DEPARTMENT OF COMPUTER SCIENCE

Soundness of an Idealised Semantics for Functional Logic Programming

[redacted]

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Wednesday 3rd May, 2023

Abstract

Despite 30 years of research into functional logic programming (FLP) languages, there exists no satisfying account of the paradigm's semantics in a general setting. This thesis presents a core FLP language, developed by Kavvos, based on the Call-By-Push-Value (CBPV) calculus by considering the non-deterministic features of FLP as computational effects. CBPV makes a distinction between values and computations, controlling effectful computation explicitly. This provides an elegant basis for the semantics of FLP languages which is not tied to an evaluation strategy.

We define an operational and denotational semantics for the core FLP language which treats non-determinism via a monad, covering interpretations through powersets, lists and other structures.

The main contribution of the thesis is a proof of the soundness of this denotational semantics with respect to an equational theory we present. In other words, we prove that the same mathematical object in the denotational semantics represents any terms equated in the equational theory. This means the equational theory for the language does not contradict the mathematical interpretation given by the denotational semantics. Additionally, we also show that the operational semantics is sound with respect to the denotational semantics, meaning the mathematical interpretation of a program is not changed after evaluating it through the big-step operational relation we present.

The semantics is idealised in that it supports infinitary non-determinism but despite this provides a useful basis for comparison with practically implementable FLP languages, in order to judge their correctness. Previous to this work such a comparison point did not exist.

Contents

1 Introduction **1**
1.1 What are Functional Logic Programming Languages? 1
1.2 Current Problems Facing FLP Languages 2
1.3 Aims and Objectives 3

2 Tools Used **4**
2.1 Call-By-Push-Value 4
2.2 Denotational Semantics and Algebraic Structures for Effects 6

3 The Language **9**

4 Idealised Semantics **11**
4.1 Semantic Structures 11
4.2 Operational Semantics 18
4.3 Denotational Semantics 20
4.4 Equational Theory 23

5 Soundness of Denotational Semantics **28**
5.1 Preliminary Results 28
5.2 Main Proof 40

6 Soundness of Operational Semantics **51**

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Alex Kavvos.

Chapter 1

Introduction

1.1 What are Functional Logic Programming Languages?

At a high-level Functional Logic Programming (FLP) languages combine the features of functional programming languages and logic programming languages. This combination of declarative programming paradigms aims to combine their key features to facilitate more expressive and concise programming.

Functional programming has long been hailed for enabling programmers to produce code which is more modular and composable [1]. Logic programming gives programmers tools to represent knowledge of a problem domain in a declarative manner and infer solutions from this knowledge by logical deduction. In theory, these paradigms aid the programmer in focusing less on specific computational details and more on the problem at hand. FLP languages aim to further this by combining the two paradigms, equipping the programmer with an enhanced set of tools making these languages suited to a wider range of problems.

For a reader unfamiliar with these two paradigms, functional programming languages are often characterised as declarative languages whose underlying model of computation is the mathematical function as opposed to the command-centred basis of imperative languages. In particular, any practical functional language treats functions as first-class citizens in the sense that they support recursive, higher-order and polymorphic functions. This enables a modular style of programming where highly-generic functions can be widely reused in composition with others to build solutions. In terms of syntax, they often take an equational style much like mathematical notation [2].

Often when discussing the functional programming paradigm, there is a focus on pure functional languages, such as Haskell, which do not allow for side effects. These languages allow for straightforward equational reasoning about their behaviour as a lack of side effects removes the need to consider wider program state [1].

Logic programming languages are characterised by modelling computation via relations, often in the forms of facts and rules by which a knowledge base is created. This knowledge base is used to return solutions to queries posed by the programmer.

In order to facilitate this computational model logic programming languages make use of predicates, logical variables and non-determinism [3]. Predicates are boolean-valued functions which are used to represent relations between arguments. Predicates which are always true are called facts, while those which are true under certain conditions are called rules. Logical variables are used to represent unknown values and can be used to query the knowledge base for solutions. Solutions (valid instantiations of logical variables) are found through unification, the process of matching terms in a query to those in the knowledge base. Logical variables are instantiated during the unification algorithm, this process is non-deterministic as there may be none, one or many solutions to a query.

It is worth noting implementations of logic programming are not always strictly declarative, in particular, Prolog allows for some imperative commands to deal with input and output as well as a *cut* operator to control backtracking [4].

The idea to combine functional and logic programming is not new, research in the area roughly spans the last three decades, with most languages proposed based on either extending existing logic languages with functional features or vice versa [5]. Functional logic languages aim to combine the strengths of both paradigms, giving the benefits of function inversion, partial data structures and logical variables over purely functional languages, while also providing the benefits of far more expressive deterministic evaluation than predicates in logic languages, which can alleviate the need for non-declarative features such as the aforementioned *cut* operator in Prolog [6].

Notable modern functional logic programming languages include Mercury [7] and Curry [8]. Mercury can be seen as an FLP language based on Prolog, with a focus on efficient implementation which results in a restriction on the logic programming features available, in particular the non-deterministic evaluation of functions. Despite this restriction it is a fully declarative language which sees niche industry usage [9]. On the other hand, Curry appears as a syntactic extension of Haskell, with fewer restrictions to the interoperation of functional and logic programming features. This allows for using logical variables to express the relationship between the arguments of a function, enabling specific FLP design patterns [10].

More recently there has been renewed interest in the functional logic paradigm by researchers at Epic Games [11]. They seek to formulate a simple core calculus for a functional logic programming language they aim to develop named *Verse*. In the draft paper, they present a semantics for a simple untyped FLP language based on rewrite rules, with the aim of showing these rules are confluent. This recent work shows the continued interest in FLP languages, particularly the formulation of a satisfying semantics which the paradigm currently lacks.

1.1.1 Example Program

To give a concrete example of a functional logic program, we use a syntax similar to Haskell. We consider the problem of extracting the last element from a list. Rather than the usual recursive definition used in functional languages, we use logical variables to define this function.

```
last :: [a] -> a
last xs = ∃ ys : [a]. ∃ y : a. xs =:= ys ++ [y]. y
```

The last function takes a list of type a called xs, it then introduces two logical variables ys of type $[a]$ and y of type a. We then use the *unification* operator =:= to equate the list xs with the concatenation of ys and y. In this way, we constrain the value of ys to be the prefix of the list xs up to the last element and the value of y to be the last element of xs. Finally, we return y which must take the last value of the inputted list.

In comparison to functional languages, the FLP approach is arguably more descriptive of the property the last function implements, namely returning the last element of a list. The purely functional approach would pattern match on the shape of xs and recursively unwrap it until the end of the list is reached, which instead more closely resembles the computational steps needed to process such a list.

1.2 Current Problems Facing FLP Languages

Critically despite the long-standing interest in functional logic programming languages, there is no well-defined semantics for the paradigm. A semantics for a programming language provides a way to reason about and understand the programs written in that language. Programming language semantics come in a number of forms, which can be broadly categorised as either operational or denotational. Operational semantics, often characterised as small-step or big-step, provide a way to reason about the behaviour of a program by describing the steps taken to evaluate a program. Denotational semantics assigns a mathematical meaning directly to expressions in the language, providing a way to reason about programs in the language by considering the mathematical objects associated with them, independent of a machine implementation. While operational semantics can be useful in understanding how a program is evaluated on a machine, denotational semantics can be a more powerful tool for proving the properties of programs.

There is some existing work which formulates an operational semantics for Curry [12], however, the proposed semantics are heavily implementation focused making them hard to use for proving properties of programs in the language. Additionally, they lack utility for functional logic programming languages in general which follow a different implementation strategy. It is evident a core calculus for functional logic programming languages is needed to provide a basis for further research in the area. Ideally, this calculus will avoid implementation details and be sufficiently general to be applicable to a wide range of functional logic programming languages. By proving properties of this core FLP language, in particular developing an operational and denotational semantics which we prove are sound, we can compare future works in the area to this core calculus. This comparison will provide a way to evaluate the design of new FLP languages and their semantics.

1.3 Aims and Objectives

We aim to work towards the formulation of a core functional logic programming language equipped with a corresponding operational and denotational semantics.

In brief, we will formulate a core idealised FLP calculus, the language will be idealised in the sense it will support infinitary non-determinism, which is unlikely to be useable for direct implementation but provides a useful model to compare against. To model the non-deterministic features of functional logic programming we will consider them as computational effects tacked onto the purely functional language. In doing so, we will base our language upon the Call-By-Push-Value (CBPV) calculus [13]. Call-By-Push-Value provides an elegant basis for the study of functional languages with effects by creating a clear separation between values and computations. The CBPV type system explicitly controls when a computation can and will evaluate with effects. Furthermore, the type system of CBPV draws a distinction between computations which are functions and those which return a value, which proves useful for defining a clean semantic interpretation. Additionally, basing our core FLP language on the CBPV paradigm enables our work to be translated to both call-by-value and call-by-name FLP languages, as the CBPV calculus subsumes both.

We will then develop both an operational and denotational semantics for the core FLP calculus, this will be based on the semantics of the CBPV calculus. An operational semantics gives a precise description of how a program is evaluated, we will take a big-step approach which describes the operation of a program in terms of the final value it evaluates to. A denotational semantics provides the expressions in the language syntax with a mathematical 'denotation' or meaning. By assigning the types and terms of the language with mathematical objects, we can make use of existing mathematical tools to reason about the language. In particular making use of the well-studied literature on algebraic effects to handle non-deterministic effects in the language.

Additionally, we abstract over the specific interpretation details of this non-deterministic behaviour by using a monad. This allows our results to be generalised over many interpretations of non-determinism. For example, we will directly consider the unordered interpretation as the powerset monad, and the ordered interpretation as the list monad. It is also clear that a tree monad could be used to provide a finer-grained trace interpretation of the non-deterministic effects, which could be useful in a pragmatic implementation for debugging. However, we do not consider the tree model in full as it lacks the associative properties we desire for non-deterministic computation in the FLP language we present. By developing our semantics with this monadic abstraction it should be straightforward to generalise our results to the tree model and other interpretations in the future.

Moreover, we will present an equational theory for the FLP language, equating certain programs with each other. This provides a means by which programs in the language can be reasoned about, free from implementation details.

The main contribution of this thesis is a proof of the soundness of the denotational semantics of the core FLP calculus. The soundness property informs us that the denotational semantics and equational theory are consistent with each other, i.e. terms which are equated by the equational theory are assigned the same denotation by the denotational semantics. This effectively supports the use of the equational theory to reason about the programs of the FLP language, as they do not cause contradictions within the mathematical meaning we give to the programs.

Furthermore, we will also show that the operational semantics are sound with respect to the denotational semantics. This means that the mathematical interpretation of programs is the same as the mathematical interpretation of the program's results under the big-step relation we present. This shows that the operational semantics defines the process of evaluation in the language in a way which is consistent with the mathematical meaning given to programs.

In summary, our aims and objectives are as follows:

1. Formulate a core functional logic programming language based on CBPV.
2. Define an operational and denotational semantics for the core FLP language.
3. Define an equational theory for the core FLP language.
4. Prove the soundness of the denotational semantics.
5. Prove the soundness of the operational semantics.

Chapter 2

Tools Used

In this chapter, we outline the existing mathematical tools and methods used within this thesis in order to aid the reader's understanding of the contributions made. This presentation aims to enable a reader unfamiliar with these tools to follow the work presented in this thesis. However, we do not aim to provide a fully motivated overview of the work we rely on, instead, the reader should refer to the relevant literature for more information.

We provide a brief overview of the call-by-push-value calculus [13], which is based on the introductory article by Levy [14]. We assume the reader is comfortable with the simply typed λ-calculus.

2.1 Call-By-Push-Value

The Call-By-Push-Value (CBPV) calculus provides a fundamental basis for the study of functional languages with computational effects. It is a form of typed λ-calculus which creates a clear separation between values and computations through the type system. Computation terms have a corresponding innate value term, which is called their thunk. To evaluate a thunk, it must be 'forced' making explicit its return to an effectful computation. CBPV subsumes the call-by-value and call-by-name evaluation strategies and can be translated to and fro while preserving the semantics of the language.

2.1.1 Syntax

The CBPV calculus makes a distinction between values and computations by the type system, which admits the following types:

$$\begin{array}{llcll} \text{value types} & A & ::= & \mathsf{Num} & \text{numbers} \\ & & & A_1 \times A_2 & \text{pairs} \\ & & & U\underline{B} & \text{thunks} \\ \text{computation types} & \underline{B} & ::= & FA & \text{returners} \\ & & & A \to \underline{B} & \text{functions} \end{array}$$

We note the definitions are mutually recursive. Each value type A has a corresponding computation type FA, the type of computations which return values. Similarly, for each computation type $\underline{B}$ there is a value type $U\underline{B}$ which corresponds to the thunks of those computations. The type $A \to \underline{B}$ is a computation type for functions that take inputs of value type A and continue as computations of type $\underline{B}$. In particular, we note any useful program will be of returner type.

We define typing contexts in much the same way as the simply-typed λ-calculus, except we limit typing contexts to only contain variables of value type:

$$\begin{array}{llcrl} \text{contexts} & \Gamma & ::= & & \text{empty context} \\ & & & \Gamma, x : A & \text{context extension} \end{array}$$

Now values and computations are built up by the following syntax:

values	V, W	$::=$	x	variables
			$\overline{n}$	number
			(V_1, V_2)	value pair
			$\mathsf{thunk}\ M$	thunk
computations	M, N	$::=$	$\mathsf{return}\ V$	produce
			$M_1\ \mathsf{to}\ x.\, M_2$	bind
			$\lambda x : A.\, M$	abstraction
			$M(V)$	application
			$\mathsf{ifz}(V; M; x.\, N)$	zero test
			$\mathsf{split}(V; x_1, x_2.\, M)$	split pair
			$\mathsf{force}\ V$	force thunk

The following typing rules define the typing relation for CBPV:

$$\textsc{Var}\ \frac{}{\Gamma, x : A \vdash^{\mathsf{v}} x : A} \qquad \textsc{Nat}\ \frac{n \in \mathbb{N}}{\Gamma \vdash^{\mathsf{v}} \overline{n} : \mathsf{Num}} \qquad \textsc{Nat}\ \frac{n \in \mathbb{N}}{\Gamma \vdash^{\mathsf{v}} \overline{n} : \mathsf{Num}} \qquad \textsc{Tuple}\ \frac{\Gamma \vdash^{\mathsf{v}} V_1 : A_1 \qquad \Gamma \vdash^{\mathsf{v}} V_2 : A_2}{\Gamma \vdash^{\mathsf{v}} (V_1, V_2) : A_1 \times A_2}$$

$$\textsc{Thunk}\ \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathsf{thunk}\ M : U\underline{B}} \qquad \textsc{Return}\ \frac{\Gamma \vdash^{\mathsf{v}} M : A}{\Gamma \vdash^{\mathsf{c}} \mathsf{return}\ M : FA} \qquad \textsc{Bind}\ \frac{\Gamma \vdash^{\mathsf{c}} M : FA \qquad \Gamma, x : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathsf{to}\ x.\, N : \underline{B}}$$

$$\textsc{Lam}\ \frac{\Gamma, x : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda x : A.\, M : A \to \underline{B}} \qquad \textsc{App}\ \frac{\Gamma \vdash^{\mathsf{c}} M : A \to \underline{B} \qquad \Gamma \vdash^{\mathsf{v}} N : A}{\Gamma \vdash^{\mathsf{c}} M(N) : \underline{B}}$$

$$\textsc{Split}\ \frac{\Gamma \vdash^{\mathsf{v}} V : A_1 \times A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathsf{split}(V; x_1, x_2.\, M) : \underline{B}}$$

$$\textsc{Ifz}\ \frac{\Gamma \vdash^{\mathsf{v}} V : \mathsf{Num} \qquad \Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma, x : \mathsf{Num} \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathsf{ifz}(V; M; x.\, N) : \underline{B}} \qquad \textsc{Force}\ \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathsf{force}\ V : \underline{B}}$$

We can now see through the typing rules how CBPV controls when and where effects may occur. In particular, the restriction of typing contexts to value types means the evaluation of a returner and the use of the value it returns must be separated with a bind term. The bind term is the only place where a returner can be evaluated and its returned value unwrapped into a value type variable. In a similar vein, lambda abstractions can only abstract over value types, separating the process of function evaluation from when the argument is evaluated. The call-by-name evaluation $(\lambda x.\, M)N$ becomes $(\lambda x.\, M)(\mathsf{thunk}\ N)$ in CBPV. Whereas the term $(\lambda x.\, M)N$ in call-by-value becomes $N\ \mathsf{to}\ y.\, (\lambda x.\, M)(y)$ in CBPV. This is the key to the control of effects in CBPV.

2.1.2 Operational Semantics

We now present a big-step operational semantics for CBPV, with an evaluation relation, $M \Downarrow T$, from closed computation terms, $\vdash^{\mathsf{c}} M : \underline{B}$, (those with empty typing contexts) to terminal terms $\vdash^{\mathsf{c}} T : \underline{B}$. Terminal terms or terminals are just those closed computation terms of the form $\mathsf{return}\ V$ or $\lambda x.\, M$, which have no further evaluation steps. The operational semantics is defined as follows:

$$\frac{M \Downarrow T}{\mathsf{ifz}(\overline{0}; M; x.\, N) \Downarrow_{\underline{B}} T} \qquad \frac{N[\overline{n}/x] \Downarrow T}{\mathsf{ifz}(\overline{n+1}; M; x.\, N) \Downarrow T} \qquad \frac{M[V_1, V_2/x_1, x_2] \Downarrow T}{\mathsf{split}((V_1, V_2); x_1, x_2.\, M) \Downarrow T}$$

$$\frac{M \Downarrow T}{\mathsf{force}\ (\mathsf{thunk}\ M) \Downarrow T} \qquad \frac{}{\lambda x : A.\, M \Downarrow \lambda x : A.\, M} \qquad \frac{}{\mathsf{return}\ V \Downarrow \mathsf{return}\ V} \qquad \frac{M \Downarrow N \qquad N[V/x] \Downarrow T}{M(V) \Downarrow T}$$

$$\frac{M \Downarrow \mathsf{return}\ V \qquad N[V/x] \Downarrow T}{M\ \mathsf{to}\ x.\, N \Downarrow T}$$

2.1.3 Adding Effects

There is not much to say about the operational semantics in the absence of any effects. Thus, we now consider CBPV extended with a simple print effect. Fix some finite alphabet $\mathcal{A}$ of symbols. We extend the syntax of CBPV with a new term constructor:

$$\frac{\Gamma \vdash^{c} M : \underline{B} \qquad a \in \Sigma}{\Gamma \vdash^{c} \mathsf{print}(a); M : \underline{B}}$$

Then to evaluate $\Gamma \vdash^{c} \mathsf{print}(a); M : \underline{B}$ we first print the character a and then evaluate M. Then to formulate an operational semantics for CBPV with print, we modify the evaluation relation to be $M \Downarrow s, T$ which evaluates closed computation terms M to an ordered pair consisting of a finite string of characters, $s \in \mathcal{A}^*$, and a terminal T. The evaluation relation is defined by extending the previously seen rules from

$$\frac{M_0 \Downarrow T_0 \qquad \ldots \qquad M_n \Downarrow T_n}{M \Downarrow T}$$

to

$$\frac{M_0 \Downarrow a_0, T_0 \qquad \ldots \qquad M_n \Downarrow a_n, T_n}{M \Downarrow a_0 \ldots a_n, T}$$

and adding the rule

$$\frac{M \Downarrow a, T}{\mathsf{print}(b); M \Downarrow ba, T}$$

Here we see the ease by which computational effects can be added to the CBPV calculus. Only a single new rule is needed to extend the operational semantics with the print effect, the nature of CBPV ensures that there are no other situations we might need to consider when a character is printed. This ease of extension with computation effects makes CBPV an ideal candidate as the basis of study for any language with computational effects. This is why make use of CBPV in our study of FLP languages.

To read a greater number of examples of the semantics of CBPV with other effects, see [13].

2.2 Denotational Semantics and Algebraic Structures for Effects

As mentioned in the introduction, the denotational semantics of a language provides a mathematical meaning or denotation to the syntactic elements of the language [15]. This is useful in providing an interpretation of the language in a context free from implementation details. This enables us to prove properties of the semantics of the language using the mathematical model we define.

In particular, we are interested in the denotational semantics of CBPV, which is a strongly-typed calculus. To do so we build up denotational definitions for types, contexts and terms. We then use these to define a denotational interpretation of the CBPV typing relation and hence the programs of the language.

Firstly each type A will have an associated set $[\![A]\!]$, known as its denotation, which we will define inductively on the definition of the type A. This can be thought of as the semantic domain for closed terms with type A. For example in CBPV with print the observable base types would take definitions:

$$[\![\mathsf{Num}]\!] \stackrel{\text{def}}{=} \mathbb{N}$$
$$[\![A_1 \times A_2]\!] \stackrel{\text{def}}{=} [\![A_1]\!] \times [\![A_2]\!]$$

The denotation of values of type Num is simply the set of natural numbers. We highlight in the second definition that the symbol $\times$ on the left-hand side is a syntactic element of the language, whereas on the right-hand side it is the normal Cartesian product on sets. Thus value terms of type $A_1 \times A_2$ are interpreted recursively as ordered pairs of elements from the sets $[\![A_1]\!]$ and $[\![A_2]\!]$.

To interpret computation types, we must consider how we represent the print effect. We make use of algebraic structures from mathematics. An algebraic structure consists of a non-empty-set X called the carrier, and one or more operations on the elements of the carrier X [16]. The structure defines how operations behave on the elements of the carrier and how they interact with each other. Common properties include commutativity, associativity, distributivity and identity elements.

For example, a monoid is an example of a specific algebraic structure where the carrier set is equipped with an associative binary operation, in addition to an identity element for the operation. Examples of monoids are the carrier set of natural numbers $\mathbb{N}$ and the operation of addition. This is because addition is a binary operation which is associative and has zero as an identity element. The set of strings over an alphabet Σ also admits a monoid structure with the carrier set Σ^* and the binary operation of concatenation. This is because concatenation is associative and the empty string ϵ acts as the identity element.

To model computation terms with printed characters, we make use of the algebraic structure on the alphabet of characters $\mathcal{A}$ which Levy calls an $\mathcal{A}$-set [13].

Definition 2.2.1. An $\mathcal{A}$-set is a pair $(X, *)$ where X is the carrier set equipped with an operation $* : \mathcal{A} \times X \to X$. Then

1. An element of $(X, *)$ is an element of the carrier X.
2. A function from a set Y to (X, *) is a function from Y to X.

It is straightforward to extend any definition of $*$ to act as $* : \mathcal{A}^* \times X \to X$ by repeated application of $*$.

A few constructions of $\mathcal{A}$-sets are:

1. The free $\mathcal{A}$-set on a set X is constructed with the carrier set being $\mathcal{A}^* \times X$ where the operation acts such that $a * (b, x) = (a * b, x)$.
2. Given any set X and an $\mathcal{A}$-set $(Y, *)$ we can define a new $\mathcal{A}$-set $X \to (Y, *)$ by defining the carrier set to be $X \to Y$ and the operation by $(a * f)(x) = a * f(x)$. Where $a * f(x)$ is defined by the $\mathcal{A}$-set $(Y, *)$.

The reader may now see how this algebraic structure can be used to model the print effect. Each computation type $\underline{B}$ will have an associated $\mathcal{A}$-set $[\![\underline{B}]\!]$, which we will define recursively on the definition of the type $\underline{B}$.

$$[\![FA]\!] \stackrel{\text{def}}{=} \text{the free } \mathcal{A}\text{-set on} [\![A]\!]$$
$$[\![A \to \underline{B}]\!] \stackrel{\text{def}}{=} \text{the } \mathcal{A}\text{-set of } [\![A]\!] \to [\![\underline{B}]\!]$$

Then the denotation of the computation thunk type $[\![U\underline{B}]\!]$ can be defined as the carrier set of the $\mathcal{A}$-set $[\![\underline{B}]\!]$. This is because the thunk type value needs no algebraic structure as its elements are values which cannot be executed without being coerced back into a computation by a force term.

Now each typing context assigns free variables to value types, hence it is natural that its denotation should be the Cartesian product of the denotations of the value types:

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \prod_{(x:A) \in \Gamma} [\![A]\!]$$

Thus elements $\rho \in [\![\Gamma]\!]$ are mappings from free variables to semantic values of the appropriate type, we call these elements *environments.*

Finally, we can define the denotation of the typing relation, which will be a set of functions from the denotation of contexts, $[\![\Gamma]\!]$, to the denotation of types. In this sense, the denotation of types acts as the domain for the denotation of terms. In particular for an environment $\rho \in [\![\Gamma]\!]$:

$$\left[\!\!\left[\Gamma \vdash^{\mathrm{v}} V : A\right]\!\!\right]\rho \in [\![A]\!]$$
$$\left[\!\!\left[\Gamma \vdash^{\mathrm{c}} M : \underline{B}\right]\!\!\right]\rho \in [\![\underline{B}]\!]$$

We recall that elements of the $\mathcal{A}$-set $[\![\underline{B}]\!]$ are just elements of the carrier set. Then we abuse notation slightly by writing $[\![M]\!]$ for $\left[\!\!\left[\Gamma \vdash^{\mathrm{c}} M : \underline{B}\right]\!\!\right]$ and similarly for value terms $[\![V]\!]$. We define the denotation of terms as follows:

$$\begin{aligned}
[\![x]\!]\rho &\overset{\mathrm{def}}{=} \rho(x) \\
[\![\overline{n}]\!]\rho &\overset{\mathrm{def}}{=} n \\
[\![(V_1, V_2)]\!]\rho &\overset{\mathrm{def}}{=} ([\![V_1]\!]\rho, [\![V_2]\!]\rho) \\
[\![\mathsf{thunk}\ M]\!]\rho &\overset{\mathrm{def}}{=} [\![M]\!]\rho \\
[\![\mathsf{return}\ V]\!]\rho &\overset{\mathrm{def}}{=} (\epsilon, [\![V]\!]\rho) \\
[\![\mathsf{force}\ V]\!]\rho &\overset{\mathrm{def}}{=} [\![V]\!]\rho \\
[\![\lambda x : A.\, M]\!]\rho &\overset{\mathrm{def}}{=} f \text{ where } f(v) \overset{\mathrm{def}}{=} [\![M]\!]\rho[x \mapsto v] \\
[\![M(V)]\!]\rho &\overset{\mathrm{def}}{=} [\![M]\!]\rho([\![V]\!]\rho) \\
[\![\mathsf{split}(V; x_1, x_2.\, M)]\!] &\overset{\mathrm{def}}{=} [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\mathrm{def}}{=} [\![V]\!]\rho \\
[\![\mathsf{ifz}(n; M; x.\, N)]\!] &\overset{\mathrm{def}}{=} \begin{cases} [\![M]\!]\rho & \text{if } [\![n]\!]\rho = 0 \\ [\![N]\!]\rho[x \mapsto n] & \text{if } [\![n]\!]\rho = n + 1 \end{cases} \\
[\![M\ \mathsf{to}\ x.\, N]\!]\rho &\overset{\mathrm{def}}{=} m * [\![N]\!]\rho[x \mapsto a] \text{ where } [\![M]\!]\rho = (m, a) \\
[\![\mathsf{print}(a); M]\!]\rho &= a * [\![M]\!]\rho
\end{aligned}$$

Here again, we see how it is straightforward to define the denotational semantics of CBPV with an effect. Once you have determined a suitable algebraic structure for the effect, there are two main steps for determining the denotation of typed terms. How the effectful terms introduce their effects into the algebraic structure, and how the end result of a computation including its effects is extracted from the algebraic structure in the bind operation.

The elegant structure of CBPV will enable us to focus our attention on how the effects of FLP languages interact in the denotational semantics, without the distraction of catching all the cases where they may occur in a language based on call-by-name or call-by-value evaluation.

Assuming the reader is comfortable with the simple example we have just seen, we can now move on to the more complex case of FLP languages.

Chapter 3

The Language

In this chapter, we define the core Functional Logic Programming language which we consider throughout the rest of this thesis. The language proposed is the work of Kavvos and uses the Call-By-Push-Value (CBPV) calculus as a basis. As demonstrated in the previous chapter, the type system of CBPV provides a separation between values and computations, making explicit when a computation may produce effects. Hence modelling the non-deterministic features of FLP as effects on top of CBPV makes for a pleasant semantic interpretation.

