iota)} \circ \alpha_{(\bullet,\, \iota)} \sim \mathrm{id}$;
- $\alpha_{(\bullet,\, T_1 \to T_2)} \circ \tilde{e}_{H \circ F;(\bullet,\, T_1);(\bullet,\, T_2)} \sim \tilde{e}_{G;(\bullet,\, T_1);(\bullet,\, T_2)} \circ (\alpha^{\dagger}_{(\bullet,\, T_1)} \Rightarrow \alpha_{(\bullet,\, T_2)})$; and
- $(HF(\tilde{\varepsilon}_{(\bullet,\, T_1);(\bullet,\, T_2)}) \circ p_{H \circ F})^* \circ \alpha^{\dagger}_{(\bullet,\, T_1 \to T_2)} \sim (\alpha_{(\bullet,\, T_1)} \Rightarrow \alpha^{\dagger}_{(\bullet,\, T_2)}) \circ (G(\tilde{\varepsilon}_{(\bullet,\, T_1);(\bullet,\, T_2)}) \circ p_G)^*$.

The first two of these properties express that at base type the natural transformations, α and $\alpha^{\dagger}$, exhibit an isomorphism. The latter two of these properties express that at arrow type the natural transformations, α and $\alpha^{\dagger}$, weakly preserve the P-Cartesian-pre-closed structure. The equivalent properties for a P-natural isomorphism between P-Cartesian-closed functors are derivable from the universal structure of P-Cartesian-closure.

Definition 7.3.19 (2-Morphisms of $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$)**.** In this definition we fix two objects, $(\mathbb{C}; F)$ and $(\mathbb{D}; G)$, and two 1-morphisms therebetwixt, $(H, \alpha, \alpha^{\dagger})$ and $(K, \beta, \beta^{\dagger})$. A 2-*morphism*, γ, from $(H, \alpha, \alpha^{\dagger})$ to $(K, \beta, \beta^{\dagger})$ is a 2-morphism in the locally-groupoidal P-bicategory of P-Cartesian-closed functors from H to K such that following properties hold:

- $\beta_{\Gamma} \circ \gamma_{F\,\Gamma} \sim \alpha_{\Gamma}$; and
- $\gamma_{F\,\Gamma} \circ \alpha^{\dagger}_{\Gamma} \sim \beta^{\dagger}_{\Gamma}$.

Theorem 7.3.20 (Universal Property of $j : \mathcal{A} \to \mathcal{F}$)**.** *The quotient functor $j : \mathcal{A} \to \mathcal{F}$, as a P-Cartesian-pre-closed coslice under $\mathcal{A}$ into $\mathcal{F}$, is P-pseudoinitial in the P-bicategory* $\mathbf{PCartPreClosFun}_{\mathcal{A}/}$.

Proof. The unique 1-morphism out from j is given by the interpretation P-functor, and by $\tilde{q}$ and $\tilde{u}$. The unique P-iso-2-morphism establishing the uniqueness of the unique 1-morphism uses the lifting of contexts, q, and u. □

CHAPTER 8

P-CATEGORICAL NORMALISATION OF SIMPLY TYPED SYNTAX

In this chapter, we give a number of normal-form algorithms using the P-categories defined in chapter 7. Each of the normal-form algorithms depends upon the universal property of $\mathcal{F}$ (the category of substitutions under $\beta\eta$-conversion), and/or the universal property of j (the quotient functor from $\mathcal{A}$ (the category of substitutions under equality) to $\mathcal{F}$). The full type-theoretic formalism of the universal property of $\mathcal{F}$ is a key contribution of this work. The categorical development and type-theoretic formalism of a universal property of j allows a categorical proof of correctness properties for some of the normal-form algorithms, and is considered novel and therefore a key contribution of this work.

8.1 NORMALISATION WITH $\mathcal{F}$

In this section, we derive normal form algorithms purely from our P-categorical development and the universal property of $\mathcal{F}$. These normal-form algorithms have much the same structure as non-categorical normalisation by evaluation algorithms, and thus can be seen as a (partial) P-categorical reconstruction of these algorithms.

In order to use the universal property of $\mathcal{F}$, we must choose a P-Cartesian-closed category and an object thereof in order to instantiate the interpretation P-functor, and a P-Cartesian-closed functor into the interpretation category. The simplest choice is probably the terminal (P-Cartesian-closed) P-category; however, since this has only trivial structure it is certainly insufficient. There are three further choices that we investigate in the following three subsections. The first of these choices demonstrates the need for presheaf categories. The second is from the work of Čubrić et al. (1998). The third is a variation on the second, and is a contribution of this work based on an observation about the nerve of the quotient functor $j : \mathcal{A} \to \mathcal{F}$. First, we give a brief overview of the putative[1] normal-form algorithm in the general case before investigating specific cases.

Construction 8.1.1 (Putative Normal-Form Algorithm). The following construction induces a $\mathbb{C}$-normal–form algorithm, given some P-Cartesian-closed functor, $F : \mathcal{F} \to \mathbb{C}$; we abbreviate $F_0(\bullet, \iota)$ as c. Considering the following observations about morphisms defined in section 7.2,

$$\begin{aligned} u_\Gamma &: F_0(\Gamma) \to [\![\Gamma]\!]_c \\ [\![\Gamma \vdash_{\text{sub}} \sigma : \Delta]\!]_c &: [\![\Gamma]\!]_c \to [\![\Delta]\!]_c \\ q_\Delta &: [\![\Delta]\!]_c \to F_0(\Delta) \end{aligned}$$

[1] There is no reason *prima facie* that the algorithm performs any normalisation; thus, we may only refer to it as such putatively.

we make the following definition.

$$\mathrm{nf}_F(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq q_\Delta \circ [\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]_c \circ u_\Gamma \,:\, F_0(\Gamma) \to F_0(\Delta)$$

Remark 8.1.2. If we wish to normalise just a single term, then either we can embed it into a context substitution into a singleton context, or we can use the interpretation of terms and the type-indexed (rather than context-indexed) versions of q and u. By abuse of notation, we pretend that we have done either of these equivalent options when normalising a single term.

Theorem 8.1.3 (Soundness). *The putative normal-form algorithm is sound;* i.e.*, it satisfies the following property.*

$$\mathrm{nf}_F(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \sim F_1(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta)$$

Proof. This follows from naturality of u, and q and u being inverses with each other. □

We first give an example of a poor choice of the P-functor, F, that does not result in a normal-form algorithm. Thereafter, we give two examples of good choices of the P-functor, F, that do result in normal-form algorithms.

Into $\mathcal{F}$

We make the following choices for "normalisation" by interpretation into $\mathcal{F}$:

- for the chosen object: the singleton context containing just the base type, *i.e.*, $(\bullet,\, \iota)$; and
- for the P-Cartesian-closed functor: the identity P-functor.

With these choices, the putative normal-form algorithm directly returns an element of the same hom-P-set in $\mathcal{F}$. However, it does not perform normalisation, as can be observed by the following non-examples of normalisation.

Non-Example 8.1.4.

$$\mathrm{nf}_{\mathrm{Id}}(\lambda_{\iota\to\iota}.\underline{i_0}) \equiv \lambda.\lambda.((\lambda.\underline{i_0})\,(\lambda.(\lambda.\underline{i_3}\;\underline{i_0})\,\underline{i_0}))\,\underline{i_0}$$

Non-Example 8.1.5.

$$\mathrm{nf}_{\mathrm{Id}}(\lambda_\iota.(\lambda_\iota.\underline{i_0})\,\underline{i_0}) \equiv \lambda.(\lambda.(\lambda.\underline{i_0})\,\underline{i_0})\,\underline{i_0}$$

As can be observed this "normal-form algorithm" is performing some η-expansion, but it is not obviously performing any β-reduction.

8.1.1 *Into* $\widehat{\mathcal{F}}$

Following Čubrić et al. (1998), we make the following choices for "normalisation" by interpretation into $\widehat{\mathcal{F}}$:

- for the chosen object: the Yoneda embedding of the singleton context containing just the base type, *i.e.*, $よ\,(\bullet,\, \iota)$; and
- for the P-Cartesian-closed functor: the Yoneda P-functor.

With these choices, the putative normal-form algorithm returns a (P-natural) transformation, *e.g.*, $よ\,(\Gamma) \Rightarrow よ\,(\Delta)$. From such a transformation we can recover a context substitution by taking the component at, *e.g.*, Γ, resulting in a (P-)function, *e.g.*, $\mathcal{F}(\Gamma,\Gamma) \to \mathcal{F}(\Gamma,\Delta)$, to which we can provide the identity substitution. We therefore, by abuse of notation, make the following definition.

Definition 8.1.6 (Normalisation in $\widehat{\mathcal{F}}$).

$$\mathrm{nf}_{\widehat{\mathcal{F}}}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq (q_\Delta \circ \widehat{\mathcal{F}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ u_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

This normal-form algorithm is worthy of being called such as it actually normalises, as can be observed by the following examples of normalisation.

Example 8.1.7.

$$\mathrm{nf}_{\widehat{\mathcal{F}}}(\lambda_{\iota\to\iota}.\underline{i_0}) \equiv \lambda.\lambda.\underline{i_1}\,\underline{i_0}$$

Example 8.1.8.

$$\mathrm{nf}_{\widehat{\mathcal{F}}}(\lambda_{\iota}.(\lambda_{\iota}.\underline{i_0})\,\underline{i_0}) \equiv \lambda.\underline{i_0}$$

8.1.2 *Into* $\widehat{\mathcal{A}}$

Alternatively, we may normalise by interpretation into $\widehat{\mathcal{A}}$. This requires finding a P-Cartesian-closed functor; the following fact provides such an example.

Fact 8.1.9. *The quotient P-functor, $j : \mathcal{A} \to \mathcal{F}$, is dominant*[2] *(it is identity-on-objects), full, and P-Cartesian; therefore its nerve, $\langle j \rangle : \mathcal{F} \to \widehat{\mathcal{A}}$, is a P-Cartesian-closed functor.*

We make the following choices for "normalisation" by interpretation into $\widehat{\mathcal{A}}$:

- for the chosen object: the nerve of the quotient P-functor at the singleton context containing just the base type, *i.e.*, $\langle j \rangle(\bullet,\ \iota)$; and
- for the P-Cartesian-closed functor: the nerve of the quotient P-functor.

With these choices, the putative normal-form algorithm returns a (P-natural) transformation, *e.g.*, $\langle j \rangle(\Gamma) \Rightarrow \langle j \rangle(\Delta)$. From such a transformation we can recover a context substitution by taking the component at, *e.g.*, Γ, resulting in a (P-)function, *e.g.*, $\mathcal{F}(j^{\mathrm{op}}(\Gamma), \Gamma) \to \mathcal{F}(j^{\mathrm{op}}(\Gamma), \Delta)$, to which we can provide the identity substitution. We therefore, by abuse of notation, make the following definition.

Definition 8.1.10 (Normalisation in $\widehat{\mathcal{A}}$)**.**

$$\mathrm{nf}_{\widehat{\mathcal{A}}}(\Gamma \vdash_{\mathrm{sub}} \sigma\ :\ \Delta) \triangleq (q_\Delta \circ \widehat{\mathcal{A}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma\ :\ \Delta]\!] \circ u_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

This normal-form algorithm is worthy of being called such as it actually normalises, as can be observed by the following examples of normalisation.

Example 8.1.11.

$$\mathrm{nf}_{\widehat{\mathcal{A}}}(\lambda_{\iota\to\iota}.\underline{i_0}) \equiv \lambda.\lambda.\underline{i_1}\,\underline{i_0}$$

Example 8.1.12.

$$\mathrm{nf}_{\widehat{\mathcal{A}}}(\lambda_{\iota}.(\lambda_{\iota}.\underline{i_0})\,\underline{i_0}) \equiv \lambda.\underline{i_0}$$

8.2 Normalisation with $\mathcal{A}$

In this section, we derive normal-form algorithms purely from our P-categorical development and the universal property of $j : \mathcal{A} \to \mathcal{F}$. These normal-form algorithms have much the same structure as non-categorical normalisation by evaluation algorithms, and thus can be seen as a (partial) P-categorical reconstruction of these algorithms.

In order to use the universal property of $j : \mathcal{A} \to \mathcal{F}$ we must choose a P-Cartesian-closed category and an object thereof in order to instantiate the interpretation P-functor, and a P-Cartesian-pre-closed functor into the interpretation category. The simplest choice is probably the terminal (P-Cartesian-closed) P-category; however, since this has only trivial structure it is certainly insufficient. There are three further choices that we investigate in the following three subsections.

[2] It is essentially surjective, but dominance is sufficient here.

There are a number of choices for the P-Cartesian-pre-closed functor that we could choose; however, many of them easily factor as the composition of a P-Cartesian-closed functor after j, and so this results in the same normal-form algorithm as if we had just used the P-Cartesian-closed functor. First, we give a brief overview of the putative[3] normal-form algorithm in the general case before investigating specific cases.

