Handling fibred algebraic effects

DANEL AHMAN, Inria Paris

We study algebraic computational effects and their handlers in the dependently typed setting. We describe computational effects using a generalisation of Plotkin and Pretnar's effect theories, whose dependently typed operations allow us to capture precise notions of computation, e.g., state with location-dependent store types and dependently typed update monads. Our treatment of handlers is based on an observation that their conventional term-level definition leads to unsound program equivalences being derivable in languages that include a notion of homomorphism. We solve this problem by giving handlers a novel type-based treatment via a new computation type, the user-defined algebra type, which pairs a value type (the carrier) with a family of value terms (the operations), capturing Plotkin and Pretnar's insight that handlers denote algebras. The conventional presentation of handlers can then be routinely derived. In addition, we demonstrate that this type-based treatment of handlers provides a useful mechanism for reasoning about effectful computations.

CCS Concepts: •**Software and its engineering** → *Functional languages;* •**Theory of computation** → *Type theory; Control primitives; Functional constructs; Type structures; Program specifications; Denotational semantics;*

ACM Reference format:
Danel Ahman. 2017. Handling fibred algebraic effects. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), 36 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An important feature of many widely-used programming languages is their support for computational effects, e.g., raising exceptions, accessing memory, performing I/O, etc., which allows programmers to write more efficient and conceptually clearer programs. Therefore, if dependently typed languages are to live up to their promise of providing a lightweight means for integrating formal verification and practical programming, we must first understand how to properly account for computational effects in such languages. While there already exists a range of work on combining these two fields (Ahman et al. 2016, 2017; Brady 2013; Casinghino 2014; Hancock and Setzer 2000; McBride 2011; Nanevski et al. 2008; Pédrot and Tabareau 2017; Pitts et al. 2015), there is still a gap between the rigorous and comprehensive understanding we have of computational effects in the simply typed setting, and what we know about them in the presence of dependent types. For example, in the above-mentioned works, either the mathematical foundations of the languages developed are not settled, the available effects are limited, or they lack a systematic treatment of (equational) effect specification. In this paper, we contribute to the intersection of these two fields by studying algebraic effects and their handlers in the dependently typed setting.

Algebraic effects form a wide class of computational effects that lend themselves to specification using operations and equations; examples include exceptions, state, input-output, nondeterminism, probability, etc. Their study originated with the pioneering work of Plotkin and Power (2001, 2002); they have since been successfully applied to, e.g., modularly combining effects (Hyland et al. 2006) and effect-dependent program optimisations (Kammar and Plotkin 2012). A key insight of Plotkin and Power was that most of Moggi's monads (Moggi 1989, 1991) are determined by algebraic effects, with the notable exception of continuations, which are not algebraic.

2017. 2475-1421/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

A significant role in the recent rise of interest in algebraic effects can be attributed to their *handlers*. These were introduced by Plotkin and Pretnar (2013) as a generalisation of exception handlers to all algebraic effects, based on the idea that handlers denote user-defined algebras for the given algebraic theory and the handling construct denotes the homomorphism induced by the universal property of the free algebra. From a programming language perspective, a handler

$$\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}}$$

provides redefinitions of the algebraic operations in the signature $\mathcal{S}_{\mathrm{eff}}$ and the handling construct

$$M \;\mathtt{handled\ with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}\; N_{\mathrm{ret}}$$

then recursively traverses the given program M, replacing each algebraic operation op with the corresponding user-defined term N_{op}, e.g., as illustrated by the following β-equation:

$$\begin{aligned}\Gamma \vdash (\mathsf{op}^{FA}_V(y'.M)) \;&\mathtt{handled\ with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}\; N_{\mathrm{ret}}\\ &= N_{\mathsf{op}}[V/x][\lambda y'\!:\!O[V/x].\mathtt{thunk}\; H/x'] : \underline{C}\end{aligned}$$

where

$$H \stackrel{\mathrm{def}}{=} M \;\mathtt{handled\ with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}\; N_{\mathrm{ret}}$$

Plotkin and Pretnar (2013, §3) also showed that handlers can be used to neatly implement timeouts, rollbacks, stream redirection, etc. More recently, handlers have also gained popularity as a practical and modular programming language abstraction, allowing programmers to write their programs generically in terms of algebraic operations, and then use handlers to modularly provide different fit-for-purpose implementations for these programs. A prototypical example of this style of programming involves implementing the global state operations (`get` and `put`) using the natural representation of stateful programs as state-passing functions $\mathrm{St} \to A \times \mathrm{St}$. In order to support this style of programming, existing languages have been extended with algebraic effects and their handlers (Hillerström and Lindley 2016; Kammar et al. 2013; Leijen 2017), and entire new languages have been built around them (Bauer and Pretnar 2015; Lindley et al. 2017).

Contributions. Our *key contribution* is an observation that the conventional term-level of definition of handlers leads to unsound program equivalences being derivable in languages that include a notion of homomorphism (§4.1). Our *other contributions* include: i) a dependently typed generalisation of Plotkin and Pretnar's effect theories (§3.1); ii) introducing a new computation type, the user-defined algebra type, so as to give a type-based treatment of handlers and solve the problem with unsound program equivalences (§4.2); iii) showing how to derive the conventional term-level definition of handlers from our type-based treatment (§4.3); iv) demonstrating that such handlers provide a useful mechanism for reasoning about effectful computations (§7); and v) equipping the resulting dependently typed language with a sound fibrational denotational semantics (§8).

2 EMLTT: THE UNDERLYING EFFECTFUL DEPENDENTLY TYPED LANGUAGE

We begin with an overview of the language we use as a basis for studying algebraic effects and their handlers in the dependently typed setting, namely, the effectful dependently typed language proposed by Ahman et al. (2016). This language is a natural extension of Martin-Löf's intensional type theory (MLTT) with computational effects. It makes a clear distinction between values and computations, both at the level of types and terms, analogously to simply typed languages such as Call-By-Push-Value (CBPV) (Levy 2004) and the Enriched Effect Calculus (EEC) (Egger et al. 2014). Specifically, we base our work on a minor extension of Ahman et al.'s dependently typed language, as made precise later in this section. In this paper, we refer to this extended language as EMLTT.

As usual for dependently typed languages, EMLTT's types and terms are defined mutually inductively. First, one assumes countable sets of *value variables* $x, y, \ldots$ and *computation variables* $z, \ldots$. Next, the grammar of *value types* $A, B, \ldots$ and *computation types* $\underline{C}, \underline{D}, \ldots$ is given by

$$\begin{array}{rcl} A & ::= & \mathsf{Nat} \mid 1 \mid 0 \mid A + B \mid \Sigma x\,{:}\,A.B \mid \Pi x\,{:}\,A.B \mid V =_A W \mid U\underline{C} \mid \underline{C} \multimap \underline{D} \\ \underline{C} & ::= & FA \mid \Sigma x\,{:}\,A.\underline{C} \mid \Pi x\,{:}\,A.\underline{C} \end{array}$$

As standard, we write $A \times B$ and $A \to B$ for $\Sigma x\,{:}\,A.B$ and $\Pi x\,{:}\,A.B$ when x is not free in B, and similarly for the computational Σ- and Π-types. Analogously to Ahman et al. (2016), we omit general inductive types and use natural numbers as a representative example. However, compared to op. cit., we further include the empty type 0, the sum type $A + B$, and the homomorphic function type $\underline{C} \multimap \underline{D}$. We include the first two as to specify signatures of algebraic effects (see §3.2); and the latter as it is useful for writing effectful code without excessive thunking and forcing, and because it enables us to eliminate values into homomorphism terms, as discussed later in this section. Finally, we note that FA is the type of possibly effectful computations that return values of type A.

Next, the grammar of EMLTT's *value terms* $V, W, \ldots$ is given by

$$\begin{array}{rl} V ::= & x \mid \star \mid \mathtt{zero} \mid \mathtt{succ}\, V \mid \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, V) \mid \mathtt{case}\, V\, \mathtt{of}_{x.A}\, () \\ \mid & \mathtt{inl}_{A+B}\, V \mid \mathtt{inr}_{A+B}\, V \mid \mathtt{case}\, V\, \mathtt{of}_{x.B}\, (\mathtt{inl}(y_1\,{:}\,A_1) \mapsto W_1, \mathtt{inr}(y_2\,{:}\,A_2) \mapsto W_2) \\ \mid & \langle V, W \rangle_{(x:A).B} \mid \mathtt{pm}\, V\, \mathtt{as}\, (x_1\,{:}\,A_1, x_2\,{:}\,A_2)\, \mathtt{in}_{y.B}\, W \mid \lambda x\,{:}\,A.V \mid V(W)_{(x:A).B} \\ \mid & \mathtt{refl}\, V \mid \mathtt{eq\text{-}elim}_A(x_1.x_2.x_3.B, y.W, V_1, V_2, V_p) \mid \mathtt{thunk}\, M \mid \lambda z\,{:}\,\underline{C}.K \end{array}$$

Observe that in addition to the introduction and elimination forms for the types inherited from MLTT, value terms also include thunks of computations and homomorphic lambda abstractions.

Regarding effectful programs, EMLTT makes a distinction between *computation terms* $M, N, \ldots$ and *homomorphism terms* $K, L, \ldots$. The latter are necessary for correctly defining the elimination form for the computational Σ-type $\Sigma x\,{:}\,A.\underline{C}$. The grammar of these two kinds of terms is given by

$$\begin{array}{rlcrl} M ::= & \mathtt{force}_{\underline{C}}\, V & \quad & K ::= & z \\ \mid & \mathtt{return}\, V \mid M\, \mathtt{to}\, x\,{:}\,A\, \mathtt{in}_{\underline{C}}\, N & & \mid & K\, \mathtt{to}\, x\,{:}\,A\, \mathtt{in}_{\underline{C}}\, M \\ \mid & \langle V, M \rangle_{(x:A).\underline{C}} \mid M\, \mathtt{to}\, (x\,{:}\,A, z\,{:}\,\underline{C})\, \mathtt{in}_{\underline{D}}\, K & & \mid & \langle V, K \rangle_{(x:A).\underline{C}} \mid K\, \mathtt{to}\, (x\,{:}\,A, z\,{:}\,\underline{C})\, \mathtt{in}_{\underline{D}}\, L \\ \mid & \lambda x\,{:}\,A.M \mid M(V)_{(x:A).\underline{C}} & & \mid & \lambda x\,{:}\,A.K \mid K(V)_{(x:A).\underline{C}} \\ \mid & V(M)_{\underline{C},\underline{D}} & & \mid & V(K)_{\underline{C},\underline{D}} \end{array}$$

Computation terms include standard combinators for effectful programming: returning a value, sequential composition, lambda abstraction, and function application. Further, they include forcing of thunked computations, introduction and elimination forms for the computational Σ-type, and homomorphic function applications. Homomorphism terms are similar, but also include computation variables z, which have to be used i) linearly and ii) in a way that ensures that the computation bound to z "happens first" in a term containing it. These restrictions on computation variables guarantee that every K denotes a homomorphism in the categorical models we study in §8.