3.0.1 Syntax

The syntax of the FLP language is defined as follows:

value types	A	$::=$	Num	numbers
			$A_1 \times A_2$	value (positive) pairs
			$U\underline{B}$	thunks
computation types	$\underline{B}$	$::=$	FA	returners
			$A \to \underline{B}$	functions
contexts	Γ	$::=$		empty context
			$\Gamma, x : A$	context extension
values	V, W	$::=$	x	variables
			$\overline{n}$	number
			(V_1, V_2)	value pair
			$\mathsf{thunk}\ M$	thunk
computations	M, N	$::=$	$\mathsf{return}\ V$	produce
			$M_1\ \mathsf{to}\ x.\, M_2$	bind (sequencing)
			$\lambda x : A.\, M$	abstraction
			$M(V)$	application
			$\mathsf{ifz}(V; M; x.\, N)$	zero test
			$\mathsf{split}(V; x_1, x_2.\, M)$	split pair
			$\mathsf{force}\ V$	force thunk
			fail	failure
			$M_1 \mathbin{[]} M_2$	binary choice
			$\exists x : A.\, M$	existential
			$V \doteq_A W;\ M$	unification

We limit our core FLP language to numbers and pairs as base value types with minimal operations, we do so to focus our attention on the essential features of FLP.

In particular, we have a binary choice operator $M_1 \mathbin{[]} M_2$, which represents a non-deterministic choice the two computations M_1 and M_2. Operationally such a non-deterministic computation can be seen as maybe executing M_1 or M_2

We also have a unification operator $V \doteq_A W;\ M$, which operationally will attempt to unify two values V and W, if this succeeds we then continue with the computation M. We only consider unification over first-order value terms (values free from thunks), this is because the unification of computation terms in general is undecidable [17]. Notably that in a call-by-value interpretation of the language, unification is possible over the result of computations which return values since when translated to CBPV this is:

$$M \text{ to } x.\, N \text{ to } y.\, x =_A y;\, P$$

Additionally, we have a failure computation, fail. This represents a failure of unification and acts as the unit for the choice operator, which will be evident in the equation theory. Operationally the failure computation will execute when unification fails. For example when executing $\vdash^c \overline{1} =_{\mathsf{Num}} \overline{2};\, M : \underline{B}$, the unification of 1 with 2 will not succeed and thus fail will execute instead of M.

Finally, the language admits an existential operator $\exists x : A.\, M$, which introduces a new bound variable x of first-order value type. This bound variable is seen to non-deterministically take all possible values of its type, the idea being it can be used in conjunction with the unification operator in order to control the values taken. We limit the variable bound by the existential operator to first-order value types, so that the number of terms that the bound variable can take are countable, and we can assign them to the bound variable in accordance with some enumeration. If we allow the existential operator to bind variables of computation type, then the number of terms that the bound variable can take could be uncountable, in particular, it is a straightforward exercise to show that the set of functions from $\mathbb{N}$ to $\mathbb{N}$ is uncountable.

In this way, we have idealised the language by allowing programs to evaluate to a countably infinite number of values. Consider the program below which can evaluate to any natural number:

$$\vdash^c \exists x : \mathsf{Num}.\, (\mathsf{return}\ x) : F(\mathsf{Num})$$

While infinitary non-determinism is likely to prove problematic in practice, the language should provide a useful comparison to more practical languages and guide their design.

The limits we impose on unification and existential quantification to first-order value types enable a straightforward formulation of big-step operational semantics. In particular, for first-order value types successful unification occurs just when the two values are syntactically identical and for existential quantification, we can enumerate the bound variable over a countable set. We leave the treatment of more general unification and existential quantification to future work.

Clearly, the language could also be extended to include more value types and operations, but we omit these to aid with a concise presentation of our results. We also omit some specific features within Epic Games' *Verse*, such as first and total extraction, we leave these and other language extensions to future work.

We make precise the notion of first-order values we described above.

Definition 3.0.1. The *first-order value types* are

$$F \in \mathsf{FO} ::= \mathsf{Num} \mid F_1 \times F_2$$

These are the values that are free from thunks. We can unify over them as mentioned previously.

Finally, the typing rules for our core FLP language are those of effect-free CBPV presented in §2.1 extended with the following rules for FLP effects.

$$\textsc{Fail}\ \frac{}{\Gamma \vdash^c \mathsf{fail} : \underline{B}} \qquad \textsc{Choice}\ \frac{\Gamma \vdash^c M_1 : \underline{B} \qquad \Gamma \vdash^c M_2 : \underline{B}}{\Gamma \vdash^c M_1 \,[]\, M_2 : \underline{B}} \qquad \textsc{Exists}\ \frac{A \in \mathsf{FO} \qquad \Gamma, x : A \vdash^c M : \underline{B}}{\Gamma \vdash^c \exists x : A.\, M : \underline{B}}$$

$$\textsc{Unify}\ \frac{\Gamma \vdash^v V_1 : A \qquad \Gamma \vdash^v V_2 : A \qquad A \in \mathsf{FO} \qquad \Gamma \vdash^c M : \underline{B}}{\Gamma \vdash^c V_1 =_A V_2;\, M : \underline{B}}$$

To ensure the reader has an operational understanding of the language, we provide small examples for the effect constructors above:

1. The program $\vdash^c \mathsf{return}\ \overline{5} \,[]\, \mathsf{return}\ \overline{8} : F(\mathsf{Num})$ can be interpreted as the computation which will non-deterministically return either 5 or 8. Resulting in the set $\{5, 8\}$ or list $[5, 8]$.

2. The program $\vdash^c \overline{4} =_{\mathsf{Num}} \overline{4};\, \mathsf{return}\ \overline{1} : F(\mathsf{Num})$ will return the number 1 since the unification of 4 with 4 is successful. Resulting in the set $\{1\}$ or list $[1]$.

3. The program $\vdash^c \exists x : \mathsf{Num}.\, x =_{\mathsf{Num}} \overline{3};\, \mathsf{return}\ (\overline{1}, \overline{1}) : F(\mathsf{Num} \times \mathsf{Num})$ will non-deterministically fail when the existential variable is not equal to 3, however when it does equal 3 the computation returns the tuple $(1, 1)$. Resulting in the set $\{(1, 1)\}$ or list $[(1, 1)]$.

Chapter 4

Idealised Semantics

To simultaneously handle the different interpretations of non-deterministic choice such as returning ordered (lists) or unordered (sets) results, we use a monad to represent the results of computations. This enables the semantics to be agnostic with respect to the desired implementation of non-determinism.

Throughout this section definitions are based on the work of Kavvos and others where stated. This thesis additionally provides proof for the concrete examples of powersets and lists as monads and as the algebraic structures defined.

4.1 Semantic Structures

As mentioned we parameterise non-determinism through a monad, which we now define:

4.1.1 Monads

Definition 4.1.1. A *monad* T (on sets) consists of the following objects:

- for each set A a set $T(A)$
- for each set A a function $\eta_A : A \to T(A)$
- for every two sets A and B, a function

$$(\gg\!=) : T(A) \to (A \to T(B)) \to T(B)$$

These objects must satisfy the following equations:

$$\begin{aligned}\eta(x) \gg\!= f &= f(x)\\ xm \gg\!= \eta &= xm\\ (xm \gg\!= f) \gg\!= g &= xm \gg\!= (x \mapsto f(x) \gg\!= g)\end{aligned}$$

In particular, we call the last equation the *bind associativity* law.

We consider the two realisations of the monad we are interested in, lists and powersets, as examples.

Example 4.1.2 (Powerset)**.** The powerset construction maps each set A to its set $\mathcal{P}A$ of subsets. We define

$$\eta_A(x) \overset{\text{def}}{=} \{x\} \qquad\qquad S \gg\!= f \overset{\text{def}}{=} \bigcup_{x \in S} f(x)$$

It is a monad.

Proof. Let f, g be functions from A to $\mathcal{P}A$, and let some $x \in A$ for an arbitrary set A. The first equation follows by:

$$\eta_A(x) \gg\!= f = \{x\} \gg\!= f$$

$$\begin{aligned}&= \bigcup_{y \in \{x\}} f(y)\\ &= f(x)\end{aligned}$$

Now let some $xm \in \mathcal{P}A$ then the second equation holds by seeing that

$$\begin{aligned}xm \gg\!= \eta_A &= xm \gg\!= x \mapsto \{x\}\\ &= \bigcup_{x \in xm} \{x\}\\ &= xm\end{aligned}$$

Where taking the union over singleton sets containing each element of the original set is simply equal to the original set.

Finally, the third equation follows by:

$$\begin{aligned}(xm \gg\!= f) \gg\!= g &= \left(\bigcup_{x \in xm} f(x)\right) \gg\!= g\\ &= \left(\bigcup_{x \in xm} f(x)\right) \gg\!= g\\ &= \bigcup_{y \in \left(\bigcup_{x \in xm} f(x)\right)} g(y)\\ &= \bigcup_{x \in xm} \bigcup_{y \in f(x)} g(y)\\ &= \bigcup_{x \in xm} f(x) \gg\!= g\\ &= xm \gg\!= (x \mapsto f(x) \gg\!= g)\end{aligned}$$

Hence, powerset is a monad. ◁

◁

Example 4.1.3 (Finite or infinite lists)**.** For each set A we define A^∞ to consist of the set of *finite or infinite lists* of elements of A. We define

$$\eta_A(x) \stackrel{\text{def}}{=} [x] \qquad\qquad xs \gg\!= f \stackrel{\text{def}}{=} \Big[y \mid x \leftarrow xs, y \leftarrow f(x)\Big]$$

The operation η returns the singleton list. The definition of $(\gg\!=)$ uses a Haskell-like notation: first iterate x over xs, in order, and compute $f(x)$. We then iterate over each $f(x)$, in order again, returning all the values in that list in order.

This is also a monad

Proof. Let A be an arbitrary set, and let f, g be functions from A to A^∞. Let $x \in A$ and $xm \in A^\infty$. Then we see the first equation holds by:

$$\begin{aligned}\eta(x) \gg\!= f &= [x] \gg\!= f\\ &= \Big[z \mid y \leftarrow [x], z \leftarrow f(y)\Big]\\ &= f(x)\end{aligned}$$

Next, the second equation holds by:

$$\begin{aligned}xm \gg\!= \eta &= xm \gg\!= x \mapsto [x]\\ &= \Big[z \mid y \leftarrow xm, z \leftarrow [y]\Big]\\ &= \Big[y \mid y \leftarrow xm\Big]\\ &= xm\end{aligned}$$

Finally, the third equation holds by:

$$\begin{aligned}(xm \ggg f) \ggg g &= \Big[z \mid y \leftarrow xm, z \leftarrow f(y)\Big] \ggg g \\ &= \Big[w \mid v \leftarrow \Big[z \mid y \leftarrow xm, z \leftarrow f(y)\Big], w \leftarrow g(v)\Big] \\ &= \Big[w \mid y \leftarrow xm, v \leftarrow f(y), w \leftarrow g(v)\Big] \\ &= \Big[w \mid y \leftarrow xm, w \leftarrow \Big[z \mid v \leftarrow f(y), z \leftarrow g(v)\Big]\Big] \\ &= \Big[w \mid y \leftarrow xm, w \leftarrow f(y) \ggg g\Big] \\ &= xm \ggg (x \mapsto f(x) \ggg g)\end{aligned}$$

Hence, lists are a monad. $\lhd$

$\lhd$

4.1.2 Structures for Handling Effects

We now define the mathematical structures we use to model computational effects within the language. There is a rich literature on the algebraic treatment of computational effects, and in this thesis we will follow the style of parameterised algebra described in [18].

The parameters for our algebras will be given by a universe of values:

Definition 4.1.4. A *universe of values* $\mathfrak{V}$ is a family of sets $(\mathfrak{V}_A)_{A \in \mathsf{FO}}$, i.e. a choice of a set $\mathfrak{V}_A$ for each first-order value type $A \in \mathsf{FO}$.

Then we will interpret FLP effects by an algebra parameterised with a universe of values. These algebras will be equipped with operations that closely resemble the choice, existential and unification operations of the language, we will call them *FLP algebras*:

Definition 4.1.5. An *FLP algebra* $(X, \varepsilon_X, \vee_X, \exists_X, \mathsf{eq}_X)$ over a universe $\mathfrak{V}$ consists of

- a *carrier set* X
- a constant $\varepsilon_X \in X$
- an operation $\vee_X : X \times X \to X$
- for each value type A an operation

$$\exists_X^A : (\mathfrak{V}_A \to X) \to X$$

- for each first-order value type $A \in \mathsf{FO}$ an operation

$$\mathsf{eq}_X^A : \mathfrak{V}_A \times \mathfrak{V}_A \times X \to X$$

satisfying the equations (omitting types everywhere)

$$x \vee (y \vee z) = (x \vee y) \vee z \tag{4.1}$$
$$x \vee \varepsilon = x \tag{4.2}$$
$$\varepsilon \vee x = x \tag{4.3}$$
$$\mathsf{eq}(v, v, x) = x \tag{4.4}$$
$$\mathsf{eq}(v, w, x_v) = \mathsf{eq}(v, w, x_w) \quad \text{for a family } (x_v)_{v \in \mathfrak{V}_A} \tag{4.5}$$
$$\mathsf{eq}(v, w, x) = \mathsf{eq}(w, v, x) \tag{4.6}$$
$$\mathsf{eq}(v_1, w_1, \mathsf{eq}(v_2, v_2, x)) = \mathsf{eq}(v_2, w_2, \mathsf{eq}(v_1, v_2, x)) \tag{4.7}$$
$$\mathsf{eq}(v, w, \varepsilon) = \varepsilon \tag{4.8}$$
$$\mathsf{eq}(v, w, x \vee y) = \mathsf{eq}(v, w, x) \vee \mathsf{eq}(v, w, y) \tag{4.9}$$

Remark 4.1.6. While not previously noted, we work within classical set theory (ZFC) for the entirety of this thesis. It is useful to note from [18] that within classical set theory the equations Eqs. (4.4), (4.5) and (4.8) prove that eq must always be defined by

$$\mathsf{eq}_X^A(v, w, x) = \begin{cases} x & \text{if } v = w \\ \varepsilon_X & \text{if } v \neq w \end{cases}$$

◁

Proof. See Staton [18, §2.6]. □

The reader should notice how the domain and codomain of the FLP Algebra operations mirror the operation of the effects we consider. Where $\vee$ will model non-deterministic choice of computations, $\exists$ models the existential operator and eq models unification. A universe of values for each first-order value type can be given by the denotation of values which inhabit that type or even just the set of closed terms of that type.

We will need to incorporate FLP algebras into the monad structure which we use in the language semantics. We do this by defining an *FLP structure* for a monad T which will provide an FLP algebra on the carrier set of TX for each set X. Additionally, this structure will provide some expected equations for the interaction of the FLP algebra with the monad operations. Before we can do this we will need to define the notion of a section-retraction pair in order to relate FLP algebras over different universes.

Definition 4.1.7. A section-retraction pair (s, r) is a pair of functions $s : X \to Y$ and $r : Y \to X$ which satisfy:

$$r \circ s = id_X$$

Each function can be seen as a one-sided inverse of the other.

Lemma 4.1.8. *Let (s, r) be a section-retraction pair. Then s is injective and r is surjective.*

Proof. Suppose a section-retraction pair $(s : X \to Y, r : Y \to X)$. Suppose some $x, x' \in X$ such that $s(x) = s(x')$. Then we see

$$\begin{aligned} & s(x) = s(x') \\ \implies & r(s(x)) = r(s(x')) \\ \implies & x = x' \end{aligned}$$

Therefore s is injective.

Suppose some $x \in X$, then we know that $s(x) \in Y$ and $r(s(x)) = x \in X$. Hence r is a surjection. □

We now define FLP structures.

Definition 4.1.9. An *FLP structure* $(\varepsilon, \vee, \exists, \mathsf{eq})$ on a monad T consists of an FLP algebra $(TX, \varepsilon_{TX}, \vee_{TX}, \exists_{TX}, \mathsf{eq}_{TX})$ for each set X over any universe $\mathfrak{V}$. Moreover, it must satisfy the equations for arbitrary sets X, Y and function $f : X \to TY$.

$$\varepsilon_{TX} \gg= f = \varepsilon_{TY} \tag{4.10}$$

$$(xm \vee_{TX} ym) \gg= f = (xm \gg= f) \vee_{TY} (ym \gg= f) \tag{4.11}$$

$$\left(\exists_{TX} h\right) \gg= f = \exists_{TY}(v \mapsto h(v) \gg= f) \tag{4.12}$$

$$\mathsf{eq}_{TX}(v, w, xm) \gg= f = \mathsf{eq}_{TY}(v, w, xm \gg= f) \tag{4.13}$$

Additionally given arbitrary universes $(\mathfrak{V}_A)_{A \in \mathsf{FO}}$, $(\mathfrak{Z}_A)_{A \in \mathsf{FO}}$, and a section-retraction pair $(s : \mathfrak{Z}_A \to \mathfrak{V}_A, r : \mathfrak{V}_A \to \mathfrak{Z}_A)$, the following equations hold:

$$\text{For all } g : \mathfrak{Z}_A \to TX,\ \exists_{TX}^{\mathfrak{Z}_A} g = \exists_{TX}^{\mathfrak{V}_A}(g \circ r) \tag{4.14}$$

$$\mathsf{eq}^{\mathfrak{V}_A}(v, w, xm) = \mathsf{eq}^{\mathfrak{Z}_A}(s(v), s(w), xm) \tag{4.15}$$

We notice that Eqs. (4.10) to (4.13) define how to bind to an element of an FLP algebra on TX in the expected way. This means can extend any function on the carrier set TX to all elements of the FLP algebra on that set TX given by the FLP structure. Then we note that Eqs. (4.14) and (4.15) show that if you have a section-retraction relation between two universes, then you can translate between the

FLP algebras over them. In particular, Eq. (4.14) makes intuitive sense as the retraction is surjective, so by pre-composing it we know that we will hit every element of the codomain of r. Then Eq. (4.15) follows intuitively from the fact that a section is injective, so we know that if two elements are equal in the codomain of s, then they must have been equal in the domain.

We now show that lists and powersets have FLP structures.

Example 4.1.10 (Powerset)**.** Continuing Example 4.1.2, the powerset monad $\mathcal{P}(-)$ has an FLP structure given by

$$\begin{aligned}
\varepsilon &\stackrel{\text{def}}{=} \emptyset \\
xm \vee ym &\stackrel{\text{def}}{=} xm \cup ym \\
\exists f &\stackrel{\text{def}}{=} \bigcup_{i \in \mho_A} f(i) \\
\mathsf{eq}^A_X(v, w, xm) &\stackrel{\text{def}}{=} \begin{cases} xm & \text{if } v = w \\ \emptyset & \text{if } v \neq w \end{cases}
\end{aligned}$$

We do not need any subscripts, since the definitions are the same for each set.

Proof. We first show that we have an FLP algebra for each set, by showing the above definition satisfies Definition 4.1.5. Suppose some set X, so that $TX = \mathcal{P}X$ is the carrier set. Then immediately we know $\emptyset \in \mathcal{P}X$, and that the operations have the correct types.

Then Eq. (4.1) is a direct consequence of the associativity of unions of sets. Also, Eqs. (4.2) and (4.3) hold trivially by unions with the empty set, suppose $xm \in \mathcal{P}X$ then

$$xm \cup \emptyset = xm = \emptyset \cup xm$$

Finally, by Remark 4.1.6 we see that $\mathsf{eq}^A_X(v, w, xm)$ takes the only possible correct definition. Hence, we have an FLP algebra on $\mathcal{P}X$ for all arbitrary sets X.

Now we show the FLP Structure properties hold. Fix some function $f : X \to \mathcal{P}Y$. Then Eq. (4.10) follows from the definitions of ε and $\gg\!=$.

$$\emptyset \gg\!= f = \bigcup_{x \in \emptyset} f(x) = \emptyset$$

Suppose some $xm, ym \in \mathcal{P}X$, then Eq. (4.11) can be seen by:

$$\begin{aligned}
(xm \vee ym) \gg\!= f &= \bigcup_{x \in (xm \cup ym)} f(x) \\
&= \left(\bigcup_{x \in xm} f(x) \right) \cup \left(\bigcup_{y \in ym} f(y) \right) \\
&= (xm \gg\!= f) \cup (ym \gg\!= f) \\
&= (xm \gg\!= f) \vee (ym \gg\!= f)
\end{aligned}$$

Next, suppose some $h \in (\mho_A \to \mathcal{P}X)$ for a first-order value type A, then Eq. (4.12) follows by:

$$\begin{aligned}
\exists h \gg\!= f &= \bigcup_{x \in \exists h} f(x) \\
&= \bigcup_{\left(x \in \bigcup_{i \in \mho_A} h(i)\right)} f(x) \\
&= \bigcup_{i \in \mho_A} \bigcup_{x \in h(i)} f(x) \\
&= \bigcup_{i \in \mho_A} (h(i) \gg\!= f) \\
&= \exists (i \mapsto h(i) \gg\!= f)
\end{aligned}$$

Now suppose some $v, w \in \mathfrak{V}_A$ and $xm \in \mathcal{P}X$, then Eq. (4.13) follows by:

$$\begin{aligned}
\mathsf{eq}_{TX}(v, w, xm) \gg\!= f &= \begin{cases} xm \gg\!= f & \text{if } v = w \\ \emptyset \gg\!= f & \text{if } v \neq w \end{cases} \\
&= \begin{cases} xm \gg\!= f & \text{if } v = w \\ \emptyset & \text{if } v \neq w \end{cases} \qquad \text{by Eq. (4.10)} \\
&= \mathsf{eq}_{TY}(v, w, xm \gg\!= f)
\end{aligned}$$

Next, suppose two universes $(\mathfrak{V}_A)_{A \in \mathsf{FO}}$, $(\mathfrak{Z}_A)_{A \in \mathsf{FO}}$, a section-retraction pair $(s : \mathfrak{Z}_A \to \mathfrak{V}_A, r : \mathfrak{V}_A \to \mathfrak{Z}_A)$ and a function $g : \mathfrak{Z}_A \to TX$, then the Eq. (4.14) follows by:

$$\begin{aligned}
\mathbb{H}^{\mathfrak{Z}_A}_{TX} g &= \bigcup_{x \in \mathfrak{Z}_A} g(x) \\
&= \bigcup_{x \in \mathfrak{V}_A} g(r(x)) \\
&= \mathbb{H}^{\mathfrak{V}_A}_{TX}(g \circ r)
\end{aligned}$$

The second line follows from the fact that r is a retraction and thus is surjective. This means r takes every value of $\mathfrak{Z}_A$ when it is applied to every value in $\mathfrak{V}_A$. Hence, we can take the union over $\mathfrak{V}_A$ when we pre-compose r.

Then in the same setting as above, we can see Eq. (4.15) follows by:

$$\begin{aligned}
\mathsf{eq}^{\mathfrak{V}_A}(v, w, xm) &= \begin{cases} xm & \text{if } v = w \\ \emptyset & \text{if } v \neq w \end{cases} \\
&= \begin{cases} xm & \text{if } s(v) = s(w) \\ \emptyset & \text{if } s(v) \neq s(w) \end{cases} \qquad \text{by injectivity of } s \\
&= \mathsf{eq}^{\mathfrak{Z}_A}(s(v), s(w), xm)
\end{aligned}$$

Where the penultimate line follows from the injectivity of s, which means that if $s(v) = s(w)$ then $v = w$.

Hence, the powerset monad $\mathcal{P}(-)$ has an FLP structure. ◁

◁

Example 4.1.11 (Lists)**.** Continuing Example 4.1.3, the list monad $(-)^\infty$ has an FLP structure, assuming we limit ourselves to countable universes which provide some fixed enumeration.

Suppose a universe $(\mathfrak{V}_A)_{A \in \mathsf{FO}}$ and for each first-order value type A we have a fixed enumeration

$$e_{\mathfrak{V}_A} : \mathbb{N} \xrightarrow{\cong} \mathfrak{V}_A$$

Then we may define

$$\begin{aligned}
\varepsilon &\overset{\text{def}}{=} [\,] \\
xm \vee ym &\overset{\text{def}}{=} xm +\!\!+\, ym \\
\mathbb{H} f &\overset{\text{def}}{=} \left[x \;\middle|\; i \leftarrow [0..], v \leftarrow e_A(i), x \leftarrow f(v) \right] \\
\mathsf{eq}(v, w, xm) &\overset{\text{def}}{=} \begin{cases} xm & \text{if } v = w \\ [\,] & \text{if } v \neq w \end{cases}
\end{aligned}$$

Again we do not need any subscripts, since the definitions are the same for each set.

Proof. We first show that the above admits an FLP algebra for any set. Suppose some set X, and consider the carrier set $TX = X^\infty$. Then immediately we know $[\,] \in X^\infty$, and that the operations have the correct types.

Then Eq. (4.1) is a direct consequence of the associativity of the concatenation of lists. Also, Eqs. (4.2) and (4.3) hold trivially by concatenation with the empty list, suppose $xm \in X^\infty$ then

$$xm +\!\!+ [\,] = xm = [\,] +\!\!+ xm$$

Finally, by Remark 4.1.6 we see that $\mathsf{eq}^A_X(v, w, xm)$ takes the only possible correct definition. Hence, we have an FLP algebra for X^∞ for all arbitrary sets X.

Now we show the FLP structure properties hold. Fix some function $f : X \to Y^\infty$. Then the Eq. (4.10) follows from the definitions of ε and $\ggg=$.

$$[\,] \ggg= f = \Big[y \mid x \leftarrow [\,], y \leftarrow f(x)\Big] = [\,]$$

Suppose some $xm, ym \in X^\infty$, then Eq. (4.11) can be seen by:

$$\begin{aligned}(xm \vee ym) \ggg= f &= \Big[y \mid x \leftarrow (xm +\!\!+\, ym), y \leftarrow f(x)\Big] \\ &= \Big[y \mid x \leftarrow xm, y \leftarrow f(x)\Big] +\!\!+ \Big[y \mid x \leftarrow ym, y \leftarrow f(x)\Big] \\ &= (xm \ggg= f) +\!\!+ (ym \ggg= f)\end{aligned}$$

Where the second line follows from the fact that we iterate $x \leftarrow (xm +\!\!+\, ym)$ by taking x in order from xm first, then ym.