Construction 8.2.1 (Putative Normal-Form Algorithm). The following construction induces a $\mathbb{C}$-normal–form algorithm, given some P-Cartesian-pre-closed functor, $F : \mathcal{A} \to \mathbb{C}$; we abbreviate $F_0(\bullet, \iota)$ as c. Considering the following observations about morphisms defined in section 7.3,

$$\begin{aligned} \tilde{u}_\Gamma &: F_0(\Gamma) \to [\![\Gamma]\!]_c \\ [\![\Gamma \vdash_{\text{sub}} \sigma : \Delta]\!]_c &: [\![\Gamma]\!]_c \to [\![\Delta]\!]_c \\ \tilde{q}_\Delta &: [\![\Delta]\!]_c \to F_0(\Delta) \end{aligned}$$

we make the following definition.

$$\text{nf}_F(\Gamma \vdash_{\text{sub}} \sigma : \Delta) \triangleq \tilde{q}_\Delta \circ [\![\Gamma \vdash_{\text{sub}} \sigma : \Delta]\!]_c \circ \tilde{u}_\Gamma : F_0(\Gamma) \to F_0(\Delta)$$

Remark 8.2.2. If we wish to normalise just a single term, then either we can embed it into a context substitution into a singleton context, or we can use the interpretation of terms and the type-indexed (rather than context-indexed) versions of $\tilde{q}$ and $\tilde{u}$. By abuse of notation, we pretend that we have done either of these equivalent options when normalising a single term.

Lemma 8.2.3 (Weak Completeness). *The putative normal-form algorithm is weakly complete;* i.e.*, it satisfies the following property.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 : \Delta \Rightarrow \text{nf}'_F(\Gamma \vdash_{\text{sub}} \sigma_1 : \Delta) \sim \text{nf}'_F(\Gamma \vdash_{\text{sub}} \sigma_2 : \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

We now proceed to give three examples of good choices of the P-functor, F, that do result in normal-form algorithms.

8.2.1 *Into* $\widehat{\mathcal{A}}$

We make the following choices for "normalisation" by interpretation into $\widehat{\mathcal{A}}$:

- for the chosen object: the Yoneda embedding of the singleton context containing just the base type, *i.e.*, $よ(\bullet, \iota)$; and
- for the P-Cartesian-pre-closed functor: the Yoneda P-functor.

With these choices, the putative normal-form algorithm returns a (P-natural) transformation, *e.g.*, $よ(\Gamma) \Rightarrow よ(\Delta)$. From such a transformation we can recover a context substitution by taking the component at, *e.g.*, Γ, resulting in a (P-)function, *e.g.*, $\mathcal{A}(\Gamma, \Gamma) \to \mathcal{A}(\Gamma, \Delta)$, to which we can provide the identity substitution. We therefore, by abuse of notation, make the following definition.

Definition 8.2.4 (Normalisation in $\widehat{\mathcal{A}}$).

$$\text{nf}'_{\widehat{\mathcal{A}}}(\Gamma \vdash_{\text{sub}} \sigma : \Delta) \triangleq (\tilde{q}_\Delta \circ \widehat{\mathcal{A}}[\![\Gamma \vdash_{\text{sub}} \sigma : \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\text{id}_\Gamma)$$

Theorem 8.2.5 (Strong Completeness). *This normal-form algorithm is strongly complete;* i.e.*, it satisfies the following property.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 : \Delta \Rightarrow \text{nf}'_{\widehat{\mathcal{A}}}(\Gamma \vdash_{\text{sub}} \sigma_1 : \Delta) = \text{nf}'_{\widehat{\mathcal{A}}}(\Gamma \vdash_{\text{sub}} \sigma_2 : \Delta)$$

[3] There is no reason *prima facie* that the algorithm performs any normalisation; thus, we may only refer to it as such putatively.

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

This normal-form algorithm is worthy of being called such as it actually normalises, as can be observed by the following examples of normalisation.

Example 8.2.6.

$$\mathrm{nf}'_{\widehat{\mathcal{A}}}(\lambda_{\iota\to\iota}.\underline{i_0}) \equiv \lambda.\lambda.\underline{i_1}\,\underline{i_0}$$

Example 8.2.7.

$$\mathrm{nf}'_{\widehat{\mathcal{A}}}(\lambda_{\iota}.(\lambda_{\iota}.\underline{i_0})\,\underline{i_0}) \equiv \lambda.\underline{i_0}$$

8.2.2 *Into* $\widehat{\mathcal{U}}$

Alternatively, we may normalise by interpretation into $\widehat{\mathcal{U}}$. This requires finding a P-Cartesian-pre-closed functor; the following fact provides such an example.

Fact 8.2.8. *The second inclusion P-functor, $i_2 : \mathcal{U} \to \mathcal{A}$, is dominant*[4] *(it is identity-on-objects) and P-Cartesian; therefore its nerve, $\langle i_2 \rangle : \mathcal{A} \to \widehat{\mathcal{U}}$, is a P-Cartesian-pre-closed functor.*

We make the following choices for "normalisation" by interpretation into $\widehat{\mathcal{U}}$:

- for the chosen object: the nerve of the second inclusion P-functor at the singleton context containing just the base type, *i.e.*, $\langle i_2 \rangle(\bullet,\ \iota)$; and
- for the P-Cartesian-pre-closed functor: the nerve of the second inclusion P-functor.

With these choices, the putative normal-form algorithm returns a (P-natural) transformation, *e.g.*, $\langle i_2 \rangle(\Gamma) \Rightarrow \langle i_2 \rangle(\Delta)$. From such a transformation we can recover a context substitution by taking the component at, *e.g.*, Γ, resulting in a (P-)function, *e.g.*, $\mathcal{A}(i_2^{\mathrm{op}}(\Gamma), \Gamma) \to \mathcal{A}(i_2^{\mathrm{op}}(\Gamma), \Delta)$, to which we can provide the identity substitution. We therefore, by abuse of notation, make the following definition.

Definition 8.2.9 (Normalisation in $\widehat{\mathcal{U}}$)**.**

$$\mathrm{nf}'_{\widehat{\mathcal{U}}}(\Gamma \vdash_{\mathrm{sub}} \sigma\ :\ \Delta) \triangleq (\tilde{q}_{\Delta} \circ \widehat{\mathcal{U}}\,[\![\Gamma \vdash_{\mathrm{sub}} \sigma\ :\ \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\mathrm{id}_{\Gamma})$$

Theorem 8.2.10 (Strong Completeness)**.** *This normal-form algorithm is strongly complete;* i.e.*, it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2\ :\ \Delta \Rightarrow \mathrm{nf}'_{\widehat{\mathcal{U}}}(\Gamma \vdash_{\mathrm{sub}} \sigma_1\ :\ \Delta) = \mathrm{nf}'_{\widehat{\mathcal{U}}}(\Gamma \vdash_{\mathrm{sub}} \sigma_2\ :\ \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

This normal-form algorithm is worthy of being called such as it does perform normalisation as can be observed by the following examples of normalisation.

Example 8.2.11.

$$\mathrm{nf}'_{\widehat{\mathcal{U}}}(\lambda_{\iota\to\iota}.\underline{i_0}) \equiv \lambda.\lambda.\underline{i_1}\,\underline{i_0}$$

Example 8.2.12.

$$\mathrm{nf}'_{\widehat{\mathcal{U}}}(\lambda_{\iota}.(\lambda_{\iota}.\underline{i_0})\,\underline{i_0}) \equiv \lambda.\underline{i_0}$$

[4] It is essentially surjective, but dominance is sufficient here.

8.2.3 *Into* $\widehat{\mathcal{R}}$

Alternatively we may normalise by interpretation into $\widehat{\mathcal{R}}$. This requires finding a P-Cartesian-pre-closed functor; the following fact provides such an example.

Fact 8.2.13. *The third inclusion P-functor, $i_3 : \mathcal{R} \to \mathcal{A}$, is dominant*[5] *(it is identity-on-objects) and P-Cartesian; therefore its nerve, $\langle i_3 \rangle : \mathcal{A} \to \widehat{\mathcal{R}}$, is a P-Cartesian-pre-closed functor.*

We make the following choices for "normalisation" by interpretation into $\widehat{\mathcal{R}}$:

- for the chosen object: the nerve of the third inclusion P-functor at the singleton context containing just the base type, *i.e.*, $\langle i_3 \rangle(\bullet, \iota)$; and
- for the P-Cartesian-pre-closed functor: the nerve of the third inclusion P-functor.

With these choices, the putative normal-form algorithm returns a (P-natural) transformation, *e.g.*, $\langle i_3 \rangle(\Gamma) \Rightarrow \langle i_3 \rangle(\Delta)$. From such a transformation we can recover a context substitution by taking the component at, *e.g.*, Γ, resulting in a (P-)function, *e.g.*, $\mathcal{A}(i_3^{\mathrm{op}}(\Gamma), \Gamma) \to \mathcal{A}(i_3^{\mathrm{op}}(\Gamma), \Delta)$, to which we can provide the identity substitution. We therefore, by abuse of notation, make the following definition.

Definition 8.2.14 (Normalisation in $\widehat{\mathcal{R}}$)**.**

$$\mathrm{nf}'_{\widehat{\mathcal{R}}}(\Gamma \vdash_{\mathrm{sub}} \sigma : \Delta) \triangleq (\tilde{q}_{\Delta} \circ \widehat{\mathcal{R}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma : \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\mathrm{id}_{\Gamma})$$

Theorem 8.2.15 (Strong Completeness)**.** *This normal-form algorithm is strongly complete;* i.e.*, it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 : \Delta \Rightarrow \mathrm{nf}'_{\widehat{\mathcal{R}}}(\Gamma \vdash_{\mathrm{sub}} \sigma_1 : \Delta) = \mathrm{nf}'_{\widehat{\mathcal{R}}}(\Gamma \vdash_{\mathrm{sub}} \sigma_2 : \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

This normal-form algorithm is worthy of being called such as it actually normalises, as can be observed by the following examples of normalisation.

Example 8.2.16.

$$\mathrm{nf}'_{\widehat{\mathcal{R}}}(\lambda_{\iota \to \iota}.\underline{i_0}) \equiv \lambda.\lambda.\underline{i_1}\,\underline{i_0}$$

Example 8.2.17.

$$\mathrm{nf}'_{\widehat{\mathcal{R}}}(\lambda_{\iota}.(\lambda_{\iota}.\underline{i_0})\,\underline{i_0}) \equiv \lambda.\underline{i_0}$$

In the preceding sections, we have synthesised a number of normal-form algorithms: section 8.1 synthesised sound normal-form algorithms; and section 8.2 synthesised strongly complete algorithms. The facility for the algorithms in section 8.1 to be sound is rooted in the families of morphisms, q and u, being both P-natural with respect to terms/context substitutions and being inverses with each other; however, this is not the case for the families of morphisms, $\tilde{q}$ and $\tilde{u}$, which are only P-natural with respect to de Bruijn indices/context renamings, and are not inverses with each other. The facility for the algorithms in section 8.2 to be strongly complete is rooted in their weak completeness specialising and being refined into strong completeness by the choice of the PER on the P-sets produced by the choice of P-functors valued in P-presheaves. Thus each set of normal-form algorithms only has one correctness property, but the two sets of normal-form algorithms have complementary correctness properties; this suggests that combining them might provide a way of synthesising a normal-form algorithm that has both strong completeness and soundness.

[5] It is essentially surjective, but dominance is sufficient here.

8.3 Correctness by Categorical Gluing

In this section we demonstrate a categorical construction, by way of parametrised p-categorical gluing, to combine the normal-form algorithms of sections 8.1 and 8.2 together in order to synthesise an algorithm that is both sound and strongly complete. In order to achieve this, we need to define a p-Cartesian-closed p-gluing category, and a p-Cartesian-pre-closed functor from $\mathcal{A}$ thereinto. We make several choices resulting in a number of instantiations of our construction.

In this section, all of our p-categorical gluing constructions depend upon the following data:

- a p-Cartesian category, $\mathbb{C}$;
- a dominant p-Cartesian functor, $F : \mathbb{C} \to \mathcal{A}$;
- a p-Cartesian-closed category, $\mathbb{S}$; and
- a p-Cartesian-closed functor, $G : \mathcal{F} \to \mathbb{S}$.

Therefore, all of our p-gluing categories have the following shape.

$$\mathcal{G} \triangleq \widehat{\mathbb{C}} \downarrow \langle G \circ j \circ F \rangle$$

We refer to our construction as parametrised p-categorical gluing as the construction is parametrised by the p-Cartesian category $\mathbb{C}$ and the dominant p-Cartesian functor $F : \mathbb{C} \to \mathcal{A}$. Because of this parametrisation, the p-gluing category $\mathcal{G}$ does not support an extension of the Yoneda embedding, but instead supports an extension of the p-nerve functor of F, *i.e.*, $\langle F \rangle$. From this definition of $\mathcal{G}$ we induce a p-functor $\overline{F}_G : \mathcal{A} \to \mathcal{G}$, according to construction 4.2.39, by making the following choices of p-functors and p-natural transformation:

- $\langle F \rangle : \mathcal{A} \to \widehat{\mathbb{C}}$;
- $G \circ j : \mathcal{A} \to \mathbb{S}$; and
- $\langle F \rangle \Rightarrow \langle G \circ j \circ F \rangle \circ G \circ j$.