As one can readily use thunking and forcing (and homomorphic functions) to eliminate values into computation terms (resp. homomorphism terms), these elimination forms are not included primitively. For example, one can eliminate natural numbers into computation terms as follows:

$$\mathtt{nat\text{-}elim}_{x.\underline{C}}(M_z, y_1.y_2.M_s, V) \stackrel{\text{def}}{=} \mathtt{force}_{\underline{C}[V/x]} \Big(\mathtt{nat\text{-}elim}_{x.U\underline{C}}(\mathtt{thunk}\, M_z, y_1.y_2.\mathtt{thunk}\, M_s, V)\Big)$$

and into homomorphism terms as follows:

$$\mathtt{nat\text{-}elim}_{\underline{C}}(K_z, y.K_s, V) \stackrel{\text{def}}{=} \Big(\mathtt{nat\text{-}elim}_{x_1.\underline{C} \multimap \underline{C}}(\lambda z\,{:}\,\underline{C}.K_z, y.x_2.\lambda z\,{:}\,\underline{C}.K_s[x_2\, z/z], V)\Big)\; z$$

Value types:

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Nat}} \qquad \frac{\Gamma \vdash V : A \quad \Gamma \vdash W : A}{\Gamma \vdash V =_A W} \qquad \frac{\Gamma \vdash \underline{C}}{\Gamma \vdash U\underline{C}} \qquad \frac{\Gamma \vdash \underline{C} \quad \Gamma \vdash \underline{D}}{\Gamma \vdash \underline{C} \multimap \underline{D}}$$

Computation types:

$$\frac{\Gamma \vdash A}{\Gamma \vdash FA} \qquad \frac{\Gamma, x \mathbin{:} A \vdash \underline{C}}{\Gamma \vdash \Sigma x \mathbin{:} A.\underline{C}} \qquad \frac{\Gamma, x \mathbin{:} A \vdash \underline{C}}{\Gamma \vdash \Pi x \mathbin{:} A.\underline{C}}$$

Value terms:

$$\frac{\vdash \Gamma \quad x \mathbin{:} A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathtt{zero} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash V : \mathsf{Nat}}{\Gamma \vdash \mathtt{succ}\ V : \mathsf{Nat}} \qquad \frac{\begin{array}{c}\Gamma, x \mathbin{:} \mathsf{Nat} \vdash A \quad \Gamma \vdash V : \mathsf{Nat} \quad \Gamma \vdash V_z : A[\mathtt{zero}/x] \\ \Gamma, y_1 \mathbin{:} \mathsf{Nat}, y_2 \mathbin{:} A[y_1/x] \vdash V_s : A[\mathtt{succ}\ y_1/x]\end{array}}{\Gamma \vdash \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, V) : A[V/x]}$$

$$\frac{\Gamma, x \mathbin{:} A \vdash V : B}{\Gamma \vdash \lambda x \mathbin{:} A.V : \Pi x \mathbin{:} A.B} \qquad \frac{\Gamma \vdash V : \Pi x \mathbin{:} A.B \quad \Gamma, x \mathbin{:} A \vdash B \quad \Gamma \vdash W : A}{\Gamma \vdash V(W)_{(x:A).B} : B[W/x]} \qquad \frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash \mathtt{thunk}\ M : U\underline{C}} \qquad \frac{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : \underline{D}}{\Gamma \vdash \lambda z \mathbin{:} \underline{C}.K : \underline{C} \multimap \underline{D}}$$

Computation terms:

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, x \mathbin{:} A \vdash N : \underline{C}}{\Gamma \vdash M\ \mathtt{to}\ x \mathbin{:} A\ \mathtt{in}_{\underline{C}}\ N : \underline{C}} \qquad \frac{\Gamma \vdash V : U\underline{C}}{\Gamma \vdash \mathtt{force}_{\underline{C}}\ V : \underline{C}} \qquad \frac{\Gamma \vdash V : \underline{C} \multimap \underline{D} \quad \Gamma \vdash M : \underline{C}}{\Gamma \vdash V(M)_{\underline{C},\underline{D}} : \underline{D}}$$

Homomorphism terms:

$$\frac{\Gamma \vdash \underline{C}}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash z : \underline{C}} \qquad \frac{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : FA \quad \Gamma \vdash \underline{D} \quad \Gamma, x \mathbin{:} A \vdash M : \underline{D}}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K\ \mathtt{to}\ x \mathbin{:} A\ \mathtt{in}_{\underline{D}}\ M : \underline{D}} \qquad \frac{\Gamma \vdash V : A \quad \Gamma, x \mathbin{:} A \vdash \underline{D} \quad \Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : \underline{D}[V/x]}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash \langle V, K \rangle_{(x:A).\underline{D}} : \Sigma x \mathbin{:} A.\underline{D}}$$

$$\frac{\Gamma \,|\, z_1 \mathbin{:} \underline{C} \vdash K : \Sigma x \mathbin{:} A.\underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x \mathbin{:} A \,|\, z_2 \mathbin{:} \underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \,|\, z_1 \mathbin{:} \underline{C} \vdash K\ \mathtt{to}\ (x \mathbin{:} A, z_2 \mathbin{:} \underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L : \underline{D}_2} \qquad \frac{\Gamma \vdash \underline{C} \quad \Gamma, x \mathbin{:} A \,|\, z \mathbin{:} \underline{C} \vdash K : \underline{D}}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash \lambda x \mathbin{:} A.K : \Pi x \mathbin{:} A.\underline{D}}$$

$$\frac{\Gamma, x \mathbin{:} A \vdash \underline{D} \quad \Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : \Pi x \mathbin{:} A.\underline{D} \quad \Gamma \vdash V : A}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K(V)_{(x:A).\underline{D}} : \underline{D}[V/x]} \qquad \frac{\Gamma \vdash V : \underline{D}_1 \multimap \underline{D}_2 \quad \Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : \underline{D}_1}{\Gamma \,|\, z \mathbin{:} \underline{C} \vdash V(K)_{\underline{D}_1,\underline{D}_2} : \underline{D}_2}$$

Fig. 1. Selected formation and typing rules for eMLTT types and terms.

where we require that $FCV(K_z) = FCV(K_s) = z$, and where x_1 and x_2 are chosen fresh.

The *well-formed syntax* of eMLTT is defined using judgments of well-formed value contexts $\vdash \Gamma$, value types $\Gamma \vdash A$, and computation types $\Gamma \vdash \underline{C}$; and well-typed value terms $\Gamma \vdash V : A$, computation terms $\Gamma \vdash M : \underline{C}$, and homomorphism terms $\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K : \underline{D}$, where the *value context* Γ is a list of distinct value variables annotated with value types; the empty context is written $\diamond$. We present selected rules for these judgments in Fig. 1; the full set of typing rules can be found in Appendix A. It is worthwhile to note that the elimination forms for the value types that eMLTT inherits from MLTT are dependently typed, e.g., see the typing rule of `nat-elim`.

Analogously to Ahman et al. (2016), we decorate value, computation, and homomorphism terms with a number of type annotations. We use these annotations to define the denotational semantics of eMLTT on raw expressions, so as to avoid well-known coherence problems arising in the interpretation of dependently typed languages; this is a standard technique in the literature (Hofmann 1997; Streicher 1991). We omit these annotations in examples.

The well-formed syntax of eMLTT is defined mutually with its *equational theory*, consisting of a collection of mutually defined equivalence relations given by *definitional equations* between well-formed value contexts, written $\vdash \Gamma_1 = \Gamma_2$; well-formed types, written $\Gamma \vdash A = B$ and $\Gamma \vdash \underline{C} = \underline{D}$; and well-typed terms, written $\Gamma \vdash V = W : A$, $\Gamma \vdash M = N : \underline{C}$, and $\Gamma \,|\, z \mathbin{:} \underline{C} \vdash K = L : \underline{D}$. We give a selection of these equations in Fig. 2; the full set of definitional equations can be found in Appendix B. The definitional equations interact with the typing rules via context and type conversion rules, e.g.,

$$\frac{\vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash V : A_1 \quad \Gamma_1 \vdash A_1 = A_2}{\Gamma_2 \vdash V : A_2} \qquad \frac{\vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash M : \underline{C}_1 \quad \Gamma_1 \vdash \underline{C}_1 = \underline{C}_2}{\Gamma_2 \vdash M : \underline{C}_2}$$

Value terms:

$$\frac{\Gamma, x\!:\!\mathsf{Nat} \vdash A \quad \Gamma \vdash V_z : A[\mathtt{zero}/x] \quad \Gamma, y_1\!:\!\mathsf{Nat}, y_2\!:\!A[y_1/x] \vdash V_s : A[\mathtt{succ}\ y_1/x]}{\Gamma \vdash \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, \mathtt{zero}) = V_z : A[\mathtt{zero}/x]}$$

$$\frac{\Gamma, x\!:\!\mathsf{Nat} \vdash A \quad \Gamma \vdash V : \mathsf{Nat} \quad \Gamma \vdash V_z : A[\mathtt{zero}/x] \quad \Gamma, y_1\!:\!\mathsf{Nat}, y_2\!:\!A[y_1/x] \vdash V_s : A[\mathtt{succ}\ y_1/x]}{\Gamma \vdash \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, \mathtt{succ}\ V) = V_s[V/y_1][\mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, V)/y_2] : A[\mathtt{succ}\ V/x]}$$

$$\frac{\Gamma \vdash A \quad \Gamma, x_1\!:\!A, x_2\!:\!A, x_3\!:\!x_1 =_A x_2 \vdash B \quad \Gamma \vdash V : A \quad \Gamma, y\!:\!A \vdash W : B[y/x_1][y/x_2][\mathtt{refl}\ y/x_3]}{\Gamma \vdash \mathtt{eq\text{-}elim}_A(x_1.x_2.x_3.B, y.W, V, V, \mathtt{refl}\ V) = W[V/y] : B[V/x_1][V/x_2][\mathtt{refl}\ V/x_3]}$$

$$\frac{\Gamma \vdash V : U\underline{C}}{\Gamma \vdash \mathtt{thunk}\ (\mathtt{force}_{\underline{C}}\ V) = V : U\underline{C}} \qquad \frac{\Gamma \vdash V : \underline{C} \multimap \underline{D}}{\Gamma \vdash V = \lambda z\!:\!\underline{C}.V(z)_{\underline{C},\underline{D}} : \underline{C} \multimap \underline{D}}$$

Computation terms:

$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash \underline{C} \quad \Gamma, x\!:\!A \vdash M : \underline{C}}{\Gamma \vdash \mathtt{return}\ V\ \mathtt{to}\ x\!:\!A\ \mathtt{in}_{\underline{C}}\ M = M[V/x] : \underline{C}} \qquad \frac{\Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma \,|\, z\!:\!FA \vdash K : \underline{C}}{\Gamma \vdash M\ \mathtt{to}\ x\!:\!A\ \mathtt{in}_{\underline{C}}\ K[\mathtt{return}\ x/z] = K[M/z] : \underline{C}}$$

$$\frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash \mathtt{force}_{\underline{C}}\ (\mathtt{thunk}\ M) = M : \underline{C}} \qquad \frac{\Gamma \vdash M : \underline{C} \quad \Gamma \,|\, z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \vdash (\lambda z\!:\!\underline{C}.K)(M)_{\underline{C},\underline{D}} = K[M/z] : \underline{D}}$$

Homomorphism terms:

$$\frac{\Gamma \vdash V : A \quad \Gamma \,|\, z_1\!:\!\underline{C} \vdash K : \underline{D}_1[V/x] \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x\!:\!A \,|\, z_2\!:\!\underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \,|\, z_1\!:\!\underline{C} \vdash \langle V, K\rangle_{(x:A).\underline{D}_1}\ \mathtt{to}\ (x\!:\!A, z_2\!:\!\underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L = L[V/x][K/z_2] : \underline{D}_2}$$

$$\frac{\Gamma, x\!:\!A \vdash \underline{D}_1 \quad \Gamma \,|\, z_1\!:\!\underline{C} \vdash K : \Sigma x\!:\!A.\underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma \,|\, z_3\!:\!\Sigma x\!:\!A.\underline{D}_1 \vdash K : \underline{D}_2}{\Gamma \,|\, z_1\!:\!\underline{C} \vdash K\ \mathtt{to}\ (x\!:\!A, z_2\!:\!\underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L[\langle x, z_2\rangle_{(x:A).\underline{D}_1}/z_3] = L[K/z_3] : \underline{D}_2}$$

$$\frac{\Gamma \vdash \underline{C} \quad \Gamma, x\!:\!A \,|\, z\!:\!\underline{C} \vdash K : \underline{D} \quad \Gamma \vdash V : A}{\Gamma \,|\, z\!:\!\underline{C} \vdash (\lambda x\!:\!A.K)(V)_{(x:A).\underline{D}} = K[V/x] : \underline{D}[V/x]} \qquad \frac{\Gamma, x\!:\!A \vdash \underline{D} \quad \Gamma \,|\, z\!:\!\underline{C} \vdash K : \Pi x\!:\!A.\underline{D}}{\Gamma \,|\, z\!:\!\underline{C} \vdash K = \lambda x\!:\!A.K(x)_{(x:A).\underline{D}} : \Pi x\!:\!A.\underline{D}}$$

Fig. 2. Selected definitional equations from the equational theory of eMLTT.

We note that as eMLTT is based on Martin-Löf's intensional type theory, the elimination form for propositional equality $V =_A W$ supports only a β-equation (see Fig. 2). Similarly, the elimination form for natural numbers also supports only β-equations. In both cases, this is done so as to avoid known sources of undecidability for the equational theory—see the analysis by Hofmann (1995), and Okada and Scott (1999), respectively. We also note that the elimination forms for all other value and computation types come equipped with both β- and η-equations.

Regarding the meta-theory of eMLTT, one can readily prove standard weakening and substitution results, the latter for both value and computation variables. For example, we write $A[V/x]$ for the substitution of V for x in A. Analogously we write $K[M/z]$ for the substitution of M for z in K. The definitions of both kinds of substitution are straightforward: they proceed by recursion on the structure of the given term, making use of the standard convention of identifying types and terms that differ only in the names of bound variables and assuming that in any definition, etc., the bound variables of types and terms are chosen to be different from free variables.

We conclude by recalling that one of the notable features of eMLTT is the computational Σ-type $\Sigma x\!:\!A.\underline{C}$, see Ahman et al. (2016). In particular, the computational Σ-type provides a uniform treatment of type-dependency in sequential composition by allowing one to "close-off" the type of the the second computation with $\Sigma x\!:\!A.\underline{C}$ before using the typing rule for sequential composition which prohibits x to appear in the type of the second computation. A similar restriction on free variables also appears in many other computational typing rules. As a result, eMLTT lends itself to a very natural general denotational semantics based on *fibred* adjunctions, as studied in detail by Ahman et al. (2016). Thus, one says that the computational effects in eMLTT are *fibred*.