Next, suppose some $h \in (\mathfrak{V}_A \to X^\infty)$ for a value type A, then Eq. (4.12) follows by:

$$\begin{aligned}\boldsymbol{\exists} h \ggg= f &= \Big[y \mid x \leftarrow \boldsymbol{\exists} h, y \leftarrow f(x)\Big] \\ &= \Big[y \mid x \leftarrow \Big[x \,\Big|\, i \leftarrow [0..], v \leftarrow e_A(i), x \leftarrow h(v)\Big], y \leftarrow f(x)\Big] \\ &= \Big[y \mid i \leftarrow [0..], v \leftarrow e_A(i), x \leftarrow h(v), y \leftarrow f(x)\Big] \\ &= \Big[y \mid i \leftarrow [0..], v \leftarrow e_A(i), y \leftarrow \Big[y \mid x \leftarrow h(v), y \leftarrow f(x)\Big]\Big] \\ &= \Big[y \mid i \leftarrow [0..], v \leftarrow e_A(i), y \leftarrow h(v) \ggg= f\Big] \\ &= \boldsymbol{\exists}(v \mapsto h(v) \ggg= f)\end{aligned}$$

Now suppose some $v, w \in \mathfrak{V}_A$ and $xm \in X^\infty$, then Eq. (4.13) follows by:

$$\begin{aligned}\mathsf{eq}_{TX}(v, w, xm) \ggg= f &= \begin{cases} xm \ggg= f & \text{if } v = w \\ [\,] \ggg= f & \text{if } v \neq w \end{cases} \\ &= \begin{cases} xm \ggg= f & \text{if } v = w \\ [\,] & \text{if } v \neq w \end{cases} \qquad \text{by Eq. (4.10)} \\ &= \mathsf{eq}_{TY}(v, w, xm \ggg= f)\end{aligned}$$

Next, suppose two universes $(\mathfrak{V}_A)_{A \in \mathsf{FO}}, (\mathfrak{Z}_A)_{A \in \mathsf{FO}}$ with enumerations $e_{\mathfrak{V}_A}, e_{\mathfrak{V}_A}$ respectively, a function $g : \mathfrak{Z}_A \to TX$ and a section-retraction pair $(s : \mathfrak{Z}_A \to \mathfrak{V}_A, r : \mathfrak{V}_A \to \mathfrak{Z}_A)$ such that $e_{\mathfrak{V}_A} = s \circ e_{\mathfrak{Z}_A}$, then Eq. (4.14) follows by:

$$\begin{aligned}\boldsymbol{\exists}^{\mathfrak{Z}_A}_{TX} g &= \Big[x \,\Big|\, i \leftarrow [0..], v \leftarrow e_{\mathfrak{Z}_A}(i), x \leftarrow g(v)\Big] \\ &= \Big[x \,\Big|\, i \leftarrow [0..], v \leftarrow e_{\mathfrak{Z}_A}(i), x \leftarrow g(r(s(v)))\Big] \\ &= \Big[x \,\Big|\, i \leftarrow [0..], v \leftarrow s(e_{\mathfrak{Z}_A}(i)), x \leftarrow g(r(v))\Big] \\ &= \Big[x \,\Big|\, i \leftarrow [0..], v \leftarrow e_{\mathfrak{V}_A}(i), x \leftarrow g(r(v))\Big] \\ &= \boldsymbol{\exists}^{\mathfrak{V}_A}_{TX}(g \circ r)\end{aligned}$$

We can move the application of s from v in the second line to $e_{\mathfrak{Z}_A}(i)$ in the third since $e_{\mathfrak{Z}_A}(i)$ produces exactly one element. In the penultimate line, we use the property $e_{\mathfrak{V}_A} = s \circ e_{\mathfrak{Z}_A}$, meaning the enumerations line up. We note that supposing this property is not particularly restrictive since we always have the bijection $(e_{\mathfrak{V}_A} \circ e^{-1}_{\mathfrak{Z}_A})$ which can act as a section with this property. In particular the two out of three property [19] tells us that any section s which satisfies $e_{\mathfrak{V}_A} = s \circ e_{\mathfrak{Z}_A}$ must be a bijection because the two enumerations themselves are bijections. Hence for the list monad, any suitable section-retraction pair will constitute a bijection between the countable universes.

Finally, in the same setting as above we can show Eq. (4.12) by:

$$\begin{aligned}
\mathsf{eq}^{\mathfrak{V}_A}(v, w, xm) &= \begin{cases} xm & \text{if } v = w \\ [\,] & \text{if } v \neq w \end{cases} \\
&= \begin{cases} xm & \text{if } s(v) = s(w) \\ [\,] & \text{if } s(v) \neq s(w) \end{cases} && \text{by injectivity of } s \\
&= \mathsf{eq}^{\mathfrak{Z}_A}(s(v), s(w), xm)
\end{aligned}$$

This follows by the same argument used in the powerset example. Hence, the list monad $(-)^{\infty}$ has an FLP structure. $\triangleleft$

$\triangleleft$

4.2 Operational Semantics

We now present an idealised operational semantics for the FLP language, as previously mentioned in Chapter 3, the operational semantics will be idealised in that we allow programs to evaluate to a countably infinite amount of terminal terms. In particular, the existential operator ranges over the countably many values for the variable it binds. Hence, we do not propose this operation semantics as a suitable model for direct implementation, but rather as a useful tool for comparing with more pragmatic models.

4.2.1 Effectful Terminals

We recall from our overview of Call-By-Push-Value in §2.1 that in defining an operational semantics we must define the terminal terms, those closed computation terms we consider to be final. Firstly we have the returner terminals:

$$\mathcal{T}_{FA} = \left\{ \mathsf{return}\ V \,\middle|\, \vdash^{\mathsf{v}} V : A \right\}$$

And the function terminals:

$$\mathcal{T}_{A \to \underline{B}} = \left\{ \lambda x : A.\, M \,\middle|\, \vdash^{\mathsf{c}} \lambda x : A.\, M : A \to \underline{B} \right\}$$

Hence the set of closed terminal terms or terminals is given by:

$$\mathcal{T}_{\underline{B}} = \left\{ T \in \mathcal{T}_{FA} \cup \mathcal{T}_{A \to \underline{B}} \,\middle|\, \vdash^{\mathsf{c}} T : \underline{B} \right\}$$

We define the separate sets for function and returner terminals in order to provide a convenient definition of the operational semantics.

Again we recall from extending CBPV with a print effect in §2.1 That in order to capture the effects of FLP in our operational semantics we must extend the evaluation relation to an appropriate codomain.

In particular, we make use of the structures defined in the previous section §4.1. In order to model FLP programs non-deterministically returning countably many values, we consider the monadic structure of terminals (henceforth monadic terminals) each term will evaluate to in $T(\mathcal{T}_{\underline{B}})$. In this way, we cover both the list and powerset non-determinism models simultaneously. For example the computation $\vdash^{\mathsf{c}} \exists x : \mathsf{Num}.\,(\mathsf{return}\ x) : F(\mathsf{Num})$ can be seen to evaluate to:

$$\{\mathsf{return}\ \bar{1}, \mathsf{return}\ \bar{2}, \mathsf{return}\ \bar{3}, \ldots\} \in \mathcal{P}\Big(\mathcal{T}_{F(\mathsf{Num})}\Big) \quad \text{or} \quad [\mathsf{return}\ \bar{1}, \mathsf{return}\ \bar{2}, \mathsf{return}\ \bar{3}, \ldots] \in \Big(\mathcal{T}_{F(\mathsf{Num})}\Big)^{\infty}$$

In order to give an operational description of these monadic terminals, we make use of an FLP algebra. In this case, we will use a universe of closed first-order value terms $(\mathcal{V}_A)_{A \in \mathsf{FO}}$ where each set is given by:

$$\mathcal{V}_A \stackrel{\text{def}}{=} \left\{ V \,\middle|\, \vdash^{\mathsf{v}} V : A \right\}$$

Then by fixing a monad T with an FLP structure, we have an FLP Algebra on the monadic set terminals over the universe $(\mathcal{V}_A)_{A \in \mathsf{FO}}$:

$$\Big(T\Big(\mathcal{T}_{\underline{B}}\Big), \varepsilon_{T(\mathcal{T}_{\underline{B}})}, \vee_{T(\mathcal{T}_{\underline{B}})}, \exists^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}, \mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}\Big)$$

At times we may abuse notation for the operations when the meaning is clear, writing ε or $\varepsilon_{\underline{B}}$ for $\varepsilon_{T(\mathcal{T}_{\underline{B}})}$ and $\mathbb{\exists}^{A}_{\underline{B}}$ or $\mathbb{\exists}$ for $\mathbb{\exists}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}$ and so on.

We are now in a position to define the evaluation relation from closed computation terms to elements of the FLP Algebra over the monadic terminals. This evaluation relation will be written as $M \Downarrow_{\underline{B}} xm$, where $\vdash^{c} M : \underline{B}$ and xm is an element of the FLP algebra over $T(\mathcal{T}_{\underline{B}})$.

$$\textsc{D-Ifz-Zero}\quad \frac{M \Downarrow_{\underline{B}} xm}{\mathsf{ifz}(\overline{0}; M; x.\, N) \Downarrow_{\underline{B}} xm} \qquad \textsc{D-Ifz-Pos}\quad \frac{N[\overline{n}/x] \Downarrow_{\underline{B}} xm}{\mathsf{ifz}(\overline{n+1}; M; x.\, N) \Downarrow_{\underline{B}} xm} \qquad \textsc{D-Pair}\quad \frac{M[V_1, V_2/x_1, x_2] \Downarrow_{\underline{B}} xm}{\mathsf{split}((V_1, V_2); x_1, x_2.\, M) \Downarrow_{\underline{B}} xm}$$

$$\textsc{D-Force}\quad \frac{M \Downarrow_{\underline{B}} xm}{\mathsf{force}\ (\mathsf{thunk}\ M) \Downarrow_{\underline{B}} xm} \qquad \textsc{D-Lam}\quad \frac{}{\lambda x : A.\, M \Downarrow_{A \to \underline{B}} \eta(\lambda x : A.\, M)} \qquad \textsc{D-Return}\quad \frac{\vdash^{v} V : A}{\mathsf{return}\ V \Downarrow_{FA} \eta(\mathsf{return}\ V)}$$

These first six rules should require no explanation as they are simple extensions of the effect-free CBPV operational semantics in §2.1.2.

$$\textsc{D-App}\quad \frac{M \Downarrow_{A \to \underline{B}} xm \qquad \vdash^{v} V : A \qquad N[V/x] \Downarrow_{\underline{B}} ym_N}{M(V) \Downarrow_{\underline{B}} xm \gg\!= (\lambda x : A.\, N \mapsto ym_N)}$$

The rule D-App is the first with substantial changes. We highlight the use of a pattern-matching notation in the conclusion. Since we know xm is an element of the FLP Algebra on $T(\mathcal{T}_{FA})$ it suffices to define the function which xm binds to for elements $\lambda x : A.\, N \in \mathcal{T}_{FA}$ only. This is because we can use the FLP structure laws in Definition 4.1.9 to apply the function $\lambda x : A.\, N \mapsto ym_N$ under the operations of the FLP algebra.

Furthermore, the rule premise $N[V/x] \Downarrow_{\underline{B}} ym_N$ is overloaded, we require it to hold for every N which is pattern-matched by the function $\lambda x : A.\, N \mapsto ym_N$. We use the subscript in ym_N to highlight that this monadic terminal depends on the term N in $N[V/x] \Downarrow_{\underline{B}} ym_N$, and in particular the function $\lambda x : A.\, N \mapsto ym_N$ gives the specific monadic terminal for each matched term N. In this way, we use pattern matching to extract N from $\lambda x : A.\, N$ and use it to define the monadic terminal ym_N. In words for each function returned by evaluating M, we evaluate it on the value V and collect the results under the monad.

$$\textsc{D-Bind}\quad \frac{M \Downarrow_{FA} xm \qquad x : A \vdash^{c} N : \underline{B} \qquad N[V/x] \Downarrow_{\underline{B}} ym_V}{M\ \mathsf{to}\ x.\, N \Downarrow_{\underline{B}} xm \gg\!= (\mathsf{return}\ V \mapsto ym_V)}$$

The rule D-Bind is very similar to D-App, except that in this case we know xm is an element of the FLP algebra on $T(\mathcal{T}_{FA})$ and so we pattern match on the terminals of shape $\mathsf{return}\ V$. The function $\mathsf{return}\ V \mapsto ym_V$ extracts the closed value term V and uses the premises $x : A \vdash^{c} N : \underline{B}$ and $N[V/x] \Downarrow_{\underline{B}} ym_V$ to return the monadic terminal ym_V. In this way, the rule implements binding by evaluating M then for each terminal in the monadic structure we substitute it into the term N, these substitutions are all evaluated and collected under the monad.

$$\textsc{D-Fail}\quad \frac{}{\mathsf{fail} \Downarrow_{\underline{B}} \varepsilon_{T(\mathcal{T}_{\underline{B}})}}$$

The rule D-Fail simply evaluates failure computations as failure units of the FLP algebra on $T(\mathcal{T}_{\underline{B}})$.

$$\textsc{D-Choice}\quad \frac{M \Downarrow_{\underline{B}} xm \qquad N \Downarrow_{\underline{B}} ym}{M \mathbin{[\,]} N \Downarrow_{\underline{B}} xm \vee_{T(\mathcal{T}_{\underline{B}})} ym}$$

The rule D-Choice evaluates choice computations by evaluating both branches and collecting the results using the choice operation of the FLP algebra.

$$\text{D-Exists}\quad \frac{M[V/x] \Downarrow_{\underline{B}} xm_V}{\exists x : A.\, M \Downarrow_{\underline{B}} \mathbb{E}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V \mapsto xm_V)}$$

The rule D-Exists implements the evaluation of an existential term by using the existential operator of the FLP algebra. The existential operator collects the results of evaluating M substituted with each value V in the countable set of first-order value terms $\mathcal{V}_A$. In this way the application of this inference rule may involve a countably infinite premises to hold, we ignore any potential issues arising from this.

$$\text{D-Unify}\quad \frac{\Gamma \vdash^{\mathsf{v}} V_1 : A \qquad \Gamma \vdash^{\mathsf{v}} V_2 : A \qquad A \in \mathsf{FO} \qquad M \Downarrow_{\underline{B}} xm}{V_1 =_A V_2;\, M \Downarrow_{\underline{B}} \mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V_1, V_2, xm)}$$

The rule D-Unify implements the unification term within the FLP language. We let eq decide if V_1 and V_2 are equal if so then $\mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V_1, V_2, xm)$ will be the monadic terminal xm, if not then it will be the failure unit for the FLP algebra, $\varepsilon_{T(\mathcal{T}_{\underline{B}})}$. In particular, by Remark 4.1.6 we know that within classical set theory we will have equality of V_1 and V_2 just when they are exactly identical. In a more pragmatic machine-based implementation, eq may need to be implemented using a unification algorithm.

4.3 Denotational Semantics

We now define a denotational semantics for language, which will take a similar shape to the CBPV denotational semantics in [13].

Firstly we suppose a monad (say list or powerset), equipped with an FLP structure. We now introduce the algebra for this monad, adapting definitions in [20] for our use.

4.3.1 Monad Algebras

Definition 4.3.1. We define the *monad multiplication* (or *join*) operation $\mu_X : TTX \to TX$ as

$$\mu_X(xmm) = xmm \gg\!= id_{TX}$$

Definition 4.3.2. Given a function $g : A \to B$, the functorial action of a monad on g is the function $Tg : TA \to TB$ defined by

$$Tg(xm) = xm \gg\!= (\eta \circ g)$$

This is the natural way to apply g to elements of TA and then re-wrap the results in the monad to get an element of TB.

Definition 4.3.3. A *monad algebra* (or *T-algebra*) for a monad T over a carrier set X is a pair (X, α) of the carrier set X and a map $\alpha : TX \to X$, which satisfies the following properties:

$$\forall x \in X.\, \alpha(\eta(x)) = x$$
$$\alpha \circ T\alpha = \alpha \circ \mu_X \in (TTX \to X)$$

The first property of T-algebra states that it unwraps the return monad operation in the expected way. The second property states that using the T-algebra twice to evaluate elements of TTX is equivalent to using the multiplication monad operation μ_X to evaluate elements of TTX to TX first, then using the monad algebra.

Theorem 4.3.4. *Given a T-algebra (X, α) and a function $f : Y \to TX$, then for all $ym \in TY$ the following property holds:*

$$\alpha(ym \gg\!= (y \mapsto \eta(\alpha(f(y))))) = \alpha(ym \gg\!= f)$$

This is the result of applying the equivalent functions $\alpha \circ T\alpha = \alpha \circ \mu_X$ to $(ym \gg\!= \eta \circ f) \in TTX$. This property will prove useful later on when we consider the associativity law for bind in the FLP language.

Proof. Fix some monad T and sets X and Y. Suppose a function $f : Y \to TX$ and some element $ym \in TY$. Then we note that:

$$(ym \gg\!= \eta \circ f) \in TTX$$

Then supposing we have a T-algebra operation α for T over X as in Definition 4.3.3, which satisfies $\alpha \circ T\alpha = \alpha \circ \mu_X$, we see that:

$$(\alpha \circ T\alpha)(ym \gg\!= \eta \circ f) = (\alpha \circ \mu_X)(ym \gg\!= \eta \circ f)$$

Now we expand each side, starting with the left to see that:

$$\begin{aligned}
(\alpha \circ T\alpha)(ym \gg\!= \eta \circ f) &= (xm \mapsto \alpha(xm \gg\!= (\eta \circ \alpha)))(ym \gg\!= \eta \circ f) \\
&= \alpha((ym \gg\!= \eta \circ f) \gg\!= (\eta \circ \alpha)) \\
&= \alpha(ym \gg\!= (y \mapsto \eta(f(y)) \gg\!= \eta \circ \alpha)) && \text{by bind associativity} \\
&= \alpha(ym \gg\!= (y \mapsto \eta(\alpha(f(y))))) && \text{as } \eta(x) \gg\!= f = f(x)
\end{aligned}$$

Now expanding the right-hand side, we see:

$$\begin{aligned}
(\alpha \circ \mu_X)(ym \gg\!= \eta \circ f) &= (xmm \mapsto \alpha(xmm \gg\!= id_{TX}))(ym \gg\!= \eta \circ f) \\
&= \alpha((ym \gg\!= \eta \circ f) \gg\!= id_{TX}) \\
&= \alpha(ym \gg\!= (y \mapsto \eta(f(y)) \gg\!= id_{TX})) && \text{by bind associativity} \\
&= \alpha(ym \gg\!= (y \mapsto id_{TX}(f(y)))) && \text{as } \eta(x) \gg\!= f = f(x) \\
&= \alpha(ym \gg\!= f)
\end{aligned}$$

Thus the result holds. □

Each monad algebra provides an interpretation of elements of TX, which are a 'T-structured' collection of elements in X, as single elements of X. In the case of powerset monad, this will be a union over the sets and in the case of list monad, this will be a concatenation of the lists. Using a tree monad to interpret non-determinism would require the use of a tree traversal which satisfies the two properties in the definition, but we will not consider this here.

4.3.2 Semantic Domains

We now define the domains which will be used to interpret the types of the FLP language.

For each value type A we inductively define a set $[\![A]\!]$. The sets of first-order values types will admit a universe of values $\left([\![A]\!]\right)_{A \in \mathsf{FO}}$. These are defined by mutual induction over value and computation types as follows:

$$\begin{aligned}
[\![\mathsf{Num}]\!] &\overset{\text{def}}{=} \mathbb{N} \\
[\![A_1 \times A_2]\!] &\overset{\text{def}}{=} [\![A_1]\!] \times [\![A_2]\!] \\
[\![U\underline{B}]\!] &\overset{\text{def}}{=} \mathfrak{X}_{[\![\underline{B}]\!]}
\end{aligned}$$

We reiterate that the symbol $\times$ on the right-hand side is distinct from the ones used in the language, this is the usual symbol for the Cartesian product on sets.

Then each computation type $\underline{B}$ will be interpreted by a structure which contains both a T-algebra for the monad in addition to an FLP algebra over $\left([\![A]\!]\right)_{A \in \mathsf{FO}}$. So $[\![\underline{B}]\!] = \left(\mathfrak{X}_{\underline{B}}, \alpha_{\underline{B}}, \varepsilon_{\underline{B}}, \vee_{\underline{B}}, \exists_{\underline{B}}, \mathsf{eq}_{\underline{B}}\right)$ where $\alpha_{\underline{B}} : T\mathfrak{X}_{\underline{B}} \to \mathfrak{X}_{\underline{B}}$ is an T-algebra over $\mathfrak{X}_{\underline{B}}$, and $\left(\mathfrak{X}_{\underline{B}}, \varepsilon_{\underline{B}}, \vee_{\underline{B}}, \exists_{\underline{B}}, \mathsf{eq}_{\underline{B}}\right)$ is an FLP algebra over the same carrier set $\mathfrak{X}_{\underline{B}}$ as $\alpha_{\underline{B}}$. We abuse notation somewhat by writing $\varepsilon_{\underline{B}}$ instead of $\varepsilon_{\mathfrak{X}_{\underline{B}}}$ and similarly for the other operations. This considerably lightens our notation and will usually not cause confusion in the cases we consider. In particular, we highlight the distinction from $\varepsilon_{T\mathfrak{X}_{\underline{B}}}$ given by the FLP structure for the FLP algebra on $T[\![\underline{B}]\!]$.

For returner computation types we define:

$$[\![FA]\!] \overset{\text{def}}{=} \left(T[\![A]\!], \mu_{T[\![A]\!]}, \varepsilon_{T[\![A]\!]}, \vee_{T[\![A]\!]}, \exists_{T[\![A]\!]}, \mathsf{eq}_{T[\![A]\!]}\right)$$

where FLP algebra $\left(T[\![A]\!], \varepsilon_{T[\![A]\!]}, \vee_{T[\![A]\!]}, \exists_{T[\![A]\!]}, \mathsf{eq}_{T[\![A]\!]}\right)$ comes from the FLP structure on T we assumed earlier. The T-algebra $(T[\![A]\!], \mu_{T[\![A]\!]})$ uses the multiplication operation of the monad we saw earlier. It is given by

$$\mu_{T[\![A]\!]} : TT[\![A]\!] \to T[\![A]\!]$$
$$\mu_{T[\![A]\!]}(xmm) \stackrel{\text{def}}{=} xmm \gg\!= \mathrm{id}_{T[\![A]\!]}$$

For function computation types we define:

$$[\![A \to \underline{B}]\!] \stackrel{\text{def}}{=} \left([\![A]\!] \to [\![\underline{B}]\!], \alpha_{A\to\underline{B}}, \varepsilon_{A\to\underline{B}}, \vee_{A\to\underline{B}}, \exists_{A\to\underline{B}}, \mathsf{eq}_{A\to\underline{B}}\right)$$

where the operations are inductively defined by:

$$\begin{aligned}
\alpha_{A\to\underline{B}}(fm)(a) &\stackrel{\text{def}}{=} \alpha_{\underline{B}}(fm \gg\!= (f \mapsto \eta(f(a))))\\
\varepsilon_{A\to\underline{B}}(a) &\stackrel{\text{def}}{=} \varepsilon_{\underline{B}}\\
(f \vee_{A\to\underline{B}} g)(a) &\stackrel{\text{def}}{=} f(a) \vee_{\underline{B}} g(a)\\
\left(\exists^{A'}_{A\to\underline{B}} f\right)(a) &\stackrel{\text{def}}{=} \exists^{A'}_{\underline{B}}(V' \mapsto f(V')(a))\\
\mathsf{eq}^{A'}_{A\to\underline{B}}(w, v, f)(a) &\stackrel{\text{def}}{=} \begin{cases} f(a) & \text{if } v = w\\ \varepsilon_{A\to\underline{B}} & \text{if } v \neq w\end{cases}
\end{aligned}$$

We now provide an interpretation of contexts and terms in the language within the semantic domains we have defined.

We recall each context Γ, is a mapping from free variables to value types, so it defines a set $[\![\Gamma]\!]$ by taking the Cartesian product over denotation of the types of variables in the context.

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \prod_{(x:A)\in\Gamma} [\![A]\!]$$

We call elements $\rho \in [\![\Gamma]\!]$ environments, which provide an element in the appropriate domain for each free variable. We will use the notation $\rho[x \mapsto v]$ to denote the extension of ρ with x mapped to v.

Given some environment $\rho \in [\![\Gamma]\!]$, we inductively define the meaning of each value and computation term, so that

$$\left[\!\!\left[\Gamma \vdash^{\mathsf{v}} V : A\right]\!\!\right]\rho \in [\![A]\!] \qquad\qquad \left[\!\!\left[\Gamma \vdash^{\mathsf{c}} M : \underline{B}\right]\!\!\right]\rho \in \mathfrak{X}_{\underline{B}}$$

We lighten our notation by writing $[\![M]\!]$ instead of $\left[\!\!\left[\Gamma \vdash^{\mathsf{c}} M : \underline{B}\right]\!\!\right]$, and similarly $[\![V]\!]$ for $\left[\!\!\left[\Gamma \vdash^{\mathsf{v}} V : A\right]\!\!\right]$. Then the denotation of terms are defined by:

$$\begin{aligned}
[\![x]\!]\rho &\stackrel{\text{def}}{=} \rho(x)\\
[\![\overline{n}]\!]\rho &\stackrel{\text{def}}{=} n\\
[\![(V_1, V_2)]\!]\rho &\stackrel{\text{def}}{=} ([\![V_1]\!]\rho, [\![V_2]\!]\rho)\\
[\![\mathsf{thunk}\ M]\!]\rho &\stackrel{\text{def}}{=} [\![M]\!]\rho
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{return}\ V]\!]\rho &\stackrel{\text{def}}{=} \eta([\![V]\!]\rho)\\
[\![\mathsf{force}\ V]\!]\rho &\stackrel{\text{def}}{=} [\![V]\!]\rho\\
[\![\lambda x : A.\, M]\!]\rho &\stackrel{\text{def}}{=} f \text{ where } f(v) \stackrel{\text{def}}{=} [\![M]\!]\rho[x \mapsto v]\\
[\![M(V)]\!]\rho &\stackrel{\text{def}}{=} [\![M]\!]\rho([\![V]\!]\rho)\\
[\![\mathsf{split}(V; x_1, x_2.\, M)]\!] &\stackrel{\text{def}}{=} [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \stackrel{\text{def}}{=} [\![V]\!]\rho\\
[\![\mathsf{ifz}(\overline{n}; M; x.\, N)]\!] &\stackrel{\text{def}}{=} \begin{cases} [\![M]\!]\rho & \text{if } [\![\overline{n}]\!]\rho = 0\\ [\![N]\!]\rho[x \mapsto n] & \text{if } [\![\overline{n}]\!]\rho = n + 1\end{cases}\\
[\![M \text{ to } x.\, N]\!]\rho &\stackrel{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))
\end{aligned}$$

$$
\begin{aligned}
[\![\mathsf{fail}]\!]\rho &\overset{\text{def}}{=} \varepsilon_{\underline{B}} \\
[\![M \mathbin{[\!]} N]\!]\rho &\overset{\text{def}}{=} [\![M]\!]\rho \vee_{\underline{B}} [\![N]\!]\rho \\
[\![\exists x : A.\, M]\!]\rho &\overset{\text{def}}{=} \exists^{A}_{\underline{B}}(v \mapsto [\![M]\!]\rho[x \mapsto v]) \\
[\![V_1 =_A V_2;\ M]\!]\rho &\overset{\text{def}}{=} \mathsf{eq}^{A}_{\underline{B}}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![M]\!]\rho)
\end{aligned}
$$

4.3.3 Substitution

We define the substitution operation in our metatheory. Given a value term $\Gamma \vdash^{\mathsf{v}} e : A$, we write $\Gamma \vdash M[e/x] : B$ for the (value or computation) term obtained by substituting e for all occurrences of the free variable x in $\Gamma, x : A \vdash M : B$. We adopt the Barendregt convention [21, §2.1], assuming that our notion of substitution never results in variable capture or referential clash, by α-conversion of any troublesome bound variables. The only quantifiers in our language are $\exists x : A.\, M$, $\lambda x : A.\, M$, $\mathsf{split}(V; x_1, x_2.\, M)$, $\mathsf{ifz}(n; M; x.\, N)$ and M to $x.\, N$.

Substitution is defined as follows, ignoring contexts and types for readability:

$$
\begin{aligned}
x[e/y] &= \begin{cases} e & \text{if } x = y \\ x & \text{if } x \neq y \end{cases} \\
\overline{n}[e/x] &= \overline{n} \\
(V_1, V_2)[e/x] &= (V_1[e/x], V_2[e/x]) \\
(\mathsf{thunk}\ M)[e/x] &= \mathsf{thunk}\ M[e/x] \\
(\mathsf{return}\ V)[e/x] &= \mathsf{return}\ V[e/x] \\
(\mathsf{force}\ V)[e/x] &= \mathsf{force}\ V[e/x] \\
(\lambda y : A.\, M)[e/x] &= \lambda y : A.\, M[e/x] \\
(M(V))[e/x] &= M[e/x](V[e/x]) \\
\mathsf{split}(V; y_1, y_2.\, M)[e/x] &= \mathsf{split}(V[e/x]; y_1, y_2.\, M[e/x]) \\
\mathsf{ifz}(n; M; y.\, N)[e/x] &= \mathsf{ifz}(n; M[e/x]; y.\, N[e/x]) \\
(M\ \mathsf{to}\ y.\, N)[e/x] &= M[e/x]\ \mathsf{to}\ y.\, N[e/x] \\
\mathsf{fail}[e/x] &= \mathsf{fail} \\
(M \mathbin{[\!]} N)[e/x] &= M[e/x] \mathbin{[\!]} N[e/x] \\
(\exists y : A.\, M)[e/x] &= \exists y : A.\, M[e/x] \\
(V_1 =_A V_2;\ M)[e/x] &= V_1[e/x] =_A V_2[e/x];\ M[e/x]
\end{aligned}
$$

4.4 Equational Theory

We now present an equational theory for the functional logic language. We write $\Gamma \vdash^{\mathsf{c}} M = N : \underline{B}$ to mean that computation terms M and N are equated under context Γ. We do not give equations for value terms, since values are not seen to 'do' anything in a CBPV-based language. In particular, by consulting the typing rules we can see no elimination rules land into a value term. As mentioned in [14], we highlight the need for the context Γ, and the type of the terms M and N, to be explicit in the equational theory. This is because the interpretation of the terms M and N depend on them, in particular, if they contain free variables or effects.