Where the p-natural transformation is the canonical construction mapping hom p-set into hom p-set by $G \circ j$. These choices of p-functors are certainly p-Cartesian functors, and so the induced p-functor $\overline{F}_G : \mathcal{A} \to \mathcal{G}$ is a p-Cartesian functor. In order to conclude that the induced p-Cartesian functor $\overline{F}_G : \mathcal{A} \to \mathcal{G}$ is a p-Cartesian-pre-closed functor, we use fact 4.4.50.

As $F : \mathbb{C} \to \mathcal{A}$ is dominant, its nerve $\langle F \rangle : \mathcal{A} \to \widehat{\mathbb{C}}$ is p-Cartesian-pre-closed, and as $G \circ j : \mathcal{A} \to \mathcal{F}$ is p-Cartesian-pre-closed (it is the composition of a p-Cartesian-closed functor after a p-Cartesian-pre-closed functor), the induced p-Cartesian functor $\overline{F}_G : \mathcal{A} \to \mathcal{G}$ is p-Cartesian-pre-closed. We can therefore instantiate a number of normal-form algorithms in the following subsections by using the universal property of $j : \mathcal{A} \to \mathcal{F}$. First, we give a definition of a putative "normalisation" algorithm that we need for our constructions in each of the following subsections.

Definition 8.3.1 (Normalisation in $\mathbb{S}$).

$$\mathrm{nf}_{\mathbb{S}}(\Gamma \vdash_{\mathrm{sub}} \sigma : \Delta) \triangleq (q_\Delta \circ \mathbb{S}[\![\Gamma \vdash_{\mathrm{sub}} \sigma : \Delta]\!] \circ u_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Where the q and u are induced by the universal property of $\mathcal{F}$ and $G : \mathcal{F} \to \mathbb{S}$ being p-Cartesian-closed.

Theorem 8.3.2 (Soundness of Normalisation in $\mathbb{S}$). *This normal-form algorithm is sound;* i.e.*, it satisfies the following property.*

$$\mathrm{nf}_{\mathbb{S}}(\sigma) \sim G_1(\sigma)$$

Proof. This follows from naturality of u, and q and u being inverses with each other. □

Remark 8.3.3. The P-Cartesian-closed category, $\mathbb{S}$, is simply used to establish soundness of the normal-form algorithm and as such there are a number of obvious choices available for $\mathbb{S}$:

- the terminal P-Cartesian-closed category, 1;
- the free P-Cartesian-closed category, $\mathcal{F}$;
- the P-category of P-presheaves over $\mathcal{F}$, $\widehat{\mathcal{F}}$; and
- the P-category of P-presheaves over $\mathcal{A}$, $\widehat{\mathcal{A}}$.

These choices of $\mathbb{S}$ would correlate, respectively, with the following choices for $G : \mathcal{F} \to \mathbb{S}$:

- the unique P-Cartesian-closed functor to the terminal P-Cartesian-closed category;
- the identity P-Cartesian-closed functor;
- the Yoneda embedding; and
- the P-Cartesian-closed nerve P-functor, $\langle j \rangle$.

The first choice should be made for concrete computation as it would minimise the computational overhead associated to using the P-Cartesian-closed category, $\mathbb{S}$, for the P-gluing construction. The second choice should be made for establishing concrete soundness.

$$\Gamma \vdash_{\text{sub}} \text{nf}_{\mathcal{F}}(\sigma) \equiv_{\beta\eta} \sigma : \Delta$$

This choice of $\mathbb{S}$ and F, as previously observed, does not actually normalise, but it has the important desired soundness property. Thus, although it has little utility in normalising terms, it does play a rôle in establishing the correctness of other normal-form algorithms. The latter two choices have no obvious useful advantage, although they do result in actual normalisation in the codomain part of the P-gluing construction.

In the following subsections, we instantiate $\mathbb{S}$ with $\mathcal{F}$, and G with the identity P-Cartesian-closed functor, so that we can establish concrete soundness for the induced normal-form algorithms. Moreover, in the following subsections all choices for $\mathbb{C}$ have contexts as their objects, and as such we elide their action on objects. Note that in full generality the abstract normalisation algorithms would have to use the dominance of F to be able to extract out a context substitution from the algorithm. As the choices for $\mathbb{C}$ have contexts as objects, and therefore the dominance is witnessed by the identity P-split-monomorphism, we can elide this use.

8.3.1 *Gluing with* $\widehat{\mathcal{A}}$

In this subsection we make the following choices for $\mathbb{C}$ and $F : \mathbb{C} \to \mathcal{A}$:

- $\mathbb{C} \triangleq \mathcal{A}$; and
- $F \triangleq \text{Id} : \mathcal{A} \to \mathcal{A}$.

This choice of $F \triangleq \text{Id}$ is dominant[6] (it is identity-on-objects) and P-Cartesian; therefore, it satisfies the requirements of our construction. We are therefore justified in making the following definitions of normal-form algorithms:

Definition 8.3.4 (Domain Normalisation in $\widehat{\mathcal{A}}$).

$$\text{nf}'_{\text{Dom}}(\Gamma \vdash_{\text{sub}} \sigma : \Delta) \triangleq \text{Dom}(\tilde{q}_{\Delta} \circ \mathcal{G}_{\mathcal{A}}[\![\Gamma \vdash_{\text{sub}} \sigma : \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\text{id}_{\Gamma})$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j : \mathcal{A} \to \mathcal{F}$ and $\overline{F} : \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

[6] It is essentially surjective, but dominance is sufficient here.

Theorem 8.3.5 (Strong Completeness of Domain Normalisation in $\widehat{\mathcal{A}}$). *This normal-form algorithm is strongly complete;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\text{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \Rightarrow \text{nf}'_{\text{Dom}}(\Gamma \vdash_{\text{sub}} \sigma_1 \,:\, \Delta) = \text{nf}'_{\text{Dom}}(\Gamma \vdash_{\text{sub}} \sigma_2 \,:\, \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

Definition 8.3.6 (Weak Codomain Normalisation in $\mathcal{F}$).

$$\text{nf}'_{\text{Cod}}(\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta) \triangleq \text{Cod}(\tilde{q}_\Delta \circ \mathcal{G}_{\mathcal{A}}[\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\text{id}_\Gamma)$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \,:\, \mathcal{A} \to \mathcal{F}$ and $\overline{F} \,:\, \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

Definition 8.3.7 (Strict Codomain Normalisation in $\mathcal{F}$).

$$\text{nf}'_{\text{Cod}'}(\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta) \triangleq (\tilde{q}_\Delta \circ \mathcal{F}[\![\Gamma \vdash_{\text{sub}} \sigma \,:\, \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\text{id}_\Gamma)$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \,:\, \mathcal{A} \to \mathcal{F}$ and $\text{Cod} \circ \overline{F} \,:\, \mathcal{A} \to \mathcal{F}$ being P-Cartesian-pre-closed (it is the composition of a P-Cartesian-closed functor after a P-Cartesian-pre-closed functor).

Lemma 8.3.8. *We have the following lemmata:*

- $\Gamma \vdash_{\text{sub}} j(\text{nf}'_{\text{Dom}}(\sigma)) \equiv_{\beta\eta} \text{nf}'_{\text{Cod}}(\sigma) \,:\, \Delta$*;*
- $\Gamma \vdash_{\text{sub}} \text{nf}'_{\text{Cod}}(\sigma) \equiv_{\beta\eta} \text{nf}'_{\text{Cod}'}(\sigma) \,:\, \Delta$ *and*
- $\Gamma \vdash_{\text{sub}} \text{nf}'_{\text{Cod}'}(\sigma) \equiv_{\beta\eta} \text{nf}_{\mathcal{F}}(\sigma) \,:\, \Delta$.

Proof. The first lemma follows from the commutation of morphism squares in P-comma categories. The second lemma follows from fact 7.3.9, and from q and u being inverses with each other. The third lemma follows from fact 7.3.10. □

We can therefore make the following conclusion about the domain normal-form algorithm.

Theorem 8.3.9 (Soundness of Domain Normalisation in $\widehat{\mathcal{A}}$). *The domain normal-form algorithm is sound;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\text{sub}} j(\text{nf}'_{\text{Dom}}(\sigma)) \equiv_{\beta\eta} \sigma \,:\, \Delta$$

Proof. This follows from normalisation in $\mathcal{F}$ being sound, and from the preceding lemma allowing us to relate normalisation in $\mathcal{F}$ with the domain normal-form algorithm in $\widehat{\mathcal{A}}$. □

8.3.2 *Gluing with* $\widehat{\mathcal{U}}$

In this subsection we make the following choices for $\mathbb{C}$ and $F \,:\, \mathbb{C} \to \mathcal{A}$:

- $\mathbb{C} \triangleq \mathcal{U}$; and
- $F \triangleq i_2 \,:\, \mathcal{U} \to \mathcal{A}$.

This choice of $F \triangleq i_2$ is dominant[7] (it is identity-on-objects) and P-Cartesian; therefore, it satisfies the requirements of our construction. We are therefore justified in making the following definitions of normal-form algorithms:

[7] It is essentially surjective, but dominance is sufficient here.

Definition 8.3.10 (Domain Normalisation in $\widehat{\mathcal{U}}$).

$$\mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta) \triangleq \mathrm{Dom}(\tilde{q}_{\Delta} \circ \mathcal{G}_{\mathcal{U}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\mathrm{id}_{\Gamma})$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \ : \ \mathcal{A} \to \mathcal{F}$ and $\overline{F} \ : \ \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

Theorem 8.3.11 (Strong Completeness of Domain Normalisation in $\widehat{\mathcal{U}}$). *This normal-form algorithm is strongly complete;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \ : \ \Delta \Rightarrow \mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma_1 \ : \ \Delta) = \mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma_2 \ : \ \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

Definition 8.3.12 (Weak Codomain Normalisation in $\mathcal{F}$).

$$\mathrm{nf}'_{\mathrm{Cod}}(\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta) \triangleq \mathrm{Cod}(\tilde{q}_{\Delta} \circ \mathcal{G}_{\mathcal{U}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\mathrm{id}_{\Gamma})$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \ : \ \mathcal{A} \to \mathcal{F}$ and $\overline{F} \ : \ \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

Definition 8.3.13 (Strict Codomain Normalisation in $\mathcal{F}$).

$$\mathrm{nf}'_{\mathrm{Cod}'}(\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta) \triangleq (\tilde{q}_{\Delta} \circ \mathcal{F}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \ : \ \Delta]\!] \circ \tilde{u}_{\Gamma})_{\Gamma}(\mathrm{id}_{\Gamma})$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \ : \ \mathcal{A} \to \mathcal{F}$ and $\mathrm{Cod} \circ \overline{F} \ : \ \mathcal{A} \to \mathcal{F}$ being P-Cartesian-pre-closed (it is the composition of a P-Cartesian-closed functor after a P-Cartesian-pre-closed functor).

Lemma 8.3.14. *We have the following lemmata:*

- $\Gamma \vdash_{\mathrm{sub}} j(\mathrm{nf}'_{\mathrm{Dom}}(\sigma)) \equiv_{\beta\eta} \mathrm{nf}'_{\mathrm{Cod}}(\sigma) \ : \ \Delta$*;*
- $\Gamma \vdash_{\mathrm{sub}} \mathrm{nf}'_{\mathrm{Cod}}(\sigma) \equiv_{\beta\eta} \mathrm{nf}'_{\mathrm{Cod}'}(\sigma) \ : \ \Delta$ *and*
- $\Gamma \vdash_{\mathrm{sub}} \mathrm{nf}'_{\mathrm{Cod}'}(\sigma) \equiv_{\beta\eta} \mathrm{nf}_{\mathcal{F}}(\sigma) \ : \ \Delta$.

Proof. The first lemma follows from the commutation of morphism squares in P-comma categories. The second lemma follows from fact 7.3.9, and from q and u being inverses with each other. The third lemma follows from fact 7.3.10. □

We can therefore make the following conclusion about the domain normal-form algorithm.