3 FIBRED ALGEBRAIC EFFECTS

We now develop a means for specifying computational effects in eMLTT using operations and equations, based on a natural dependently typed generalisation of the effect theories of Plotkin and Pretnar (2013). We note that Ahman et al. (2016) discussed adding algebraic effects only informally.

3.1 Fibred effect theories

We begin by identifying the fragment of eMLTT which we use to define the types of our operations.

A value type is *pure* if it is built up from only Nat, 1, $\Sigma x : A.B$, $\Pi x : A.B$, 0, $A + B$, and $V =_A W$, where V, W, and A are all pure in propositional equality $V =_A W$. A value term is *pure* if it does not contain thunked computations and homomorphic lambda abstractions, and all its type annotations are pure. This notion of pureness extends straightforwardly to contexts—a value context Γ is *pure* if A_i is pure for every $x_i : A_i \in \Gamma$. Note that this fragment of eMLTT corresponds precisely to MLTT.

Assuming a countable set of *effect variables* $w, \ldots$, we now define our notion of fibred effect theory. We begin by defining corresponding signatures of operation symbols and then add equations between derivable effect terms, so as to specify both the effects at hand and their behaviour.

A *fibred effect signature* $\mathcal{S}_{\text{eff}}$ consists of a finite set of typed operation symbols $\mathsf{op} : (x : I) \longrightarrow O$, where $\diamond \vdash I$ and $x : I \vdash O$ are required to be pure value types, called the *input* and *output* type of op.

The *effect terms* T that one can derive from the given fibred effect signature $\mathcal{S}_{\text{eff}}$ are given by

$$\begin{array}{rcl} T & ::= & w\,(V) \mid \mathsf{op}_V(y.T) \mid \mathsf{pm}\ V\ \mathsf{as}\ (x_1 : A_1, x_2 : A_2)\ \mathsf{in}\ T \\ & \mid & \mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(x_1 : A_1) \mapsto T_1, \mathsf{inr}(x_2 : A_2) \mapsto T_2) \end{array}$$

with the involved value types and value terms all required to be pure. We use the convention of omitting V in $\mathsf{op}_V(y.T)$ when the input type of op is 1, and y when the output type of op is 1.

An *effect context* Δ is a list of distinct effect variables annotated with pure value types. We say that Δ is *well-formed* in a pure value context Γ, written $\Gamma \vdash \Delta$, if $\vdash \Gamma$ and $\Gamma \vdash A_i$ for every $w_j : A_j \in \Delta$.

The *well-formed* effect terms are defined using the judgement $\Gamma \mid \Delta \vdash T$ as follows:

$$\frac{\Gamma \vdash \Delta_1, w : A, \Delta_2 \quad \Gamma \vdash V : A}{\Gamma \mid \Delta_1, w : A, \Delta_2 \vdash w\,(V)} \qquad \frac{\Gamma \vdash V : \Sigma x_1 : A_1.A_2 \quad \Gamma \vdash \Delta \quad \Gamma, x_1 : A_1, x_2 : A_2 \mid \Delta \vdash T}{\Gamma \mid \Delta \vdash \mathsf{pm}\ V\ \mathsf{as}\ (x_1 : A_1, x_2 : A_2)\ \mathsf{in}\ T}$$

$$\frac{\Gamma \vdash V : I \quad \Gamma \vdash \Delta \quad \Gamma, y : O[V/x] \mid \Delta \vdash T}{\Gamma \mid \Delta \vdash \mathsf{op}_V(y.T)} \qquad \frac{\Gamma \vdash V : A_1 + A_2 \quad \Gamma \vdash \Delta \quad \Gamma, x_i : A_i \mid \Delta \vdash T_i \quad i \in \{1, 2\}}{\Gamma \mid \Delta \vdash \mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(x_1 : A_1) \mapsto T_1, \mathsf{inr}(x_2 : A_2) \mapsto T_2)}$$

Finally, a *fibred effect theory* $\mathcal{T}_{\text{eff}} = (\mathcal{S}_{\text{eff}}, \mathcal{E}_{\text{eff}})$ is given by a fibred effect signature $\mathcal{S}_{\text{eff}}$ and a finite set $\mathcal{E}_{\text{eff}}$ of equations $\Gamma \mid \Delta \vdash T_1 = T_2$ between well-formed effect terms $\Gamma \mid \Delta \vdash T_1$ and $\Gamma \mid \Delta \vdash T_2$.

In order to simplify the presentation of typing rules involving fibred effect theories, we assume $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and $\Delta = w_1 : A'_1, \ldots, w_m : A'_m$ when quantifying over the variables of Γ, Δ.

3.2 Examples of fibred effect theories

As our fibred effect theories are a natural dependently typed generalisation of Plotkin and Pretnar's effect theories, we can capture all the effects they can, e.g., assuming a pure value type $\diamond \vdash \mathsf{Exc}$ of exception names, the *theory $\mathcal{T}_{EXC}$ of exceptions* is given by one operation symbol $\mathsf{raise} : \mathsf{Exc} \longrightarrow 0$ and no equations. Another standard example is the *theory $\mathcal{T}_{ND}$ of nondeterminism*, which is given by one operation symbol $\mathsf{or} : 1 \longrightarrow 1 + 1$ and three equations that make or into a semilattice.

However, it is worth noting that compared to Plotkin and Pretnar's effect theories, our dependently typed operation symbols allow us to capture more precise notions of computation. We discuss two such examples below: i) global state in which the store types are dependent on locations; and ii) dependently typed update monads that model state in which the store is changed not by overwriting but instead by applying (store-dependent) updates to it, examples of which include non-overflowing buffers and non-underflowing stacks—see Ahman and Uustalu (2014).

Global state. Assuming given pure value types of *memory locations* and *values* stored at them:

$$\diamond \vdash \mathsf{Loc} \qquad x\,{:}\,\mathsf{Loc} \vdash \mathsf{Val}$$

the fibred effect signature $\mathcal{S}_{GS}$ of global state is given by the following two operation symbols:

$$\mathsf{get} : (x\,{:}\,\mathsf{Loc}) \longrightarrow \mathsf{Val} \qquad \mathsf{put} : \Sigma x\,{:}\,\mathsf{Loc}.\mathsf{Val} \longrightarrow 1$$

The idea here is that get denotes an effectful command that returns the current value of the store at the given location; and put denotes a command that overwrites the store at the given location.

The corresponding fibred effect theory $\mathcal{T}_{\mathrm{GS}}$ is then given by the following five equations:

$$x\,{:}\,\mathsf{Loc} \mid w\,{:}\,1 \vdash \mathsf{get}_x(y.\mathsf{put}_{\langle x,y\rangle}(w\,(\star))) = w\,(\star)$$

$$x\,{:}\,\mathsf{Loc}, y\,{:}\,\mathsf{Val} \mid w\,{:}\,\mathsf{Val} \vdash \mathsf{put}_{\langle x,y\rangle}(\mathsf{get}_x(y'.w\,(y'))) = \mathsf{put}_{\langle x,y\rangle}(w\,(y))$$

$$x\,{:}\,\mathsf{Loc}, y_1\,{:}\,\mathsf{Val}, y_2\,{:}\,\mathsf{Val} \mid w\,{:}\,1 \vdash \mathsf{put}_{\langle x,y_1\rangle}(\mathsf{put}_{\langle x,y_2\rangle}(w\,(\star))) = \mathsf{put}_{\langle x,y_2\rangle}(w\,(\star))$$

$$\begin{aligned} x_1\,{:}\,\mathsf{Loc}, x_2\,{:}\,\mathsf{Loc} \mid w\,{:}\,\mathsf{Val}[x_1/x] \times \mathsf{Val}[x_2/x] \vdash\ & \mathsf{get}_{x_1}(y_1.\mathsf{get}_{x_2}(y_2.w\,(\langle y_1, y_2\rangle))) \\ =\ & \mathsf{get}_{x_2}(y_2.\mathsf{get}_{x_1}(y_1.w\,(\langle y_1, y_2\rangle))) \qquad (x_1 \neq x_2) \end{aligned}$$

$$\begin{aligned} x_1\,{:}\,\mathsf{Loc}, x_2\,{:}\,\mathsf{Loc}, y_1\,{:}\,\mathsf{Val}[x_1/x], y_2\,{:}\,\mathsf{Val}[x_2/x] \mid w\,{:}\,1 \vdash\ & \mathsf{put}_{\langle x_1,y_1\rangle}(\mathsf{put}_{\langle x_2,y_2\rangle}(w\,(\star))) \\ =\ & \mathsf{put}_{\langle x_2,y_2\rangle}(\mathsf{put}_{\langle x_1,y_1\rangle}(w\,(\star))) \qquad (x_1 \neq x_2) \end{aligned}$$

where the last two include a side-condition that requires the locations x_1 and x_2 to be different. Similarly to Plotkin and Pretnar's effect theories, this an informal shorthand notation. Formally, we require the type Loc to come with decidable equality (for simplicity, boolean-valued), given by a pure value $\diamond \vdash \mathsf{eq} : \mathsf{Loc} \to \mathsf{Loc} \to 1 + 1$, and then write the right-hand sides of these equations using case analysis on $\mathsf{eq}\,x_1\,x_2$, e.g., the right-hand side of the last equation is formally written as

$$\mathsf{case}\ (\mathsf{eq}\,x_1\,x_2)\ \mathsf{of}\ \left(\mathsf{inl}(x_1') \mapsto \mathsf{put}_{\langle x_1,y_1\rangle}(\mathsf{put}_{\langle x_2,y_2\rangle}(w\,(\star))), \mathsf{inr}(x_2') \mapsto \mathsf{put}_{\langle x_2,y_2\rangle}(\mathsf{put}_{\langle x_1,y_1\rangle}(w\,(\star)))\right)$$

Observe that these five equations describe the expected behaviour of global state: trivial store changes are not observable (1st equation); `get` returns the most recent value the store has been set to (2nd equation); `put` overwrites the content of the store (3rd equation); and `gets` and `puts` at different locations are independent and commute with each other (4th and 5th equation).

Dependently typed update monads. We assume given pure value types

$$\diamond \vdash \mathsf{St} \qquad x\,{:}\,\mathsf{St} \vdash \mathsf{Upd}$$

of *store values* and *store updates*, respectively, together with well-typed closed pure value terms

$$\downarrow\,: \Pi x\,{:}\,\mathsf{St}.\mathsf{Upd} \to \mathsf{St} \qquad \mathsf{o} : \Pi x\,{:}\,\mathsf{St}.\mathsf{Upd} \qquad \oplus : \Pi x\,{:}\,\mathsf{St}.\Pi y\,{:}\,\mathsf{Upd}.\mathsf{Upd}[x \downarrow y/x] \to \mathsf{Upd}$$

satisfying the following definitional equations (in the pure fragment of eMLTT's equational theory; for better readability, we omit contexts and types, and write the first argument to $\oplus$ as a subscript):

$$V \downarrow (\mathsf{o}\,V) = V \qquad V \downarrow (W_1 \oplus_V W_2) = (V \downarrow W_1) \downarrow W_2$$

$$W \oplus_V (\mathsf{o}\,(V \downarrow W)) = W \qquad (\mathsf{o}\,V) \oplus_V W = W \qquad (W_1 \oplus_V W_2) \oplus_V W_3 = W_1 \oplus_V (W_2 \oplus_{V \downarrow W_1} W_3)$$

The signature $\mathcal{S}_{\mathrm{UPD}}$ of a dependently typed update monad is then given by two operation symbols:

$$\mathsf{lookup} : 1 \longrightarrow \mathsf{St} \qquad \mathsf{update} : \Pi x\,{:}\,\mathsf{St}.\mathsf{Upd} \longrightarrow 1$$

The high-level idea is that $(\mathsf{Upd}, \mathsf{o}, \oplus)$ forms a dependently typed monoid of updates which can be applied to the store values via its action $\downarrow$ on St; lookup denotes an effectful command that returns the current value of the store; and update denotes an effectful command that applies an appropriate update to the current store (from the family of updates given as its input). The dependency of Upd on St gives us fine-grain control over which updates are applicable to which store values, and allows this to be enforced during typechecking. It is worth noting that in the literature the 5-tuple $(\mathsf{St}, \mathsf{Upd}, \downarrow, \mathsf{o}, \oplus)$ is also known under the name of *directed containers* (Ahman et al. 2014).