4.4.1 General laws for CBPV

First, we have the standard CBPV equations, which hold in any language based upon it. These describe the basic eliminations and computation steps in a CBPV language. For further discussion of these basic laws, see Levy [13, §4.3].

β-laws

Eq-Beta-1
$$\frac{\Gamma \vdash^{v} V : A \qquad \Gamma, x : A \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} \mathsf{return}\ V\ \mathsf{to}\ x.\, M = M[V/x] : \underline{B}}$$

Eq-Beta-2
$$\frac{\Gamma \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} \mathsf{force}\ (\mathsf{thunk}\ M) = M : \underline{B}}$$

Eq-Beta-3
$$\frac{\Gamma \vdash^{v} V_1 : A_1 \qquad \Gamma \vdash^{v} V_2 : A_2 \qquad \Gamma, x_1 : A_1, x_2 : A_2 \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} \mathsf{split}((V_1, V_2); x_1, x_2.\, M) = M[V_1, V_2/x_1, x_2] : \underline{B}}$$

Eq-Beta-4
$$\frac{\Gamma, x : A \vdash^{c} M : \underline{B} \qquad \Gamma \vdash^{v} V : A}{\Gamma \vdash^{c} (\lambda x : A.\, M)(V) = M[V/x] : \underline{B}}$$

Eq-Beta-5
$$\frac{\Gamma \vdash^{c} M : \underline{B} \qquad \Gamma, x : \mathsf{Num} \vdash^{c} N : \underline{B}}{\Gamma \vdash^{c} \mathsf{ifz}(\overline{0}; M; x.\, N) = M : \underline{B}}$$

Eq-Beta-6
$$\frac{\Gamma \vdash^{c} M : \underline{B} \qquad \Gamma, x : \mathsf{Num} \vdash^{c} N : \underline{B}}{\Gamma \vdash^{c} \mathsf{ifz}(\overline{n+1}; M; x.\, N) = N[\overline{n}/x] : \underline{B}}$$

η-laws

Eq-Eta
$$\frac{\Gamma \vdash^{v} V : A_1 \times A_2 \qquad \Gamma, p : A_1 \times A_2 \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} M[V/p] = \mathsf{split}(V; x_1, x_2.\, M[(x_1, x_2)/p]) : \underline{B}}$$

sequencing laws

Eq-Seq-1
$$\frac{\Gamma \vdash^{c} M : FA}{\Gamma \vdash^{c} M = M\ \mathsf{to}\ x.\, \mathsf{return}\ x : FA}$$

Eq-Seq-2
$$\frac{\Gamma \vdash^{c} M : FA_1 \qquad \Gamma, x_1 : A_1 \vdash^{c} N : FA_2 \qquad \Gamma, x_2 : A_2 \vdash^{c} P : \underline{B}}{\Gamma \vdash^{c} (M\ \mathsf{to}\ x_1.\, N)\ \mathsf{to}\ x_2.\, P = M\ \mathsf{to}\ x_1.\, (N\ \mathsf{to}\ x_2.\, P) : \underline{B}}$$

Eq-Seq-3
$$\frac{\Gamma \vdash^{c} M : FA_1 \qquad \Gamma, x : A_1 \vdash^{c} N : A_2 \to \underline{B} \qquad \Gamma \vdash^{v} V : A_2}{\Gamma \vdash^{c} (M\ \mathsf{to}\ x.\, N)(V) = M\ \mathsf{to}\ x.\, N(V) : \underline{B}}$$

4.4.2 Effect laws for FLP

We now present the laws relating to the effects of the FLP language. Firstly how effects interact with lambda abstraction and secondly how effects interact with each other and themselves. This second set of equations will mirror many of the properties of the FLP algebras and FLP structures we defined earlier.

Lambda vs. effects

The following set of equations describes the property that all FLP effects commute with abstraction.

Eq-Lam-Fail
$$\frac{}{\Gamma \vdash^{c} \lambda x : A.\, \mathsf{fail} = \mathsf{fail} : \underline{B}}$$

The Eq-Lam-Fail rule relates different types of fail terms. By inversion of the left-hand term, we can see that the terms have function type $\underline{B} = A \to \underline{B_2}$. Thus, the left-hand term is a function from A to the failure of type $\underline{B_2}$, whereas the right-hand term is the failure of function type $A \to \underline{B_2}$. It is evident that these terms should behave identically, and hence we equate them.

Eq-Lam-Choice
$$\frac{\Gamma, x : A \vdash^{c} M : \underline{B} \qquad \Gamma, x : A \vdash^{c} N : \underline{B}}{\Gamma \vdash^{c} \lambda x : A.\, (M \mathbin{[]} N) = \lambda x : A.\, M \mathbin{[]} \lambda x : A.\, N : A \to \underline{B}}$$

The Eq-Lam-Choice rule relates the lambda abstraction over a choice term with the choice over two lambda abstractions. The commutativity of choice and lambda abstraction describes that it does not matter whether the non-deterministic choice is made before or after the lambda abstraction. The two terms act identically when applied to a value V of type A, either we substitute V into $M \mathbin{[]} N$, then carry out the non-deterministic choice. Or, we carry out the non-deterministic choice to select which abstraction to apply, then substitute V into the appropriate term.

$$\textsc{Eq-Lam-Exists}\quad \frac{\Gamma, x_1 : A_1, x_2 : A_2 \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda x_1 : A_1.\, (\exists x_2 : A_2.\, M) = \exists x_2 : A_2.\, (\lambda x_1 : A_1.\, M) : A_1 \to \underline{B}}$$

The Eq-Lam-Exists rule relates the lambda abstraction over an existential term with the existential over a lambda abstraction. The commutativity of existential and lambda abstraction describes the notion that does not matter whether the existential quantification occurs over or under lambda abstraction when evaluating the terms. The two terms act identically when applied to a value V of type A_1, either we substitute V into $\exists x_2 : A_2.\, M$, then carry out the existential quantification for the bound variable x_2. Or, we carry out the existential quantification for x_2 over the lambda abstraction, then substitute V into the lambdas non-deterministically.

$$\textsc{Eq-Lam-Unify}\quad \frac{\Gamma \vdash^{\mathsf{v}} V_1 : A_1 \qquad \Gamma \vdash^{\mathsf{v}} V_2 : A_1 \qquad \Gamma, x : A_2 \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda x : A_2.\, (V_1 \doteq V_2;\ M) = V_1 \doteq V_2;\ \lambda x : A_2.\, M : A_2 \to \underline{B}}$$

The Eq-Lam-Unify rule relates the lambda abstraction over a unification term with the unification over a lambda abstraction. The commutativity of unification and lambda abstraction describes the notion that does not matter whether we unify the value terms before or after the lambda abstraction. The two terms act identically when applied to a value V of type A_2, either we substitute V into $V_1 \doteq V_2;\ M$, then carry out the unification. Or, we carry out the unification over the lambda abstraction, then if it is successful, substitute V into the lambda abstraction. It does not matter if we substitute V for the bound variable x before unification since the rule requires that x is not free in V_1 and V_2. We note that if the unification of the value terms fails, then we are in the same case as Eq-Lam-Fail, and hence the terms are equivalent.

FLP algebra laws

Many of the laws in this section are derived from those given in the definition of FLP algebras Definition 4.1.5. Others come from the paper by Staton [18].

$$\textsc{Eq-Choice-Assoc}\quad \frac{\Gamma \vdash^{\mathsf{c}} M_1 : \underline{B} \qquad \Gamma \vdash^{\mathsf{c}} M_2 : \underline{B} \qquad \Gamma \vdash^{\mathsf{c}} M_3 : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M_1 \mathbin{[\!]} (M_2 \mathbin{[\!]} M_3) = (M_1 \mathbin{[\!]} M_2) \mathbin{[\!]} M_3 : \underline{B}}$$

The Eq-Choice-Assoc rule describes the associativity of choice terms. Both the powerset and list interpretations of non-determinism which we have considered are associative, and so the choice term will be associative in these interpretations. We note that the associativity of choice terms is not a property of the choice term itself, but rather a property of the interpretation of the language's non-determinism. If we were to consider an interpretation using a tree monad, then clearly the nesting of choices would not be associative.

$$\textsc{Eq-Choice-Unit}\quad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M \mathbin{[\!]} \mathsf{fail} = \mathsf{fail} \mathbin{[\!]} M : \underline{B}}$$

The Eq-Choice-Unit rule describes how failure acts as a unit for choice. While in general, the choice of any two terms is not commutative due to this contradicting the list interpretation of non-determinism, the choice of a term with failure is commutative. This is because the choice of a term with failure can be seen as a non-choice, there is no computation to be done in the case of failure and so we always choose M. In particular, the empty list commutes with list concatenation so this makes sense.

The following set of equations covers unification.

$$\textsc{Eq-Unify-Refl}\quad \frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} V \doteq V;\ M = M : \underline{B}}$$

The Eq-Unify-Refl rule describes the reflexivity of unification. If we unify a value term with itself, then unification always succeeds, and we always continue as the computation M.

$$\textsc{Eq-Unify-Subst}\quad \frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{v}} W : A \qquad \Gamma, x : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} V \doteq W;\ M[V/x] = V \doteq W;\ M[W/x] : \underline{B}}$$

The Eq-Unify-Subst rule describes the substitution of unified value terms into the subsequent computations. If the value terms can be unified, then they are equal, and so we can substitute either in the subsequent computation for the same evaluation. If the value terms cannot be unified, then the unification fails, so the subsequent computation is not evaluated, and thus the substitutions are irrelevant. Hence in either case both terms are equivalent.

Eq-Unify-Comm

$$\frac{\Gamma \vdash^{\mathsf{v}} V_1 : A_1 \qquad \Gamma \vdash^{\mathsf{v}} W_1 : A_1 \qquad \Gamma \vdash^{\mathsf{v}} V_2 : A_2 \qquad \Gamma \vdash^{\mathsf{v}} W_2 : A_2 \qquad \Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} V_1 \doteq W_1;\ V_2 \doteq W_2;\ M = V_2 \doteq W_2;\ V_1 \doteq W_1;\ M : \underline{B}}$$

The Eq-Unify-Comm rule describes the commutativity of unification. If we unify two pairs of value terms, then the order in which we unify them is irrelevant. If both the value pairs can be unified successfully then in both cases the terms will continue as M, and so the order of unification is irrelevant. If one or both of the pairs cannot be unified, then the unification fails, and so the subsequent computation is not evaluated. Hence both terms will fail before reaching the computation M, and so they are equivalent.

Eq-Unify-Choice

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{v}} W : A \qquad \Gamma \vdash^{\mathsf{c}} M : \underline{B} \qquad \Gamma \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} V \doteq W;\ (M \mathbin{[]} N) = (V \doteq W;\ M) \mathbin{[]} (V \doteq W;\ N) : \underline{B}}$$

The Eq-Unify-Choice rule describes the distributivity of unification over choice. If we unify two value terms and then continue with a choice of two computations, then this is equivalent to evaluating a choice of two of the same unifications but with each followed by one of the distinct computations. Since if the unification in the left term fails then both choices will fail in the right-hand term. If the unification in the left term succeeds, then the right-hand term will act the same as $M \mathbin{[]} N$, since both unifications in the right-hand term will also succeed. Hence both terms are equivalent.

Eq-Unify-Fail

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \qquad \Gamma \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{c}} V \doteq W;\ \mathsf{fail} = \mathsf{fail} : \underline{B}}$$

The Eq-Unify-Fail rule describes the failure of unification. The unification of two terms followed by a failure computation is equivalent to just failing. This is because if the value terms can be successfully unified and then the subsequent computation will be carried out which is fail. If the value terms cannot be unified, then the unification will fail and so the computation also results in failure. Hence the left-hand term acts identically as a plain failure computation, and so the terms are equivalent.

The following laws relate to sequencing and finitary choice. They follow from the definition of FLP structures (Definition 4.1.9).

Eq-Bind-Fail

$$\frac{\Gamma, x : FA \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathsf{fail\ to}\ x.\, M = \mathsf{fail} : \underline{B}}$$

The Eq-Bind-Fail rule describes what happens when we bind a failure computation to a variable. If we bind a failure computation of returner type to a variable, then when the term is executed we will need to execute the returner type computation first to get the value to bind to the variable. However, since the failure computation is executed the whole term will fail. Hence, binding a failure computation to a variable is equivalent to just failing.

Eq-Bind-Choice

$$\frac{\Gamma \vdash^{\mathsf{c}} M_1 : FA \qquad \Gamma \vdash^{\mathsf{c}} M_2 : FA \qquad \Gamma, x : FA \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} (M_1 \mathbin{[]} M_2)\ \mathsf{to}\ x.\, N = M_1\ \mathsf{to}\ x.\, N \mathbin{[]} M_2\ \mathsf{to}\ x.\, N : \underline{B}}$$

The Eq-Bind-Choice rule describes the distributivity of binding over choice. If we bind a choice of two returner type computations to a variable, then this is equivalent to the choice over the separate binding of each of the returner type computations to the variable and then continuing with the same computation.

Notably, the following rules which are found in the paper by Staton [18], do not hold for our language, because the list interpretation of non-determinism does not satisfy them (but the powerset does). For some rules, this is because the concatenation of two lists with the same contents is a list with the contents repeated twice, whereas, the union of two sets with the same contents is the same as one of the initial sets. In other rules, this is due to the unordered nature of sets or the uniqueness of the elements in a set. We present these rules below, despite them not holding in our generalised setting which includes the list interpretation of non-determinism.

$$\textsc{Eq-Choice-Comm}\quad \frac{\Gamma \vdash^{c} M : \underline{B} \qquad \Gamma \vdash^{c} N : \underline{B}}{\Gamma \vdash^{c} M \talloblong N = N \talloblong M : \underline{B}}$$

$$\textsc{Eq-Choice-Idem}\quad \frac{\Gamma \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} M \talloblong M = M : \underline{B}}$$

$$\textsc{Eq-Exists-Choice-1}\quad \frac{\Gamma \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} (\exists x : A.\, M) \talloblong M = M : \underline{B}}$$

$$\textsc{Eq-Exists-Choice-2}\quad \frac{\Gamma \vdash^{v} V : A \qquad \Gamma, x : A \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} M[V/x] \talloblong (\exists x : A.\, M) = \exists x : A.\, M : \underline{B}}$$

$$\textsc{Eq-Exists-Choice-3}\quad \frac{\Gamma, x : A \vdash^{c} M : \underline{B} \qquad \Gamma, x : A \vdash^{c} N : \underline{B}}{\Gamma \vdash^{c} \exists x : A.\, (M \talloblong N) = (\exists x : A.\, M) \talloblong (\exists x : A.\, N) : \underline{B}}$$

$$\textsc{Eq-Unify-Choice-2}\quad \frac{\Gamma \vdash^{v} V : A \qquad \Gamma \vdash^{v} W : A \qquad \Gamma \vdash^{c} M : \underline{B}}{\Gamma \vdash^{c} (V \doteq W;\, M) \talloblong M = M : \underline{B}}$$

Chapter 5

Soundness of Denotational Semantics

In this chapter, we prove the soundness of the denotational semantics of the core FLP language, which means if two terms are equal within the equational theory then they are represented by the same object in the denotational semantics. This is a useful result which tells us our equational theory is consistent up to our beliefs on mathematical consistency of the theory we work in (ZFC) [22].

5.1 Preliminary Results

Before we prove soundness we will need to prove a few smaller results, namely:

1. A substitution lemma for values,
2. A denotational equivalence between terms with pair type in the context and the same term substituted with separate variables. I.e the notion of a pair in the language coincides with the mathematical notion of ordered pairs.
3. Some naturality properties for FLP structure operations with monad algebras.

Lemma 5.1.1 (Substitution Lemma)**.** *Let $\Gamma \vdash^v V : A$ be a value, and a computation term $\Gamma, x : A \vdash^c M : \underline{B}$. Then the substitution of V for x in M is a term $\Gamma \vdash^c M[V/x] : \underline{B}$, and the following holds:*

$$[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Additionally, let $\Gamma \vdash^v V : A$ be a value, and suppose a value term $\Gamma, x : A \vdash^v M : A_2$. Then the substitution of V for x in M is a term $\Gamma \vdash^v M[V/x] : A_2$, and the following holds:

$$[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Proof. We prove the two results by mutual induction, thus we work by considering the typing derivation for M.

Case(Var). Suppose the derivation of $\Gamma, x : A \vdash^v M : A_2$ has the shape

$$\frac{}{\Gamma, x : A, y : A_2 \vdash^v y : A_2}\ \textsc{Var}$$

Then we consider two cases. Suppose $x \neq y$. Then by definition we have $y[V/x] = y$. Then by the definition of free variables in the denotational semantics, we have:

$$[\![y[V/x]]\!]\rho = [\![y]\!]\rho = \rho(y) = \rho[x \mapsto [\![V]\!]\rho](y) = [\![y]\!]\rho[x \mapsto [\![V]\!]\rho]$$

So the result holds for $x \neq y$.

Now suppose $x = y$, so $y[V/x] = V$. Then we see that:

$$[\![y[V/x]]\!]\rho = [\![V]\!]\rho = \rho[x \mapsto [\![V]\!]\rho](x) = \rho[x \mapsto [\![V]\!]\rho](y) = [\![y]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Thus the result holds if $x = y$, and hence for the rule Var.

CASE(NAT). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{v}} M : A_2$ has the shape

$$\dfrac{n \in \mathbb{N}}{\Gamma, x : A \vdash^{\mathrm{v}} \overline{n} : \mathsf{Num}}\ \textsc{Nat}$$

Now by definition we have that $\overline{n}[V/x] = \overline{n}$, and so we see

$$[\![\overline{n}[V/x]]\!]\rho = [\![\overline{n}]\!]\rho = n$$

Furthermore, since the denotational semantics of a Num type value do not depend on the environment ρ, the result follows immediately

$$[\![\overline{n}[V/x]]\!]\rho = [\![\overline{n}]\!]\rho = n = [\![\overline{n}]\!]\rho[x \mapsto [\![V]\!]\rho]$$

CASE(TUPLE). Suppose the derivation of M has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{\mathrm{v}} V_1 : A_1} \qquad \dfrac{\vdots}{\Gamma \vdash^{\mathrm{v}} V_2 : A_2}}{\Gamma, x : A \vdash^{\mathrm{v}} (V_1, V_2) : A_1 \times A_2}\ \textsc{Tuple}$$

Then we know by definition that $(V_1, V_2)[V/x] = (V_1[V/x], V_2[V/x])$, and so we have by the denotational semantics of tuples that:

$$\begin{aligned}[\![(V_1, V_2)[V/x]]\!]\rho &= [\![(V_1[V/x], V_2[V/x])]\!]\rho \\ &= ([\![V_1[V/x]]\!]\rho, [\![V_2[V/x]]\!]\rho)\end{aligned}$$

Now by the induction hypothesis we have that $[\![V_1[V/x]]\!]\rho = [\![V_1]\!]\rho[x \mapsto [\![V]\!]\rho]$, and similarly for V_2. Thus using this with the above and the definitions in the denotational semantics we see that:

$$\begin{aligned}[\![(V_1, V_2)[V/x]]\!]\rho &= ([\![V_1[V/x]]\!]\rho, [\![V_2[V/x]]\!]\rho) \\ &= ([\![V_1]\!]\rho[x \mapsto [\![V]\!]\rho], [\![V_2]\!]\rho[x \mapsto [\![V]\!]\rho]) \\ &= [\![(V_1, V_2)]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

So the result holds in this case

CASE(THUNK). Suppose the derivation of M has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A \vdash^{\mathrm{v}} \mathsf{thunk}\ M : U\underline{B}}\ \textsc{Thunk}$$

Then we know that $(\mathsf{thunk}\ M)[V/x] = \mathsf{thunk}\ M[V/x]$. Additionally by the induction hypothesis, we have that $[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$, and so combining these facts with the definition of the denotational semantics of a thunk we see that:

$$\begin{aligned}[\![(\mathsf{thunk}\ M)[V/x]]\!]\rho &= [\![\mathsf{thunk}\ M[V/x]]\!]\rho \\ &= [\![M[V/x]]\!]\rho \\ &= [\![M]\!]\rho[x \mapsto [\![V]\!]\rho] \\ &= [\![\mathsf{thunk}\ M]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

So the result holds for this case.

CASE(RETURN). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{v}} W : A_2}}{\Gamma, x : A \vdash^{\mathrm{c}} \mathsf{return}\ W : FA_2}\ \textsc{Return}$$

Then we know we also have

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{\mathrm{v}} W[V/x] : A_2}}{\Gamma \vdash^{\mathrm{c}} \mathsf{return}\ W[V/x] : FA_2}\ \textsc{Return}$$

By definition for the denotation of return terms, we have that:

$$[\![\mathsf{return}\ W[V/x]]\!]\rho = \eta([\![W[V/x]]\!]\rho)$$

By the induction hypothesis, we have that:

$$[\![W[V/x]]\!]\rho = [\![W]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Thus we immediately see that the result holds in this case.

$$[\![\mathsf{return}\ W[V/x]]\!]\rho = \eta([\![W]\!]\rho[x \mapsto [\![V]\!]\rho]) = [\![\mathsf{return}\ W]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Case(Bind). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{c}} M : FA_2} \qquad \dfrac{\vdots}{\Gamma, x : A, y : A_2 \vdash^{\mathrm{c}} N : \underline{B}}}{\Gamma \vdash^{\mathrm{c}} M\ \mathsf{to}\ y.\, N : \underline{B}}\ \textsc{Bind}$$

We know that $M\ \mathsf{to}\ y.\, N[V/x] = M[V/x]\ \mathsf{to}\ y.\, N[V/x]$. Using this and the definition of the denotational semantics for bind terms we see that:

$$[\![M[V/x]\ \mathsf{to}\ y.\, N[V/x]]\!]\rho = \alpha_{\underline{B}}([\![M[V/x]]\!]\rho \gg\!= (v \mapsto \eta([\![N[V/x]]\!]\rho[y \mapsto v])))$$

Also by the induction hypothesis, we have that:

$$[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$$

$$[\![N[V/x]]\!]\rho = [\![N]\!]\rho[x \mapsto [\![V]\!]\rho]$$

Thus we see that

$$\begin{aligned}[\![M\ \mathsf{to}\ y.\, N[V/x]]\!]\rho &= [\![M[V/x]\ \mathsf{to}\ y.\, N[V/x]]\!]\rho \\ &= \alpha_{\underline{B}}([\![M]\!]\rho[x \mapsto [\![V]\!]\rho] \gg\!= (v \mapsto \eta([\![N]\!]\rho[y \mapsto v, x \mapsto [\![V]\!]\rho]))) \\ &= [\![M\ \mathsf{to}\ y.\, N]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

So the result holds for this case.

Case(Lam). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A, y : A_2 \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A \vdash^{\mathrm{c}} \lambda y : A_2.\, M : A_2 \to \underline{B}}\ \textsc{Lam}$$

By the definition of substitution we know that $(\lambda y : A_2.\, M)[V/x] = \lambda y : A_2.\, M[V/x]$. By the induction hypothesis we have that $[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$. Now using the denotational semantics of lambda terms we see that:

$$\begin{aligned}[\![(\lambda y : A_2.\, M)[V/x]]\!]\rho &= [\![\lambda y : A_2.\, M[V/x]]\!]\rho \\ &= f \text{ where } f(v) \overset{\text{def}}{=} [\![M[V/x]]\!]\rho[y \mapsto v] \\ &= f \text{ where } f(v) \overset{\text{def}}{=} [\![M]\!]\rho[x \mapsto [\![V]\!]\rho, y \mapsto v] \\ &= [\![\lambda y : A_2.\, M]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

Thus the result holds.

Case(App). Suppose the derivation of $\Gamma, x : A \vdash^{c} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : A_2 \to \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{v} N : A_2}}{\Gamma, x : A \vdash^{c} M(N) : \underline{B}}\ \textsc{App}$$

Then we know that $M(N)[V/x] = M[V/x](N[V/x])$. By the induction hypothesis we have that $\llbracket M[V/x] \rrbracket \rho = \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]$, similarly for N. By the denotational semantics of application we have that:

$$\begin{aligned}
\llbracket M(N)[V/x] \rrbracket \rho &= \llbracket M[V/x](N[V/x]) \rrbracket \rho \\
&= \llbracket M[V/x] \rrbracket \rho (\llbracket N[V/x] \rrbracket \rho) \\
&= \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho](\llbracket N \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]) \\
&= \llbracket M(N) \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]
\end{aligned}$$

Hence the result holds.

Case(Split). Suppose the derivation of $\Gamma, x : A \vdash^{c} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{v} W : A_1 \times A_2} \qquad \dfrac{\vdots}{\Gamma, x : A, x_1 : A_1, x_2 : A_2 \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \mathsf{split}(W; x_1, x_2.\, M) : \underline{B}}\ \textsc{Split}$$

Then we know that $\mathsf{split}(W; x_1, x_2.\, M)[V/x] = \mathsf{split}(W[V/x]; x_1, x_2.\, M[V/x])$. By the induction hypothesis we have that $\llbracket M[V/x] \rrbracket \rho = \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]$, similarly for W. By the denotational semantics of split we have that:

$$\begin{aligned}
\llbracket \mathsf{split}(W; x_1, x_2.\, M)[V/x] \rrbracket \rho &= \llbracket \mathsf{split}(W[V/x]; x_1, x_2.\, M[V/x]) \rrbracket \rho \\
&= \llbracket M[V/x] \rrbracket \rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\text{def}}{=} \llbracket W[V/x] \rrbracket \rho \\
&= \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho, x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\text{def}}{=} \llbracket W \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho] \\
&= \llbracket \mathsf{split}(W; x_1, x_2.\, M) \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]
\end{aligned}$$

So the result holds in this case.

Case(Ifz). Suppose the derivation of $\Gamma, x : A \vdash^{c} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{v} W : \mathsf{Num}} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A, y : \mathsf{Num} \vdash^{c} N : \underline{B}}}{\Gamma, x : A \vdash^{c} \mathsf{ifz}(W; M; y.\, N) : \underline{B}}\ \textsc{Ifz}$$

Then we know that $\mathsf{ifz}(W; M; y.\, N)[V/x] = \mathsf{ifz}(W[V/x]; M[V/x]; y.\, N[V/x])$. By the induction hypothesis we have that $\llbracket M[V/x] \rrbracket \rho = \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]$, similarly for N and W. By the denotational semantics of if zero terms we have that:

$$\begin{aligned}
\llbracket \mathsf{ifz}(W; M; y.\, N)[V/x] \rrbracket \rho &= \llbracket \mathsf{ifz}(W[V/x]; M[V/x]; y.\, N[V/x]) \rrbracket \rho \\
&= \begin{cases} \llbracket M[V/x] \rrbracket \rho & \text{if } \llbracket W[V/x] \rrbracket \rho = 0 \\ \llbracket N[V/x] \rrbracket \rho[x \mapsto n] & \text{if } \llbracket W[V/x] \rrbracket \rho = n + 1 \end{cases} \\
&= \begin{cases} \llbracket M \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho] & \text{if } \llbracket W \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho] = 0 \\ \llbracket N \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho, x \mapsto n] & \text{if } \llbracket W \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho] = n + 1 \end{cases} \\
&= \llbracket \mathsf{ifz}(W; M; y.\, N)[V/x] \rrbracket \rho[x \mapsto \llbracket V \rrbracket \rho]
\end{aligned}$$

So the result holds in this case.

Case(Force). Suppose the derivation of $\Gamma, x : A \vdash^{c} M : \underline{B}$ has the shape

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{v} W : U\underline{B}}}{\Gamma, x : A \vdash^{c} \mathsf{force}\ W : \underline{B}}\ \textsc{Force}$$

Then we know that $\mathsf{force}\ W[V/x] = \mathsf{force}\ W[V/x]$. By the induction hypothesis we have that $[\![W[V/x]]\!]\rho = [\![W]\!]\rho[x \mapsto [\![V]\!]\rho]$. By the denotational semantics of force we have that:

$$\begin{aligned}[\![\mathsf{force}\ W[V/x]]\!]\rho &= [\![\mathsf{force}\ W[V/x]]\!]\rho \\ &= [\![W[V/x]]\!]\rho \\ &= [\![W]\!]\rho[x \mapsto [\![V]\!]\rho] \\ &= [\![\mathsf{force}\ W]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

Hence the result holds in this case.