Theorem 8.3.15 (Soundness of Domain Normalisation in $\widehat{\mathcal{U}}$). *The domain normal-form algorithm is sound;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} j(\mathrm{nf}'_{\mathrm{Dom}}(\sigma)) \equiv_{\beta\eta} \sigma \ : \ \Delta$$

Proof. This follows from normalisation in $\mathcal{F}$ being sound, and from the preceding fact allowing us to relate normalisation in $\mathcal{F}$ with the domain normal-form algorithm in $\widehat{\mathcal{U}}$. □

8.3.3 *Gluing with* $\widehat{\mathcal{R}}$

In this subsection we make the following choices for $\mathbb{C}$ and $F \ : \ \mathbb{C} \to \mathcal{A}$:

- $\mathbb{C} \triangleq \mathcal{R}$; and
- $F \triangleq i_3 \ : \ \mathcal{R} \to \mathcal{A}$.

This choice of $F \triangleq i_3$ is dominant[8] (it is identity-on-objects) and P-Cartesian; therefore, it satisfies the requirements of our construction. We are therefore justified in making the following definitions of normal-form algorithms:

Definition 8.3.16 (Domain Normalisation in $\widehat{\mathcal{R}}$).

$$\mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq \mathrm{Dom}(\tilde{q}_\Delta \circ \mathcal{G}_{\mathcal{R}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \,:\, \mathcal{A} \to \mathcal{F}$ and $\overline{F} \,:\, \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

Theorem 8.3.17 (Strong Completeness of Domain Normalisation in $\widehat{\mathcal{R}}$). *This normal-form algorithm is strongly complete;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma_1 \equiv_{\beta\eta} \sigma_2 \,:\, \Delta \Rightarrow \mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma_1 \,:\, \Delta) = \mathrm{nf}'_{\mathrm{Dom}}(\Gamma \vdash_{\mathrm{sub}} \sigma_2 \,:\, \Delta)$$

Proof. This follows from $\tilde{q}$, $[\![-]\!]$, and $\tilde{u}$ being P-well-defined. □

Definition 8.3.18 (Weak Codomain Normalisation in $\mathcal{F}$).

$$\mathrm{nf}'_{\mathrm{Cod}}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq \mathrm{Cod}(\tilde{q}_\Delta \circ \mathcal{G}_{\mathcal{R}}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \,:\, \mathcal{A} \to \mathcal{F}$ and $\overline{F} \,:\, \mathcal{A} \to \mathcal{G}$ being P-Cartesian-pre-closed.

Definition 8.3.19 (Strict Codomain Normalisation in $\mathcal{F}$).

$$\mathrm{nf}'_{\mathrm{Cod}'}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq (\tilde{q}_\Delta \circ \mathcal{F}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \tilde{u}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Where the $\tilde{q}$ and $\tilde{u}$ are induced by the universal property of $j \,:\, \mathcal{A} \to \mathcal{F}$ and $\mathrm{Cod} \circ \overline{F} \,:\, \mathcal{A} \to \mathcal{F}$ being P-Cartesian-pre-closed (it is the composition of a P-Cartesian-closed functor after a P-Cartesian-pre-closed functor).

Lemma 8.3.20. *We have the following lemmata:*

- $\Gamma \vdash_{\mathrm{sub}} j(\mathrm{nf}'_{\mathrm{Dom}}(\sigma)) \equiv_{\beta\eta} \mathrm{nf}'_{\mathrm{Cod}}(\sigma) \,:\, \Delta$*;*
- $\Gamma \vdash_{\mathrm{sub}} \mathrm{nf}'_{\mathrm{Cod}}(\sigma) \equiv_{\beta\eta} \mathrm{nf}'_{\mathrm{Cod}'}(\sigma) \,:\, \Delta$*; and*
- $\Gamma \vdash_{\mathrm{sub}} \mathrm{nf}'_{\mathrm{Cod}'}(\sigma) \equiv_{\beta\eta} \mathrm{nf}_{\mathcal{F}}(\sigma) \,:\, \Delta$.

Proof. The first lemma follows from the commutation of morphism squares in P-comma categories. The second lemma follows from fact 7.3.9, and from q and u being inverses with each other. The third lemma follows from fact 7.3.10. □

We can therefore make the following conclusion about the domain normal-form algorithm.

Theorem 8.3.21 (Soundness of Domain Normalisation in $\widehat{\mathcal{R}}$). *The domain normal-form algorithm is sound;* i.e., *it satisfies the following property.*

$$\Gamma \vdash_{\mathrm{sub}} j(\mathrm{nf}'_{\mathrm{Dom}}(\sigma)) \equiv_{\beta\eta} \sigma \,:\, \Delta$$

Proof. This follows from normalisation in $\mathcal{F}$ being sound, and from the preceding fact allowing us to relate normalisation in $\mathcal{F}$ with the domain normal-form algorithm in $\widehat{\mathcal{R}}$. □

[8] It is essentially surjective, but dominance is sufficient here.

8.3.4 *Comments*

In each of the three preceding subsections, we have succeeded in synthesising a normal-form algorithm that is both sound and strongly complete by using parametrised p-categorical gluing. The constructions rest on having p-functorial maps from $\mathcal{A}$ into a p-gluing category that are p-Cartesian-pre-closed, and on the universal property of $j : \mathcal{A} \to \mathcal{F}$. This overcomes the non-categorical nature of the proof of strong completeness found in Čubrić et al. (1998). Nonetheless, it is not clear how to adapt these constructions to track neutrality and normality to allow for the characterisation of the outputs of the normal-form algorithms, and to allow for a proof of stability. The final construction using $\widehat{\mathcal{R}}$ is similar to standard gluing constructions for categorical normalisation by evaluation results; however, our construction differs in that our assignment into the gluing category is functorial, and thus it is able to use the universal property of $j : \mathcal{A} \to \mathcal{F}$. One disadvantage of this approach is that it is not clear how the choices of $\mathbb{C}$, F, $\mathbb{S}$, and G affect the algorithm. It should be established that the domain normal-form algorithms are independent of the choice of $\mathbb{S}$, and thus that the normal-form algorithms induced by choosing the terminal p-Cartesian-closed category are the same as the normal-form algorithms induced by picking the free p-Cartesian-closed category, $\mathcal{F}$.

8.4 Normalisation by Categorical Gluing

In this section, we present three normal form algorithms using p-categorical gluing. All three normal-form algorithms depend upon several useful p-presheaves in certain p-presheaf p-categories. These p-presheaves are then extended to produce objects in the p-gluing categories. The base p-categories of the p-presheaf p-categories are all p-Cartesian categories whose objects are contexts. For the latter two the base p-category's morphisms, crucially, preserve neutral and normal substitutions. Although we use the universal property of $\mathcal{F}$ to characterise some morphisms, building upon the work of Fiore (2002, 2022), it is not used to induce the normal-form algorithms.

In our first normal-form algorithm we choose the p-category of context substitutions. This choice acts as an informal bridge between the p-categorical gluing constructions of section 8.3 and the constructions that follow in this section. Frequently, the p-category of renamings has been chosen as such a base p-category, and we present this choice in our second p-gluing category. Additionally, there is a useful third possibility for the base p-category – the p-category of neutral substitutions – and we present this choice in our third p-gluing category. The latter two normal-form algorithms presented in this section differ from the normal-form algorithms in sections 8.1, 8.2, and 8.3 as they produce terms that are provably characterised as being normal. This latter choice of base p-category has a number of advantages over the second choice of base p-category, which are discussed in subsection 8.4.7.

Throughout this section we define a number of p-presheaves. We define their p-sets, *i.e.* their underlying carrier type and PER, by a dependent pairing $(-;\ -)$.

8.4.1 *p-Presheaves in* $\widehat{\mathcal{A}}$

In this subsection, we define a number of p-presheaves upon which we depend for our first normalisation by gluing construction. The p-presheaves in this subsection all have their action given by context substitutions; thus, they are p-functors in $\widehat{\mathcal{A}} \equiv [\mathcal{A}^{\mathrm{op}}, \mathbf{PSet}]$.

We now proceed to define p-presheaves of terms and context substitutions; for which we have a choice as to what the PER for the p-sets are: it can be either equality (for the discrete p-set), or $\beta\eta$-conversion. We choose the former.

Definition 8.4.1 (p-Presheaf of Terms). The *p-presheaf of terms* at type T, $\mathcal{L}_T$, is a type-indexed family of p-presheaves defined by the sequel:

- $\mathcal{L}_T(\Gamma) \triangleq (\Gamma \vdash - : T; =)$; and
- $\mathcal{L}_T(\Gamma \vdash_{\mathrm{sub}} \sigma : \Theta)(\Theta \vdash t : T) \triangleq \Gamma \vdash t[\sigma] : T$.

Definition 8.4.2 (P-Presheaf of Substitutions). The *P-presheaf of substitutions* at context Δ, $\mathcal{L}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{L}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\text{sub}} - \,:\, \Delta; =)$; and
- $\mathcal{L}_\Delta(\Gamma \vdash_{\text{sub}} \sigma \,:\, \Theta)(\Theta \vdash_{\text{sub}} \psi \,:\, \Delta) \triangleq \Gamma \vdash \psi \circ \sigma \,:\, \Delta$.

Remark 8.4.3. We use the same notation for the P-presheaf of renamings as that for the P-presheaf of variables, as we think of $\mathcal{L}_\Delta$ as the Δ-fold product of $\mathcal{L}_T$ for each T in Δ. We continue this notational abuse for all similarly related type-indexed and context-indexed families of P-presheaves and glued objects.

Remark 8.4.4. The P-presheaves $\mathcal{L}_T$ and $\mathcal{L}_\Delta$ are equivalent to the Yoneda embedding at $(\bullet,\, T)$ and at Δ respectively.

Construction 8.4.5 (Term Application Morphism). There is a P-well-defined application morphism for terms.

$$\mathsf{App}_{T_1;T_2} \,:\, \mathcal{L}_{T_1 \to T_2} \times \mathcal{L}_{T_1} \to \mathcal{L}_{T_2}$$

Construction 8.4.6 (Term Abstraction Morphism). There is a P-well-defined abstraction morphism for terms.

$$\mathsf{Abs}_{T_1;T_2} \,:\, (\mathcal{L}_{T_1} \Rightarrow \mathcal{L}_{T_2}) \to \mathcal{L}_{T_1 \to T_2}$$

8.4.2 *First Gluing Category*

In this subsection, we define the first P-gluing category that we use to establish a normal-form algorithm, and objects and morphisms thereof.

Remark 8.4.7. In the sequel we fix a pointed P-Cartesian-closed category, $\mathbb{S}$, with chosen object s.

Definition 8.4.8 (First P-Gluing Category). The *first P-gluing category*, which we denote by $\mathcal{G}_1$, is the following P-Cartesian-closed P-comma category.

$$\mathcal{G}_1 \triangleq (\mathrm{Id}_{\widehat{\mathcal{A}}} \downarrow \langle [\![-]\!]_s \circ j \rangle)$$

We now proceed to define objects and morphisms in the P-gluing category $\mathcal{G}_1$.

Definition 8.4.9 (Glued Object of Terms). The *glued object of terms* at type T, $\overline{\mathcal{L}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_1$, and is defined by the sequel.

$$\overline{\mathcal{L}}_T \triangleq \begin{pmatrix} \mathcal{L}_T \\ \big| \\ [\![\Gamma \vdash - \,:\, T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{pmatrix}$$

Definition 8.4.10 (Glued Object of Substitutions). The *glued object of substitutions* at context Δ, $\overline{\mathcal{L}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_1$, and is defined by the sequel.

$$\overline{\mathcal{L}}_\Delta \triangleq \begin{pmatrix} \mathcal{L}_\Delta \\ \big| \\ [\![\Gamma \vdash_{\text{sub}} - \,:\, \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{pmatrix}$$

Construction 8.4.11 (Term Application Morphism). There is a P-well-defined application morphism for neutral terms with normal terms.

$$\overline{\mathsf{App}}_{T_1;T_2} \,:\, \overline{\mathcal{L}}_{T_1 \to T_2} \times \overline{\mathcal{L}}_{T_1} \to \overline{\mathcal{L}}_{T_2}$$

Construction 8.4.12 (Term Abstraction Morphism)**.** There is a P-well-defined abstraction morphism for normal terms.

$$\overline{\mathsf{Abs}}_{T_1;T_2} \,:\, (\overline{\mathcal{L}}_{T_1} \Rightarrow \overline{\mathcal{L}}_{T_2}) \to \overline{\mathcal{L}}_{T_1 \to T_2}$$

The domain component of this morphism is given by the composition of the relevant morphism for P-presheaves with one of the canonical maps induced by the pullback-definition of exponentials in P-gluing categories.

Definition 8.4.13 (Interpretation in $\mathcal{G}_1$)**.** We can interpret $\mathcal{F}$ into $\mathcal{G}_1$ by choosing $\overline{\mathcal{L}}_\iota$ as the chosen object for interpreting ι.