The corresponding fibred effect theory $\mathcal{T}_{\mathrm{UPD}}$ is then given by the following three equations:

$$\diamond \mid w\,{:}\,1 \vdash \mathsf{lookup}(x.\mathsf{update}_{\lambda y:\mathsf{St}.\mathsf{o}\, y}(w\,(\star))) = w\,(\star)$$

$$\begin{aligned} x\,{:}\,(\Pi x'\,{:}\,\mathsf{St}.\mathsf{Upd}[x'/x]) \mid w\,{:}\,\mathsf{St}\times\mathsf{St} \vdash\ & \mathsf{lookup}(y.\mathsf{update}_x(\mathsf{lookup}(y'.w\,(\langle y, y'\rangle)))) \\ & = \mathsf{lookup}(y.\mathsf{update}_x(w\,(\langle y, y \downarrow (x\, y)\rangle))))\end{aligned}$$

$$\begin{aligned} x\,{:}\,(\Pi x'\,{:}\,\mathsf{St}.\mathsf{Upd}[x'/x]), y\,{:}\,(\Pi y'\,{:}\,\mathsf{St}.\mathsf{Upd}[y'/x]) \mid w\,{:}\,1 \vdash\ & \mathsf{update}_x(\mathsf{update}_y(w\,(\star))) \\ & = \mathsf{update}_{\lambda x''.(x\, x'') \oplus_{x''} (y\,(x'' \downarrow (x\, x'')))}(w\,(\star))\end{aligned}$$

These equations are similar to the first three equations of the global state theory $\mathcal{T}_{\mathrm{GS}}$, but instead of an overwriting behaviour, they describe how the store is changed using updates. In particular, observe how $\oplus$ is used to combine subsequent updates, and how o gives us "do nothing" updates.

3.3 Extending eMLTT with fibred algebraic effects

We now show how to extend eMLTT with algebraic effects given by a fibred effect theory $\mathcal{T}_{\mathrm{eff}}$.

First, we extend the grammar of eMLTT's computation terms with *algebraic operations*:

$$M \ ::= \ \ldots \ \mid \ \mathsf{op}^{\underline{C}}_V(y.M)$$

for all $\mathsf{op} : (x\,{:}\,I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$ and computation types $\underline{C}$.

Next, in order to extend the well-formed syntax of eMLTT with a corresponding typing rule and definitional equations, we first define a translation of effect terms into value terms. In particular, given an effect term $\Gamma \mid \Delta \vdash T$, a value type A, value terms V_i (for all $x_i\,{:}\,A_i \in \Gamma$), value terms V'_j (for all $w_j\,{:}\,A'_j \in \Delta$), and value terms W_{op} (for all $\mathsf{op} : (x\,{:}\,I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$), we define the *translation* of T into a value term $(\!|\, T \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\mathrm{op}}}}$ by recursion on the structure of T, as given in detail in Fig. 3. For better readability, we write $\overrightarrow{V_i}$ for $\{V_1, \ldots, V_n\}$ in the subscripts, and analogously for $\overrightarrow{V'_j}$ and $\overrightarrow{W_{\mathrm{op}}}$.

While we omit the subscripts in Fig. 3 so as to improve readability, it is important to note that in the cases where the given effect term involves variable bindings, the set of value terms $\overrightarrow{V_i}$ is extended with the corresponding variables in the right-hand side, e.g., in the second case we have

$$(\!|\, \mathsf{op}_V(y.T) \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\mathrm{op}}}} \overset{\mathrm{def}}{=} W_{\mathrm{op}}\ \langle V[\overrightarrow{V_i}/\overrightarrow{x_i}], \lambda y\,{:}\,O[V[\overrightarrow{V_i}/\overrightarrow{x_i}]/x].(\!|\, T \,|\!)_{A;\overrightarrow{V_i},y;\overrightarrow{V'_j};\overrightarrow{W_{\mathrm{op}}}}\rangle$$

It is also worth observing that while it might be more intuitive and natural to translate effect terms directly into computation terms, giving the translation from effect terms into value terms allows us to reuse it later in §4 where we extend eMLTT with handlers of fibred algebraic effects.

Further, we only translate well-formed effect terms $\Gamma \mid \Delta \vdash T$ because it makes it easier to account for substituting value terms for effect variables—various subsequent results refer to substituting value terms for all effect variables in Δ, not just for the free effect variables appearing in T.

$$
\begin{array}{ll}
(\!| w_j\,(V) |\!) & \stackrel{\text{def}}{=} V_j'\ (V[\overrightarrow{V_i}/\overrightarrow{x_i}]) \\
(\!| \mathsf{op}_V(y.T) |\!) & \stackrel{\text{def}}{=} W_{\mathsf{op}}\ \langle V[\overrightarrow{V_i}/\overrightarrow{x_i}], \lambda y\!:\!O[V[\overrightarrow{V_i}/\overrightarrow{x_i}]/x].(\!| T |\!)\rangle \\
(\!| \mathsf{pm}\ V\ \mathsf{as}\ (y_1\!:\!B_1, y_2\!:\!B_2)\ \mathsf{in}\ T |\!) & \stackrel{\text{def}}{=} \mathtt{pm}\ V[\overrightarrow{V_i}/\overrightarrow{x_i}]\ \mathtt{as}\ (y_1\!:\!B_1[\overrightarrow{V_i}/\overrightarrow{x_i}], y_2\!:\!B_2[\overrightarrow{V_i}/\overrightarrow{x_i}])\ \mathtt{in}\ (\!| T |\!) \\
(\!| \mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(y_1\!:\!B_1) \mapsto T_1, \mathsf{inr}(y_2\!:\!B_2) \mapsto T_2) |\!) & \stackrel{\text{def}}{=} \mathtt{case}\ V[\overrightarrow{V_i}/\overrightarrow{x_i}]\ \mathtt{of}\ (\mathtt{inl}(y_1\!:\!B_1[\overrightarrow{V_i}/\overrightarrow{x_i}]) \mapsto (\!| T_1 |\!), \\
& \qquad\qquad \mathtt{inr}(y_2\!:\!B_2[\overrightarrow{V_i}/\overrightarrow{x_i}]) \mapsto (\!| T_2 |\!))
\end{array}
$$

Fig. 3. Translation of effect terms into value terms.

Typing rule for algebraic operations:

$$\frac{\Gamma \vdash V : I \quad \Gamma \vdash \underline{C} \quad \Gamma, y\!:\!O[V/x] \vdash M : \underline{C}}{\Gamma \vdash \mathsf{op}^{\underline{C}}_V(y.M) : \underline{C}}$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$.

Congruence equations:

$$\frac{\Gamma \vdash V = W : I \quad \Gamma \vdash \underline{C} = \underline{D} \quad \Gamma, y\!:\!O[V/x] \vdash M = N : \underline{C}}{\Gamma \vdash \mathsf{op}^{\underline{C}}_V(y.M) = \mathsf{op}^{\underline{D}}_W(y.N) : \underline{C}}$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$.

General algebraicity equations:

$$\frac{\Gamma \vdash V : I \quad \Gamma, y\!:\!O[V/x] \vdash M : \underline{C} \quad \Gamma \,|\, z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \vdash K[\mathsf{op}^{\underline{C}}_V(y.M)/z] = \mathsf{op}^{\underline{D}}_V(y.K[M/z]) : \underline{D}}$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$.

Equations of the given fibred effect theory:

$$\frac{\begin{array}{ll} \mathit{Vars}(\Gamma') \cap \mathit{Vars}(\Gamma) = \emptyset & \\ \Gamma' \vdash V_i : A_i[V_1/x_1, \ldots, V_{i-1}/x_{i-1}] & (1 \le i \le n) \\ \Gamma' \vdash \underline{C} \quad \Gamma' \vdash V_j' : A_j'[\overrightarrow{V_i}/\overrightarrow{x_i}] \to U\underline{C} & (1 \le j \le m) \end{array}}{\Gamma' \vdash (\!| T_1 |\!)_{U\underline{C};\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{W_{\mathsf{op}}}} = (\!| T_2 |\!)_{U\underline{C};\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{W_{\mathsf{op}}}} : U\underline{C}}$$

for all $\Gamma \,|\, \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$, with the value terms $\Gamma' \vdash W_{\mathsf{op}} : (\Sigma x\!:\!I.O \to U\underline{C}) \to U\underline{C}$ given by

$$W_{\mathsf{op}} \stackrel{\text{def}}{=} \lambda x'\!:\!(\Sigma x\!:\!I.O \to U\underline{C}).\mathtt{pm}\ x'\ \mathtt{as}\ (x\!:\!I, y\!:\!O \to U\underline{C})\ \mathtt{in}\ \mathtt{thunk}\ (\mathsf{op}^{\underline{C}}_x(y'.\mathtt{force}_{\underline{C}}\ (y\ y')))$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$.

Fig. 4. Rules for extending EMLTT with fibred algebraic effects.

Using this translation of effect terms into value terms, we can now define the typing rule and definitional equations for algebraic operations, as given in Fig. 4. It is worth noting that for presentational convenience we include the equations given in $\mathcal{E}_{\text{eff}}$ as definitional equations between value terms. The corresponding equations between computation terms are easily derivable, e.g.,

$$\Gamma \vdash \mathtt{get}^{\underline{C}}_V(y.\mathtt{put}^{\underline{C}}_{\langle V, y\rangle}(M)) = M : \underline{C}$$

can be derived from the translation of the corresponding equation in the global state theory $\mathcal{T}_{\text{GS}}$.

Finally, it is important to note that we impose a disjointness requirement on value contexts in the last group of equations in Fig 4. We do so to ensure that the substitution theorem (Thm. 6.3) goes through. While at first sight this requirement might seem limiting on the number of definitional equations we can derive, it turns out that after having proved the substitution theorem, the corresponding definitional equation without the disjointness requirement is derivable—see Prop. 6.7.

4 HANDLERS VIA THE USER-DEFINED ALGEBRA TYPE

4.1 A problem with adding conventional handlers to eMLTT

Before we show how to extend eMLTT with handlers of fibred algebraic effects using our user-defined algebra type, we first explain how extending eMLTT with the conventional term-level definition of handlers leads to *unsound* program equivalences becoming derivable.

First, recall from §1 that Plotkin and Pretnar (and others since) include handlers in effectful languages by extending the syntax of computation terms with the following handling construct:

$$M \;\texttt{handled with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \;\texttt{to}\; y{:}A \;\texttt{in}\; N_{\mathrm{ret}}$$

whose semantics is given using the mediating homomorphism from the free algebra over A to the algebra denoted by the handler $\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}}$. However, when extending a language that includes a notion of homomorphism, such as eMLTT with its homomorphism terms, this algebraic understanding of handlers suggests that one ought to also extend the given notion of homomorphism with a corresponding handling construct. Unfortunately, if one simply adds

$$K \;\texttt{handled with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \;\texttt{to}\; y{:}A \;\texttt{in}\; N_{\mathrm{ret}}$$

to eMLTT, the combination of the general algebraicity equation (Fig. 4) and the β-equations associated with the handling construct (§1) now gives rise to unsound definitional equations.

To explain this problem in more detail, let us consider the theory $\mathcal{T}_{\mathrm{I/O}}$ of *interactive input-output of bits*, given by two operation symbols $\mathsf{read} : 1 \longrightarrow 1 + 1$ and $\mathsf{write} : 1 + 1 \longrightarrow 1$, and no equations.

Next, let us consider a handler that negates all bits written to the output, as given by

$$\{\; \mathtt{read}(x') \mapsto \mathtt{read}^{F1}(y.\mathtt{force}\;(x'\,y)) \;,\; \mathtt{write}_x(x') \mapsto \mathtt{write}^{F1}_{\neg x}(\mathtt{force}\;(x'\,\star)) \;\}$$

where $\neg : 1 + 1 \to 1 + 1$ is a pure value term that swaps the left and right injections.

Now, let us consider handling a simple program, $\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\;\star)$, using the handler we defined above. On the one hand, using the β-equations for handling (see §1), we can prove that

$$\begin{aligned}\Gamma \vdash\; & (\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\;\star)) \;\texttt{handled with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{I/O}}} \;\texttt{to}\; y{:}1 \;\texttt{in return}\;\star \\ & = \mathtt{write}^{F1}_{\mathtt{inr}\,\star}(\mathtt{return}\;\star) : F1\end{aligned}$$

On the other hand, using the general algebraicity equation (see Fig. 4), which ensures that homomorphism terms indeed behave as if they were algebra homomorphisms, we can prove that

$$\begin{aligned}\Gamma \vdash\; & (\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\;\star)) \;\texttt{handled with}\; \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{I/O}}} \;\texttt{to}\; y{:}1 \;\texttt{in return}\;\star \\ & = \mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\;\star) : F1\end{aligned}$$

Clearly, this equation is sound only if we would have $\mathsf{inl}\,\star = \mathsf{inr}\,\star$ in the semantics. The reason for this discrepancy lies in the term-level definition of handlers in their conventional presentation. In particular, while the homomorphic behaviour of homomorphism terms is determined exclusively by the computation types involved (via the general algebraicity equation), the type of the above handling construct contains no trace of the algebra denoted by the handler $\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{I/O}}}$.