CASE(FAIL). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\frac{}{\Gamma, x : A \vdash^{\mathrm{c}} \mathsf{fail} : \underline{B}}\ \textsc{Fail}$$

Then we know that $\mathsf{fail}[V/x] = \mathsf{fail}$. By the denotational semantics of fail we trivially see that:

$$\begin{aligned}[\![\mathsf{fail}[V/x]]\!]\rho &= [\![\mathsf{fail}]\!]\rho \\ &= \varepsilon_{\underline{B}} \\ &= [\![\mathsf{fail}]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

Hence the result holds in this case.

CASE(CHOICE). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\frac{\dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{c}} M_1 : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{c}} M_2 : \underline{B}}}{\Gamma, x : A \vdash^{\mathrm{c}} M_1 \mathbin{[\!]} M_2 : \underline{B}}\ \textsc{Choice}$$

Then we know that $(M_1 \mathbin{[\!]} M_2)[V/x] = M_1[V/x] \mathbin{[\!]} M_2[V/x]$. By the induction hypothesis we have that $[\![M_1[V/x]]\!]\rho = [\![M_1]\!]\rho[x \mapsto [\![V]\!]\rho]$ and $[\![M_2[V/x]]\!]\rho = [\![M_2]\!]\rho[x \mapsto [\![V]\!]\rho]$. By the denotational semantics of choice we have that:

$$\begin{aligned}[\![(M_1 \mathbin{[\!]} M_2)[V/x]]\!]\rho &= [\![M_1[V/x] \mathbin{[\!]} M_2[V/x]]\!]\rho \\ &= [\![M_1[V/x]]\!]\rho \vee_{\underline{B}} [\![M_2[V/x]]\!]\rho \\ &= [\![M_1]\!]\rho[x \mapsto [\![V]\!]\rho] \vee_{\underline{B}} [\![M_2]\!]\rho[x \mapsto [\![V]\!]\rho] \\ &= [\![M_1 \mathbin{[\!]} M_2]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

So the result holds in this case.

CASE(EXISTS). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\frac{A_2 \in \mathsf{FO} \qquad \dfrac{\vdots}{\Gamma, x : A, y : A_2 \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A \vdash^{\mathrm{c}} \exists y : A_2.\, M : \underline{B}}\ \textsc{Exists}$$

Now we have that $\exists y : A_2.\, M[V/x] = \exists y : A_2.\, M[V/x]$. By the induction hypothesis we have that $[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$. By the denotational semantics we see:

$$\begin{aligned}[\![\exists y : A_2.\, M[V/x]]\!]\rho &= [\![\exists y : A_2.\, M[V/x]]\!]\rho \\ &= \exists_{\underline{B}}[\![M[V/x]]\!]\rho \\ &= \exists_{\underline{B}}[\![M]\!]\rho[x \mapsto [\![V]\!]\rho] \\ &= [\![\exists y : A_2.\, M]\!]\rho[x \mapsto [\![V]\!]\rho]\end{aligned}$$

So the result holds in this case.

CASE(UNIFY). Suppose the derivation of $\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}$ has the shape

$$\frac{\dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{v}} V_1 : A_2} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{v}} V_2 : A_2} \qquad A_2 \in \mathsf{FO} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A \vdash^{\mathrm{c}} V_1 =_{A_2} V_2;\ M : \underline{B}}\ \textsc{Unify}$$

Then we have that $(V_1 =_{A_2} V_2;\, M)[V/x] = V_1[V/x] =_{A_2} V_2[V/x];\, M[V/x]$. By the induction hypothesis we have that $[\![M[V/x]]\!]\rho = [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]$ and similarly for V_1 and V_2. By the denotational semantics of unification terms we have that:

$$\begin{aligned}
[\![(V_1 =_{A_2} V_2;\, M)[V/x]]\!]\rho &= [\![V_1[V/x] =_{A_2} V_2[V/x];\, M[V/x]]\!]\rho \\
&= \mathsf{eq}^{A_2}_{\underline{B}}([\![V_1[V/x]]\!]\rho, [\![V_2[V/x]]\!]\rho, [\![M[V/x]]\!]\rho) \\
&= \mathsf{eq}^{A_2}_{\underline{B}}([\![V_1]\!]\rho[x \mapsto [\![V]\!]\rho], [\![V_2]\!]\rho[x \mapsto [\![V]\!]\rho], [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]) \\
&= [\![V_1 =_{A_2} V_2;\, M]\!]\rho[x \mapsto [\![V]\!]\rho]
\end{aligned}$$

So the result holds in this case, completing the proof. ☐

Lemma 5.1.2. *Let M be a term which satisfies.*

$$\Gamma, x : A_1 \times A_2 \vdash^c M : \underline{B}$$

Then we have:

$$\left[\!\!\left[\Gamma, x : A_1 \times A_2 \vdash^c M : \underline{B}\right]\!\!\right]\rho[x \mapsto (v_1, v_2)] = \left[\!\!\left[\Gamma, x_1 : A_1, x_2 : A_2 \vdash^c M[(x_1, x_2)/x] : \underline{B}\right]\!\!\right]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]$$

Proof. We work by induction on the typing derivation of M, we will additionally show the result holds for value terms which will form base cases in the induction.

Case(Var). Suppose that the derivation of M takes the shape:

$$\frac{}{\Gamma, x : A_1 \times A_2, y : A_3 \vdash^{\mathsf{v}} y : A_3}\ \textsc{Var}$$

As usual, we consider two cases. Suppose $x \neq y$ and so $y[(x_1, x_2)/x] = y$. Then the result is immediate by considering

$$\begin{aligned}
[\![y[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![y]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= \rho[x_1 \mapsto v_1, x_2 \mapsto v_2](y) \\
&= \rho(y) \\
&= \rho[x \mapsto (v_1, v_2)](y) \\
&= [\![y]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

Where we assume x_1, x_2 are not equal to y by the Barendregt convention in the third line. We now consider the other case where $x = y$ and so $y[(x_1, x_2)/x] = (x_1, x_2)$. Now we see that

$$\begin{aligned}
[\![y[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![(x_1, x_2)]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= ([\![x_1]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2], [\![x_2]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]) \\
&= (\rho[x_1 \mapsto v_1, x_2 \mapsto v_2](x_1), \rho[x_1 \mapsto v_1, x_2 \mapsto v_2](x_2)) \\
&= (v_1, v_2) \\
&= \rho[x \mapsto (v_1, v_2)](x) \\
&= \rho[x \mapsto (v_1, v_2)](y) \\
&= [\![y]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

Hence the result holds in this case overall.

Case(Nat). Suppose that the derivation of M takes the shape:

$$\frac{n \in \mathbb{N}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} \overline{n} : \mathsf{Num}}\ \textsc{Nat}$$

Then we know that $\overline{n}[(x_1, x_2)/x] = \overline{n}$ and so the result is immediate by the definition of the denotational semantics of natural numbers:

$$\left[\!\!\left[\overline{n}[(x_1, x_2)/x]\right]\!\!\right]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![\overline{n}]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = n = [\![\overline{n}]\!]\rho[x \mapsto (v_1, v_2)]$$

CASE(TUPLE). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} V_1 : A_3} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} V_2 : A_4}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} (V_1, V_2) : A_3 \times A_4}\ \textsc{Tuple}$$

Then we know that $(V_1, V_2)[(x_1, x_2)/x] = (V_1[(x_1, x_2)/x], V_2[(x_1, x_2)/x])$. Furthermore, by the induction hypothesis we have that:

$$\begin{aligned}
[\![V_1[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![V_1]\!]\rho[x \mapsto (v_1, v_2)] \\
[\![V_2[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![V_2]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

Hence, we can use the definition of the denotational semantics of tuples to conclude:

$$\begin{aligned}
[\![(V_1, V_2)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![(V_1[(x_1, x_2)/x], V_2[(x_1, x_2)/x])]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= ([\![V_1[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2], [\![V_2[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]) \\
&= ([\![V_1]\!]\rho[x \mapsto (v_1, v_2)], [\![V_2]\!]\rho[x \mapsto (v_1, v_2)]) \\
&= [\![(V_1, V_2)]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

CASE(THUNK). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{c}} M : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} \mathsf{thunk}\ M : U\underline{B}}\ \textsc{Thunk}$$

Now we know that $(\mathsf{thunk}\ M)[(x_1, x_2)/x] = \mathsf{thunk}\ M[(x_1, x_2)/x]$. Also, by the induction hypothesis we have that $[\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![M]\!]\rho[x \mapsto (v_1, v_2)]$. Using these facts along with the denotational semantics of thunks we obtain the result as follows:

$$\begin{aligned}
[\![(\mathsf{thunk}\ M)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![\mathsf{thunk}\ M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![M]\!]\rho[x \mapsto (v_1, v_2)] \\
&= [\![\mathsf{thunk}\ M]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

CASE(RETURN). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} V : A}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} \mathsf{return}\ V : FA}\ \textsc{Return}$$

Now by definition we have that $(\mathsf{return}\ V)[(x_1, x_2)/x] = \mathsf{return}\ V[(x_1, x_2)/x]$. Also, by the induction hypothesis we have that $[\![V[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![V]\!]\rho[x \mapsto (v_1, v_2)]$. Using these facts along with the denotational semantics of returners we obtain the result as follows:

$$\begin{aligned}
[\![(\mathsf{return}\ V)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![\mathsf{return}\ V[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= \eta[\![V[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= \eta[\![V]\!]\rho[x \mapsto (v_1, v_2)] \\
&= [\![\mathsf{return}\ V]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

CASE(BIND). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{c}} M : FA} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2, y : A \vdash^{\mathsf{c}} N : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{c}} M\ \mathsf{to}\ y.\, N : \underline{B}}\ \textsc{Bind}$$

By definition we have that $(M \text{ to } y.\, N)[(x_1, x_2)/x] = M[(x_1, x_2)/x] \text{ to } y.\, N[(x_1, x_2)/x]$. Also, by the induction hypothesis we have that:

$$[\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![M]\!]\rho[x \mapsto (v_1, v_2)]$$

$$[\![N[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![N]\!]\rho[x \mapsto (v_1, v_2)]$$

Using these facts along with the denotational semantics of bind we can see:

$$\begin{aligned}
&[\![(M \text{ to } y.\, N)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
=&[\![M[(x_1, x_2)/x] \text{ to } y.\, N[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
=&\alpha_{\underline{B}}([\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \gg\!= (v \mapsto \eta([\![N[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2, y \mapsto v]))) \\
=&\alpha_{\underline{B}}([\![M]\!]\rho[x \mapsto (v_1, v_2)] \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto (v_1, v_2), y \mapsto v]))) \\
=&[\![M \text{ to } y.\, N]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

So the lemma holds for this case.

CASE(LAM). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2, y : A \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \lambda y : A.\, M : A \to \underline{B}}\ \textsc{Lam}$$

Now by definition we have that $(\lambda y : A.\, M)[(x_1, x_2)/x] = \lambda y : A.\, M[(x_1, x_2)/x]$. Also, by the induction hypothesis we have that $[\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![M]\!]\rho[x \mapsto (v_1, v_2)]$. Using these facts along with the denotational semantics of lambda terms we obtain the result by:

$$\begin{aligned}
[\![(\lambda y : A.\, M)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![\lambda y : A.\, M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= f \text{ where } f(v) \overset{\text{def}}{=} [\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2, y \mapsto v] \\
&= f \text{ where } f(v) \overset{\text{def}}{=} [\![M]\!]\rho[x \mapsto (v_1, v_2), y \mapsto v] \\
&= [\![\lambda y : A.\, M]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

CASE(APP). Suppose that the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M : A \to \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{v}} N : A}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M(N) : \underline{B}}\ \textsc{App}$$

Now by definition we have that $(M(N))[(x_1, x_2)/x] = M[(x_1, x_2)/x](N[(x_1, x_2)/x])$. Also, by the induction hypothesis, we have that:

$$\begin{aligned}
[\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M]\!]\rho[x \mapsto (v_1, v_2)] \\
[\![N[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![N]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

Using these facts along with the denotational semantics of function application we see that:

$$\begin{aligned}
[\![(M(N))[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M[(x_1, x_2)/x](N[(x_1, x_2)/x])]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]([\![N[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]) \\
&= [\![M]\!]\rho[x \mapsto (v_1, v_2)]([\![N]\!]\rho[x \mapsto (v_1, v_2)]) \\
&= [\![M(N)]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

So the result holds.

CASE(SPLIT). Suppose the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{v}} V : A_3 \times A_4} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2, x_3 : A_3, x_4 : A_4 \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \text{split}(V; x_3, x_4.\, M) : \underline{B}}\ \textsc{Split}$$

Now by definition, we have that $\mathsf{split}(V; x_3, x_4.\, M)[(x_1, x_2)/x] = \mathsf{split}(V[(x_1, x_2)/x]; x_3, x_4.\, M[(x_1, x_2)/x])$. Also, by the induction hypothesis, we have that

$$\begin{aligned}\llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)]\\ \llbracket M[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= \llbracket M\rrbracket\rho[x \mapsto (v_1, v_2)]\end{aligned}$$

Using these facts along with the denotational semantics of split we see that:

$$\begin{aligned}&\llbracket\mathsf{split}(V; x_3, x_4.\, M)[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]\\ =&\llbracket\mathsf{split}(V[(x_1, x_2)/x]; x_3, x_4.\, M[(x_1, x_2)/x])\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]\\ =&\llbracket M[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2, x_3 \mapsto v_3, x_4 \mapsto v_4] \text{ where } (v_3, v_4) \overset{\text{def}}{=} \llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]\\ =&\llbracket M\rrbracket\rho[x \mapsto (v_1, v_2), x_3 \mapsto v_3, x_4 \mapsto v_4] \text{ where } (v_3, v_4) \overset{\text{def}}{=} \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)]\\ =&\llbracket\mathsf{split}(V; x_3, x_4.\, M)\rrbracket\rho[x \mapsto (v_1, v_2)]\end{aligned}$$

So the result holds in this case.

CASE(IFZ). Suppose the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{v}} V : \mathsf{Num}} \quad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M : \underline{B}} \quad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2, y : \mathsf{Num} \vdash^{\mathrm{c}} N : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \mathsf{ifz}(V; M; y.\, N) : \underline{B}}\ \textsc{Ifz}$$

Now by definition, we have that the following substitution is

$$\mathsf{ifz}(V; M; y.\, N)[(x_1, x_2)/x] = \mathsf{ifz}(V[(x_1, x_2)/x]; M[(x_1, x_2)/x]; y.\, N[(x_1, x_2)/x])$$

Also by the induction hypothesis, we have that

$$\begin{aligned}\llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)]\\ \llbracket M[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= \llbracket M\rrbracket\rho[x \mapsto (v_1, v_2)]\\ \llbracket N[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= \llbracket N\rrbracket\rho[x \mapsto (v_1, v_2)]\end{aligned}$$

Hence by the denotational semantics of if-zero, we have that:

$$\begin{aligned}&\llbracket\mathsf{ifz}(V; M; y.\, N)[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]\\ =&\llbracket\mathsf{ifz}(V[(x_1, x_2)/x]; M[(x_1, x_2)/x]; y.\, N[(x_1, x_2)/x])\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]\\ =&\begin{cases}\llbracket M[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] & \text{if } \llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = 0\\ \llbracket N[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2, y \mapsto n] & \text{if } \llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = n + 1\end{cases}\\ =&\begin{cases}\llbracket M\rrbracket\rho[x \mapsto (v_1, v_2)] & \text{if } \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)] = 0\\ \llbracket N\rrbracket\rho[x \mapsto (v_1, v_2), y \mapsto n] & \text{if } \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)] = n + 1\end{cases}\\ =&\llbracket\mathsf{ifz}(V; M; y.\, N)\rrbracket\rho[x \mapsto (v_1, v_2)]\end{aligned}$$

Hence the result holds in this case.

CASE(FORCE). Suppose the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{v}} V : U\underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \mathsf{force}\ V : \underline{B}}\ \textsc{Force}$$

Now by definition, we have that the following substitution is

$$(\mathsf{force}\ V)[(x_1, x_2)/x] = \mathsf{force}\ V[(x_1, x_2)/x]$$

Also by the induction hypothesis, we have that

$$\llbracket V[(x_1, x_2)/x]\rrbracket\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = \llbracket V\rrbracket\rho[x \mapsto (v_1, v_2)]$$

Hence by the denotational semantics of force terms, we have that:

$$\begin{aligned}
[\![(\mathsf{force}\ V)[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![\mathsf{force}\ V[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![V[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![V]\!]\rho[x \mapsto (v_1,v_2)] \\
&= [\![\mathsf{force}\ V]\!]\rho[x \mapsto (v_1,v_2)]
\end{aligned}$$

Hence the result holds in this case.

CASE(FAIL). Suppose the derivation of M takes the shape:

$$\dfrac{}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \mathsf{fail} : \underline{B}}\ \textsc{Fail}$$

Now by definition, we have that the following substitution is:

$$\mathsf{fail}[(x_1,x_2)/x] = \mathsf{fail}$$

Hence by the denotational semantics of fail we have that:

$$[\![\mathsf{fail}[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![\mathsf{fail}]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = \varepsilon_{\underline{B}} = [\![\mathsf{fail}]\!]\rho[x \mapsto (v_1,v_2)]$$

Hence the result trivially holds in this case.

CASE(CHOICE). Suppose the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M_1 : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M_2 : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} M_1 \mathbin{[\!]} M_2 : \underline{B}}\ \textsc{Choice}$$

Now by definition, we have that the following substitution is

$$(M_1 \mathbin{[\!]} M_2)[(x_1,x_2)/x] = M_1[(x_1,x_2)/x] \mathbin{[\!]} M_2[(x_1,x_2)/x]$$

Also by the induction hypothesis, we have that

$$\begin{aligned}
[\![M_1[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M_1]\!]\rho[x \mapsto (v_1,v_2)] \\
[\![M_2[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M_2]\!]\rho[x \mapsto (v_1,v_2)]
\end{aligned}$$

Hence by the denotational semantics of choice, we have that:

$$\begin{aligned}
[\![M_1 \mathbin{[\!]} M_2[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M_1[(x_1,x_2)/x] \mathbin{[\!]} M_2[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= [\![M_1[(x_1,x_2)/x]]\!]\rho[x \mapsto (v_1,v_2)] \vee [\![M_2[(x_1,x_2)/x]]\!]\rho[x \mapsto (v_1,v_2)] \\
&= [\![M_1]\!]\rho[x \mapsto (v_1,v_2)] \vee [\![M_2]\!]\rho[x \mapsto (v_1,v_2)] \\
&= [\![M_1 \mathbin{[\!]} M_2]\!]\rho[x \mapsto (v_1,v_2)]
\end{aligned}$$

Therefore the result holds in this case.

CASE(EXISTS). Suppose the derivation of M takes the shape:

$$\dfrac{A \in \mathsf{FO} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2, y : A \vdash^{\mathrm{c}} M : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathrm{c}} \exists y : A.\, V : \underline{B}}\ \textsc{Exists}$$

Now by definition, we have that the following substitution is:

$$(\exists y : A.\, M)[(x_1,x_2)/x] = \exists y : A.\, M[(x_1,x_2)/x]$$

Also by the induction hypothesis, we have that:

$$[\![M[(x_1,x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] = [\![M]\!]\rho[x \mapsto (v_1,v_2)]$$

Now by the definitions of the denotational semantics, we have that:

$$\begin{aligned}
[\![(\exists y : A.\, M[(x_1, x_2)/x])]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![\exists y : A.\, M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
&= \mathfrak{A}^{A}_{\underline{B}}(v \mapsto [\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2, y \mapsto v]) \\
&= \mathfrak{A}^{A}_{\underline{B}}(v \mapsto [\![M]\!]\rho[x \mapsto (v_1, v_2), y \mapsto v]) \\
&= [\![\exists y : A.\, M]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

Therefore the result holds in this case.

CASE(UNIFY). Suppose the derivation of M takes the shape:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} V_1 : A} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{v}} V_2 : A} \qquad A \in \mathsf{FO} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{c}} M : \underline{B}}}{\Gamma, x : A_1 \times A_2 \vdash^{\mathsf{c}} V_1 =_A V_2;\ M : \underline{B}}\ \text{UNIFY}$$

Now by definition, we have that the following substitution is:

$$(V_1 =_A V_2;\ M)[(x_1, x_2)/x] = V_1 =_A V_2;\ M[(x_1, x_2)/x]$$

Additionally, by the induction hypothesis, we have that:

$$\begin{aligned}
[\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![M]\!]\rho[x \mapsto (v_1, v_2)] \\
[\![V_1[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![V_1]\!]\rho[x \mapsto (v_1, v_2)] \\
[\![V_2[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] &= [\![V_2]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

We now use the above alongside the definition of the denotational semantics to see that:

$$\begin{aligned}
&[\![(V_1 =_A V_2;\ M)[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
=&[\![V_1[(x_1, x_2)/x] =_A V_2[(x_1, x_2)/x];\ M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \\
=&\mathsf{eq}^{A}_{\underline{B}}([\![V_1]\!]\rho[x \mapsto (v_1, v_2)], [\![V_2]\!]\rho[x \mapsto (v_1, v_2)], [\![M[(x_1, x_2)/x]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]) \\
=&\mathsf{eq}^{A}_{\underline{B}}([\![V_1]\!]\rho[x \mapsto (v_1, v_2)], [\![V_2]\!]\rho[x \mapsto (v_1, v_2)], [\![M]\!]\rho[x \mapsto (v_1, v_2)]) \\
=&[\![V_1 =_A V_2;\ M]\!]\rho[x \mapsto (v_1, v_2)]
\end{aligned}$$

□

For the following lemma we drop our notational abuse of computation subscripts for the operators of FLP algebras and monad algebras to ensure clarity.

Lemma 5.1.3. *Suppose a monad T with an FLP structure $(\varepsilon, \vee, \mathfrak{A}, \mathsf{eq})$ that gives an FLP algebra over the carrier TX for every set X. Then, given a monad algebra $(\mathfrak{X}_{\underline{B}}, \alpha_{\mathfrak{X}_{\underline{B}}} : T(\mathfrak{X}_{\underline{B}}) \to \mathfrak{X}_{\underline{B}})$ for each computation type as defined in §4.3.2, the following hold:*

$$\begin{aligned}
\alpha_{\mathfrak{X}_{\underline{B}}}(\varepsilon_{T(\mathfrak{X}_{\underline{B}})}) &= \varepsilon_{\mathfrak{X}_{\underline{B}}} \\
\alpha_{\mathfrak{X}_{\underline{B}}}(xm \vee_{T(\mathfrak{X}_{\underline{B}})} ym) &= \alpha_{\mathfrak{X}_{\underline{B}}}(xm) \vee_{\mathfrak{X}_{\underline{B}}} \alpha_{\mathfrak{X}_{\underline{B}}}(ym) \\
\alpha_{\mathfrak{X}_{\underline{B}}}\left(\mathfrak{A}_{T(\mathfrak{X}_{\underline{B}})} f\right) &= \mathfrak{A}_{\mathfrak{X}_{\underline{B}}}(\alpha_{\mathfrak{X}_{\underline{B}}} \circ f) \\
\alpha_{\mathfrak{X}_{\underline{B}}}(\mathsf{eq}_{T(\mathfrak{X}_{\underline{B}})}(v, w, xm)) &= \mathsf{eq}_{\mathfrak{X}_{\underline{B}}}(v, w, \alpha_{\mathfrak{X}_{\underline{B}}}(xm))
\end{aligned}$$

Proof. We prove the properties by induction on the computation type. We start with the property $\alpha_{\mathfrak{X}_{\underline{B}}}(\varepsilon_{T(\mathfrak{X}_{\underline{B}})}) = \varepsilon_{\mathfrak{X}_{\underline{B}}}$

CASE(FA). Suppose the computation type takes shape FA, and we recall that

$$[\![FA]\!] \overset{\text{def}}{=} \left(T[\![A]\!], \mu_{T[\![A]\!]}, \varepsilon_{T[\![A]\!]}, \vee_{T[\![A]\!]}, \mathfrak{A}_{T[\![A]\!]}, \mathsf{eq}_{T[\![A]\!]}\right)$$

In particular we have that $\mu_{T[\![A]\!]}(xmm) = xmm \gg\!= id_{T[\![A]\!]}$. So in this case we see that:

$$\alpha_{\mathfrak{X}_{FA}}(\varepsilon_{T(\mathfrak{X}_{FA})}) = \mu_{T[\![A]\!]}(\varepsilon_{TT[\![A]\!]})$$

$$\begin{aligned} &= \varepsilon_{TT[\![A]\!]} \gg= id_{T[\![A]\!]} \\ &= \varepsilon_{T[\![A]\!]} \quad \text{by FLP structure Eq. (4.10)} \end{aligned}$$

So the property holds in this case.

CASE($A \to \underline{B}$). Suppose the computation type takes shape $A \to \underline{B}$. Then we have by definition that

$$\alpha_{\mathfrak{X}_{A \to \underline{B}}}(xm) = a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(xm \gg= (f \mapsto \eta(f(a))))$$

So in this case we see that:

$$\begin{aligned} \alpha_{\mathfrak{X}_{A \to \underline{B}}}(\varepsilon_{T(\mathfrak{X}_{A \to \underline{B}})}) &= a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(\varepsilon_{T(\mathfrak{X}_{A \to \underline{B}})} \gg= \underbrace{(f \mapsto \eta(f(a))))}_{\mathfrak{X}_{A \to \underline{B}} \to T(\mathfrak{X}_{\underline{B}})} \\ &= a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(\varepsilon_{T(\mathfrak{X}_{\underline{B}})}) \quad \text{by FLP structure Eq. (4.10)} \\ &= a \mapsto \varepsilon_{\mathfrak{X}_{\underline{B}}} \quad \text{by the induction hypothesis} \\ &= \varepsilon_{\mathfrak{X}_{A \to \underline{B}}} \quad \text{by definition of } [\![A \to \underline{B}]\!] \text{ in §4.3.2} \end{aligned}$$

So the property holds in this case, and thus all computation types by induction.

We now prove the property that $\alpha_{\mathfrak{X}_{\underline{B}}}(xm \vee_{T(\mathfrak{X}_{\underline{B}})} ym) = \alpha_{\mathfrak{X}_{\underline{B}}}(xm) \vee_{\mathfrak{X}_{\underline{B}}} \alpha_{\mathfrak{X}_{\underline{B}}}(ym)$.

CASE(FA). Suppose the computation type takes shape FA. In this case, we see that

$$\begin{aligned} \alpha_{\mathfrak{X}_{FA}}(xm \vee_{T(\mathfrak{X}_{FA})} ym) &= \mu_{T[\![A]\!]}(xm \vee_{TT[\![A]\!]} ym) \\ &= (xm \vee_{TT[\![A]\!]} ym) \gg= id_{T[\![A]\!]} \\ &= (xm \gg= id_{T[\![A]\!]}) \vee_{T[\![A]\!]} (ym \gg= id_{T[\![A]\!]}) \quad \text{by FLP structure Eq. (4.11)} \\ &= \mu_{T[\![A]\!]}(xm) \vee_{T[\![A]\!]} \mu_{T[\![A]\!]}(ym) \end{aligned}$$

So the second property holds in this case.