Construction 8.4.14 (In/Out Morphisms)**.** There are P-well-defined type-indexed and context-indexed families of morphisms:

- $\overline{\mathsf{in}}_T \,:\, \overline{\mathcal{L}}_T \to \mathcal{G}_1[\![T]\!]$;
- $\overline{\mathsf{in}}_\Delta \,:\, \overline{\mathcal{L}}_\Delta \to \mathcal{G}_1[\![\Delta]\!]$;
- $\overline{\mathsf{out}}_T \,:\, \mathcal{G}_1[\![T]\!] \to \overline{\mathcal{L}}_T$; and
- $\overline{\mathsf{out}}_\Delta \,:\, \mathcal{G}_1[\![\Delta]\!] \to \overline{\mathcal{L}}_\Delta$.

Remark 8.4.15. We denote the domain parts of the $\overline{\mathsf{in}}$ and $\overline{\mathsf{out}}$ morphisms by in and out.

Fact 8.4.16. *The following facts hold of the codomain parts of the* $\overline{\mathsf{in}}$ *and* $\overline{\mathsf{out}}$ *morphisms:*

- $\mathrm{Cod}(\overline{\mathsf{in}}_T) \sim q_T$*;*
- $\mathrm{Cod}(\overline{\mathsf{in}}_\Delta) \sim q_\Delta$*;*
- $\mathrm{Cod}(\overline{\mathsf{out}}_T) \sim u_T$*; and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta) \sim u_\Delta$*;*

where the q *and the* u *are induced by the universal property with respect to the following* P*-Cartesian-closed functor.*

$$\mathrm{Cod} \circ \mathcal{G}_1[\![-]\!] \,:\, \mathcal{F} \to \mathbb{S}$$

We therefore have the following facts as corollaries:

- $\mathrm{Cod}(\overline{\mathsf{out}}_T \circ \mathcal{G}_1[\![\Gamma \vdash t \,:\, T]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash t \,:\, T]\!]$*; and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta \circ \mathcal{G}_1[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]$.

Definition 8.4.17 (Normalisation in $\mathcal{G}_1$)**.**

$$\mathrm{nf}_{\mathcal{G}_1}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq (\mathsf{out}_\Delta \circ \mathrm{Dom}(\mathcal{G}_1[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]) \circ \mathsf{in}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Remark 8.4.18. Note that definition 8.4.17 for normalisation in $\mathcal{G}_1$ implicitly depends upon a pointed P-Cartesian-closed category; *viz.*, $\mathbb{S}$ with chosen point s. For concrete computation, it suffices to instantiate these with the terminal P-Cartesian-closed category and the terminal object thereof.

Fact 8.4.19. *The following fact holds.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma \equiv_{\beta\eta} \sigma' \,:\, \Delta \Rightarrow \mathbb{S}[\![\mathrm{nf}_{\mathcal{G}_1}(\sigma)]\!] \sim \mathbb{S}[\![\sigma']\!]$$

Fact 8.4.20. *The following facts hold:*

- $\Delta \vdash i_n \,:\, T \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma \,:\, \Delta \Rightarrow \mathsf{in}_{T,\Gamma}(i[\sigma]) \sim \mathrm{Dom}(\mathcal{G}_1[\![\Gamma \vdash i_n \,:\, T]\!])_\Gamma(\mathsf{in}_{\Delta,\Gamma}(\sigma))$*;*

- $\Delta \vdash_{\mathcal{M}} t : T \Rightarrow \mathsf{in}_{T,\Gamma}(t) \sim \mathrm{Dom}(\mathcal{G}_1[\![\Gamma \vdash t : T]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))$*; and*
- $\Delta \vdash_{\mathcal{N}} t : T \Rightarrow \mathsf{out}_{T,\Gamma}(\mathrm{Dom}(\mathcal{G}_1[\![\Gamma \vdash t : \Delta]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))) = t$*.*

Theorem 8.4.21 (Soundness). *The normal-form algorithm of definition 8.4.17 is sound;* i.e.*, the output is $\beta\eta$-convertible with the input. This can be derived from substituting $\mathcal{F}$ and $(\bullet, \iota)$ for $\mathbb{S}$ and s respectively, and from noting that interpretation into $\mathcal{F}$ is injective on substitutions.*

Theorem 8.4.22 (Stability). *The normal-form algorithm of definition 8.4.17 is stable;* i.e.*, for normal inputs the output is equal with the input. This can be derived from fact 8.4.20.*

8.4.3 *P-Presheaves in $\widehat{\mathcal{R}}$*

In this subsection, we define a number of P-presheaves upon which we depend for our second normalisation by gluing construction. The P-presheaves in this subsection all have their action given by context renamings; thus, they are P-functors in $\widehat{\mathcal{R}} \equiv [\mathcal{R}^{\mathrm{op}}, \mathbf{PSet}]$.

Definition 8.4.23 (P-Presheaf of de Bruijn Indices/Variables). The *P-presheaf of de Bruijn indices/variables* at type T, $\mathcal{V}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{V}_T(\Gamma) \triangleq (\Gamma \vdash i_\_ : T; =)$; and
- $\mathcal{V}_T(\Gamma \vdash_{\mathrm{ren}} \rho : \Theta)(\Theta \vdash i_n : T) \triangleq \Gamma \vdash i_n[\rho] : T$.

Definition 8.4.24 (P-Presheaf of Renamings). The *P-presheaf of renamings* at context Δ, $\mathcal{V}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{V}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\mathrm{ren}} - : \Delta; =)$; and
- $\mathcal{V}_\Delta(\Gamma \vdash_{\mathrm{ren}} \rho : \Theta)(\Theta \vdash_{\mathrm{ren}} \phi : \Delta) \triangleq \Gamma \vdash \phi \circ \rho : \Delta$.

Remark 8.4.25. We use the same notation for the P-presheaf of renamings as that for the P-presheaf of variables, as we think of $\mathcal{V}_\Delta$ as the Δ-fold product of $\mathcal{V}_T$ for each T in Δ. We continue this notational abuse for all similarly related type-indexed and context-indexed families of P-presheaves and glued objects.

Remark 8.4.26. The P-presheaves $\mathcal{V}_T$ and $\mathcal{V}_\Delta$ are equivalent to the Yoneda embedding at $(\bullet, T)$ and at Δ respectively.

We now proceed to define P-presheaves of terms and context substitutions, for which we have a choice as to what the PER for the P-sets are: it can be either equality (for the discrete P-set), or $\beta\eta$-conversion. We choose the former.

Definition 8.4.27 (P-Presheaf of Terms). The *P-presheaf of terms* at type T, $\mathcal{L}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{L}_T(\Gamma) \triangleq (\Gamma \vdash - : T; =)$; and
- $\mathcal{L}_T(\Gamma \vdash_{\mathrm{ren}} \rho : \Theta)(\Theta \vdash t : T) \triangleq \Gamma \vdash t[\rho] : T$.

Definition 8.4.28 (P-Presheaf of Substitutions). The *P-presheaf of substitutions* at context Δ, $\mathcal{L}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{L}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\mathrm{sub}} - : \Delta; =)$; and
- $\mathcal{L}_\Delta(\Gamma \vdash_{\mathrm{ren}} \rho : \Theta)(\Theta \vdash_{\mathrm{sub}} \sigma : \Delta) \triangleq \Gamma \vdash \sigma \circ \rho : \Delta$.

Construction 8.4.29 (de Bruijn Index/Renaming Inclusion Morphisms). There are P-well-defined inclusion morphisms:

- $\mathsf{Var}_T : \mathcal{V}_T \hookrightarrow \mathcal{L}_T$; and

- $\mathsf{Var}_\Delta \,:\, \mathcal{V}_\Delta \hookrightarrow \mathcal{L}_\Delta$.

Construction 8.4.30 (Term Application Morphism). There is a P-well-defined application morphism for terms.

$$\mathsf{App}_{T_1;T_2} \,:\, \mathcal{L}_{T_1 \to T_2} \times \mathcal{L}_{T_1} \to \mathcal{L}_{T_2}$$

Construction 8.4.31 (Term Abstraction Morphism). There is a P-well-defined abstraction morphism for terms.

$$\mathsf{Abs}_{T_1;T_2} \,:\, (\mathcal{V}_{T_1} \Rightarrow \mathcal{L}_{T_2}) \to \mathcal{L}_{T_1 \to T_2}$$

Definition 8.4.32 (P-Presheaf of Neutral Terms). The *P-presheaf of neutral terms* at type T, $\mathcal{M}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{M}_T(\Gamma) \triangleq (\Gamma \vdash - \,:\, T; t\, t' \mapsto \Gamma \vdash_{\mathcal{M}} t \,:\, T \wedge \Gamma \vdash_{\mathcal{M}} t' \,:\, T \wedge t = t')$; and
- $\mathcal{M}_T(\Gamma \vdash_{\text{ren}} \rho \,:\, \Theta)(\Theta \vdash t \,:\, T) \triangleq \Gamma \vdash t[\rho] \,:\, T$.

Definition 8.4.33 (P-Presheaf of Neutral Substitutions). The *P-presheaf of neutral substitutions* at context Δ, $\mathcal{M}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{M}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\text{sub}} - \,:\, \Delta; \sigma\, \sigma' \mapsto \Gamma \vdash_{\mathcal{M}\,\text{sub}} \sigma \,:\, \Delta \wedge \Gamma \vdash_{\mathcal{M}\,\text{sub}} \sigma' \,:\, \Delta \wedge \sigma = \sigma')$; and
- $\mathcal{M}_\Delta(\Gamma \vdash_{\text{ren}} \rho \,:\, \Theta)(\Theta \vdash_{\text{sub}} \sigma \,:\, \Delta) \triangleq \Gamma \vdash \sigma \circ \rho \,:\, \Delta$.

Construction 8.4.34 (de Bruijn Index/Renaming Inclusion Morphisms). There are P-well-defined inclusion morphisms:

- $\mathsf{Var}_T \,:\, \mathcal{V}_T \hookrightarrow \mathcal{M}_T$; and
- $\mathsf{Var}_\Delta \,:\, \mathcal{V}_\Delta \hookrightarrow \mathcal{M}_\Delta$.

Definition 8.4.35 (P-Presheaf of Normal Terms). The *P-presheaf of normal terms* at type T, $\mathcal{N}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{N}_T(\Gamma) \triangleq (\Gamma \vdash - \,:\, T; t\, t' \mapsto \Gamma \vdash_{\mathcal{N}} t \,:\, T \wedge \Gamma \vdash_{\mathcal{N}} t' \,:\, T \wedge t = t')$; and
- $\mathcal{N}_T(\Gamma \vdash_{\text{ren}} \rho \,:\, \Theta)(\Theta \vdash t \,:\, T) \triangleq \Gamma \vdash t[\rho] \,:\, T$.

Definition 8.4.36 (P-Presheaf of Normal Substitutions). The *P-presheaf of normal substitutions* at context Δ, $\mathcal{N}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{N}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\text{sub}} - \,:\, \Delta; \sigma\, \sigma' \mapsto \Gamma \vdash_{\mathcal{N}\,\text{sub}} \sigma \,:\, \Delta \wedge \Gamma \vdash_{\mathcal{N}\,\text{sub}} \sigma' \,:\, \Delta \wedge \sigma = \sigma')$; and
- $\mathcal{N}_\Delta(\Gamma \vdash_{\text{ren}} \rho \,:\, \Theta)(\Theta \vdash_{\text{sub}} \sigma \,:\, \Delta) \triangleq \Gamma \vdash \sigma \circ \rho \,:\, \Delta$.

Construction 8.4.37 (Neutral/Normal Term Application Morphism). There is a P-well-defined application morphism for neutral terms with normal terms.

$$\mathsf{App}_{T_1;T_2} \,:\, \mathcal{M}_{T_1 \to T_2} \times \mathcal{N}_{T_1} \to \mathcal{M}_{T_2}$$

Construction 8.4.38 (Normal Term Abstraction Morphism). There is a P-well-defined abstraction morphism for normal terms.

$$\mathsf{Abs}_{T_1;T_2} \,:\, (\mathcal{V}_{T_1} \Rightarrow \mathcal{N}_{T_2}) \to \mathcal{N}_{T_1 \to T_2}$$

8.4.4 *Second Gluing Category*

In this subsection we define the second P-gluing category that we use to establish a normal-form algorithm, and objects and morphisms thereof.

Remark 8.4.39. In the sequel we fix a pointed P-Cartesian-closed category, $\mathbb{S}$, with chosen object s.

Definition 8.4.40 (Second P-Gluing Category). The *second P-gluing category*, which we denote by $\mathcal{G}_2$, is the P-Cartesian-closed P-comma category:

$$\mathcal{G}_2 \triangleq (\mathrm{Id}_{\widehat{\mathcal{R}}} \downarrow \langle [\![-]\!]_s \circ j \circ i_2 \circ i_1 \rangle)$$

We now proceed to define objects and morphisms in the P-gluing category $\mathcal{G}_1$.