It is worth noting that this problem is not inherent to eMLTT but would also arise in the simply typed setting, e.g., when combining handlers of algebraic effects with CBPV and its stack terms, or with EEC and its linear (computation) terms. Finally, we note that the reason why Plotkin and Pretnar (2013) were able to give a sound denotational semantics to their language was precisely due to their choice of using CBPV *without* stack terms, i.e., with only value and computation terms.

4.2 Extending eMLTT with the user-defined algebra type

In this section we solve the problems of §4.1 by giving handlers a novel type-based treatment that internalises Plotkin and Pretnar's insight that they denote algebras for the given effect theory.

First, given a fibred effect theory $\mathcal{T}_{\text{eff}}$, we extend eMLTT with the *user-defined algebra type*:

$$\underline{C} \ ::= \ \ldots \ \mid \langle A, \{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle$$

which pairs a value type A (the carrier) with a family of value terms $\{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}}$ (the operations).

We also extend eMLTT's computation and homomorphism terms with two *composition operations*:

$$M \ ::= \ \ldots \ \mid M \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N \qquad\qquad K \ ::= \ \ldots \ \mid K \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$$

which provide elimination forms for the user-defined algebra type when $\underline{C}$ is $\langle A, \{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle$.

It is worth noting that in principle we could have restricted these composition operations to only the user-defined algebra type $\langle A, \{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle$, but then we would not have been able to derive a useful computation type isomorphism to coerce computations between a general $\underline{C}$ and its canonical representation as a user-defined algebra type. We construct this type isomorphism in Prop. 4.1.

Conceptually, these composition operations are a form of explicit substitution of thunked computations for value variables. For example, in this extension of eMLTT we will be able to show that the computation term $M \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$ is definitionally equal to $N[\mathtt{thunk}\, M/x]$. As such, the value variable x refers to the whole of (the thunk of) M, compared to, e.g., sequential composition $M \,\mathtt{to}\, x \,{:}\, A \,\mathtt{in}\, N$, where the value variable x is used to bind the value produced by M.

It is however important to note that we impose some conditions on how the value variables x can be used in these composition operations. In particular, the typing rules of $M \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$ and $K \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$ require that x is used in N as if it was a computation variable, in that x must not be duplicated or discarded arbitrarily. We do so as to ensure that N behaves as if it was a homomorphism term, meaning that in $M \,\mathtt{as}\, x \,{:}\, U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$ the effects of M are guaranteed to "happen before" those of N. However, rather than trying to extend eMLTT further with some form of linearity for such value variables, we impose these requirements via equational proof obligations, requiring that N commutes with algebraic operations (when substituted for x using thunking).

We make this discussion formal in Fig. 5 by giving the rules for extending eMLTT's well-formed syntax and equational theory with the user-defined algebra type and composition operations.

In the rules concerning the user-defined algebra type, we use the following *auxiliary judgment*:

$$\Gamma' \vdash \{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \,\mathsf{on}\, A$$

which holds iff the value terms V_{op} form an algebra *on* A, i.e., iff $\Gamma' \vdash A$, $\Gamma' \vdash V_{\text{op}} : (\Sigma x \,{:}\, I.O \to A) \to A$ (for all $\text{op} : (x \,{:}\, I) \longrightarrow O$ in $\mathcal{S}_{\text{eff}}$), and we can prove for all $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$ that

$$\Gamma', \Gamma \vdash \overrightarrow{\lambda x_{w_j} \,{:}\, A'_j \to A.} (\!| \, T_1 \, |\!)_{A; \overrightarrow{x_i}; \overrightarrow{x_{w_j}}; \overrightarrow{V_{\text{op}}}} = \overrightarrow{\lambda x_{w_j} \,{:}\, A'_j \to A.} (\!| \, T_2 \, |\!)_{A; \overrightarrow{x_i}; \overrightarrow{x_{w_j}}; \overrightarrow{V_{\text{op}}}} : \overrightarrow{A'_j \to A} \to A$$

where we write $\overrightarrow{\lambda x_{w_j} \,{:}\, A'_j \to A.}$ for the lambda abstractions $\lambda x_{w_1} : A'_1 \to A. \ldots \lambda x_{w_m} : A'_m \to A.$, and $\overrightarrow{A'_j \to A}$ for the corresponding sequence of function types $(A'_1 \to A) \to \ldots \to (A'_m \to A)$.

It is worth noting that if one works exclusively with equation-free fibred effect theories, e.g., as used in simply typed languages (Bauer and Pretnar 2015; Hillerström and Lindley 2016) and discussed in §1, then the equational proof obligations given by the judgement $\Gamma' \vdash \{V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \,\mathsf{on}\, A$ hold vacuously, and thus do not put any additional burden on the programmer. However, the

Formation rule for the user-defined algebra type:

$$\frac{\Gamma \vdash \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \text{ on } A}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}$$

Typing rules for the composition operations:

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}}{\Gamma \vdash M \text{ as } x\!:\!U\underline{C} \text{ in}_{\underline{D}}\ N : \underline{D}} \qquad \frac{\Gamma \,|\, z\!:\!\underline{C} \vdash K : \underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x\!:\!U\underline{D}_1 \vdash_{\mathsf{hom}} M : \underline{D}_2}{\Gamma \,|\, z\!:\!\underline{C} \vdash K \text{ as } x : U\underline{D}_1 \text{ in}_{\underline{D}_2}\ M : \underline{D}_2}$$

Congruence equations:

$$\frac{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle \quad \Gamma \vdash \langle B, \{W_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle \quad \Gamma \vdash A = B \qquad \Gamma \vdash V_{\mathsf{op}} = W_{\mathsf{op}} : (\Sigma x\!:\!I.O \to A) \to A \quad (\mathsf{op} : (x\!:\!I) \longrightarrow O)}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle = \langle B, \{W_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}$$

plus similar two equations for composition operations.

β-equation for the user-defined algebra type:

$$\frac{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}{\Gamma \vdash U\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle = A}$$

β-equation for the composition operations:

$$\frac{\Gamma \vdash V : U\underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x\!:\!U\underline{C} \vdash_{\mathsf{hom}} M : \underline{D}}{\Gamma \vdash (\mathsf{force}_{\underline{C}}\, V) \text{ as } x\!:\!U\underline{C} \text{ in}_{\underline{D}}\ M = M[V/x] : \underline{D}}$$

η-equations for the composition operations:

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \,|\, z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \vdash M \text{ as } x\!:\!U\underline{C} \text{ in}_{\underline{D}}\ K[\mathsf{force}_{\underline{C}}\, x/z] = K[M/z] : \underline{D}}$$

$$\frac{\Gamma \,|\, z_1\!:\!\underline{C} \vdash K : \underline{D}_1 \quad \Gamma \,|\, z_2\!:\!\underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \,|\, z_1\!:\!\underline{C} \vdash K \text{ as } x\!:\!U\underline{D}_1 \text{ in}_{\underline{D}_2}\ L[\mathsf{force}_{\underline{D}_1}\, x/z_2] = L[K/z_2] : \underline{D}_2}$$

η-equation for algebraic operations:

$$\frac{\Gamma \vdash V : I \quad \Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle \quad \Gamma, y\!:\!O[V/x] \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}{\Gamma \vdash \mathsf{op}_V^{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}(y.M) = \mathsf{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}\,(V_{\mathsf{op}}\, \langle V, \lambda y\!:\!O[V/x].\mathsf{thunk}\, M \rangle) : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle}$$

Fig. 5. Rules for extending eMLTT with the user-defined algebra type.

possibility of also being able to specify effects using equations ensures that the fit-for-purpose handler implementations of given notion of computation (say, global state) are indeed correct.

In the rules concerning the composition operations, we use the following *auxiliary judgment*:

$$\Gamma, y\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$$

which holds iff N behaves like a homomorphism from the algebra denoted by $\underline{C}$ to the algebra denoted by $\underline{D}$, i.e., iff $\Gamma, y\!:\!U\underline{C} \vdash N : \underline{D}$ and we can prove for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$ that

$$\Gamma \vdash \lambda x.\lambda x'.N[\mathsf{thunk}\,(\mathsf{op}_x^{\underline{C}}(y'.\mathsf{force}_{\underline{C}}\,(x'\,y')))/y] = \lambda x.\lambda x'.\mathsf{op}_x^{\underline{D}}(y'.N[x'\,y'/y]) : \Pi x\!:\!I.(O \to U\underline{C}) \to \underline{D}$$

where the omitted type annotations on x and x' are given by the value types I and $O \to U\underline{C}$.

It is worth noting that the equational proof obligations given by the judgement $\Gamma, y\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$ are reminiscent of the equational conditions used by Pédrot and Tabareau (2017) to require linearity in type-dependence so as to ensure the correctness of their monadic translation of type theory.

Regarding the definitional equations we presented in Fig. 5, observe that the β-equation for the user-defined algebra type captures the intuition that the value type A denotes the carrier of the algebra denoted by $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$. Analogously, the η-equation for algebraic operations captures the intuition that V_{op} denote the operations associated with the algebra denoted by $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$.

It is also worth noting that we do not include an η-equation for $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$. We do so because it does not hold in the natural denotational semantics we develop for this extension of eMLTT in §8. Instead, as promised earlier, we can construct a corresponding computation type isomorphism.

Proposition 4.1. *Given a computation type* $\Gamma \vdash \underline{C}$*, there exists a computation type isomorphism* $\Gamma \vdash \underline{C} \cong \langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}}\rangle$*, where each value term* $\Gamma \vdash V_{\mathsf{op}} : (\Sigma x\!:\!I.O \to U\underline{C}) \to U\underline{C}$ *is given by*

$$V_{\mathsf{op}} \stackrel{\mathit{def}}{=} \lambda y\!:\!(\Sigma x\!:\!I.O \to U\underline{C}).\mathtt{pm}\; y \;\mathtt{as}\; (x\!:\!I, x'\!:\!O \to U\underline{C}) \;\mathtt{in}\; \mathtt{thunk}(\mathtt{op}^{\underline{C}}_{x}(y'.\mathtt{force}_{\underline{C}}\,(x'\,y')))$$

Proof. This type isomorphism is witnessed by the following homomorphic functions:

$$\Gamma \vdash \lambda z\!:\!\underline{C}\,.\,z \;\mathtt{as}\; x\!:\!U\underline{C} \;\mathtt{in}\; \mathtt{force}_{\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\, x : \underline{C} \multimap \langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$$

$$\Gamma \vdash \lambda z\!:\!\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle\,.\,z \;\mathtt{as}\; x\!:\!U\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle \;\mathtt{in}\; \mathtt{force}_{\underline{C}}\, x : \langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle \multimap \underline{C}$$

□

4.3 Deriving the term-level definition of handlers

In this section we show how to derive the conventional term-level definition of handlers from our type-based treatment. In particular, we define the handling construct using sequential composition:

$$M \;\mathtt{handled\;with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}_{\underline{C}}\; N_{\mathsf{ret}}$$
$$\stackrel{\mathrm{def}}{=}$$
$$\mathtt{force}_{\underline{C}}\,(\mathtt{thunk}\,(M \;\mathtt{to}\; y\!:\!A \;\mathtt{in}\; \mathtt{force}_{\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\,(\mathtt{thunk}\; N_{\mathsf{ret}})))$$

where each well-formed value term $\Gamma \vdash V_{\mathsf{op}} : (\Sigma x\!:\!I.O \to U\underline{C}) \to U\underline{C}$ is defined as follows:

$$V_{\mathsf{op}} \stackrel{\mathrm{def}}{=} \lambda y'\!:\!(\Sigma x\!:\!I.O \to U\underline{C}).\mathtt{pm}\; y' \;\mathtt{as}\; (x\!:\!I, x'\!:\!O \to U\underline{C}) \;\mathtt{in}\; \mathtt{thunk}\; N_{\mathsf{op}}$$

The expected typing rule and the two β-equations are then derivable for this handling construct.