CASE($A \to \underline{B}$). Suppose the type takes shape $A \to \underline{B}$. Then we have by definition that

$$\alpha_{\mathfrak{X}_{A \to \underline{B}}}(xm) = a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(xm \gg= (f \mapsto \eta(f(a))))$$

So in this case we see that:

$$\begin{aligned} \alpha_{\mathfrak{X}_{A \to \underline{B}}}(xm \vee_{T(\mathfrak{X}_{A \to \underline{B}})} ym) &= a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}((xm \vee_{T(\mathfrak{X}_{A \to \underline{B}})} ym) \gg= \underbrace{(f \mapsto \eta(f(a))))}_{\mathfrak{X}_{A \to \underline{B}} \to T(\mathfrak{X}_{\underline{B}})} \\ &= a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(xm \gg= (f \mapsto \eta(f(a))) \vee_{T(\mathfrak{X}_{\underline{B}})} ym \gg= (f \mapsto \eta(f(a)))) \text{ by Eq. (4.11)} \\ &= a \mapsto \alpha_{\underline{B}}(xm \gg= (f \mapsto \eta(f(a)))) \vee_{\mathfrak{X}_{\underline{B}}} \alpha_{\underline{B}}(ym \gg= (f \mapsto \eta(f(a)))) \text{ by induction hypothesis} \\ &= a \mapsto \alpha_{\mathfrak{X}_{A \to \underline{B}}}(xm)(a) \vee_{\mathfrak{X}_{\underline{B}}} \alpha_{\mathfrak{X}_{A \to \underline{B}}}(ym)(a) \\ &= \alpha_{\mathfrak{X}_{A \to \underline{B}}}(xm) \vee_{\mathfrak{X}_{A \to \underline{B}}} \alpha_{\mathfrak{X}_{A \to \underline{B}}}(ym) \text{ by definition of } [\![A \to \underline{B}]\!] \text{ in §4.3.2} \end{aligned}$$

So the property holds in this case, and thus for all computation types by induction.

We now prove the third property that $\alpha_{\mathfrak{X}_{\underline{B}}}\left(\mathbb{\exists}_{T(\mathfrak{X}_{\underline{B}})} x \mapsto f(x)\right) = \mathbb{\exists}_{\mathfrak{X}_{\underline{B}}} x \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(f(x))$.

CASE(FA). Suppose the computation type takes shape FA. In this case, we see that

$$\begin{aligned} \alpha_{\mathfrak{X}_{FA}}\left(\mathbb{\exists}_{T(\mathfrak{X}_{FA})}(x \mapsto f(x))\right) &= \left(\mathbb{\exists}_{TT[\![A]\!]}(x \mapsto f(x))\right) \gg= id_{T[\![A]\!]} \\ &= \mathbb{\exists}_{T[\![A]\!]}\left(x \mapsto f(x) \gg= id_{T[\![A]\!]}\right) \quad \text{by FLP structure Eq. (4.12)} \\ &= \mathbb{\exists}_{T[\![A]\!]}(x \mapsto \alpha_{\mathfrak{X}_{FA}}(f(x))) \end{aligned}$$

So the property holds in this case.

CASE($A \to \underline{B}$). Suppose the type takes shape $A \to \underline{B}$. Then we have by definition that

$$\alpha_{\mathfrak{X}_{A \to \underline{B}}}(fm) = a \mapsto \alpha_{\mathfrak{X}_{\underline{B}}}(fm \gg= (f \mapsto \eta(f(a))))$$

So in this case we can see that:

$$\begin{aligned}
\alpha_{\mathfrak{X}_{A\to\underline{B}}}\Big(\mathbb{\exists}_{T(\mathfrak{X}_{A\to\underline{B}})}(x\mapsto f(x))\Big) &= a\mapsto \alpha_{\mathfrak{X}_{\underline{B}}}\Big(\mathbb{\exists}_{T(\mathfrak{X}_{A\to\underline{B}})}(x\mapsto f(x)) \gg= \underbrace{(g\mapsto\eta(g(a)))}_{\mathfrak{X}_{A\to\underline{B}}\to T(\mathfrak{X}_{\underline{B}})}\Big)\\
&= a\mapsto \alpha_{\mathfrak{X}_{\underline{B}}}\Big(\mathbb{\exists}_{T(\mathfrak{X}_{\underline{B}})}(x\mapsto f(x)\gg=(g\mapsto\eta(g(a))))\Big) \quad \text{by FLP structure Eq. (4.1}\\
&= a\mapsto \mathbb{\exists}_{\mathfrak{X}_{\underline{B}}}(x\mapsto\alpha_{\mathfrak{X}_{\underline{B}}}(f(x)\gg=(g\mapsto\eta(g(a))))) \quad \text{by induction hypothesis}\\
&= a\mapsto \mathbb{\exists}_{\mathfrak{X}_{\underline{B}}}(x\mapsto\alpha_{\mathfrak{X}_{A\to\underline{B}}}(f(x))(a))\\
&= \mathbb{\exists}_{\mathfrak{X}_{A\to\underline{B}}}(\alpha_{\mathfrak{X}_{A\to\underline{B}}}\circ f) \quad \text{by definition of } [\![A\to\underline{B}]\!] \text{ in §4.3.2}
\end{aligned}$$

So the property holds in this case, and thus for all computation types by induction.

We now show the fourth property that $\alpha_{\mathfrak{X}_{\underline{B}}}(\mathsf{eq}_{T(\mathfrak{X}_{\underline{B}})}(v,w,xm)) = \mathsf{eq}_{\mathfrak{X}_{\underline{B}}}(v,w,\alpha_{\mathfrak{X}_{\underline{B}}}(xm))$, again working by induction on the computation type.

Case(FA). Suppose the computation type takes shape FA, then we have that:

$$\begin{aligned}
\alpha_{\mathfrak{X}_{FA}}(\mathsf{eq}_{T(\mathfrak{X}_{FA})}(v,w,xm)) &= \mathsf{eq}_{TT[\![A]\!]}(v,w,xm)\gg= id_{T[\![A]\!]}\\
&= \begin{cases} xm\gg= id_{T[\![A]\!]} & \text{if } v=w\\ \varepsilon_{TT[\![A]\!]}\gg= id_{T[\![A]\!]} & \text{if } v\neq w\end{cases} \quad \text{by Remark 4.1.6}\\
&= \begin{cases} xm\gg= id_{T[\![A]\!]} & \text{if } v=w\\ \varepsilon_{T[\![A]\!]} & \text{if } v\neq w\end{cases} \quad \text{by FLP structure Eq. (4.10)}\\
&= \mathsf{eq}_{T[\![A]\!]}(v,w,\alpha_{\mathfrak{X}_{FA}}(xm))
\end{aligned}$$

So the result holds for this case.

Case($A\to\underline{B}$). Suppose the computation type takes shape $A\to\underline{B}$, then we have that:

$$\begin{aligned}
\alpha_{\mathfrak{X}_{A\to\underline{B}}}(\mathsf{eq}_{T(\mathfrak{X}_{A\to\underline{B}})}(v,w,xm)) &= a\mapsto\alpha_{\mathfrak{X}_{\underline{B}}}(\mathsf{eq}_{T(\mathfrak{X}_{A\to\underline{B}})}(v,w,xm)\gg=\underbrace{(g\mapsto\eta(g(a))))}_{\mathfrak{X}_{A\to\underline{B}}\to T(\mathfrak{X}_{\underline{B}})} \text{ by definition of } \alpha_{\mathfrak{X}_{A\to\underline{B}}}\\
&= a\mapsto\alpha_{\mathfrak{X}_{\underline{B}}}(\mathsf{eq}_{T(\mathfrak{X}_{\underline{B}})}(v,w,xm\gg=(f\mapsto\eta(f(a))))) \text{ by FLP structure Eq. (4.13)}\\
&= a\mapsto\mathsf{eq}_{T(\mathfrak{X}_{\underline{B}})}(v,w,\alpha_{\mathfrak{X}_{\underline{B}}}(xm\gg=(f\mapsto\eta(f(a))))) \text{ by induction hypothesis}\\
&= a\mapsto\mathsf{eq}_{T(\mathfrak{X}_{\underline{B}})}(v,w,\alpha_{\mathfrak{X}_{A\to\underline{B}}}(xm)(a)) \text{ by definition of } \alpha_{\mathfrak{X}_{A\to\underline{B}}}\\
&= \begin{cases} a\mapsto\alpha_{\mathfrak{X}_{A\to\underline{B}}}(xm)(a) & \text{if } v=w\\ a\mapsto\varepsilon_{T(\mathfrak{X}_{\underline{B}})} & \text{if } v\neq w\end{cases} \quad \text{by Remark 4.1.6}\\
&= \begin{cases} a\mapsto\alpha_{\mathfrak{X}_{A\to\underline{B}}}(xm)(a) & \text{if } v=w\\ \varepsilon_{T(\mathfrak{X}_{A\to\underline{B}})} & \text{if } v\neq w\end{cases} \quad \text{by definition of } [\![A\to\underline{B}]\!] \text{ in §4.3.2}\\
&= \mathsf{eq}_{T(\mathfrak{X}_{A\to\underline{B}})}(v,w,\alpha_{\mathfrak{X}_{A\to\underline{B}}}(xm)) \text{ by definition of } [\![A\to\underline{B}]\!] \text{ in §4.3.2}
\end{aligned}$$

Hence the fourth property holds for all computation types.

We have now shown all the properties hold for all computation types, so the lemma holds. □

5.2 Main Proof

We now have all the pieces we need in order to prove the soundness of the denotational semantics of the FLP language.

Theorem 5.2.1 (Soundness of Denotational Semantics). *The denotational semantics of the FLP language are sound, so if* $\Gamma\vdash^{c} M = N : \underline{B}$ *then* $[\![M]\!] = [\![N]\!]$. *This means that if two terms are equated by the equational theory, then the denotations of the two terms are the same function.*

Proof. Suppose some environment ρ. We work by induction on the derivation of $\Gamma\vdash^{c} M = N : \underline{B}$.

CASE(EQ-BETA-1). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \mathsf{return}\ V\ \mathsf{to}\ x.\, M = M[V/x] : \underline{B}}\ \text{EQ-BETA-1}$$

We must take care to treat the monadic effects correctly, the necessary definition from the denotational semantics is:

$$[\![M\ \mathsf{to}\ x.\, N]\!]\rho \overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))$$

So in this case we have:

$$[\![\mathsf{return}\ V\ \mathsf{to}\ x.\, M]\!] = \alpha_{\underline{B}}([\![\mathsf{return}\ V]\!]\rho \gg\!\!= (v \mapsto \eta([\![M]\!]\rho[x \mapsto v])))$$

Also by the definition of the denotational semantics for return terms we have:

$$[\![\mathsf{return}\ V]\!]\rho = \eta([\![V]\!]\rho)$$

So, now recalling the monad property that $\eta(x) \gg\!\!= f = f(x)$ we see:

$$\begin{aligned}
[\![\mathsf{return}\ V\ \mathsf{to}\ x.\, M]\!]\rho &= \alpha_{\underline{B}}(\eta([\![V]\!]\rho) \gg\!\!= (v \mapsto \eta([\![M]\!]\rho[x \mapsto v])))\\
&= \alpha_{\underline{B}}(\eta([\![M]\!]\rho[x \mapsto [\![V]\!]\rho]))\\
&= [\![M]\!]\rho[x \mapsto [\![V]\!]\rho]\\
&= [\![M[V/x]]\!]\rho \text{ by Substitution Lemma 5.1.1}
\end{aligned}$$

Where the penultimate line follows from the monad algebra property that $\alpha(\eta(x)) = x$. Hence, this case holds.

CASE(EQ-BETA-2). Assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \mathsf{force}\ (\mathsf{thunk}\ M) = M : \underline{B}}\ \text{EQ-BETA-2}$$

Then by the definitions of the denotational semantics, we have:

$$[\![\mathsf{force}\ (\mathsf{thunk}\ M)]\!] = [\![\mathsf{thunk}\ M]\!] = [\![M]\!]$$

So, the result holds in this case.

CASE(EQ-BETA-3). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V_1 : A_1} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} V_2 : A_2} \qquad \dfrac{\vdots}{\Gamma, x_1 : A_1, x_2 : A_2 \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \mathsf{split}((V_1, V_2); x_1, x_2.\, M) = M[V_1, V_2/x_1, x_2] : \underline{B}}\ \text{EQ-BETA-3}$$

Then by the definitions of the denotational semantics, we have:

$$[\![\mathsf{split}((V_1, V_2); x_1, x_2.\, M)]\!]\rho \overset{\text{def}}{=} [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\text{def}}{=} [\![(V_1, V_2)]\!]\rho$$

We also have that,

$$[\![(V_1, V_2)]\!]\rho \overset{\text{def}}{=} ([\![V_1]\!]\rho, [\![V_2]\!]\rho)$$

So combining the two equalities above we obtain:

$$\begin{aligned}
[\![\mathsf{split}((V_1, V_2); x_1, x_2.\, M)]\!]\rho &= [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } v_1 = [\![V_1]\!]\rho, v_2 = [\![V_2]\!]\rho\\
&= [\![M]\!]\rho[x_1 \mapsto [\![V_1]\!]\rho, x_2 \mapsto [\![V_2]\!]\rho]
\end{aligned}$$

Now by applying the Substitution Lemma 5.1.1 twice we see that:

$$[\![M[V_1, V_2/x_1, x_2]]\!]\rho = [\![M[V_2/x_2]]\!]\rho[x_1 \mapsto [\![V_1]\!]\rho] = [\![M]\!]\rho[x_1 \mapsto [\![V_1]\!]\rho, x_2 \mapsto [\![V_2]\!]\rho]$$

So the result holds in this case by seeing that:

$$\begin{aligned}
[\![\mathsf{split}((V_1, V_2); x_1, x_2.\, M)]\!]\rho &= [\![M]\!]\rho[x_1 \mapsto [\![V_1]\!]\rho, x_2 \mapsto [\![V_2]\!]\rho]\\
&= [\![M[V_1, V_2/x_1, x_2]]\!]\rho
\end{aligned}$$

Case(Eq-Beta-4). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : \underline{B}} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} V : A}}{\Gamma \vdash^{c} (\lambda x : A.\, M)(V) = M[V/x] : \underline{B}}\ \textsc{Eq-Beta-4}$$

Now by the definitions of the denotational semantics, we have:

$$\begin{aligned} [\![(\lambda x : A.\, M)(V)]\!]\rho &= [\![\lambda x : A.\, M]\!]\rho([\![V]\!]\rho) \\ &= f([\![V]\!]\rho) \text{ where } f(v) \overset{\text{def}}{=} [\![M]\!]\rho[x \mapsto v] \\ &= [\![M]\!]\rho[x \mapsto [\![V]\!]\rho] \\ &= [\![M[V/x]]\!]\rho \end{aligned}$$

The last equality follows from the Substitution Lemma 5.1.1. So the result holds in this case.

Case(Eq-Beta-5). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : \mathsf{Num} \vdash^{c} N : \underline{B}}}{\Gamma \vdash^{c} \mathsf{ifz}(\overline{0}; M; x.\, N) = M : \underline{B}}\ \textsc{Eq-Beta-5}$$

Now by the definitions of the denotational semantics, we have:

$$[\![\mathsf{ifz}(n; M; x.\, N)]\!]\rho \overset{\text{def}}{=} \begin{cases} [\![M]\!]\rho & \text{if } [\![\overline{n}]\!]\rho = 0 \\ [\![N]\!]\rho[x \mapsto n] & \text{if } [\![\overline{n}]\!]\rho = n + 1 \end{cases}$$

So in this case where $[\![\overline{0}]\!]\rho = 0$ the result is immediate:

$$[\![\mathsf{ifz}(\overline{0}; M; x.\, N)]\!]\rho = [\![M]\!]\rho$$

Case(Eq-Beta-6). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : \mathsf{Num} \vdash^{c} N : \underline{B}}}{\Gamma \vdash^{c} \mathsf{ifz}(\overline{n + 1}; M; x.\, N) = N[\overline{n}/x] : \underline{B}}\ \textsc{Eq-Beta-5}$$

As in the previous case we use the definition from the denotational semantics:

$$[\![\mathsf{ifz}(n; M; x.\, N)]\!]\rho \overset{\text{def}}{=} \begin{cases} [\![M]\!]\rho & \text{if } [\![\overline{n}]\!]\rho = 0 \\ [\![N]\!]\rho[x \mapsto n] & \text{if } [\![\overline{n}]\!]\rho = n + 1 \end{cases}$$

So in this case we have:

$$\begin{aligned} [\![\mathsf{ifz}(\overline{n + 1}; M; x.\, N)]\!]\rho &= [\![N]\!]\rho[x \mapsto n] \\ &= [\![N[\overline{n}/x]]\!]\rho \end{aligned}$$

Where the last equality follows from the Substitution Lemma 5.1.1. So the result holds in this case.

Case(Eq-Eta). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A_1 \times A_2} \qquad \dfrac{\vdots}{\Gamma, p : A_1 \times A_2 \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} M[V/p] = \mathsf{split}(V; x_1, x_2.\, M[(x_1, x_2)/p]) : \underline{B}}\ \textsc{Eq-Eta}$$

By the definitions of the denotational semantics, we have:

$$[\![\mathsf{split}(V; x_1, x_2.\, M[(x_1, x_2)/p])]\!]\rho = [\![M[(x_1, x_2)/p]]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\text{def}}{=} [\![V]\!]\rho$$

$$\begin{aligned} &= [\![M]\!]\rho[p \mapsto [\![V]\!]\rho] \\ &= [\![M[V/p]]\!]\rho \end{aligned}$$

Where the second equality follows directly from the Lemma 5.1.2 which we proved in the previous section. The final equality is due to Substitution Lemma 5.1.1. Hence the result holds in this case.

CASE(EQ-SEQ-1). We assume the derivation of $\Gamma \vdash^c M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^c M : FA}}{\Gamma \vdash^c M = M \text{ to } x.\, \text{return } x : FA} \;\textsc{Eq-Seq-1}$$

We make use of the following definitions in the denotational semantics:

$$\begin{aligned} [\![M \text{ to } x.\, N]\!]\rho &\overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \\ [\![\text{return } x]\!]\rho &\overset{\text{def}}{=} \eta([\![x]\!]\rho) \\ [\![x]\!]\rho &\overset{\text{def}}{=} \rho(x) \end{aligned}$$

We see that:

$$[\![\text{return } x]\!]\rho[x \mapsto y] = \eta([\![x]\!]\rho[x \mapsto y]) = \eta(\rho[x \mapsto y](x)) = \eta(y)$$

Then using this and the above definitions we observe:

$$\begin{aligned} [\![M \text{ to } x.\, \text{return } x]\!]\rho &= \alpha_{FA}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![\text{return } x]\!]\rho[x \mapsto v]))) \\ &= \mu_{T[\![A]\!]}([\![M]\!]\rho \gg\!= (v \mapsto \eta_{T[\![A]\!]}(\eta_{[\![A]\!]}(v)))) \end{aligned}$$

Now we apply the definition of the multiplication operation $\mu_{T[\![A]\!]}(xmm) \overset{\text{def}}{=} xmm \gg\!= id_{T[\![A]\!]}$, and the associativity of bind to see that:

$$\begin{aligned} [\![M \text{ to } x.\, \text{return } x]\!]\rho &= \mu_{T[\![A]\!]}([\![M]\!]\rho \gg\!= \eta_{T[\![A]\!]} \circ \eta_{[\![A]\!]}) \\ &= ([\![M]\!]\rho \gg\!= \eta_{T[\![A]\!]} \circ \eta_{[\![A]\!]}) \gg\!= id_{T[\![A]\!]} \\ &= [\![M]\!]\rho \gg\!= (v \mapsto \eta_{T[\![A]\!]}(\eta_{[\![A]\!]}(v)) \gg\!= id_{T[\![A]\!]}) \\ &= [\![M]\!]\rho \gg\!= (v \mapsto id_{T[\![A]\!]}(\eta_{[\![A]\!]}(v))) \\ &= [\![M]\!]\rho \gg\!= \eta_{[\![A]\!]} \\ &= [\![M]\!]\rho \end{aligned}$$

Where the antepenultimate and final lines follow from the following laws for monads:

$$\begin{aligned} \eta(x) \gg\!= f &= f(x) \\ xm \gg\!= \eta &= xm \end{aligned}$$

So the result holds in this case.

CASE(EQ-SEQ-2). We assume the derivation of $\Gamma \vdash^c M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^c M : FA_1} \qquad \dfrac{\vdots}{\Gamma, x_1 : A_1 \vdash^c N : FA_2} \qquad \dfrac{\vdots}{\Gamma, x_2 : A_2 \vdash^c P : \underline{B}}}{\Gamma \vdash^c (M \text{ to } x_1.\, N) \text{ to } x_2.\, P = M \text{ to } x_1.\, (N \text{ to } x_2.\, P) : \underline{B}} \;\textsc{Eq-Seq-2}$$

Now by the definitions in the denotational semantics, we know:

$$[\![M \text{ to } x.\, N]\!]\rho \overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))$$

For conciseness let us define

$$f : [\![A_2]\!] \to T[\![\underline{B}]\!] \text{ by } f(v_2) = \eta([\![P]\!]\rho[x_2 \mapsto v_2])$$

$$g : [\![A_1]\!] \to T[\![A_2]\!] \text{ by } g(v_1) = [\![N]\!]\rho[x_1 \mapsto v_1]$$

Then in this case we see:

$$\begin{aligned}
[\![(M \text{ to } x_1.\, N) \text{ to } x_2.\, P]\!]\rho &= \alpha_{\underline{B}}([\![M \text{ to } x_1.\, N]\!]\rho \gg= (v_2 \mapsto \eta([\![P]\!]\rho[x_2 \mapsto v_2]))) \\
&= \alpha_{\underline{B}}(\alpha_{FA_2}([\![M]\!]\rho \gg= (v_1 \mapsto \eta([\![N]\!]\rho[x_1 \mapsto v_1]))) \gg= (v_2 \mapsto \eta([\![P]\!]\rho[x_2 \mapsto v_2]))) \\
&= \alpha_{\underline{B}}\Big(\mu_{[\![A_2]\!]}([\![M]\!]\rho \gg= (v_1 \mapsto \eta(g(v_1)))) \gg= f\Big) \\
&= \alpha_{\underline{B}}\Big(\mu_{[\![A_2]\!]}([\![M]\!]\rho \gg= \eta \circ g) \gg= f\Big) \\
&= \alpha_{\underline{B}}\Big(([\![M]\!]\rho \gg= \eta \circ g) \gg= id_{T[\![A_2]\!]} \gg= f\Big) \\
&= \alpha_{\underline{B}}\Big(([\![M]\!]\rho \gg= (v_1 \mapsto \eta(g(v_1)) \gg= id_{T[\![A_2]\!]})) \gg= f\Big) \quad \text{by bind associativity} \\
&= \alpha_{\underline{B}}\Big(([\![M]\!]\rho \gg= (v_1 \mapsto id_{T[\![A_2]\!]}(g(v_1)))) \gg= f\Big) \quad \text{by } \eta(x) \gg= f = f(x) \\
&= \alpha_{\underline{B}}(([\![M]\!]\rho \gg= g) \gg= f) \\
&= \alpha_{\underline{B}}\Big([\![M]\!]\rho \gg= \underbrace{(v_1 \mapsto g(v_1) \gg= f)}_{[\![A_1]\!] \to T[\![\underline{B}]\!]}\Big) \quad \text{by bind associativity}
\end{aligned}$$

Working from the other direction

$$\begin{aligned}
[\![M \text{ to } x_1.\, (N \text{ to } x_2.\, P)]\!]\rho &= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v_1 \mapsto \eta([\![N \text{ to } x_2.\, P]\!]\rho[x_1 \mapsto v_1]))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v_1 \mapsto \eta(\alpha_{\underline{B}}([\![N]\!]\rho[x_1 \mapsto v_1] \gg= (v_2 \mapsto \eta([\![P]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2])))))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v_1 \mapsto \eta(\alpha_{\underline{B}}([\![N]\!]\rho[x_1 \mapsto v_1] \gg= (v_2 \mapsto \eta([\![P]\!]\rho[x_2 \mapsto v_2])))))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v_1 \mapsto \eta(\alpha_{\underline{B}}(g(v_1) \gg= f)))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v_1 \mapsto g(v_1) \gg= f))
\end{aligned}$$

We can assume by the Barendregt convention that x_1 is not free in P and x_2 is not free in N and P. This enables us to drop x_1 from the environment used to evaluate P in the third line. The final line follows from our proof of Theorem 4.3.4.

Hence, the result holds in this case.

CASE(EQ-SEQ-3). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M : FA_1} \qquad \dfrac{\vdots}{\Gamma, x : A_1 \vdash^{c} N : A_2 \to \underline{B}} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} V : A_2}}{\Gamma \vdash^{c} (M \text{ to } x.\, N)(V) = M \text{ to } x.\, N(V) : \underline{B}}\ \textsc{Eq-Seq-3}$$

We now turn to the definitions in the denotational semantics, recalling that

$$[\![M \text{ to } x.\, N]\!]\rho \overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))$$

Also, we recall the definition of algebras for function types:

$$\alpha_{A \to \underline{B}}(xm) = a \mapsto \alpha_{\underline{B}}(xm \gg= (f \mapsto \eta(f(a))))$$

Using these we see

$$\begin{aligned}
[\![(M \text{ to } x.\, N)(V)]\!]\rho &= ([\![M \text{ to } x.\, N]\!]\rho)([\![V]\!]\rho) \\
&= \alpha_{A_2 \to \underline{B}}([\![M]\!]\rho \gg= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))([\![V]\!]\rho) \\
&= (w \mapsto (\alpha_{\underline{B}}(([\![M]\!]\rho \gg= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \gg= (f \mapsto \eta(f(w))))))([\![V]\!]\rho) \\
&= \alpha_{\underline{B}}(([\![M]\!]\rho \gg= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \gg= (f \mapsto \eta(f([\![V]\!]\rho)))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v \mapsto (\eta([\![N]\!]\rho[x \mapsto v]) \gg= (f \mapsto \eta(f([\![V]\!]\rho)))))) \quad \text{by bind associativity} \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg= (v \mapsto (f \mapsto \eta(f([\![V]\!]\rho)))([\![N]\!]\rho[x \mapsto v]))) \quad \text{by } \eta(x) \gg= f = f(x)
\end{aligned}$$

$$
\begin{aligned}
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]([\![V]\!]\rho)))) \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]([\![V]\!]\rho[x \mapsto v])))) \quad \text{as } x \text{ not free in } V \\
&= \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N(V)]\!]\rho[x \mapsto v]))) \\
&= [\![M \ \mathsf{to}\ x.\, N(V)]\!]\rho
\end{aligned}
$$

Where the antepenultimate equality follows from the Barendregt convention that x is not free in V, meaning we can extend ρ with x mapping to anything. Hence the result holds in this case.

CASE(EQ-LAM-FAIL). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$
\frac{}{\Gamma \vdash^{c} \lambda x : A.\, \mathsf{fail} = \mathsf{fail} : \underline{B}}\ \textsc{Eq-Lam-Fail}
$$

We must take care in this instance to consider the types of the fail terms. First we construct the typing derivation for left-hand term $\lambda x : A.\, \mathsf{fail}$:

$$
\frac{\dfrac{}{\Gamma, x : A \vdash^{c} \mathsf{fail} : \underline{C}}\ \textsc{Fail}}{\Gamma \vdash^{c} \lambda x : A.\, \mathsf{fail} : A \to \underline{C}}\ \textsc{Lam}
$$

So we see that $\underline{B} = A \to \underline{C}$. Now using the definitions of the denotational semantics we have:

$$
\begin{aligned}
\left[\!\!\left[\Gamma \vdash^{c} \lambda x : A.\, \mathsf{fail} : A \to \underline{C}\right]\!\!\right]\rho &= (v \mapsto \left[\!\!\left[\Gamma \vdash^{c} \mathsf{fail} : \underline{C}\right]\!\!\right]\rho[x \mapsto v]) \\
&= (v \mapsto \varepsilon_{\underline{C}})
\end{aligned}
$$

Now looking at the right-hand term fail we have:

$$
\left[\!\!\left[\Gamma \vdash^{c} \mathsf{fail} : A \to \underline{C}\right]\!\!\right]\rho = \varepsilon_{A \to \underline{C}} = (v \mapsto \varepsilon_{\underline{C}})
$$

Where we have recalled from the definitions for the FLP algebras that $\varepsilon_{A \to \underline{B}}(a) \overset{\text{def}}{=} \varepsilon_{\underline{B}}$. So the result holds in this case.