Definition 8.4.41 (Glued Object of de Bruijn Indices/Variables). The *glued object of de Bruijn indices/variables* at type T, $\overline{\mathcal{V}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{V}}_T \triangleq \begin{pmatrix} \mathcal{V}_T \\ \downarrow{\scriptstyle [\![\Gamma \vdash i_- : T]\!]_s} \\ [\![T]\!]_s \end{pmatrix}$$

Definition 8.4.42 (Glued Object of Renamings). The *glued object of renamings* at context Δ, $\overline{\mathcal{V}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{V}}_\Delta \triangleq \begin{pmatrix} \mathcal{V}_\Delta \\ \downarrow{\scriptstyle [\![\Gamma \vdash_{\mathrm{ren}} - : \Delta]\!]_s} \\ [\![\Delta]\!]_s \end{pmatrix}$$

Definition 8.4.43 (Glued Object of Terms). The *glued object of terms* at type T, $\overline{\mathcal{L}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{L}}_T \triangleq \begin{pmatrix} \mathcal{L}_T \\ \downarrow{\scriptstyle [\![\Gamma \vdash - : T]\!]_s} \\ [\![T]\!]_s \end{pmatrix}$$

Definition 8.4.44 (Glued Object of Substitutions). The *glued object of substitutions* at context Δ, $\overline{\mathcal{L}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{L}}_\Delta \triangleq \begin{pmatrix} \mathcal{L}_\Delta \\ \downarrow{\scriptstyle [\![\Gamma \vdash_{\mathrm{sub}} - : \Delta]\!]_s} \\ [\![\Delta]\!]_s \end{pmatrix}$$

Definition 8.4.45 (Glued Object of Neutral Terms). The *glued object of neutral terms* at type T, $\overline{\mathcal{M}}_T$, is an extension of

the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{M}}_T \triangleq \left(\begin{array}{c} \mathcal{M}_T \\ \downarrow \\ [\![\Gamma \vdash - : T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{array} \right)$$

Definition 8.4.46 (Glued Object of Neutral Substitutions). The *glued object of neutral substitutions* at context Δ, $\overline{\mathcal{M}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{M}}_\Delta \triangleq \left(\begin{array}{c} \mathcal{M}_\Delta \\ \downarrow \\ [\![\Gamma \vdash_{\text{sub}} - : \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{array} \right)$$

Construction 8.4.47 (de Bruijn Index Inclusion Morphisms). There are P-well-defined inclusion morphisms:

- $\overline{\mathsf{Var}}_T \ : \ \overline{\mathcal{V}}_T \hookrightarrow \overline{\mathcal{M}}_T$; and
- $\overline{\mathsf{Var}}_\Delta \ : \ \overline{\mathcal{V}}_\Delta \hookrightarrow \overline{\mathcal{M}}_\Delta$.

Definition 8.4.48 (Glued Object of Normal Terms). The *glued object of normal terms* at type T, $\overline{\mathcal{N}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{N}}_T \triangleq \left(\begin{array}{c} \mathcal{N}_T \\ \downarrow \\ [\![\Gamma \vdash - : T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{array} \right)$$

Definition 8.4.49 (Glued Object of Normal Substitutions). The *glued object of normal substitutions* at context Δ, $\overline{\mathcal{N}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_2$, and is defined by the sequel:

$$\overline{\mathcal{N}}_\Delta \triangleq \left(\begin{array}{c} \mathcal{N}_\Delta \\ \downarrow \\ [\![\Gamma \vdash_{\text{sub}} - : \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{array} \right)$$

Construction 8.4.50 (Neutral/Normal Term Application Morphism). There is a P-well-defined application morphism for neutral terms with normal terms.

$$\overline{\mathsf{App}}_{T_1;T_2} \ : \ \overline{\mathcal{M}}_{T_1 \to T_2} \times \overline{\mathcal{N}}_{T_1} \to \overline{\mathcal{M}}_{T_2}$$

Construction 8.4.51 (Normal Term Abstraction Morphism). There is a P-well-defined abstraction morphism for normal terms.

$$\overline{\mathsf{Abs}}_{T_1;T_2} \ : \ (\overline{\mathcal{V}}_{T_1} \Rightarrow \overline{\mathcal{N}}_{T_2}) \to \overline{\mathcal{N}}_{T_1 \to T_2}$$

Definition 8.4.52 (Interpretation in $\mathcal{G}_2$). We can interpret $\mathcal{F}$ into $\mathcal{G}_2$ by choosing $\overline{\mathcal{M}}_\iota$ as the chosen object for interpreting ι.

Construction 8.4.53 (In/Out Morphisms). There are P-well-defined type-indexed and context-indexed families of morphisms:

- $\overline{\mathsf{in}}_T \,:\, \overline{\mathcal{M}}_T \to \mathcal{G}_2[\![T]\!]$;
- $\overline{\mathsf{in}}_\Delta \,:\, \overline{\mathcal{M}}_\Delta \to \mathcal{G}_2[\![\Delta]\!]$;
- $\overline{\mathsf{out}}_T \,:\, \mathcal{G}_2[\![T]\!] \to \overline{\mathcal{N}}_T$; and
- $\overline{\mathsf{out}}_\Delta \,:\, \mathcal{G}_2[\![\Delta]\!] \to \overline{\mathcal{N}}_\Delta$.

Remark 8.4.54. We denote the domain parts of the $\overline{\mathsf{in}}$ and $\overline{\mathsf{out}}$ morphisms by in and out.

Fact 8.4.55. *The following facts hold of the codomain parts of the* $\overline{\mathsf{in}}$ *and* $\overline{\mathsf{out}}$ *morphisms:*

- $\mathrm{Cod}(\overline{\mathsf{in}}_T) \sim q_T$*;*
- $\mathrm{Cod}(\overline{\mathsf{in}}_\Delta) \sim q_\Delta$*;*
- $\mathrm{Cod}(\overline{\mathsf{out}}_T) \sim u_T$*; and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta) \sim u_\Delta$*;*

where the q *and the* u *are induced by the universal property with respect to the following* P*-Cartesian-closed functor.*

$$\mathrm{Cod} \circ \mathcal{G}_2[\![-]\!] \,:\, \mathcal{F} \to \mathbb{S}$$

We therefore have the following facts as corollaries:

- $\mathrm{Cod}(\overline{\mathsf{out}}_T \circ \mathcal{G}_2[\![\Gamma \vdash t \,:\, T]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash t \,:\, T]\!]$*; and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta \circ \mathcal{G}_2[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]$.

Definition 8.4.56 (Normalisation in $\mathcal{G}_2$).

$$\mathrm{nf}_{\mathcal{G}_2}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq (\mathsf{out}_\Delta \circ \mathrm{Dom}(\mathcal{G}_2[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]) \circ \mathsf{in}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Remark 8.4.57. Note that definition 8.4.56 for normalisation in $\mathcal{G}_2$ implicitly depends upon a pointed P-Cartesian-closed category; *viz.*, $\mathbb{S}$ with chosen point s. For concrete computation, it suffices to instantiate these with the terminal P-Cartesian-closed category and the terminal object thereof.

Fact 8.4.58. *The following fact holds.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma \equiv_{\beta\eta} \sigma' \,:\, \Delta \Rightarrow \mathbb{S}[\![\mathrm{nf}_{\mathcal{G}_2}(\sigma)]\!] \sim \mathbb{S}[\![\sigma']\!]$$

Fact 8.4.59. *The following facts hold:*

- $\Delta \vdash i_n \,:\, T \wedge \Gamma \vdash_{\overline{\mathcal{M}}\,\mathrm{sub}} \sigma \,:\, \Delta \Rightarrow \mathsf{in}_{T,\Gamma}(i[\sigma]) \sim \mathrm{Dom}(\mathcal{G}_2[\![\Gamma \vdash i_n \,:\, T]\!])_\Gamma(\mathsf{in}_{\Delta,\Gamma}(\sigma))$*;*
- $\Delta \vdash_{\overline{\mathcal{M}}} t \,:\, T \Rightarrow \mathsf{in}_{T,\Gamma}(t) \sim \mathrm{Dom}(\mathcal{G}_2[\![\Gamma \vdash t \,:\, T]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))$*; and*
- $\Delta \vdash_{\overline{\mathcal{N}}} t \,:\, T \Rightarrow \mathsf{out}_{T,\Gamma}(\mathrm{Dom}(\mathcal{G}_2[\![\Gamma \vdash t \,:\, \Delta]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))) = t$*.*

Theorem 8.4.60 (Soundness). *The normal-form algorithm of definition 8.4.56 is sound;* i.e.*, the output is* $\beta\eta$*-convertible with the input. This can be derived from substituting* $\mathcal{F}$ *and* $(\bullet,\, \iota)$ *for* $\mathbb{S}$ *and* s *respectively, and from noting that interpretation into* $\mathcal{F}$ *is injective on substitutions.*

Theorem 8.4.61 (Stability). *The normal-form algorithm of definition 8.4.56 is stable;* i.e.*, for normal inputs the output is equal with the input. This can be derived from fact 8.4.59.*

8.4.5 *P-Presheaves in* $\widehat{\mathcal{U}}$

In this subsection, we define a number of P-presheaves upon which we depend for our third normalisation by gluing construction. The P-presheaves in this subsection all have their action given by neutral context substitution; thus, they are P-functors in $\widehat{\mathcal{U}} \equiv [\mathcal{U}^{\mathrm{op}}, \mathbf{PSet}]$.

We now proceed to define P-presheaves of terms and context substitutions, for which we have a choice as to what the PER for the P-sets are: it can be either equality (for the discrete P-set), or $\beta\eta$-conversion. We choose the former.

Definition 8.4.62 (P-Presheaf of Terms). The *P-presheaf of terms* at type T, $\mathcal{L}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{L}_T(\Gamma) \triangleq (\Gamma \vdash - \,:\, T; =)$; and
- $\mathcal{L}_T(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Theta)(\Theta \vdash t \,:\, T) \triangleq \Gamma \vdash t[\sigma] \,:\, T$.

Definition 8.4.63 (P-Presheaf of Substitutions). The *P-presheaf of substitutions* at context Δ, $\mathcal{L}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{L}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\mathrm{sub}} - \,:\, \Delta; =)$; and
- $\mathcal{L}_\Delta(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Theta)(\Theta \vdash_{\mathrm{sub}} \psi \,:\, \Delta) \triangleq \Gamma \vdash \psi \circ \sigma \,:\, \Delta$.

Remark 8.4.64. We use the same notation for the P-presheaf of renamings as that for the P-presheaf of variables, as we think of $\mathcal{L}_\Delta$ as the Δ-fold product of $\mathcal{L}_T$ for each T in Δ. We continue this notational abuse for all similarly related type-indexed and context-indexed families of P-presheaves and glued objects.

Construction 8.4.65 (Term Application Morphism). There is a P-well-defined application morphism for terms.

$$\mathsf{App}_{T_1;T_2} \,:\, \mathcal{L}_{T_1 \to T_2} \times \mathcal{L}_{T_1} \to \mathcal{L}_{T_2}$$

Definition 8.4.66 (P-Presheaf of Neutral Terms). The *P-presheaf of neutral terms* at type T, $\mathcal{M}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{M}_T(\Gamma) \triangleq (\Gamma \vdash - \,:\, T; t\, t' \mapsto \Gamma \vdash_{\mathcal{M}} t \,:\, T \wedge \Gamma \vdash_{\mathcal{M}} t' \,:\, T \wedge t = t')$; and
- $\mathcal{M}_T(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Theta)(\Theta \vdash t \,:\, T) \triangleq \Gamma \vdash t[\sigma] \,:\, T$.

Definition 8.4.67 (P-Presheaf of Neutral Substitutions). The *P-presheaf of neutral substitutions* at context Δ, $\mathcal{M}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{M}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\mathrm{sub}} - \,:\, \Delta; \sigma\, \sigma' \mapsto \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma \,:\, \Delta \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma' \,:\, \Delta \wedge \sigma = \sigma')$; and
- $\mathcal{M}_\Delta(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Theta)(\Theta \vdash_{\mathrm{sub}} \psi \,:\, \Delta) \triangleq \Gamma \vdash \psi \circ \sigma \,:\, \Delta$.

Remark 8.4.68. The P-presheaves $\mathcal{M}_T$ and $\mathcal{M}_\Delta$ are equivalent to the Yoneda embedding at $(\bullet, T)$ and at Δ respectively.

Construction 8.4.69 (Term Abstraction Morphism). There is a P-well-defined abstraction morphism for terms.

$$\mathsf{Abs}_{T_1;T_2} \,:\, (\mathcal{M}_{T_1} \Rightarrow \mathcal{L}_{T_2}) \to \mathcal{L}_{T_1 \to T_2}$$

Definition 8.4.70 (P-Presheaf of Normal Terms). The *P-presheaf of normal terms* at type T, $\mathcal{N}_T$, is a type-indexed family of P-presheaves defined by the sequel:

- $\mathcal{N}_T(\Gamma) \triangleq (\Gamma \vdash - \,:\, T; t\, t' \mapsto \Gamma \vdash_{\mathcal{N}} t \,:\, T \wedge \Gamma \vdash_{\mathcal{N}} t' \,:\, T \wedge t = t')$; and
- $\mathcal{N}_T(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Theta)(\Theta \vdash t \,:\, T) \triangleq \Gamma \vdash t[\sigma] \,:\, T$.