Proposition 4.2. *The following typing rule is derivable:*

$$\frac{\begin{array}{c}\Gamma, x\!:\!I, x'\!:\!O \to U\underline{C} \vdash N_{\mathsf{op}} : \underline{C} \quad (\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\mathit{eff}}) \\ \Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, y\!:\!A \vdash N_{\mathsf{ret}} : \underline{C} \quad \Gamma \vdash \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}} \;\mathtt{on}\; U\underline{C}\end{array}}{\Gamma \vdash M \;\mathtt{handled\;with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}_{\underline{C}}\; N_{\mathsf{ret}} : \underline{C}}$$

where each value term V_{op} *is derived from the corresponding computation term* N_{op}*, as defined above.*

Proposition 4.3. *The following definitional* β*-equations are derivable:*

$$\frac{\begin{array}{c}\Gamma, x\!:\!I, x'\!:\!O \to U\underline{C} \vdash N_{\mathsf{op}} : \underline{C} \qquad (\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\mathit{eff}}) \\ \Gamma \vdash V : I \quad \Gamma, y'\!:\!O[V/x] \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, y\!:\!A \vdash N_{\mathsf{ret}} : \underline{C} \quad \Gamma \vdash \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}} \;\mathtt{on}\; U\underline{C}\end{array}}{\begin{array}{l}\Gamma \vdash (\mathtt{op}^{FA}_{V}(y'.M)) \;\mathtt{handled\;with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}_{\underline{C}}\; N_{\mathsf{ret}} \\ \quad = N_{\mathsf{op}}[V/x][\lambda y'\!:\!O[V/x].\mathtt{thunk}\; H/x'] : \underline{C}\end{array}}$$

where

$$H \stackrel{\mathit{def}}{=} M \;\mathtt{handled\;with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}_{\underline{C}}\; N_{\mathsf{ret}}$$

$$\frac{\Gamma, x\!:\!I, x'\!:\!O \to U\underline{C} \vdash N_{\mathsf{op}} : \underline{C} \qquad (\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\mathit{eff}}) \qquad \Gamma \vdash V : A \quad \Gamma \vdash \underline{C} \quad \Gamma, y\!:\!A \vdash N_{\mathsf{ret}} : \underline{C} \quad \Gamma \vdash \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathit{eff}}} \;\mathtt{on}\; U\underline{C}}{\Gamma \vdash (\mathtt{return}\; V)\;\mathtt{handled}\;\mathtt{with}\;\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathit{eff}}}\;\mathtt{to}\; y\!:\!A\;\mathtt{in}_{\underline{C}}\; N_{\mathsf{ret}} = N_{\mathsf{ret}}[V/y] : \underline{C}}$$

It is worth recalling that Plotkin and Pretnar do not enforce the correctness of their handlers during typechecking as it is in general undecidable (Plotkin and Pretnar 2013, §6). In particular, they do not require the family of user-defined terms N_{op} to satisfy the equations given in $\mathcal{E}_{\mathsf{eff}}$. We will address decidable typechecking in future extensions of this work. For example, one could develop a normaliser that is optimised for important fibred effect theories (e.g., for state, as studied in the simply typed setting by Ahman and Staton (2013, §5.2)) and require programmers to manually prove equations that can not be established automatically. To enable the latter, we could change eMLTT to use propositional equalities in proof obligations instead of definitional equations.

4.4 Handling computations into values

We conclude this section by noting that in addition to the standard "handle into computation terms" handling construct, as discussed in §4.3, we can also use the user-defined algebra type and the composition operations to define a handling construct that handles computations directly into value terms, e.g., as briefly discussed by Ahman and Staton (2013, §6) in the context of Levy's fine-grain call-by-value language (Levy 2004, Appendix A.3.2). This handling construct is given by

$$M\;\mathtt{handled}\;\mathtt{with}\;\{\mathsf{op}_x(x') \mapsto V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\;\mathtt{to}\; y\!:\!A\;\mathtt{in}_B\; V_{\mathsf{ret}}$$
$$\stackrel{\text{def}}{=}$$
$$\mathtt{thunk}\;(M\;\mathtt{to}\; y\!:\!A\;\mathtt{in}\;\mathtt{force}_{\langle B, \{\lambda x''.\mathtt{pm}\; x''\;\mathtt{as}\;(x,x')\;\mathtt{in}\; V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\; V_{\mathsf{ret}})$$

and it satisfies the expected typing rules and β-equations, e.g.,

$$\frac{\Gamma, x\!:\!I, x'\!:\!O \to B \vdash V_{\mathsf{op}} : B \quad (\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\mathsf{eff}}) \qquad \Gamma \vdash M : FA \quad \Gamma \vdash B \quad \Gamma, y\!:\!A \vdash V_{\mathsf{ret}} : B \qquad \Gamma \vdash \{\lambda x''\!:\!(\Sigma x\!:\!I.O \to B).\mathtt{pm}\; x''\;\mathtt{as}\;(x\!:\!I, x'\!:\!O \to B)\;\mathtt{in}\; V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\;\mathtt{on}\; B}{\Gamma \vdash M\;\mathtt{handled}\;\mathtt{with}\;\{\mathsf{op}_x(x') \mapsto V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\;\mathtt{to}\; y\!:\!A\;\mathtt{in}_B\; V_{\mathsf{ret}} : B}$$

We will later use this handling construct in §7 to define predicates on effectful computations.

5 AN ALTERNATIVE PRESENTATION

It is worth noting that as the only computational effects we consider in eMLTT are algebraic, it would be possible to give eMLTT a different presentation. Namely, we could omit computation variables z and homomorphism terms K, and instead use value variables x and the auxiliary judgement $\Gamma, x\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$ to define and type the elimination form for the computational Σ-type $\Sigma x\!:\!A.\underline{C}$, analogously to the composition operations introduced in §4.2 . In more detail, this alternative presentation of eMLTT would involve the following elimination form for $\Sigma x\!:\!A.\underline{C}$:

$$\frac{\Gamma \vdash M : \Sigma x\!:\!A.\underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x\!:\!A, y\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}}{\Gamma \vdash M\;\mathtt{to}\;(x\!:\!A, y\!:\!U\underline{C})\;\mathtt{in}_{\underline{D}}\; N : \underline{D}}$$

where M is now eliminated into a pair of values, but with the judgement $\Gamma, x\!:\!A, y\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$ ensuring that the computation term N behaves in y as if it was a homomorphism term.

We note that in this paper we choose to include both homomorphism terms and the auxiliary judgement $\Gamma, x\!:\!A, y\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$ for two reasons. First, as the main aim of this paper is to extend the language of Ahman et al. (2016) with handlers of fibred algebraic effects, we wanted to keep the underlying language close to op. cit. Second, aesthetically, using homomorphism terms provides a cleaner presentation of the elimination form for $\Sigma x\!:\!A.\underline{C}$, compared to equational proof obligations.

6 META-THEORY

6.1 Weakening and substitution

We begin by noting that, as expected, weakening is admissible for value variables. However, due to the disjointness condition we imposed on value contexts in Fig. 4, we require the value variable x (with which we weaken the given judgement) to be fresh with respect to the equations in $\mathcal{E}_{\text{eff}}$.

Theorem 6.1 (Weakening). *Given $\Gamma_1, \Gamma_2 \vdash B$, $\Gamma_1 \vdash A$, and x such that $x \notin Vars(\Gamma_1, \Gamma_2)$ and $x \notin Vars(\Gamma)$ (for all $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{eff}$), then $\Gamma_1, x\!:\!A, \Gamma_2 \vdash B$, and similarly for other judgements.*

Next, we note that, as also expected, substitution is admissible for both value and computation variables. As various typing rules and definitional equations now include (translations of) effect terms, we also need to prove the corresponding property for the translation of effect terms.

Proposition 6.2. *Given $\Gamma \mid \Delta \vdash T$, y, and W such that $y \notin Vars(\Gamma)$ and $FV(W) \cap Vars(\Gamma) = \emptyset$, then*

$$(\!| T |\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\text{op}}}}[W/y] = (\!| T |\!)_{A[W/y];\overrightarrow{V_i[W/y]};\overrightarrow{V'_j[W/y]};\overrightarrow{W_{\text{op}}[W/y]}}$$

Theorem 6.3 (Substitution).

- *Given $\Gamma_1, x\!:\!A, \Gamma_2 \vdash B$ and $\Gamma_1 \vdash V : A$, then $\Gamma_1, \Gamma_2[V/x] \vdash B[V/x]$, and similarly for other judgments of types, terms, and definitional equations.*
- *Given $\Gamma \mid z\!:\!\underline{C} \vdash K : \underline{D}$ and $\Gamma \vdash M : \underline{C}$, then $\Gamma \vdash K[M/z] : \underline{D}$.*
- *Given $\Gamma \mid z_2\!:\!\underline{D}_1 \vdash L : \underline{D}_2$ and $\Gamma \mid z_1\!:\!\underline{C} \vdash K : \underline{D}_1$, then $\Gamma \mid z_1\!:\!\underline{C} \vdash L[K/z_2] : \underline{D}_2$.*

Finally, we note that judgments of well-formed types, etc. only refer to well-formed contexts, etc. To this end, we also need to show that under suitable assumptions, $(\!| T |\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\text{op}}}}$ is well-typed.

Proposition 6.4. *Given $\Gamma \mid \Delta \vdash T$ and Γ' such that $Vars(\Gamma') \cap Vars(\Gamma) = \emptyset$, $\Gamma' \vdash A$, and the value terms in the subscripts are well-typed in Γ' (as in Fig. 4), then we have $\Gamma' \vdash (\!| T |\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\text{op}}}} : A$.*

Proposition 6.5. *Given $\Gamma \mid \Delta \vdash T$ and Γ' such that $Vars(\Gamma') \cap Vars(\Gamma) = \emptyset$, $\Gamma' \vdash A = B$, and the corresponding value terms in the subscripts are definitionally equal in Γ', then we have*

$$\Gamma' \vdash (\!| T |\!)_{A;\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{V_{\text{op}}}} = (\!| T |\!)_{B;\overrightarrow{W_i};\overrightarrow{W'_j};\overrightarrow{W_{\text{op}}}} : A$$

Theorem 6.6. *Given $\Gamma \vdash V : A$, then $\vdash \Gamma$ and $\Gamma \vdash A$, and similarly for other judgments.*

6.2 Derivable definitional equations

We begin by showing that Fig. 4's disjointness requirement on value contexts can be omitted.

Proposition 6.7. *For any equation $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{eff}$, the following rule is derivable:*

$$\frac{\begin{array}{ll} \Gamma' \vdash V_i : A_i[V_1/x_1, \ldots, V_{i-1}/x_{i-1}] & \qquad (1 \leq i \leq n) \\ \Gamma' \vdash \underline{C} \quad \Gamma' \vdash V'_j : A'_j[\overrightarrow{V_i/x_i}] \to U\underline{C} & \qquad (1 \leq j \leq m) \end{array}}{\Gamma' \vdash (\!| T_1 |\!)_{U\underline{C};\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\text{op}}}} = (\!| T_2 |\!)_{U\underline{C};\overrightarrow{V_i};\overrightarrow{V'_j};\overrightarrow{W_{\text{op}}}} : U\underline{C}}$$

Proof. There are two cases to consider. If Γ' and Γ are disjoint, we use the restricted rule from Fig. 4. If Γ' and Γ overlap, we first systematically replace the overlapping variables with fresh ones using the weakening and substitution theorems, and then use the restricted rule from Fig. 4. □

Next, we note that one can derive specialised versions of the general algebraicity equation.

Proposition 6.8. *We can derive the following specialised algebraicity equation:*

$$\frac{\Gamma \vdash V : I \quad \Gamma, y\!:\!O[V/x] \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, y'\!:\!A \vdash N : \underline{C}}{\Gamma \vdash \mathsf{op}^{FA}_V(y.M)\ \mathsf{to}\ y'\!:\!A\ \mathsf{in}\ N = \mathsf{op}^{\underline{C}}_V(y.M\ \mathsf{to}\ y'\!:\!A\ \mathsf{in}\ N) : \underline{C}}$$

and analogously for other computation term formers.

Finally, we present some useful derivable equations for composition operations.

Proposition 6.9. *We can derive the following unit and associativity equations:*

$$\frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash M\ \mathsf{as}\ x\!:\!U\underline{C}\ \mathsf{in}\ \mathsf{force}_{\underline{C}}\ x = M : \underline{C}}$$

$$\frac{\Gamma \vdash M : \underline{C}_1 \quad \Gamma \vdash \underline{C}_2 \quad \Gamma \vdash \underline{D} \quad \Gamma, x_1\!:\!U\underline{C}_1 \vdash_{\mathsf{hom}} N_1 : \underline{C}_2 \quad \Gamma, x_2\!:\!U\underline{C}_2 \vdash_{\mathsf{hom}} N_2 : \underline{D}}{\Gamma \vdash M\ \mathsf{as}\ x_1\!:\!U\underline{C}_1\ \mathsf{in}\ (N_1\ \mathsf{as}\ x_2\!:\!U\underline{C}_2\ \mathsf{in}\ N_2) = (M\ \mathsf{as}\ x_1\!:\!U\underline{C}_1\ \mathsf{in}\ N_1)\ \mathsf{as}\ x_2\!:\!U\underline{C}_2\ \mathsf{in}\ N_2 : \underline{D}}$$

and analogously for the composition operation for homomorphism terms.