CASE(EQ-LAM-CHOICE). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$
\frac{\dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : \underline{B}} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{c} N : \underline{B}}}{\Gamma \vdash^{c} \lambda x : A.\, (M \talloblong N) = \lambda x : A.\, M \talloblong \lambda x : A.\, N : A \to \underline{B}}\ \textsc{Eq-Lam-Choice}
$$

Then by the definitions in the denotational semantics we have:

$$
\begin{aligned}
[\![\lambda x : A.\, (M \talloblong N)]\!]\rho &= v \mapsto [\![M \talloblong N]\!]\rho[x \mapsto v] \\
&= v \mapsto ([\![M]\!]\rho[x \mapsto v] \vee_{\underline{B}} [\![N]\!]\rho[x \mapsto v])
\end{aligned}
$$

Working from the other direction we see:

$$
\begin{aligned}
[\![\lambda x : A.\, M \talloblong \lambda x : A.\, N]\!]\rho &= [\![\lambda x : A.\, M]\!]\rho \vee_{A \to \underline{B}} [\![\lambda x : A.\, N]\!]\rho \\
&= (v \mapsto [\![M]\!]\rho[x \mapsto v]) \vee_{A \to \underline{B}} (v \mapsto [\![N]\!]\rho[x \mapsto v])
\end{aligned}
$$

Now we recall that $(f \vee_{A \to \underline{B}} g)(a) \overset{\text{def}}{=} f(a) \vee_{\underline{B}} g(a)$, using this we can reach the result.

$$
\begin{aligned}
[\![\lambda x : A.\, (M \talloblong N)]\!]\rho &= v \mapsto ([\![M]\!]\rho[x \mapsto v] \vee_{\underline{B}} [\![N]\!]\rho[x \mapsto v]) \\
&= v \mapsto [\![M]\!]\rho[x \mapsto v] \vee_{A \to \underline{B}} v \mapsto [\![N]\!]\rho[x \mapsto v] \\
&= [\![\lambda x : A.\, M \talloblong \lambda x : A.\, N]\!]\rho
\end{aligned}
$$

So the result holds in this case.

CASE(EQ-LAM-EXISTS). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma, x_1 : A_1, x_2 : A_2 \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \lambda x_1 : A_1.\, (\exists x_2 : A_2.\, M) = \exists x_2 : A_2.\, (\lambda x_1 : A_1.\, M) : A_1 \to \underline{B}}\ \textsc{Eq-Lam-Exists}$$

Then by the definitions in the denotational semantics we have that $[\![\exists x : A.\, M]\!]\rho \overset{\text{def}}{=} \exists_{\underline{B}}^{A}(v \mapsto [\![M]\!]\rho[x \mapsto v])$, so in this case we see:

$$\begin{aligned}[\![\lambda x_1 : A_1.\, (\exists x_2 : A_2.\, M)]\!]\rho &= v_1 \mapsto [\![\exists x_2 : A_2.\, M]\!]\rho[x_1 \mapsto v_1] \\ &= v_1 \mapsto \exists_{\underline{B}}^{A_2}(v_2 \mapsto [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2])\end{aligned}$$

Working from the other direction we see:

$$\begin{aligned}[\![\exists x_2 : A_2.\, (\lambda x_1 : A_1.\, M)]\!]\rho &= \exists_{A_1 \to \underline{B}}^{A_2}(v_2 \mapsto [\![\lambda x_1 : A_1.\, M]\!]\rho[x_2 \mapsto v_2]) \\ &= \exists_{A_1 \to \underline{B}}^{A_2}(v_2 \mapsto (v_1 \mapsto [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]))\end{aligned}$$

We recall that $\left(\exists_{A \to \underline{B}}^{A'} f\right)(a) \overset{\text{def}}{=} \exists_{\underline{B}}^{A'}(V' \mapsto f(V')(a))$ and so it's clear to see that:

$$\begin{aligned}[\![\exists x_2 : A_2.\, (\lambda x_1 : A_1.\, M)]\!]\rho &= \exists_{A_1 \to \underline{B}}^{A_2}(v_2 \mapsto (v_1 \mapsto [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2])) \\ &= v_1 \mapsto \exists_{\underline{B}}^{A_2}(v_2 \mapsto [\![M]\!]\rho[x_1 \mapsto v_1, x_2 \mapsto v_2]) \\ &= [\![\lambda x_1 : A_1.\, (\exists x_2 : A_2.\, M)]\!]\rho\end{aligned}$$

So the result holds in this case.

CASE(EQ-LAM-UNIFY). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V_1 : A_1} \quad \dfrac{\vdots}{\Gamma \vdash^{v} V_2 : A_1} \quad \dfrac{\vdots}{\Gamma, x : A_2 \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \lambda x : A_2.\, (V_1 \doteq V_2;\ M) = V_1 \doteq V_2;\ \lambda x : A_2.\, M : A_2 \to \underline{B}}\ \textsc{Eq-Lam-Unify}$$

Then we work from the definitions in the denotational semantics, specifically recalling that

$$[\![V_1 \doteq_A V_2;\ M]\!]\rho \overset{\text{def}}{=} \mathsf{eq}_{\underline{B}}^{A}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![M]\!]\rho)$$

So we have in this case:

$$\begin{aligned}[\![\lambda x : A_2.\, (V_1 \doteq V_2;\ M)]\!]\rho &= v \mapsto [\![V_1 \doteq V_2;\ M]\!]\rho[x \mapsto v] \\ &= v \mapsto \mathsf{eq}_{\underline{B}}^{A_1}([\![V_1]\!]\rho[x \mapsto v], [\![V_2]\!]\rho[x \mapsto v], [\![M]\!]\rho[x \mapsto v]) \\ &= v \mapsto \mathsf{eq}_{\underline{B}}^{A_1}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![M]\!]\rho[x \mapsto v])\end{aligned}$$

Where the last equality follows by the Barendregt convention allowing us to assume x is not free in V_1 or V_2. Working from the other direction we see that:

$$\begin{aligned}[\![V_1 \doteq V_2;\ \lambda x : A_2.\, M]\!]\rho &= \mathsf{eq}_{A_2 \to \underline{B}}^{A_1}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![\lambda x : A_2.\, M]\!]\rho) \\ &= \mathsf{eq}_{A_2 \to \underline{B}}^{A_1}([\![V_1]\!]\rho, [\![V_2]\!]\rho, (v \mapsto [\![M]\!]\rho[x \mapsto v]))\end{aligned}$$

Now we know that

$$\mathsf{eq}_{A \to \underline{B}}^{A'}(w, v, f)(a) \overset{\text{def}}{=} \begin{cases} f(a) & \text{if } v = w \\ \varepsilon_{A \to \underline{B}} & \text{if } v \neq w \end{cases}$$

So we consider each case in turn. Suppose $[\![V_1]\!]\rho = [\![V_2]\!]\rho$, then we have by the above and Remark 4.1.6 that:

$$[\![V_1 \doteq V_2;\ \lambda x : A_2.\, M]\!]\rho = \mathsf{eq}_{A_2 \to \underline{B}}^{A_1}([\![V_1]\!]\rho, [\![V_2]\!]\rho, (v \mapsto [\![M]\!]\rho[x \mapsto v]))$$

$$\begin{aligned}
&= v \mapsto [\![M]\!]\rho[x \mapsto v] \\
&= v \mapsto \mathsf{eq}^{A_1}_{\underline{B}}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![M]\!]\rho[x \mapsto v]) \\
&= [\![\lambda x : A_2.\, (V_1 \doteq V_2;\ M)]\!]\rho
\end{aligned}$$

Now suppose $[\![V_1]\!]\rho \neq [\![V_2]\!]\rho$ then using $\varepsilon_{A \to \underline{B}}(a) \overset{\text{def}}{=} \varepsilon_{\underline{B}}$ and Remark 4.1.6 we see:

$$\begin{aligned}
[\![V_1 \doteq V_2;\ \lambda x : A_2.\, M]\!]\rho &= \mathsf{eq}^{A_1}_{A_2 \to \underline{B}}([\![V_1]\!]\rho, [\![V_2]\!]\rho, (v \mapsto [\![M]\!]\rho[x \mapsto v])) \\
&= \varepsilon_{A \to \underline{B}} \\
&= v \mapsto \varepsilon_{\underline{B}} \\
&= v \mapsto \mathsf{eq}^{A_1}_{\underline{B}}([\![V_1]\!]\rho, [\![V_2]\!]\rho, [\![M]\!]\rho[x \mapsto v]) \\
&= [\![\lambda x : A_2.\, (V_1 \doteq V_2;\ M)]\!]\rho
\end{aligned}$$

So the result holds in this case, no matter what V_1 and V_2 are.

CASE(EQ-CHOICE-ASSOC). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M_1 : \underline{B}} \qquad \dfrac{\vdots}{\Gamma \vdash^{c} M_2 : \underline{B}} \qquad \dfrac{\vdots}{\Gamma \vdash^{c} M_3 : \underline{B}}}{\Gamma \vdash^{c} M_1 \mathbin{[\!]} (M_2 \mathbin{[\!]} M_3) = (M_1 \mathbin{[\!]} M_2) \mathbin{[\!]} M_3 : \underline{B}} \ \text{EQ-CHOICE-ASSOC}$$

Then we have by the definition of the denotational semantics, and associativity of choice in Definition 4.1.5 that:

$$\begin{aligned}
[\![M_1 \mathbin{[\!]} (M_2 \mathbin{[\!]} M_3)]\!]\rho &= [\![M_1]\!]\rho \vee_{\underline{B}} [\![M_2 \mathbin{[\!]} M_3]\!]\rho \\
&= [\![M_1]\!]\rho \vee_{\underline{B}} ([\![M_2]\!]\rho \vee_{\underline{B}} [\![M_3]\!]\rho) \\
&= ([\![M_1]\!]\rho \vee_{\underline{B}} [\![M_2]\!]\rho) \vee_{\underline{B}} [\![M_3]\!]\rho \quad \text{by bind associativity} \\
&= [\![M_1 \mathbin{[\!]} M_2]\!]\rho \vee_{\underline{B}} [\![M_3]\!]\rho \\
&= [\![(M_1 \mathbin{[\!]} M_2) \mathbin{[\!]} M_3]\!]\rho
\end{aligned}$$

So the result holds in this case.

CASE(EQ-CHOICE-UNIT). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} M \mathbin{[\!]} \mathsf{fail} = \mathsf{fail} \mathbin{[\!]} M : \underline{B}} \ \text{EQ-CHOICE-UNIT}$$

Then by using the definitions of the denotational semantics and properties of FLP algebras defined in Definition 4.1.5 we have:

$$\begin{aligned}
[\![M \mathbin{[\!]} \mathsf{fail}]\!]\rho &= [\![M]\!]\rho \vee_{\underline{B}} [\![\mathsf{fail}]\!]\rho \\
&= [\![M]\!]\rho \vee_{\underline{B}} \varepsilon_{\underline{B}} \\
&= [\![M]\!]\rho \\
&= \varepsilon_{\underline{B}} \vee_{\underline{B}} [\![M]\!]\rho \\
&= [\![\mathsf{fail} \mathbin{[\!]} M]\!]\rho
\end{aligned}$$

So the result holds in this case.

CASE(EQ-UNIFY-REFL). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A} \qquad \dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} V \doteq V;\ M = M : \underline{B}} \ \text{EQ-UNIFY-REFL}$$

Now the result directly follows from the definition of the denotational semantics and Remark 4.1.6.

$$\begin{aligned}[\![V \doteq V;\ M]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![V]\!]\rho, [\![M]\!]\rho) \\ &= [\![M]\!]\rho \text{ as } [\![V]\!]\rho = [\![V]\!]\rho\end{aligned}$$

So the result holds in this case.

Case(Eq-Unify-Subst). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} W : A} \qquad \dfrac{\vdots}{\Gamma, x : A \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} V \doteq W;\ M[V/x] = V \doteq W;\ M[W/x] : \underline{B}}\ \textsc{Eq-Unify-Subst}$$

Now we work from the definitions in the denotational semantics to see:

$$[\![V \doteq W;\ M[V/x]]\!]\rho = \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M[V/x]]\!]\rho)$$

We use Remark 4.1.6 and consider the two cases. Suppose $[\![V]\!]\rho = [\![W]\!]\rho$. Then we have:

$$\begin{aligned}[\![V \doteq W;\ M[V/x]]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M[V/x]]\!]\rho) \\ &= [\![M[V/x]]\!]\rho \\ &= [\![M]\!]\rho[x \mapsto [\![V]\!]\rho] \text{ by Substitution Lemma 5.1.1} \\ &= [\![M]\!]\rho[x \mapsto [\![W]\!]\rho] \\ &= [\![M[W/x]]\!]\rho \\ &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M[W/x]]\!]\rho) \\ &= [\![V \doteq W;\ M[W/x]]\!]\rho\end{aligned}$$

Suppose $[\![V]\!]\rho \neq [\![W]\!]\rho$. Then we have:

$$\begin{aligned}[\![V \doteq W;\ M[V/x]]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M[V/x]]\!]\rho) \\ &= \varepsilon_{\underline{B}} \\ &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M[W/x]]\!]\rho) \\ &= [\![V \doteq W;\ M[W/x]]\!]\rho\end{aligned}$$

So, overall this case holds.

Case(Eq-Unify-Comm). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V_1 : A_1} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} W_1 : A_1} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} V_2 : A_2} \qquad \dfrac{\vdots}{\Gamma \vdash^{v} W_2 : A_2} \qquad \dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} V_1 \doteq W_1;\ V_2 \doteq W_2;\ M = V_2 \doteq W_2;\ V_1 \doteq W_1;\ M : \underline{B}}\ \textsc{Eq-Unify-Comm}$$

Now we work from the definitions in the denotational semantics to see:

$$\begin{aligned}[\![V_1 \doteq W_1;\ V_2 \doteq W_2;\ M]\!]\rho &= \mathsf{eq}^{A_1}_{\underline{B}}([\![V_1]\!]\rho, [\![W_1]\!]\rho, [\![V_2 \doteq W_2;\ M]\!]\rho) \\ &= \mathsf{eq}^{A_1}_{\underline{B}}([\![V_1]\!]\rho, [\![W_1]\!]\rho, \mathsf{eq}^{A_2}_{\underline{B}}([\![V_2]\!]\rho, [\![W_2]\!]\rho, [\![M]\!]\rho)) \\ &= \mathsf{eq}^{A_2}_{\underline{B}}([\![V_2]\!]\rho, [\![W_2]\!]\rho, \mathsf{eq}^{A_1}_{\underline{B}}([\![V_1]\!]\rho, [\![W_1]\!]\rho, [\![M]\!]\rho)) \\ &= \mathsf{eq}^{A_2}_{\underline{B}}([\![V_2]\!]\rho, [\![W_2]\!]\rho, [\![V_1 \doteq W_1;\ M]\!]\rho) \\ &= [\![V_2 \doteq W_2;\ V_1 \doteq W_1;\ M]\!]\rho\end{aligned}$$

Where the third equality follows from Definition 4.1.5, specifically the property that:

$$\mathsf{eq}(v_1, w_1, \mathsf{eq}(v_2, v_2, x)) = \mathsf{eq}(v_2, w_2, \mathsf{eq}(v_1, v_2, x))$$

So the result holds for this case.

CASE(EQ-UNIFY-CHOICE). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A} \quad \dfrac{\vdots}{\Gamma \vdash^{v} W : A} \quad \dfrac{\vdots}{\Gamma \vdash^{c} M : \underline{B}} \quad \dfrac{\vdots}{\Gamma \vdash^{c} N : \underline{B}}}{\Gamma \vdash^{c} V \doteq W;\ (M \mathbin{[\,]} N) = (V \doteq W;\ M) \mathbin{[\,]} (V \doteq W;\ N) : \underline{B}}\ \textsc{Eq-Unify-Choice}$$

Now we use the definitions in the denotational semantics and FLP algebra properties in Definition 4.1.5 to see:

$$\begin{aligned}
[\![V \doteq W;\ (M \mathbin{[\,]} N)]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![(M \mathbin{[\,]} N)]\!]\rho) \\
&= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M]\!]\rho \vee_{\underline{B}} [\![N]\!]\rho) \\
&= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![M]\!]\rho) \vee_{\underline{B}} \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![N]\!]\rho) \\
&= [\![V \doteq W;\ M]\!]\rho \vee_{\underline{B}} [\![V \doteq W;\ N]\!]\rho \\
&= [\![(V \doteq W;\ M) \mathbin{[\,]} (V \doteq W;\ N)]\!]\rho
\end{aligned}$$

So the result holds in this case.

CASE(EQ-UNIFY-FAIL). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{v} V : A} \quad \dfrac{\vdots}{\Gamma \vdash^{v} W : A}}{\Gamma \vdash^{c} V \doteq W;\ \mathsf{fail} = \mathsf{fail} : \underline{B}}\ \textsc{Eq-Unify-Fail}$$

Now it is straightforward to see the result by considering the two cases in Remark 4.1.6. Suppose $[\![V]\!]\rho = [\![W]\!]\rho$, then:

$$\begin{aligned}
[\![V \doteq W;\ \mathsf{fail}]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![\mathsf{fail}]\!]\rho) \\
&= [\![\mathsf{fail}]\!]\rho
\end{aligned}$$

In the case where $[\![V]\!]\rho \neq [\![W]\!]\rho$, we have:

$$\begin{aligned}
[\![V \doteq W;\ \mathsf{fail}]\!]\rho &= \mathsf{eq}^{A}_{\underline{B}}([\![V]\!]\rho, [\![W]\!]\rho, [\![\mathsf{fail}]\!]\rho) \\
&= \varepsilon_{\underline{B}} \\
&= [\![\mathsf{fail}]\!]\rho
\end{aligned}$$

So overall the result holds in this case.

CASE(EQ-BIND-FAIL). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$\dfrac{\dfrac{\vdots}{\Gamma, x : FA \vdash^{c} M : \underline{B}}}{\Gamma \vdash^{c} \mathsf{fail\ to}\ x.\, M = \mathsf{fail} : \underline{B}}\ \textsc{Eq-Bind-Fail}$$

Then by the definitions of the denotational semantics we know that

$$[\![M\ \mathsf{to}\ x.\, N]\!]\rho \overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))$$

$$\left[\!\!\left[\Gamma \vdash^{c} \mathsf{fail} : \underline{B}\right]\!\!\right]\rho = \varepsilon_{\underline{B}}$$

So in this case we see that

$$\begin{aligned}
[\![\mathsf{fail\ to}\ x.\, M]\!]\rho &= \alpha_{\underline{B}}\left(\left[\!\!\left[\Gamma \vdash^{c} \mathsf{fail} : FA\right]\!\!\right]\rho \gg\!= (v \mapsto \eta([\![M]\!]\rho[x \mapsto v]))\right) \\
&= \alpha_{\underline{B}}\left(\varepsilon_{T[\![A]\!]} \gg\!= \underbrace{(v \mapsto \eta([\![M]\!]\rho[x \mapsto v]))}_{[\![A]\!] \to T[\![\underline{B}]\!]}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \alpha_{\underline{B}}\Big(\varepsilon_{T[\![\underline{B}]\!]}\Big) && \text{by FLP structure laws} \\
&= \varepsilon_{\underline{B}} \\
&= [\![\mathsf{fail}]\!]\rho
\end{aligned}
$$

Where the penultimate line follows from Lemma 5.1.3, so the result holds in this case.

CASE(EQ-BIND-CHOICE). We assume the derivation of $\Gamma \vdash^{c} M = N : \underline{B}$ is of the form:

$$
\dfrac{\dfrac{\vdots}{\Gamma \vdash^{c} M_1 : FA} \qquad \dfrac{\vdots}{\Gamma \vdash^{c} M_2 : FA} \qquad \dfrac{\vdots}{\Gamma, x : FA \vdash^{c} N : \underline{B}}}{\Gamma \vdash^{c} (M_1 \mathbin{[\!]} M_2) \ \mathsf{to}\ x.\, N = M_1 \ \mathsf{to}\ x.\, N \mathbin{[\!]} M_2 \ \mathsf{to}\ x.\, N : \underline{B}} \ \textsc{Eq-Bind-Choice}
$$

Then by the definitions of the denotational semantics we know that

$$
[\![M \ \mathsf{to}\ x.\, N]\!]\rho \overset{\text{def}}{=} \alpha_{\underline{B}}([\![M]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))
$$

$$
[\![M \mathbin{[\!]} N]\!]\rho \overset{\text{def}}{=} [\![M]\!]\rho \vee_{\underline{B}} [\![N]\!]\rho
$$

We see that:

$$
\begin{aligned}
[\![(M_1 \mathbin{[\!]} M_2) \ \mathsf{to}\ x.\, N]\!]\rho &= \alpha_{\underline{B}}([\![M_1 \mathbin{[\!]} M_2]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \\
&= \alpha_{\underline{B}}\Big(([\![M_1]\!]\rho \vee_{T[\![A]\!]} [\![M_2]\!]\rho) \gg\!= \underbrace{(v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))}_{[\![A]\!] \to T[\![\underline{B}]\!]}\Big) \\
&= \alpha_{\underline{B}}\Big([\![M_1]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])) \vee_{T[\![\underline{B}]\!]} [\![M_2]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))\Big) \\
&= \alpha_{\underline{B}}([\![M_1]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \vee_{\underline{B}} \alpha_{\underline{B}}([\![M_2]\!]\rho \gg\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v]))) \\
&= [\![M_1 \ \mathsf{to}\ x.\, N]\!]\rho \vee_{\underline{B}} [\![M_2 \ \mathsf{to}\ x.\, N]\!]\rho \\
&= [\![M_1 \ \mathsf{to}\ x.\, N \mathbin{[\!]} M_2 \ \mathsf{to}\ x.\, N]\!]\rho
\end{aligned}
$$

Where the antepenultimate line follows from Lemma 5.1.3, so the result holds in this case.

□

Hence we have shown that the denotational semantics we define are sound with respect to the equational theory.

Chapter 6

Soundness of Operational Semantics

In this chapter, we will prove the soundness of the operational semantics defined in §4.2. This result tells us that if a closed term M evaluates to a monadic terminal xm, then the interpretation of these two objects through the denotational semantics is equal. This solidifies that the idealised operational semantics provides a valid interpretation of how programs in the FLP language behave in the sense that their mathematical interpretation is not altered by the evaluation relation.

The operational semantics provides an evaluation relation which maps closed computation terms to elements of the FLP algebra over monadic terminals. In order to prove a soundness result for the operational semantics, we want to interpret the codomain of the relation through our denotational semantics. To do this we define a semantic map whose domain is restricted to terminals. We can do this easily since terminals are closed terms and any closed term can be typed with an empty context and then interpreted through the denotational semantic map with an empty environment, $[\![-]\!]\emptyset$. The closed semantic map on terminals is defined by:

$$[\![-]\!]_t : \mathcal{T}_{\underline{B}} \to [\![\underline{B}]\!]$$
$$[\![M]\!]_t \overset{\text{def}}{=} \left[\!\!\left[\vdash^{\mathsf{c}} M : \underline{B}\right]\!\!\right]\emptyset$$

Then to consider monadic terminals we use the standard monadic extension of the above function as in Definition 4.3.2:

$$T[\![-]\!]_t : T(\mathcal{T}_{\underline{B}}) \to T[\![\underline{B}]\!]$$

$$T[\![xm]\!]_t \overset{\text{def}}{=} T([\![-]\!]\emptyset)(xm) = xm \gg\!= \eta \circ [\![-]\!]\emptyset$$

It is straightforward to interpret elements of $T[\![\underline{B}]\!]$ as elements of $[\![\underline{B}]\!]$ by the operation of the monad algebra, $\alpha_{\underline{B}}$. This is the final step in order to compare the monadic terminal to the closed computation which produced it.

We prove a short lemma on the bijectivity of the closed semantic map on first-order value terms, which will be useful in the proof of the soundness of the operational semantics.

Definition 6.0.1. The closed semantic map for first-order value terms is $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$, and is defined by $[\![V]\!]_V = [\![V]\!]\emptyset$.

Lemma 6.0.2. *The function $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$ is bijective for all $A \in \mathsf{FO}$.*

Proof. First we prove $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$ is injective. We work by induction on $A \in \mathsf{FO}$.

Case($A = \mathsf{Num}$). Suppose some $V, W \in \mathcal{V}_{\mathsf{Num}}$ such that

$$[\![V]\!]_V = [\![W]\!]_V = n \in \mathbb{N} = [\![\mathsf{Num}]\!]$$

Then by looking at the denotational semantics, the only way this is possible is if

$$[\![V]\!]_V = [\![W]\!]_V = [\![\overline{n}]\!]_V = n$$

Hence the only possible term is $V = W = \overline{n}$, so the result holds for $A = \mathsf{Num}$

CASE($A = A_1 \times A_2$). Suppose some $V, W \in \mathcal{V}_{A_1 \times A_2}$ such that

$$[\![V]\!]_V = [\![W]\!]_V = (v_1, v_2) \in [\![A_1 \times A_2]\!] = [\![A_1]\!] \times [\![A_2]\!]$$

Then by inspection of the denotational semantics, the only way this is possible is:

$$\begin{aligned}[\![V]\!]_V &= [\![(V_1, V_2)]\!]_V = ([\![V_1]\!]_V, [\![V_2]\!]_V) = (v_1, v_2)\\ [\![W]\!]_V &= [\![(W_1, W_2)]\!]_V = ([\![W_1]\!]_V, [\![W_2]\!]_V) = (v_1, v_2)\end{aligned}$$

This implies that $[\![V_1]\!]_V = [\![W_1]\!]_V = v_1$ and $[\![V_2]\!]_V = [\![W_2]\!]_V = v_2$. Then by the induction hypothesis we have that $V_1 = W_1$ and $V_2 = W_2$. Hence $V = (V_1, V_2) = (W_1, W_2) = W$, so the result holds for $A = A_1 \times A_2$.

Thus $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$ is injective for all $A \in \mathsf{FO}$.

We now show surjectivity, again by induction on $A \in \mathsf{FO}$.

CASE($A = \mathsf{Num}$). Suppose some $n \in [\![\mathsf{Num}]\!] = \mathbb{N}$. Then by inspection of the denotational semantics, the term $V = \overline{n} \in \mathcal{V}_{\mathsf{Num}}$ takes value n under the map $[\![\overline{n}]\!]_V = n \in \mathbb{N}$, so the result holds for $A = \mathsf{Num}$.

CASE($A = A_1 \times A_2$). Suppose some $(v_1, v_2) \in [\![A_1 \times A_2]\!] = [\![A_1]\!] \times [\![A_2]\!]$. Since $v_1 \in [\![A_1]\!]$ and $v_2 \in [\![A_2]\!]$, by the induction hypothesis there exists $V_1 \in \mathcal{V}_{A_1}$ and $V_2 \in \mathcal{V}_{A_2}$ such that $[\![V_1]\!]_V = v_1$ and $[\![V_2]\!]_V = v_2$. Then by inspection of the denotational semantics, the term $V = (V_1, V_2) \in \mathcal{V}_{A_1 \times A_2}$ takes value (v_1, v_2) under the map $[\![(V_1, V_2)]\!]_V = ([\![V_1]\!]_V, [\![V_2]\!]_V) = (v_1, v_2)$, so the result holds for $A = A_1 \times A_2$.

Hence the function $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$ is a bijection for all $A \in \mathsf{FO}$. □

Corollary 6.0.3. *The bijection $[\![-]\!]_V : \mathcal{V}_A \to [\![A]\!]$ can be used to define a section-retraction pair for any $A \in \mathsf{FO}$. Let $\mathsf{reify} : [\![A]\!] \to \mathcal{V}_A$ be defined by $\mathsf{reify}(v) \overset{def}{=} [\![v]\!]_V^{-1}$. Then $(\mathsf{reify}, [\![-]\!]_V)$ and $([\![-]\!]_V, \mathsf{reify})$ are section-retraction pairs.*

Proof. We have that $(\mathsf{reify}, [\![-]\!]_V)$ is a section-retraction pair by:

$$[\![\mathsf{reify}(v)]\!]_V = [\![[\![v]\!]_V^{-1}]\!]_V = v$$

We have that $([\![-]\!]_V, \mathsf{reify})$ is a section-retraction pair by:

$$\mathsf{reify}([\![V]\!]_V) = [\![[\![V]\!]_V]\!]_V^{-1} = V$$

□

We are ready to prove that the operational semantics we define are sound with respect to the denotational semantics.

Theorem 6.0.4 (Soundness of Operational Semantics). *If $M \Downarrow_{\underline{B}} xm$ then $[\![M]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm]\!]_t)$.*

Proof. We work by induction on the derivation of $M \Downarrow_{\underline{B}} xm$.