Definition 8.4.71 (P-Presheaf of Normal Substitutions). The *P-presheaf of normal substitutions* at context Δ, $\mathcal{N}_\Delta$, is a context-indexed family of P-presheaves defined by the sequel:

- $\mathcal{N}_\Delta(\Gamma) \triangleq (\Gamma \vdash_{\text{sub}} - \,:\, \Delta; \sigma\, \sigma' \mapsto \Gamma \vdash_{\mathcal{N}\text{ sub}} \sigma \,:\, \Delta \wedge \Gamma \vdash_{\mathcal{N}\text{ sub}} \sigma' \,:\, \Delta \wedge \sigma = \sigma')$; and
- $\mathcal{N}_\Delta(\Gamma \vdash_{\text{sub}} \sigma \,:\, \Theta)(\Theta \vdash_{\text{sub}} \psi \,:\, \Delta) \triangleq \Gamma \vdash \psi \circ \sigma \,:\, \Delta$.

Construction 8.4.72 (Neutral/Normal Term Application Morphism). There is a P-well-defined application morphism for neutral terms with normal terms.

$$\mathsf{App}_{T_1;T_2} \,:\, \mathcal{M}_{T_1 \to T_2} \times \mathcal{N}_{T_1} \to \mathcal{M}_{T_2}$$

Construction 8.4.73 (Normal Term Abstraction Morphism). There is a P-well-defined abstraction morphism for normal terms.

$$\mathsf{Abs}_{T_1;T_2} \,:\, (\mathcal{M}_{T_1} \Rightarrow \mathcal{N}_{T_2}) \to \mathcal{N}_{T_1 \to T_2}$$

8.4.6 *Third Gluing Category*

In this subsection, we define the third P-gluing category that we use to establish a normal-form algorithm, and objects and morphisms thereof.

Remark 8.4.74. In the sequel we fix a pointed P-Cartesian-closed category, $\mathbb{S}$, with chosen object s.

Definition 8.4.75 (Third P-Gluing Category). The *Third P-gluing category*, which we denote by $\mathcal{G}_3$, is the P-Cartesian-closed P-comma category:

$$\mathcal{G}_3 \triangleq (\mathrm{Id}_{\widehat{\mathcal{U}}} \downarrow \langle [\![-]\!]_s \circ j \circ i_2 \rangle)$$

We now proceed to define objects and morphisms in the P-gluing category $\mathcal{G}_3$.

Definition 8.4.76 (Glued Object of Terms). The *glued object of terms* at type T, $\overline{\mathcal{L}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{L}}_T \triangleq \begin{pmatrix} \mathcal{L}_T \\ \downarrow \\ [\![\Gamma \vdash - \,:\, T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{pmatrix}$$

Definition 8.4.77 (Glued Object of Substitutions). The *glued object of substitutions* at context Δ, $\overline{\mathcal{L}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{L}}_\Delta \triangleq \begin{pmatrix} \mathcal{L}_\Delta \\ \downarrow \\ [\![\Gamma \vdash_{\text{sub}} - \,:\, \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{pmatrix}$$

Definition 8.4.78 (Glued Object of Neutral Terms). The *glued object of neutral terms* at type T, $\overline{\mathcal{M}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{M}}_T \triangleq \begin{pmatrix} \mathcal{M}_T \\ \downarrow \\ [\![\Gamma \vdash - \,:\, T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{pmatrix}$$

Definition 8.4.79 (Glued Object of Neutral Substitutions). The *glued object of neutral substitutions* at context Δ, $\overline{\mathcal{M}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{M}}_\Delta \triangleq \left(\begin{array}{c} \mathcal{M}_\Delta \\ \big| \\ [\![\Gamma \vdash_{\text{sub}} - : \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{array} \right)$$

Definition 8.4.80 (Glued Object of Normal Terms). The *glued object of normal terms* at type T, $\overline{\mathcal{N}}_T$, is an extension of the type-indexed family of P-presheaves to a type-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{N}}_T \triangleq \left(\begin{array}{c} \mathcal{N}_T \\ \big| \\ [\![\Gamma \vdash - : T]\!]_s \\ \downarrow \\ [\![T]\!]_s \end{array} \right)$$

Definition 8.4.81 (Glued Object of Normal Substitutions). The *glued object of normal substitutions* at context Δ, $\overline{\mathcal{N}}_\Delta$, is an extension of the context-indexed family of P-presheaves to a context-indexed family of objects in $\mathcal{G}_3$, and is defined by the sequel:

$$\overline{\mathcal{N}}_\Delta \triangleq \left(\begin{array}{c} \mathcal{N}_\Delta \\ \big| \\ [\![\Gamma \vdash_{\text{sub}} - : \Delta]\!]_s \\ \downarrow \\ [\![\Delta]\!]_s \end{array} \right)$$

Construction 8.4.82 (Neutral/Normal Term Application Morphism). There is a P-well-defined application morphism for neutral terms with normal terms.

$$\overline{\mathsf{App}}_{T_1;T_2} \; : \; \overline{\mathcal{M}}_{T_1 \to T_2} \times \overline{\mathcal{N}}_{T_1} \to \overline{\mathcal{M}}_{T_2}$$

Construction 8.4.83 (Normal Term Abstraction Morphism). There is a P-well-defined abstraction morphism for normal terms.

$$\overline{\mathsf{Abs}}_{T_1;T_2} \; : \; (\overline{\mathcal{M}}_{T_1} \Rightarrow \overline{\mathcal{N}}_{T_2}) \to \overline{\mathcal{N}}_{T_1 \to T_2}$$

Definition 8.4.84 (Interpretation in $\mathcal{G}_3$). We can interpret $\mathcal{F}$ into $\mathcal{G}_3$ by choosing $\overline{\mathcal{M}}_\iota$ as the chosen object for interpreting ι.

Construction 8.4.85 (In/Out Morphisms). There are P-well-defined type-indexed and context-indexed families of morphisms:

- $\overline{\mathsf{in}}_T \; : \; \overline{\mathcal{M}}_T \to \mathcal{G}_3[\![T]\!]$;
- $\overline{\mathsf{in}}_\Delta \; : \; \overline{\mathcal{M}}_\Delta \to \mathcal{G}_3[\![\Delta]\!]$;
- $\overline{\mathsf{out}}_T \; : \; \mathcal{G}_3[\![T]\!] \to \overline{\mathcal{N}}_T$; and
- $\overline{\mathsf{out}}_\Delta \; : \; \mathcal{G}_3[\![\Delta]\!] \to \overline{\mathcal{N}}_\Delta$.

Remark 8.4.86. We denote the domain parts of the $\overline{\mathsf{in}}$ and $\overline{\mathsf{out}}$ morphisms by in and out.

Fact 8.4.87. *The following facts hold of the codomain parts of the* $\overline{\mathsf{in}}$ *and* $\overline{\mathsf{out}}$ *morphisms:*

- $\mathrm{Cod}(\overline{\mathsf{in}}_T) \sim q_T$;

- $\mathrm{Cod}(\overline{\mathsf{in}}_\Delta) \sim q_\Delta$;
- $\mathrm{Cod}(\overline{\mathsf{out}}_T) \sim u_T$; *and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta) \sim u_\Delta$;

where the q and the u are induced by the universal property with respect to the following p*-Cartesian-closed functor.*

$$\mathrm{Cod} \circ \mathcal{G}_3[\![-]\!] \,:\, \mathcal{F} \to \mathbb{S}$$

We therefore have the following facts as corollaries:

- $\mathrm{Cod}(\overline{\mathsf{out}}_T \circ \mathcal{G}_3[\![\Gamma \vdash t \,:\, T]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash t \,:\, T]\!]$; *and*
- $\mathrm{Cod}(\overline{\mathsf{out}}_\Delta \circ \mathcal{G}_3[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!] \circ \overline{\mathsf{in}}_\Gamma) \sim \mathbb{S}[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]$.

Definition 8.4.88 (Normalisation in $\mathcal{G}_3$)**.**

$$\mathrm{nf}_{\mathcal{G}_3}(\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta) \triangleq (\mathsf{out})_\Delta \circ \mathrm{Dom}(\mathcal{G}_3[\![\Gamma \vdash_{\mathrm{sub}} \sigma \,:\, \Delta]\!]) \circ \mathsf{in}_\Gamma)_\Gamma(\mathrm{id}_\Gamma)$$

Remark 8.4.89. Note that definition 8.4.88 for normalisation in $\mathcal{G}_3$ implicitly depends upon a pointed p-Cartesian-closed category; *viz.*, $\mathbb{S}$ with chosen point s. For concrete computation, it suffices to instantiate these with the terminal p-Cartesian-closed category and the terminal object thereof.

Fact 8.4.90. *The following fact holds.*

$$\Gamma \vdash_{\mathrm{sub}} \sigma \equiv_{\beta\eta} \sigma' \,:\, \Delta \Rightarrow \mathbb{S}[\![\mathrm{nf}_{\mathcal{G}_3}(\sigma)]\!] \sim \mathbb{S}[\![\sigma']\!]$$

Fact 8.4.91. *The following facts hold:*

- $\Delta \vdash i_n \,:\, T \wedge \Gamma \vdash_{\mathcal{M}\,\mathrm{sub}} \sigma \,:\, \Delta \Rightarrow \mathsf{in}_{T,\Gamma}(i[\sigma]) \sim \mathrm{Dom}(\mathcal{G}_3[\![\Gamma \vdash i_n \,:\, T]\!])_\Gamma(\mathsf{in}_{\Delta,\Gamma}(\sigma))$;
- $\Delta \vdash_{\mathcal{M}} t \,:\, T \Rightarrow \mathsf{in}_{T,\Gamma}(t) \sim \mathrm{Dom}(\mathcal{G}_3[\![\Gamma \vdash t \,:\, T]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))$; *and*
- $\Delta \vdash_{\mathcal{N}} t \,:\, T \Rightarrow \mathsf{out}_{T,\Gamma}(\mathrm{Dom}(\mathcal{G}_3[\![\Gamma \vdash t \,:\, \Delta]\!])_\Gamma(\mathsf{in}_{\Gamma,\Gamma}(\mathrm{id}_\Gamma))) = t$.

Theorem 8.4.92 (Soundness)**.** *The normal-form algorithm of definition 8.4.88 is sound;* i.e.*, the output is $\beta\eta$-convertible with the input. This can be derived from substituting $\mathcal{F}$ and $(\bullet,\ \iota)$ for $\mathbb{S}$ and s respectively, and from noting that interpretation into $\mathcal{F}$ is injective on substitutions.*

Theorem 8.4.93 (Stability)**.** *The normal-form algorithm of definition 8.4.88 is stable;* i.e.*, for normal inputs the output is equal with the input. This can be derived from fact 8.4.91.*

8.4.7 *Advantages of $\widehat{\mathcal{U}}$ over $\widehat{\mathcal{R}}$ and $\mathcal{G}_3$ over $\mathcal{G}_2$*

The third p-gluing construction has two main advantages associated thereto:

- fewer objects in the p-gluing category are needed; and
- no deep coercions are needed.

The first advantage arises from the p-presheaf of neutral terms/context substitutions over neutral context substitutions being the Yoneda embedding, and therefore these p-presheaves have two rôles within the construction: the domain p-presheaf for normal term abstraction, and the domain for the in morphisms. Since fewer objects in $\mathcal{G}_3$ are needed, we need only define fewer p-presheaves. Thus, the presentation and Rocq formalism of $\mathcal{G}_3$ is much shorter than for $\mathcal{G}_2$. The second

advantage arises from the i_1 inclusion P-functor having to coerce context renamings into context substitutions, which requires a deep recursive computation; the i_2 inclusion P-functor does not have to do any such computation, as it is simply forgetting logical property; *viz.*, the neutrality of the substitution. The presence of the coercion requires proving that the coercion respects the structures of context renamings within the structure of context substitutions which complicates the formalism. This computational efficiency arises from our choice of how to express the sub-P-set of neutral context substitutions; we could have defined a structure of intrinsically neutral context substitutions, but then we would have had to perform coercions into arbitrary context substitutions. Arguably, we could have defined a P-category of renamings where the underlying type for the morphisms is all context substitutions, but where the PER is restricted to only relate the substitutions that are composed of variables; this would have resolved some of the computational inefficiencies associated to using the P-category of renamings.

These advantages are also manifest in $\mathcal{G}_1$; however, $\mathcal{G}_1$ has the disadvantage that it does not automatically prove that the outputs of the associated normal-form algorithm are normal.

CHAPTER 9

Conclusions

In this chapter, we offer some conclusions to the body of the thesis. We summarise the contents of the thesis, emphasise the main contributions, and, finally, suggest some avenues for further work.