Proposition 6.10. *We can derive the following associativity equations:*

$$\frac{\Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x_1\!:\!A \vdash N_1 : \underline{C} \quad \Gamma, x_2\!:\!U\underline{C} \vdash_{\mathsf{hom}} N_2 : \underline{D}}{\Gamma \vdash M\ \mathsf{to}\ x_1\!:\!A\ \mathsf{in}\ (N_1\ \mathsf{as}\ x_2\!:\!U\underline{C}\ \mathsf{in}\ N_2) = (M\ \mathsf{to}\ x_1\!:\!A\ \mathsf{in}\ N_1)\ \mathsf{as}\ x_2\!:\!U\underline{C}\ \mathsf{in}\ N_2 : \underline{D}}$$

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma, x_1\!:\!U\underline{C} \vdash_{\mathsf{hom}} N_1 : FA \quad \Gamma \vdash \underline{D} \quad \Gamma, x_2\!:\!A \vdash N_2 : \underline{D}}{\Gamma \vdash M\ \mathsf{as}\ x_1\!:\!U\underline{C}\ \mathsf{in}\ (N_1\ \mathsf{to}\ x_2\!:\!A\ \mathsf{in}\ N_2) = (M\ \mathsf{as}\ x_1\!:\!U\underline{C}\ \mathsf{in}\ N_1)\ \mathsf{to}\ x_2\!:\!A\ \mathsf{in}\ N_2 : \underline{D}}$$

and analogously for computational pattern-matching, and the corresponding homomorphism terms.

Proposition 6.11. *The composition operations commute with computational pairing, computational lambda abstraction, and computational and homomorphic function applications, e.g., we have*

$$\frac{\begin{array}{c}\Gamma \vdash M : \underline{C} \quad \Gamma \vdash V : A \\ \Gamma, y\!:\!A \vdash \underline{D} \quad \Gamma, x\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}[V/y]\end{array}}{\begin{array}{l}\Gamma \vdash M\ \mathsf{as}\ x\!:\!U\underline{C}\ \mathsf{in}\ \langle V, N\rangle \\ \quad = \langle V, M\ \mathsf{as}\ x\!:\!U\underline{C}\ \mathsf{in}\ N\rangle : \Sigma y\!:\!A.\underline{D}\end{array}} \qquad \frac{\begin{array}{c}\Gamma \vdash M : \underline{C} \quad \Gamma \vdash V : \underline{D}_1 \multimap \underline{D}_2 \\ \Gamma, y_1\!:\!U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}_1\end{array}}{\begin{array}{l}\Gamma \vdash M\ \mathsf{as}\ y_1\!:\!U\underline{C}\ \mathsf{in}\ V\,N \\ \quad = V\,(M\ \mathsf{as}\ y_1\!:\!U\underline{C}\ \mathsf{in}\ N) : \underline{D}_2\end{array}}$$

7 USING HANDLERS TO REASON ABOUT EFFECTFUL COMPUTATIONS

In this section we demonstrate that our type-based treatment of handlers provides a useful mechanism for reasoning about effectful computations. In particular, we show that the "handle into value terms" handling construct (§4.4) provides the programmer with a useful alternative to defining predicates on effectful computations using propositional equality on thunks.

In order to facilitate reasoning based on the "handle into value terms" handling construct, we first introduce a *universe* à la Tarski (Martin-Löf 1984), by extending eMLTT with a *value universe* VU, the corresponding *decoding function* $\mathsf{El}(V)$, and the corresponding *codes of value types*:

$$A ::= \ \ldots \ \mid \mathsf{VU} \mid \mathsf{El}(V)$$
$$V ::= \ \ldots \ \mid \mathsf{unit\text{-}c} \mid \mathsf{empty\text{-}c} \mid \mathsf{coprod\text{-}c(V,W)} \mid \mathsf{sig\text{-}c}(V, x.W) \mid \mathsf{pi\text{-}c}(V, x.W)$$

We also extend eMLTT with corresponding typing rules and definitional equations, e.g.,

$$\frac{\Gamma \vdash V : \mathsf{VU} \quad \Gamma, x\!:\!\mathsf{El}(V) \vdash W : \mathsf{VU}}{\Gamma \vdash \mathsf{pi\text{-}c}(V, x.W) : \mathsf{VU}} \qquad \frac{\Gamma \vdash V : \mathsf{VU} \quad \Gamma, x\!:\!\mathsf{El}(V) \vdash W : \mathsf{VU}}{\Gamma \vdash \mathsf{El}(\mathsf{pi\text{-}c}(V, x.W)) = \Pi x\!:\!\mathsf{El}(V).\mathsf{El}(W)}$$

Using this value universe, we can now define predicates on effectful computations (of type FA) as value terms of the form $\Gamma \vdash V : UFA \to \mathsf{VU}$, with the aim of using these predicates to refine (thunks of) computations using value Σ-types, i.e., as $\Sigma x\!:\!UFA.\mathsf{El}(V x)$. In more detail, we define the predicates $\Gamma \vdash V : UFA \to \mathsf{VU}$ by equipping VU with an algebra for the given notion of computation, and by using the "handle into value terms" handling construct from §4.4.

It is worth noting that our approach of defining predicates on computations (essentially, defining value types that depend on effectful computations in a "well-behaved" manner) by equipping the universe VU with an algebra structure is reminiscent of the recent work by Pédrot and Tabareau (2017) we mentioned earlier. In particular, their monadic translation of type theory also crucially relies on equipping the types of their language with an algebra structure for the given monad.

Below, we give two kinds of examples of predicates on effectful computations: i) lifting predicates from return values to computations (§7.1); and ii) specifying patterns of allowed effects (§7.2).

7.1 Lifting predicates from return values to effectful computations

Lifting predicates from return values to computations is easiest when the given fibred effect theory does not contain equations. Thus, let us consider the theory $\mathcal{T}_{\mathrm{I/O}}$ of *input-output of bits* from §4.1 for our first example; other equation-free fibred algebraic effects can be reasoned about similarly.

Then, assuming given a predicate $\Gamma \vdash V_P : A \to \mathsf{VU}$, we lift V_P to a predicate $V_{\widehat{P}}$ on UFA by

$$V_{\widehat{P}} \stackrel{\text{def}}{=} \lambda y\!:\!UFA\,.\,(\mathtt{force}_{FA}\, y)\ \mathtt{handled\ with}\ \{\mathsf{op}_x(x') \mapsto V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{I/O}}}\ \mathtt{to}\ y'\!:\!A\ \mathtt{in}_{\mathsf{VU}}\ (V_P\, y')$$

where we define the code of bits as $\mathtt{bit\text{-}c} \stackrel{\text{def}}{=} \mathtt{coprod\text{-}c}(\mathtt{unit\text{-}c}, \mathtt{unit\text{-}c})$, and where

$$x\!:\!1, x'\!:\!1+1 \to \mathsf{VU} \vdash V_{\mathsf{read}} \stackrel{\text{def}}{=} \mathtt{sig\text{-}c}(\mathtt{bit\text{-}c}, y'.\, x'\, y') : \mathsf{VU}$$

$$x\!:\!1+1, x'\!:\!1 \to \mathsf{VU} \vdash V_{\mathsf{write}} \stackrel{\text{def}}{=} x'\, \star$$

On closer inspection, we see that $V_{\widehat{P}}$ agrees with the possibility modality from Evaluation Logic (Pitts 1991) in that a computation term satisfies $V_{\widehat{P}}$ if there *exists* a return value that satisfies V_P. Further, observe that if we were to replace `sig-c` (code for value Σ-type) with `pi-c` (code for value Π-type), we would get a modality that holds if *all* the return values of the given computation satisfy V_P.

For our second example of lifting predicates from return values to computations, let us consider a fibred effect theory that also includes equations, namely, the theory $\mathcal{T}_{\mathrm{GS}}$ of *global state* from §3.2.

In particular, when we define the type of states as $\mathsf{S} \stackrel{\text{def}}{=} \Pi x\!:\!\mathsf{Loc}.\mathsf{Val}$, then assuming given a predicate $\Gamma \vdash V_Q : A \to \mathsf{S} \to \mathsf{VU}$ on return values and *final states*, we can define a predicate

$$\begin{aligned} &V_{\widehat{Q}} \stackrel{\text{def}}{=} \lambda y\!:\!UFA\,.\,\lambda x_S\!:\!\mathsf{S}\,.\\ &\quad \mathtt{fst}\ \Big(\Big(\mathtt{thunk}\ \Big((\mathtt{force}_{FA}\, y)\ \mathtt{handled\ with}\ \{\mathsf{op}_x(x') \mapsto V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathrm{GS}}}\ \mathtt{to}\ y'\!:\!A\ \mathtt{in}_{\mathsf{S}\to\mathsf{VU}\times\mathsf{S}}\ V_{\mathsf{ret}}\Big)\Big)\ x_S\Big)\end{aligned}$$

on (thunks of) computations and *initial states*, with V_{get} and V_{put} defined using the natural representation of stateful programs as functions $\mathsf{S} \to \mathsf{VU} \times \mathsf{S}$, and where V_{ret} is defined as follows:

$$\Gamma, y\!:\!UFA, x_S\!:\!\mathsf{S}, y'\!:\!A \vdash V_{\mathsf{ret}} \stackrel{\text{def}}{=} \lambda x'_S\!:\!\mathsf{S}\,.\,\langle V_Q\, y'\, x'_S, x'_S\rangle : \mathsf{S} \to \mathsf{VU} \times \mathsf{S}$$

In other words, V_{get} and V_{put} are defined as if they were operations of the free algebra on VU for an equational theory of global state corresponding to the fibred effect theory $\mathcal{T}_{\mathrm{GS}}$.

On closer inspection, we can see that the predicate $V_{\widehat{Q}}$ corresponds to Dijkstra's weakest precondition semantics of stateful programs (Dijkstra 1975). For example, taking $\mathsf{Loc} \stackrel{\text{def}}{=} 1$ and omitting the trivial location arguments, we can prove that the following definitional equations hold:

$$\Gamma \vdash V_{\widehat{Q}}\ (\mathtt{thunk}\ (\mathtt{return}\ V))\ V_S = V_Q\, V\, V_S : \mathsf{VU}$$

$$\Gamma \vdash V_{\widehat{Q}}\,(\mathtt{thunk}\;(\mathtt{get}^{FA}(y.M)))\,V_S = V_{\widehat{Q}}\,(\mathtt{thunk}\;M[V_S/y])\,V_S : \mathsf{VU}$$

$$\Gamma \vdash V_{\widehat{Q}}\,(\mathtt{thunk}\;(\mathtt{put}^{FA}_{V'_S}(M)))\,V_S = V_{\widehat{Q}}\,(\mathtt{thunk}\;M)\,V'_S : \mathsf{VU}$$

7.2 Specifying patterns of allowed effects in computations

Analogously to lifting predicates from return values to effectful computations, specifying patterns of allowed effects is easiest when the given fibred effect theory does not contain any equations. Thus, for simplicity, we again consider the theory $\mathcal{T}_{\mathrm{I/O}}$ of *input-output of bits* for our examples.

As a first example, we consider a very coarse grained specification, namely, disallowing all writes:

$$V_{\text{no-w}} \stackrel{\text{def}}{=} \lambda y\!:\!UFA\,.\,(\mathtt{force}_{FA}\,y)\;\mathtt{handled}\;\mathtt{with}\;\{\mathtt{op}_x(x') \mapsto V_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{I/O}}}\;\mathtt{to}\;y'\!:\!A\;\mathtt{in}_{\mathsf{VU}}\;\mathtt{unit\text{-}c}$$

where

$$x\!:\!1, x'\!:\!1+1 \to \mathsf{VU} \vdash V_{\mathsf{read}} \stackrel{\text{def}}{=} \mathtt{pi\text{-}c}(\mathtt{bit\text{-}c}, y'.\,x'\,y')$$

$$x\!:\!1+1, x'\!:\!1 \to \mathsf{VU} \vdash V_{\mathsf{write}} \stackrel{\text{def}}{=} \mathtt{empty\text{-}c}$$

For example, we can then show that $\mathtt{read}^{FA}(x.\mathtt{write}^{FA}_V(M))$ does not satisfy $V_{\text{no-w}}$ because

$$\Gamma \vdash \mathsf{El}(V_{\text{no-w}}(\mathtt{thunk}(\mathtt{read}^{FA}(y.\mathtt{write}^{FA}_V(M))))) = \Pi y\!:\!1{+}1.0 \cong 0$$

As a more involved example, we consider specifications on I/O-effects in the style of session types (Honda et al. 1998). First, we assume an inductive type $\diamond \vdash \mathsf{Protocol}$ with three constructors:

$$\mathsf{r} : (1{+}1 \to \mathsf{Protocol}) \to \mathsf{Protocol} \qquad \mathsf{w} : (1{+}1 \to \mathsf{VU}) \to \mathsf{Protocol} \to \mathsf{Protocol} \qquad \mathsf{e} : \mathsf{Protocol}$$

describing *patterns of allowed I/O-effects*. Intuitively, r specifies that the next allowed I/O-effect is reading; w specifies that the next allowed I/O-effect is writing, with the value written required to satisfy the predicate given as an argument to w; and e specifies that no further communication must happen (i.e., end of communication). The Protocol-valued arguments of r and w specify how the computation is allowed to evolve after reading and writing, respectively.