CASE(D-RETURN). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{\vdash^{\mathsf{v}} V : A}}{\mathsf{return}\ V \Downarrow_{FA} \eta(\mathsf{return}\ V)}\ \text{D-RETURN}$$

Now considering the monadic closed terminal we see that

$$\begin{aligned}\alpha_{\underline{B}}(T[\![\eta(\mathsf{return}\ V)]\!]_t) &= \alpha_{\underline{B}}(\eta(\mathsf{return}\ V) \gg\!= (x \mapsto \eta([\![x]\!]\emptyset)))\\ &= \alpha_{\underline{B}}(\eta([\![\mathsf{return}\ V]\!]\emptyset)) \quad \text{by } \eta(x) \gg\!= f = f(x)\\ &= [\![\mathsf{return}\ V]\!]\emptyset \quad \text{by } \alpha(\eta(x)) = x\end{aligned}$$

Thus the result holds for this case.

CASE(D-LAM). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{}{\lambda x : A.\, M \Downarrow_{A \to \underline{B}} \eta(\lambda x : A.\, M)}\ \textsc{D-Lam}$$

Now we work from the right-hand side to see that

$$\begin{aligned}\alpha_{\underline{B}}(T[\![\eta(\lambda x : A.\, M)]\!]_t) &= \alpha_{\underline{B}}(\eta(\lambda x : A.\, M) \gg\!= (x \mapsto \eta([\![x]\!]\emptyset)))\\ &= \alpha_{\underline{B}}(\eta([\![\lambda x : A.\, M]\!]\emptyset)) \quad \text{by } \eta(x) \gg\!= f = f(x)\\ &= [\![\lambda x : A.\, M]\!]\emptyset \quad \text{by } \alpha(\eta(x)) = x\end{aligned}$$

Hence, the result holds for this case.

CASE(D-FAIL). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{}{\mathsf{fail} \Downarrow_{\underline{B}} \varepsilon_{T(\mathcal{T}_{\underline{B}})}}\ \textsc{D-Fail}$$

Now we consider the monadic closed terminal, we clarify that $\varepsilon_{T(\mathcal{T}_{\underline{B}})}$ is an element of the FLP algebra over $T(\mathcal{T}_{\underline{B}})$ whereas $\varepsilon_{\underline{B}}$ is an element of the FLP algebra over $[\![\underline{B}]\!]$.

$$\begin{aligned}\alpha_{\underline{B}}\left(T\left[\!\left[\varepsilon_{T(\mathcal{T}_{\underline{B}})}\right]\!\right]_t\right) &= \alpha_{\underline{B}}\left(\varepsilon_{T(\mathcal{T}_{\underline{B}})} \gg\!= \underbrace{(x \mapsto \eta([\![x]\!]\emptyset))}_{\mathcal{T}_{\underline{B}} \to T[\![\underline{B}]\!]}\right)\\ &= \alpha_{\underline{B}}\left(\varepsilon_{T[\![\underline{B}]\!]}\right) \quad \text{by FLP structure Eq. (4.10)}\\ &= \varepsilon_{\underline{B}} \quad \text{by Lemma 5.1.3}\\ &= [\![\mathsf{fail}]\!]\emptyset\end{aligned}$$

Where the penultimate line follows from Lemma 5.1.3. Thus we see that the result holds in this case.

CASE(D-IFZ-ZERO). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M \Downarrow_{\underline{B}} xm}}{\mathsf{ifz}(\overline{0}; M; x.\, N) \Downarrow_{\underline{B}} xm}\ \textsc{D-Ifz-Zero}$$

By the induction hypothesis, $[\![M]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm]\!]_t)$. Now since we have by the denotational semantics that $[\![\mathsf{ifz}(\overline{0}; M; x.\, N)]\!]\emptyset = [\![M]\!]\emptyset$, we see that the result holds in this case.

CASE(D-IFZ-POS). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{N[\overline{n}/x] \Downarrow_{\underline{B}} xm}}{\mathsf{ifz}(\overline{n+1}; M; x.\, N) \Downarrow_{\underline{B}} xm}\ \textsc{D-Ifz-Pos}$$

By the induction hypothesis, $[\![N[\overline{n}/x]]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm]\!]_t)$. Now since we have by the denotational semantics that $[\![\mathsf{ifz}(\overline{n+1}; M; x.\, N)]\!]\rho = [\![N]\!]\rho[x \mapsto n]$ for all ρ. Using the Substitution Lemma 5.1.1 this which gives us that :

$$[\![N]\!]\rho[x \mapsto n] = [\![N]\!]\rho\left[x \mapsto [\![\overline{n}]\!]\rho\right] = \left[\!\left[N[\overline{n}/x]\right]\!\right]\rho$$

So in this case we see that:

$$[\![\mathsf{ifz}(\overline{n+1}; M; x.\, N)]\!]\emptyset = [\![N[\overline{n}/x]]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm]\!]_t)$$

Hence the result holds in this case.

Case(D-Pair). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M[V_1, V_2/x_1, x_2] \Downarrow_{\underline{B}} xm}}{\mathsf{split}((V_1, V_2); x_1, x_2.\, M) \Downarrow_{\underline{B}} xm}\ \textsc{D-Pair}$$

Now by the induction hypothesis, we have that

$$\llbracket M[V_1, V_2/x_1, x_2] \rrbracket \emptyset = \alpha_{\underline{B}}(T\llbracket xm \rrbracket_t)$$

Now using the denotational semantics know that

$$\llbracket \mathsf{split}((V_1, V_2); x_1, x_2.\, M) \rrbracket \emptyset \overset{\text{def}}{=} \llbracket M \rrbracket [x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } (v_1, v_2) \overset{\text{def}}{=} \llbracket (V_1, V_2) \rrbracket \emptyset$$

and

$$\llbracket (V_1, V_2) \rrbracket \emptyset \overset{\text{def}}{=} (\llbracket V_1 \rrbracket \emptyset, \llbracket V_2 \rrbracket \emptyset)$$

So combining the two equalities above we obtain:

$$\begin{aligned}
\llbracket \mathsf{split}((V_1, V_2); x_1, x_2.\, M) \rrbracket \emptyset &= \llbracket M \rrbracket [x_1 \mapsto v_1, x_2 \mapsto v_2] \text{ where } v_1 = \llbracket V_1 \rrbracket \emptyset, v_2 = \llbracket V_2 \rrbracket \emptyset \\
&= \llbracket M \rrbracket [x_1 \mapsto \llbracket V_1 \rrbracket \emptyset, x_2 \mapsto \llbracket V_2 \rrbracket \emptyset] \\
&= \llbracket M[V_2/x_2] \rrbracket [x_1 \mapsto \llbracket V_1 \rrbracket \emptyset] \quad \text{by Substitution Lemma (5.1.1)} \\
&= \llbracket M[V_1, V_2/x_1, x_2] \rrbracket \emptyset \quad \text{by Substitution Lemma (5.1.1)} \\
&= \alpha_{\underline{B}}(T\llbracket xm \rrbracket_t) \quad \text{by induction hypothesis}
\end{aligned}$$

Hence the result holds

Case(D-Force). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M \Downarrow_{\underline{B}} xm}}{\mathsf{force}\ (\mathsf{thunk}\ M) \Downarrow_{\underline{B}} xm}\ \textsc{D-Force}$$

By the induction hypothesis, $\llbracket M \rrbracket \emptyset = \alpha_{\underline{B}}(T\llbracket xm \rrbracket_t)$. Now immediately from the denotational semantics we know that $\llbracket \mathsf{force}\ (\mathsf{thunk}\ M) \rrbracket \emptyset = \llbracket M \rrbracket \emptyset$, so we see that the result holds in this case.

Case(D-App). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M \Downarrow_{A \to \underline{B}} xm} \qquad \dfrac{\vdots}{\vdash^{\mathsf{v}} V : A} \qquad \dfrac{\vdots}{N[V/x] \Downarrow_{\underline{B}} ym_N}}{M(V) \Downarrow_{\underline{B}} xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto ym_N)}\ \textsc{D-App}$$

By the induction hypothesis, $\llbracket M \rrbracket \emptyset = \alpha_{A \to \underline{B}}(T\llbracket xm \rrbracket_t)$ and $\llbracket N[V/x] \rrbracket \emptyset = \alpha_{\underline{B}}(T\llbracket ym_N \rrbracket_t)$. Now we work from the denotational semantics to see that:

$$\begin{aligned}
\llbracket M(V) \rrbracket \emptyset &= \llbracket M \rrbracket (\llbracket V \rrbracket \emptyset) \\
&= \alpha_{A \to \underline{B}}(T\llbracket xm \rrbracket_t)(\llbracket V \rrbracket \emptyset) \\
&= \alpha_{\underline{B}}(T\llbracket xm \rrbracket_t \mathbin{>\!\!>\!=} (f \mapsto \eta(f(\llbracket V \rrbracket \emptyset)))) \quad \text{by definition of } \alpha_{A \to \underline{B}} \text{ in §4.3.2} \\
&= \alpha_{\underline{B}}((xm \mathbin{>\!\!>\!=} \eta \circ \llbracket - \rrbracket \emptyset) \mathbin{>\!\!>\!=} (f \mapsto \eta(f(\llbracket V \rrbracket \emptyset)))) \quad \text{by definition of } T\llbracket - \rrbracket_t \\
&= \alpha_{\underline{B}}(xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto \eta(\llbracket \lambda x : A.\, N \rrbracket \emptyset) \mathbin{>\!\!>\!=} (f \mapsto \eta(f(\llbracket V \rrbracket \emptyset))))) \quad \text{by bind associativity} \\
&= \alpha_{\underline{B}}(xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto (\eta(\llbracket \lambda x : A.\, N \rrbracket \emptyset(\llbracket V \rrbracket \emptyset))))) \quad \text{by } \eta(x) \mathbin{>\!\!>\!=} f = f(x) \\
&= \alpha_{\underline{B}}(xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto (\eta(\llbracket N \rrbracket \emptyset[x \mapsto \llbracket V \rrbracket \emptyset])))) \quad \text{by denotational definition} \\
&= \alpha_{\underline{B}}(xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto (\eta(\llbracket N[V/x] \rrbracket \emptyset))) \quad \text{by Substitution Lemma 5.1.1} \\
&= \alpha_{\underline{B}}\Big(xm \mathbin{>\!\!>\!=} (\lambda x : A.\, N \mapsto (\eta(\alpha_{\underline{B}}\big(\underbrace{T\llbracket ym_N \rrbracket_t}_{\mathcal{T}_0 \to T\llbracket \underline{B} \rrbracket}\big))))\Big) \quad \text{by induction hypothesis}
\end{aligned}$$

$$\begin{aligned}
&= \alpha_{\underline{B}}(xm \gg\!\!= (\lambda x : A.\, N \mapsto T[\![ym_N]\!]_t))) && \text{by Theorem 4.3.4}\\
&= \alpha_{\underline{B}}(xm \gg\!\!= (\lambda x : A.\, N \mapsto (ym_N \gg\!\!= \eta \circ [\![-]\!]\emptyset))) && \text{by definition of } T[\![-]\!]_t\\
&= \alpha_{\underline{B}}((xm \gg\!\!= \lambda x : A.\, N \mapsto ym_N) \gg\!\!= \eta \circ [\![-]\!]\emptyset) && \text{by bind associativity}\\
&= \alpha_{\underline{B}}([\![xm \gg\!\!= (\lambda x : A.\, N \mapsto ym_N)]\!]\emptyset)
\end{aligned}$$

In particular, we can pattern match for $\lambda x : A.\, N$ within $xm \in T(\mathcal{T}_{A\to\underline{B}})$ as we can always apply to terms of this shape by using FLP structure laws. Thus the result holds in this case.

Case(D-Bind). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M \Downarrow_{FA} xm} \qquad \dfrac{\vdots}{x : A \vdash^{c} N : \underline{B}} \qquad \dfrac{\vdots}{N[V/x] \Downarrow_{\underline{B}} ym_V}}{M \text{ to } x.\, N \Downarrow_{\underline{B}} xm \gg\!\!= (\text{return } V \mapsto ym_V)}\ \text{D-Bind}$$

By the induction hypothesis, $[\![M]\!]\emptyset = \alpha_{FA}(T[\![xm]\!]_t)$ and $[\![N[V/x]]\!]\emptyset = \alpha_{\underline{B}}(T[\![ym_V]\!]_t)$. Using the Substitution Lemma 5.1.1 this means we have that for all $\rho \in [\![\Gamma]\!]$:

$$[\![N]\!]\rho[x \mapsto [\![V]\!]\rho] = [\![N[V/x]]\!]\rho$$

Now by the denotational semantics of $M \text{ to } x.\, N$, we have that:

$$[\![M \text{ to } x.\, N]\!]\rho = \alpha_{\underline{B}}([\![M]\!]\rho \gg\!\!= (v \mapsto \eta([\![N]\!]\rho[x \mapsto v])))$$

These both hold for all environments ρ, in particular for closed terms with an empty environment $\rho = \emptyset$.

We can also unwrap the induction hypothesis for $[\![M]\!]\emptyset$ to get:

$$\begin{aligned}
[\![M]\!]\emptyset &= \alpha_{FA}(T[\![xm]\!]_t)\\
&= T[\![xm]\!]_t \gg\!\!= id_{T[\![A]\!]}\\
&= (xm \gg\!\!= x \mapsto \eta([\![x]\!]\emptyset)) \gg\!\!= id_{T[\![A]\!]}\\
&= xm \gg\!\!= (x \mapsto \eta([\![x]\!]\emptyset) \gg\!\!= id_{T[\![A]\!]}) && \text{by bind associativity}\\
&= xm \gg\!\!= x \mapsto id_{T[\![A]\!]}([\![x]\!]\emptyset) && \text{by } \eta(x) \gg\!\!= f = f(x)\\
&= xm \gg\!\!= [\![-]\!]\emptyset
\end{aligned}$$

Now working backwards from the monadic terminal which $[\![M \text{ to } x.\, N]\!]\emptyset$ evaluates to, we see that:

$$\begin{aligned}
&\alpha_{\underline{B}}(T[\![xm \gg\!\!= (\text{return } V \mapsto ym_V)]\!]_t)\\
=&\alpha_{\underline{B}}((xm \gg\!\!= \text{return } V \mapsto ym_V) \gg\!\!= \eta \circ [\![-]\!]\emptyset)\\
=&\alpha_{\underline{B}}\Big(\underbrace{xm}_{T(\mathcal{T}[\underline{B}])} \gg\!\!= \underbrace{(\text{return } V \mapsto ym_V \gg\!\!= \eta \circ [\![-]\!]\emptyset)}_{\mathcal{T}[\underline{B}]\to T[\![\underline{B}]\!]}\Big) \text{by bind associativity}\\
=&\alpha_{\underline{B}}(xm \gg\!\!= (\text{return } V \mapsto \eta(\alpha_{\underline{B}}((ym_V \gg\!\!= \eta \circ [\![-]\!]\emptyset))))) \text{ by Theorem 4.3.4}\\
=&\alpha_{\underline{B}}(xm \gg\!\!= (\text{return } V \mapsto \eta([\![N[V/x]]\!]\emptyset))) \text{ by induction hypothesis}\\
=&\alpha_{\underline{B}}(xm \gg\!\!= (\text{return } V \mapsto \eta([\![N]\!][x \mapsto [\![V]\!]\emptyset]))) \text{ by Substitution Lemma 5.1.1}\\
=&\alpha_{\underline{B}}(xm \gg\!\!= (\text{return } V \mapsto \eta([\![V]\!]\emptyset) \gg\!\!= (v \mapsto \eta([\![N]\!][x \mapsto v])))) \text{ by } \eta(x) \gg\!\!= f = f(x)\\
=&\alpha_{\underline{B}}(xm \gg\!\!= (\text{return } V \mapsto [\![\text{return } V]\!]\emptyset \gg\!\!= (v \mapsto \eta([\![N]\!][x \mapsto v])))) \text{ by definition of semantics for return}\\
=&\alpha_{\underline{B}}((xm \gg\!\!= [\![-]\!]\emptyset) \gg\!\!= (v \mapsto \eta([\![N]\!][x \mapsto v]))) \text{ by bind associativity}\\
=&\alpha_{\underline{B}}([\![M]\!]\emptyset \gg\!\!= (v \mapsto \eta([\![N]\!][x \mapsto v]))) \text{ by induction hypothesis}\\
=&[\![M \text{ to } x.\, N]\!]\emptyset
\end{aligned}$$

Notably, we can pattern match for $\text{return } V$ since we know that xm is an element of the FLP algebra over $T(\mathcal{T}_{FA})$. Hence the result holds in this case.

Case(D-Choice). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M \Downarrow_{\underline{B}} xm} \qquad \dfrac{\vdots}{N \Downarrow_{\underline{B}} ym}}{M \,[\!]\, N \Downarrow_{\underline{B}} xm \vee ym}\ \text{D-Choice}$$

Then by the induction hypothesis, we have:

$$\begin{aligned}
[\![M]\!]\emptyset &= \alpha_{\underline{B}}(T[\![xm]\!]_t) \\
[\![N]\!]\emptyset &= \alpha_{\underline{B}}(T[\![ym]\!]_t)
\end{aligned}$$

Now the result follows immediately from the denotational semantics of choice and FLP structure properties:

$$\begin{aligned}
[\![M \mathbin{[\!]} N]\!]\emptyset &= [\![M]\!]\emptyset \vee [\![N]\!]\emptyset \\
&= \alpha_{\underline{B}}(T[\![xm]\!]_t) \vee \alpha_{\underline{B}}(T[\![ym]\!]_t) \quad \text{by induction hypothesis} \\
&= \alpha_{\underline{B}}(T[\![xm]\!]_t \vee T[\![ym]\!]_t) \quad \text{by Lemma 5.1.3} \\
&= \alpha_{\underline{B}}((xm \gg\!= \eta \circ [\![-]\!]\emptyset) \vee (ym \gg\!= \eta \circ [\![-]\!]\emptyset)) \\
&= \alpha_{\underline{B}}((xm \vee ym) \gg\!= \eta \circ [\![-]\!]\emptyset) \quad \text{by FLP Structure Eq. (4.11)} \\
&= \alpha_{\underline{B}}(T[\![xm \vee ym]\!]_t)
\end{aligned}$$

Hence the result holds in this case.

Case(D-Exists). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{M[V/x] \Downarrow_{\underline{B}} xm_V}}{\exists x : A.\, M \Downarrow_{\underline{B}} \exists\!\!\exists^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V \mapsto xm_V)}\ \text{D-Exists}$$

Then by the induction hypothesis we have that $[\![M[V/x]]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm_V]\!]_t)$. Working backwards from the right-hand side we see that

$$\begin{aligned}
\alpha_{\underline{B}}\Big(T\Big[\!\!\Big[\exists\!\!\exists^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V \mapsto xm_V)\Big]\!\!\Big]_t\Big) &= \alpha_{\underline{B}}\Big(\Big(\exists\!\!\exists^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V \mapsto xm_V)\Big) \gg\!= \underbrace{\eta \circ [\![-]\!]\emptyset}_{\mathcal{T}_{\underline{B}} \to T[\![\underline{B}]\!]}\Big) \\
&= \alpha_{\underline{B}}\Big(\exists\!\!\exists^{\mathcal{V}_A}_{T[\![\underline{B}]\!]}(V \mapsto xm_V \gg\!= \eta \circ [\![-]\!]\emptyset)\Big) \quad \text{by FLP structure laws 4.1.9} \\
&= \alpha_{\underline{B}}\Big(\exists\!\!\exists^{\mathcal{V}_A}_{T[\![\underline{B}]\!]}(V \mapsto T[\![xm_V]\!]_t)\Big) \\
&= \exists\!\!\exists^{\mathcal{V}_A}_{[\![\underline{B}]\!]}(V \mapsto \alpha_{\underline{B}}(T[\![xm_V]\!]_t)) \quad \text{by Lemma 5.1.3} \\
&= \exists\!\!\exists^{\mathcal{V}_A}_{[\![\underline{B}]\!]}(V \mapsto [\![M[V/x]]\!]\emptyset) \quad \text{by induction hypothesis} \\
&= \exists\!\!\exists^{\mathcal{V}_A}_{[\![\underline{B}]\!]}(V \mapsto [\![M]\!][x \mapsto [\![V]\!]_V]) \quad \text{by Substitution Lemma 5.1.1} \\
&= \exists\!\!\exists^{[\![A]\!]}_{[\![\underline{B}]\!]}(v \mapsto [\![M]\!][x \mapsto v]) \quad \text{by FLP structure Eq. (4.14) and Corollary 6.0.3} \\
&= [\![\exists x : A.\, M]\!]\emptyset
\end{aligned}$$

We highlight the use of the section-retraction pair $(\mathsf{reify}, [\![-]\!]_V)$ from Corollary 6.0.3 with the FLP structure Eq. (4.14) to remove the retraction while swapping the universe which the existential operator ranges over in the penultimate line. We see that the result holds in this case.

Case(D-Unify). Suppose the derivation of $M \Downarrow_{\underline{B}} xm$ is of the form

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^{\mathsf{v}} V_1 : A} \qquad \dfrac{\vdots}{\Gamma \vdash^{\mathsf{v}} V_2 : A} \qquad A \in \mathsf{FO} \qquad \dfrac{\vdots}{M \Downarrow_{\underline{B}} xm}}{V_1 \doteq_A V_2;\ M \Downarrow_{\underline{B}} \mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V_1, V_2, xm)}\ \text{D-Unify}$$

By the induction hypothesis we have that $[\![M]\!]\emptyset = \alpha_{\underline{B}}(T[\![xm]\!]_t)$. Then by the denotational semantics of unification, we have:

$$\begin{aligned}
[\![V_1 \doteq_A V_2;\ M]\!]\emptyset &= \mathsf{eq}^{[\![A]\!]}_{\underline{B}}([\![V_1]\!]\emptyset, [\![V_2]\!]\emptyset, [\![M]\!]\emptyset) \\
&= \mathsf{eq}^{[\![A]\!]}_{\underline{B}}([\![V_1]\!]\emptyset, [\![V_2]\!]\emptyset, \alpha_{\underline{B}}(T[\![xm]\!]_t))
\end{aligned}$$

$$
\begin{aligned}
&= \alpha_{\underline{B}}\left(\mathsf{eq}^{[\![A]\!]}_{T[\![\underline{B}]\!]}([\![V_1]\!]_V, [\![V_2]\!]_V, xm \gg\!= \eta \circ [\![-]\!]\emptyset)\right) \text{ by Lemma 5.1.3} \\
&= \alpha_{\underline{B}}\left(\mathsf{eq}^{[\![A]\!]}_{T(\mathcal{T}_{\underline{B}})}([\![V_1]\!]_V, [\![V_2]\!]_V, xm) \gg\!= \eta \circ [\![-]\!]\emptyset\right) \text{ by FLP structure Eq. (4.13)} \\
&= \alpha_{\underline{B}}\left(\mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V_1, V_2, xm) \gg\!= \eta \circ [\![-]\!]\emptyset\right) \text{ by FLP Structure Eq. (4.15) and Corollary 6.0.3} \\
&= \alpha_{\underline{B}}\left(T\left[\!\!\left[\mathsf{eq}^{\mathcal{V}_A}_{T(\mathcal{T}_{\underline{B}})}(V_1, V_2, xm)\right]\!\!\right]_t\right)
\end{aligned}
$$

Where in the penultimate line we use the section-retraction pair $([\![-]\!]_V, \mathsf{reify})$ from Corollary 6.0.3 with the FLP structure Eq. (4.15) to remove the section $[\![-]\!]_V$ and change the universe of values. Hence the result holds in this case, and thus so does the theorem. □

We note that we have used the semantic map for closed first-order value terms $[\![-]\!]_V$ as a section in the case of D-UNIFY and as a retraction in the case of D-EXISTS. Hence, this proof of the soundness of the operational semantics directly relies on the bijection shown in Lemma 6.0.2. One wonders if a proof exists that only requires a section-retraction pair between $\mathcal{V}_A$ and $[\![A]\!]$ for all first-order value types, but we leave this for future consideration.

Bibliography

[1] John Hughes. “Why functional programming matters”. In: *The computer journal* 32.2 (1989), pp. 98–107 (cit. on p. 1).

[2] Paul Hudak. “Conception, evolution, and application of functional programming languages”. In: *ACM Computing Surveys (CSUR)* 21.3 (1989), pp. 359–411 (cit. on p. 1).

[3] John W Lloyd. *Foundations of logic programming.* Springer Science & Business Media, 2012 (cit. on p. 1).

[4] Krzysztof R Apt. “Logic Programming.” In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)* 1990 (1990), pp. 493–574 (cit. on p. 1).

[5] Michael Hanus. “Multi-paradigm declarative languages”. In: *Logic Programming: 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007. Proceedings 23.* Springer. 2007, pp. 45–75 (cit. on p. 1).

[6] Michael Hanus. “The integration of functions into logic programming: From theory to practice”. In: *The Journal of Logic Programming* 19 (1994), pp. 583–628 (cit. on p. 2).

[7] Zoltan Somogyi, Fergus J Henderson, and Thomas Charles Conway. “Mercury, an efficient purely declarative logic programming language”. In: *Australian Computer Science Communications* 17 (1995), pp. 499–512 (cit. on p. 2).

[8] Michael Hanus. “Functional logic programming: From theory to Curry”. In: *Programming Logics: Essays in Memory of Harald Ganzinger* (2013), pp. 123–168 (cit. on p. 2).

[9] YesLogic. *PrinceXML Documentation: Acknowledgments.* 2023. URL: https://www.princexml.com/doc/acknowledgements/ (visited on 07/03/2023) (cit. on p. 2).

[10] Sergio Antoy and Michael Hanus. “Functional logic design patterns”. In: *FLOPS.* Vol. 2441. Springer. 2002, pp. 67–87 (cit. on p. 2).

[11] Lennart Augustsson, Joachim Breitner, Koen Claessen, Ranjit Jhala, Simon Peyton Jones, Olin Shivers, and Tim Sweeney. “The Verse Calculus: a Core Calculus for Functional Logic Programming”. Draft Paper. 2023. URL: https://simon.peytonjones.org/assets/pdfs/verse-conf.pdf (cit. on p. 2).

[12] Elvira Albert, Michael Hanus, Frank Huch, Javier Oliver, and Germán Vidal. “Operational semantics for declarative multi-paradigm languages”. In: *Journal of Symbolic Computation* 40.1 (2005), pp. 795–829 (cit. on p. 2).

[13] Paul Blain Levy. *Call-by-Push-Value: A Functional-Imperative Synthesis.* Semantic Structures in Computation. Springer Dordrecht, 2003. DOI: 10.1007/978-94-007-0954-6 (cit. on pp. 3, 4, 6, 7, 20, 23).

[14] Paul Levy. “SIGLOG Semantics Column: Call-by-push-value”. In: *ACM SIGLOG News* 9.2 (2022), p. 7 (cit. on pp. 4, 23).

[15] R. D. Tennent. “The Denotational Semantics of Programming Languages”. In: *Commun. ACM* 19.8 (Aug. 1976), pp. 437–453. ISSN: 0001-0782. DOI: 10.1145/360303.360308. URL: https://doi.org/10.1145/360303.360308 (cit. on p. 6).

[16] Paul Moritz Cohn. *Universal algebra.* Vol. 6. Springer Science & Business Media, 2012 (cit. on p. 7).

[17] Franz Baader and Jörg H Siekmann. *Unification theory.* 1994 (cit. on p. 10).

[18] Sam Staton. “An Algebraic Presentation of Predicate Logic”. In: *Foundations of Software Science and Computation Structures.* Ed. by Frank Pfenning. Vol. 7794. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 401–417. DOI: 10.1007/978-3-642-37075-5_26 (cit. on pp. 13, 14, 25, 26).

[19] nLab authors. *two-out-of-three*. https://ncatlab.org/nlab/show/two-out-of-three. Revision 4. May 2023. (Visited on 02/05/2023) (cit. on p. 17).

[20] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 2013 (cit. on p. 20).

[21] Henk P Barendregt. “Lambda calculi with types”. In: (1992) (cit. on p. 23).

[22] Kurt Gödel. *On formally undecidable propositions of Principia Mathematica and related systems*. Courier Corporation, 1992 (cit. on p. 28).