9.1 Summary and Contributions

This thesis has covered two main aspects: formal P-category theory and normalisation of simple type theory. These two aspects have been joined together with a full formalism in Rocq.

9.1.1 *Summary*

Chapters 1 and 2 presented introductory and background material for the thesis. As well as providing preliminaries for the body of the thesis, it reviewed the *status quo* on type-theoretic formalisms of category theory, and *status quo* on categorical normalisation by evaluation for simple type theories. Our work builds on these two areas by contributing a fuller theoretical development and type-theoretic formalism of P-category theory in Rocq, and by suggesting new categorical techniques for normalisation by evaluation for simple type theories.

P-Category theory, in chapter 4, has been extended from the nascent theory in Čubrić et al. (1998) into a more developed theory. In particular, we have contributed a definition of P-comma categories, arguing the correctness of the definition by comparing with the definition of P-functor categories using arrow P-categories as a mediating comparator. Additionally, we have contributed a definition of P-equalisers, thus characterising all finite limits within the setting of P-category theory (the work of Čubrić et al. (1998) already gave definitions for finite products). We have extended the theory of the P-category of P-sets by establishing P-end and P-coend constructions therefor, thereby establishing the completeness and cocompleteness thereof, and a construction of P-equalisers therefor, thereby establishing the finite completeness thereof. Moreover, we have provided a formal, and more philosophical, argument for the superiority of P-sets over E-sets, and thereby P-categories over E-categories, than the argument in Čubrić et al. (1998) given for the switch from E-categories to P-categories, by appealing to the limit and colimit theories of the (E/P)-categories of P-sets. Furthermore, we have connected the definition of P-categories give in Čubrić et al. (1998) with enriched and internal definitions placing them within the broader landscape of category theory.

In chapters 6 and 7, we formalised the syntax and P-categorical structure of simple type theory. Our formalism followed the technique of Benton et al. (2012) in formalising renamings first, and thereafter formalising substitutions. These formalisms formed the basis for formalising their respective categorical structures. We analysed the P-categorical structures of the P-categories of renamings, substitutions under α-equivalence, and substitutions under $\beta\eta$-conversion. Our analysis of substitutions with α-equivalence produced a novel universal property for the quotient map from substitutions under α-equivalence to substitutions under $\beta\eta$-conversion. This formalism has supported the formalisation of the normalisation construction from Čubrić et al. (1998), the formalisation of a novel universal property for unquotiented syntax allowing

for a novel categorical normalisation construction, and the formalisation of an adaptation of the gluing construction of Fiore (2002, 2022) to P-category theory. Our adaptation of the gluing construction of Fiore (2002, 2022), followed a more P-theoretic path by considering the sets of normal and neutral terms not to be their own inductively-defined structure, but to be sub-P-sets of the discrete P-set of terms. Moreover, it required further adaptation to the dependent type-theory of Rocq, allowing us to characterise some of the maps induced in the construction as canonical maps from the universal property of the P-category of substitutions under $\beta\eta$-conversion.

Chapter 5 extended P-category theory into the realm of bicategory theory, thereby demonstrating its utility as a variation on standard category theory, and also allowing some formal P-category theory to be developed. We contributed definitions of P-bicategories, P-pseudofunctors, P-modifications, and P-pseudonatural adjoint equivalences. We assembled these definitions into their respective P-categorical and P-bicategorical structures. These structures were then used to produce definitions of finite pseudo-products in P-bicategories, thereby suggesting the correctness of the P-theoretic definitions by their ability to be used similarly to how standard bicategorical definitions are used. Furthermore, we contributed a definition of P-pseudo-comma bicategories allowing us to define succinctly the P-bicategory of pointed P-categories, pointed P-functors, and pointed P-natural transformations.

Finally, we combined much of the formalism throughout the thesis to state and prove the P-bicategorical pseudo-initiality properties of $\mathcal{F}$ and $j : \mathcal{A} \to \mathcal{F}$.

9.1.2 *Contributions*

The primary contributions of this thesis are the development of P-category theory and P-bicategory theory, and the full formalism of the P-bicategorical pseudo-initiality properties of the categorical structure of simple type theory. The development of P-category theory as an alternative to E-category theory for formalism in the non-univalent setting was defended from a philosophical perspective, emphasising its structural separation of computational parts from logical parts. This separation is well-motivated in the computational setting of the Rocq formalism.

9.2 Future Work

There are a number of avenues for further work beyond that which is contained within this thesis.

A fully-P-theoretic form of category theory should be developed incorporating partiality for objects by using an explicit predicate for P-well-definition of objects. P-categories should be recoverable from this form of category by taking Σ-types of objects with their P-well-definition predicate.

The P-bicategory theory should be extended to form a larger formalisation of various P-bicategorical constructions, including the structure of the P-bicategory of P-categories and the structure of **PCat**-valued P-pseudofunctors.

The normalisation constructions could be extended to allow for freeness over other generating data. The simple type theory could be extended to include sum types, or coproduct types incorporating the work of Altenkirch et al. (2001) and Balat et al. (2004a,b). The inclusion of algebraic datatypes, such as the natural numbers, with recursion thereover in the simple type theory should be investigated. Specifically it should be studied whether the functorial approach of Čubrić et al. (1998) extend to theories with datatypes of various descriptions. Moreover, does the approach to Cartesian-pre-closed structure extend to these settings. Additionally, the type theory could be extended into the non-simple setting involving polymorphism, following the work of Altenkirch et al. (1996), or dependent types, following the work of Altenkirch et al. (2016). Polymorphism would require a notion of P-functor from P-categories to P-bicategories; such dimension-raising functors should also be studied abstractly within the P-setting (possibly only within a fully P-theoretic framework). Finally, the approaches to normalisation in this thesis should be tried for their capacity to be extended to normalisation for dependent type theories. For this a notion of model for dependent type theory would have to be translated into the P-setting. Natural models, being more categorical in structure, could provide a strong basis for such investigations. A notion of non-discrete dependent product of P-sets should also be developed that provides an equivalence between fibred and indexed approaches to families of (P-)sets.

The P-categorical analysis of the syntax of simple type theory should be extended to include analyses of extrinsically well-scoped and extrinsically well-typed syntax by placing the well-scoping and well-typing requirements into an appropriate PER. The changes required for this should facilitate extensions to non-simple type theories, and particularly to dependent type theories where extrinsically well-typed syntax is more well-understood than intrinsically well-typed syntax.

The P-bicategorical initiality property of the quotient functor, $j : \mathcal{A} \to \mathcal{F}$, could be made cleaner and less *ad hoc*. The approach towards gluing resulting from this universal property could be more closely connected with the traditional gluing techniques of Fiore (2002, 2022).

Finally, some of the categorical results obtained in this thesis are independent of our choice of P-category theory as a categorical framework for type-theoretic formalism. As a demonstration of this, they should be translated into alternative categorical frameworks such as univalent category theory.

Bibliography

Altenkirch, T., Hofmann, M., and Streicher, T. (1996). "Reduction-free normalisation for a polymorphic system". In: *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pp. 98–106.

Altenkirch, T., Dybjer, P., Hofmann, M., and Scott, P. (2001). "Normalization by evaluation for typed lambda calculus with coproducts". In: *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*, pp. 303–310.

Altenkirch, T., Hofmann, M., and Streicher, T. (1995). "Categorical reconstruction of a reduction free normalization proof". In: *Category Theory and Computer Science*. Ed. by D. Pitt, D. E. Rydeheard, and P. Johnstone. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 182–199. ISBN: 978-3-540-44661-3.

Altenkirch, T. and Kaposi, A. (2016). "Normalisation by Evaluation for Dependent Types". In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*. Ed. by D. Kesner and B. Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 6:1–6:16. ISBN: 978-3-95977-010-1.

Balat, V., Di Cosmo, R., and Fiore, M. (2004a). "Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums". In: *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '04. Venice, Italy: Association for Computing Machinery, pp. 64–76. ISBN: 158113729X.

Balat, V., Di Cosmo, R., and Fiore, M. (2004b). "Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums". In: *SIGPLAN Not.* 39.1, pp. 64–76. ISSN: 0362-1340.

Barthe, G., Capretta, V., and Pons, O. (2003). "Setoids in type theory". In: *Journal of Functional Programming* 13.2, pp. 261–293. DOI: 10.1017/S0956796802004501.

Bénabou, J. (1967). "Introduction to bicategories". In: *Reports of the Midwest Category Seminar*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–77. ISBN: 978-3-540-35545-8.

Benton, N., Hur, C.-K., Kennedy, A. J., and McBride, C. (2012). "Strongly Typed Term Representations in Coq". In: *Journal of Automated Reasoning* 49.2, pp. 141–159. ISSN: 0168-7433.

Berger, U. and Schwichtenberg, H. (1991). "An inverse of the evaluation functional for typed lambda λ-calculus". In: *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pp. 203–211.

Berry, D. (2026). "Code supporting thesis 'Formal p-Category Theory and Normalisation for Simple Type Theory"'. In: DOI: 10.17863/CAM.124568.

Bishop, E. (1967). *Foundations of constructive analysis*. McGraw-Hill series in higher mathematics. New York: McGraw-Hill.

Church, A. (1940). "A formulation of the simple theory of types". In: *Journal of Symbolic Logic* 5.2, pp. 56–68.

Cottrell, T., Fujii, S., and Power, J. (2017). "Enriched and internal categories: an extensive relationship". In: *Tbilisi Mathematical Journal* 10.3, pp. 239–254.

Čubrić, D., Dybjer, P., and Scott, P. (1998). "Normalization and the Yoneda embedding". In: *Mathematical Structures in Computer Science* 8.2, pp. 153–192.

Eilenberg, S. and MacLane, S. (1945). "General Theory of Natural Equivalences". In: *Transactions of the American Mathematical Society* 58.2, pp. 231–294. ISSN: 00029947, 10886850.

Fiore, M. (2002). "Semantic analysis of normalisation by evaluation for typed lambda calculus". In: *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP '02. Pittsburgh, PA, USA: Association for Computing Machinery, pp. 26–37. ISBN: 1581135289.

Fiore, M. (2022). "Semantic analysis of normalisation by evaluation for typed lambda calculus". In: *Mathematical Structures in Computer Science* 32.8, pp. 1028–1065.

Gilbert, G., Cockx, J., Sozeau, M., and Tabareau, N. (2019). "Definitional proof-irrelevance without K". In: *Proc. ACM Program. Lang.* 3.POPL.

Gross, J., Chlipala, A., and Spivak, D. I. (2014). "Experience Implementing a Performant Category-Theory Library in Coq". In: *Interactive Theorem Proving*. Ed. by G. Klein and R. Gamboa. Cham: Springer International Publishing, pp. 275–291. ISBN: 978-3-319-08970-6.

Hofmann, M. and University of Edinburgh College of Science and Engineering School of Informatics (1995). "Extensional concepts in intensional type theory". PhD thesis. Edinburgh, UK: University of Edinburgh.

Hu, J. Z. S. and Carette, J. (2021). "Formalizing category theory in Agda". In: *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2021. Virtual, Denmark: Association for Computing Machinery, pp. 327–342. ISBN: 9781450382991.

Huet, G. and Saïbi, A. (2000). "Constructive Category Theory". In: *Proof, Language, and Interaction: Essays in Honour of Robin Milner*. The MIT Press. ISBN: 9780262281676.

Jay, C. B. and Ghani, N. (1995). "The virtues of eta-expansion". In: *Journal of Functional Programming* 5.2, pp. 135–154.

Kahl, W. (1998). "Relational treatment of term graphs with bound variables". In: *Logic Journal of the IGPL* 6.2, pp. 259–303. ISSN: 1367-0751.

Lack, S. G. and University of Cambridge Department of Pure Mathematics and Mathematical Statistics (1996). "The algebra of distributive and extensive categories". PhD thesis. Cambridge, UK: University of Cambridge.

Palmgren, E. (2019). "On Equality of Objects in Categories in Constructive Type Theory". In: *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*. Ed. by A. Abel, F. Nordvall Forsberg, and A. Kaposi. Vol. 104. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 7:1–7:7. ISBN: 978-3-95977-071-2.

Rothgang, C. and Rabe, F. (2025). *Subtyping in DHOL – Extended preprint*. arXiv: 2507.02855 [cs.LO].

Scott, D. (1979). "Identity and existence in intuitionistic logic". In: *Applications of Sheaves: Proceedings of the Research Symposium on Applications of Sheaf Theory to Logic, Algebra, and Analysis, Durham, July 9–21, 1977*. Ed. by M. Fourman, C. Mulvey, and D. Scott. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 660–696. ISBN: 978-3-540-34849-8.

Sterling, J. and Carnegie Mellon University Department of Computer Science (2021). "First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory". PhD thesis. Pittsburgh, PA, USA: Carnegie Mellon University.

The 1Lab Development Team (2024). *The 1Lab*. URL: https://1lab.dev.