Then, given some particular protocol $\Gamma \vdash V_{\mathsf{pr}} : \mathsf{Protocol}$, we can define a predicate

$$V_{\widehat{\mathsf{pr}}} \stackrel{\text{def}}{=} \lambda y\!:\!UFA\,.\;\Big(\mathtt{thunk}\;\Big((\mathtt{force}_{FA}\,y)\;\mathtt{handled}\;\mathtt{with}\;\{\mathtt{op}_x(x') \mapsto V_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{I/O}}}\;\mathtt{to}\;y'\!:\!A\;\mathtt{in}_{\mathsf{Protocol}\to\mathsf{VU}}\;V_{\mathsf{ret}}\Big)\Big)\,V_{\mathsf{pr}}$$

where the value terms V_{read}, V_{write}, and V_{ret} are defined as follows (for better readability, we give their structural-recursive definitions by pattern-matching on their arguments of type Protocol):

$$\Gamma, y\!:\!UFA, y'\!:\!A \vdash V_{\mathsf{ret}}\quad \mathsf{e} \stackrel{\text{def}}{=} \mathtt{unit\text{-}c}$$

$$\Gamma, y\!:\!UFA, x\!:\!1, x'\!:\!1+1 \to \mathsf{Protocol} \to \mathsf{VU} \vdash V_{\mathsf{read}}\;(\mathsf{r}\;V'_{\mathsf{pr}}) \stackrel{\text{def}}{=} \mathtt{pi\text{-}c}(\mathtt{bit\text{-}c}, y.(x'\,y)\,(V'_{\mathsf{pr}}\,y))$$

$$\Gamma, y\!:\!UFA, x\!:\!1+1, x'\!:\!1 \to \mathsf{Protocol} \to \mathsf{VU} \vdash V_{\mathsf{write}}\;(\mathsf{w}\;V_P\;V'_{\mathsf{pr}}) \stackrel{\text{def}}{=} \mathtt{sig\text{-}c}(V_P\,x, y.\,x' \star V'_{\mathsf{pr}})$$

with all other cases defined as empty-c. As a result, a computation satisfies the predicate $V_{\widehat{\mathsf{pr}}}$ only if its I/O-effects precisely follow the specific pattern of I/O-effects specified by the given protocol V_{pr}.

It is worth noting that this example can be easily extended to account for *sets* of patterns of allowed I/O-effects. For example, we could extend the inductive type Protocol with a fourth constructor $\mathsf{or} : \mathsf{Protocol} \times \mathsf{Protocol} \longrightarrow \mathsf{Protocol}$, and the above definitions of value terms V_{read}, V_{write}, and V_{ret} with the corresponding case $V\,(V'_{\mathsf{pr}}\;\mathsf{or}\;V''_{\mathsf{pr}}) \stackrel{\text{def}}{=} \mathtt{coprod\text{-}c}(V\,V'_{\mathsf{pr}}, V\,V''_{\mathsf{pr}})$.

Finally, we highlight that it is easily combine these specifications with those discussed in §7.1, namely, by replacing unit-c in the definition of V_{ret} with a suitable predicate V_P on return values.

8 SEMANTICS

We conclude by describing how to give a natural denotational semantics to our extension of eMLTT with fibred algebraic effects and their handlers. The semantics we develop is an instance of a more general class of models of eMLTT, based on *fibrations* (functors with extra structure for modelling substitution, Σ- and Π-types, etc.) and *adjunctions* between them, as studied by Ahman et al. (2016).

We proceed in three steps. First, we recall from the work of Ahman et al. (2016) how the pure fragment of eMLTT is interpreted in the families of sets fibration. Next, we show how to derive a countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ from the given fibred effect theory $\mathcal{T}_{\text{eff}}$. Finally, we define the interpretation of the rest of eMLTT using the models of this countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$.

We leave the denotational semantics of the extension of eMLTT with universes for future work, expecting it to closely follow the fibrational treatment of induction-recursion (Ghani et al. 2013).

8.1 Families fibrations

We begin by giving a brief overview of the kinds of fibrations we use for defining the denotational semantics of our extension of eMLTT. For a much more detailed treatment of fibrations and their use in modelling various type theories and logics, we suggest the book by Jacobs (1999).

First, given a category C, it is well-known that one can define a new category $\mathsf{Fam}(C)$ whose objects are pairs (X, A) of a set X and a functor $A : X \longrightarrow C$ (treating X as a discrete category); and the morphisms $(X, A) \to (Y, B)$ are pairs of a function $f : X \longrightarrow Y$ and a natural transformation $g : A \longrightarrow B \circ f$. The corresponding *$C$-valued families fibration* $\mathsf{fam}_C : \mathsf{Fam}(C) \longrightarrow \mathsf{Set}$ is then defined on objects as $\mathsf{fam}_C(X, A) \stackrel{\text{def}}{=} X$ and on morphisms as $\mathsf{fam}_C(f, g) \stackrel{\text{def}}{=} f$.

Next, for any set X, the category $\mathsf{Fam}_X(C)$ is called the *fibre* over X; this is a subcategory of $\mathsf{Fam}(C)$ whose objects and morphisms are of the form (X, A) and (id_X, g). Given a function $f : X \longrightarrow Y$, the corresponding *reindexing functor* $f^* : \mathsf{Fam}_Y(C) \longrightarrow \mathsf{Fam}_X(C)$ is given by $f^*(Y, A) \stackrel{\text{def}}{=} (X, A \circ f)$ and analogously on morphisms. As standard in the literature, we write $\overline{f}(Y, A) \stackrel{\text{def}}{=} (f, (\mathsf{id}_{A(f(x))})_x) : f^*(Y, A) \longrightarrow (Y, A)$ for the *Cartesian morphism* over $f : X \longrightarrow Y$.

It is worth noting that we get a prototypical model of dependent types when we take $C \stackrel{\text{def}}{=} \mathsf{Set}$. In this case, there also exists a pair of adjunctions $\mathsf{fam}_{\mathsf{Set}} \dashv 1 \dashv \{-\}$, where the *terminal object functor* $1 : \mathsf{Set} \longrightarrow \mathsf{Fam}(\mathsf{Set})$ is given by $1(X) \stackrel{\text{def}}{=} (X, x \mapsto \{\star\})$ and the *comprehension functor* $\{-\} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}$ by $\{(X, A)\} \stackrel{\text{def}}{=} \coprod_{x \in X} A(x)$. Further, the latter provides semantics to context extensions $\Gamma, x \!:\! A$, and it also gives us canonical *projection maps* $\pi_{(X,A)} : \{(X, A)\} \longrightarrow X$.

8.2 Interpretation of the pure fragment of eMLTT

As a first step, we recall from the work of Ahman et al. (2016) how the pure fragment of eMLTT is interpreted in the families of sets fibration $\mathsf{fam}_{\mathsf{Set}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}$ we described above.

In detail, the interpretation of the pure fragment of eMLTT is defined as a *partial interpretation function* $[\![-]\!]$, which, if defined, maps a context Γ to a set $[\![\Gamma]\!]$, a context Γ and value type A to an object $[\![\Gamma; A]\!]$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Set})$, and a context Γ and value term V to $[\![\Gamma; V]\!] : 1([\![\Gamma]\!]) \longrightarrow ([\![\Gamma]\!], A)$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Set})$, for some $A : [\![\Gamma]\!] \longrightarrow \mathsf{Set}$. For better readability, we denote the first and second components of $[\![\Gamma; A]\!]$ and $[\![\Gamma; V]\!]$ using subscripts 1, 2, i.e., we write $([\![\Gamma; A]\!]_1, [\![\Gamma; A]\!]_2)$ for $[\![\Gamma; A]\!]$.

First, the types Nat, 1, 0, and $A + B$ are interpreted by using the corresponding categorical structure in the fibres of $\mathsf{Fam}(\mathsf{Set})$. For example, we have $[\![\Gamma; \mathsf{Nat}]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \mathbb{N})$ and

$$[\![\Gamma; 0]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \emptyset) \qquad [\![\Gamma; A + B]\!] \stackrel{\text{def}}{=} [\![\Gamma; A]\!] + [\![\Gamma; B]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto [\![\Gamma; A]\!]_2(\gamma) + [\![\Gamma; B]\!]_2(\gamma))$$

Next, assuming that $[\![\Gamma; A]\!]$ and $[\![\Gamma, x\!:\!A; B]\!]$ are defined, and $[\![\Gamma, x\!:\!A; B]\!]_1 = \coprod_{\gamma \in [\![\Gamma]\!]} [\![\Gamma; A]\!]_2(\gamma)$, then

$$[\![\Gamma; \Sigma x\!:\!A.B]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \textstyle\coprod_{a \in [\![\Gamma;A]\!]_2(\gamma)} [\![\Gamma, x\!:\!A; B]\!]_2(\langle \gamma, a \rangle))$$
$$[\![\Gamma; \Pi x\!:\!A.B]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \textstyle\prod_{a \in [\![\Gamma;A]\!]_2(\gamma)} [\![\Gamma, x\!:\!A; B]\!]_2(\langle \gamma, a \rangle))$$

Finally, propositional equality is interpreted in terms of equality of the denotations of terms, i.e.,

$$[\![\Gamma; V =_A W]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \{ \star \mid ([\![\Gamma; V]\!]_2)_\gamma(\star) = ([\![\Gamma; W]\!]_2)_\gamma(\star) \})$$

As an example of the interpretation of value terms, $\mathtt{inl}_{A+B}\ V$ is interpreted as follows:

$$[\![\Gamma; \mathtt{inl}_{A+B}\ V]\!]_1 \stackrel{\text{def}}{=} \mathsf{id}_{[\![\Gamma]\!]} \qquad ([\![\Gamma; \mathtt{inl}_{A+B}\ V]\!]_2)_\gamma \stackrel{\text{def}}{=} \star \mapsto \mathsf{inl}\,(([\![\Gamma; V]\!]_2)_\gamma(\star))$$

assuming that $[\![\Gamma; V]\!] : 1([\![\Gamma]\!]) \longrightarrow [\![\Gamma; A]\!]$ and $[\![\Gamma; B]\!]$ are defined—see Ahman et al. (2016) for details.

The soundness theorem for EMLTT (Ahman et al. 2016) then tells us that $[\![-]\!]$ is in fact defined on well-formed pure syntax and that it validates the corresponding definitional equations.

8.3 Deriving a countable Lawvere theory from a fibred effect theory

We now show how to derive a countable Lawvere theory from the given fibred effect theory $\mathcal{T}_{\text{eff}}$.

We begin by recalling some basic definitions and results about countable Lawvere theories—see Power (2006) for a detailed study of countable Lawvere theories.

A *countable Lawvere theory* consists of a small category $\mathcal{L}$ with countable products and a strict countable-product preserving identity-on-objects functor $I : \aleph_1^{\text{op}} \longrightarrow \mathcal{L}$, where $\aleph_1$ is the skeleton of the category of countable sets. A *model* of a countable Lawvere theory $\mathcal{L}$ in a category C with countable products is given by a countable-product preserving functor $\mathcal{M} : \mathcal{L} \longrightarrow C$. A *morphism* of such models from $\mathcal{M}_1 : \mathcal{L} \longrightarrow C$ to $\mathcal{M}_2 : \mathcal{L} \longrightarrow C$ is given by a natural transformation $h : \mathcal{M}_1 \longrightarrow \mathcal{M}_2$. Models and morphisms between them form the category $\mathsf{Mod}(\mathcal{L}, C)$.

Conceptually, a countable Lawvere theory is nothing but an abstract category-theoretic description of the clone of countable equational theories (Grätzer 1979). In particular, one usually thinks of the morphisms $n \longrightarrow 1$ in $\mathcal{L}$ as terms in n free variables, and of the morphisms $n \longrightarrow m$ as m-tuples of terms in n free variables. Analogously, the models of a countable Lawvere theory correspond to the models of the countable equational theories whose clone this countable Lawvere theory is.

We also recall that there exists a canonical forgetful functor $U_{\mathcal{L}} : \mathsf{Mod}(\mathcal{L}, C) \longrightarrow C$, given on objects by $U_{\mathcal{L}}(\mathcal{M}) \stackrel{\text{def}}{=} \mathcal{M}(1)$. A well known result then states that if C is locally countably presentable (Adamek and Rosicky 1994), the functor $U_{\mathcal{L}}$ has a left adjoint $F_{\mathcal{L}} : C \longrightarrow \mathsf{Mod}(\mathcal{L}, C)$. Importantly for the purposes of this paper, the category Set of sets and functions is locally countably presentable. A further useful property of $\mathsf{Mod}(\mathcal{L}, \mathrm{Set})$ is that it is also both complete and cocomplete. For better readability, we write Mod for $\mathsf{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \mathrm{Set})$ in the rest of this paper.

Next, in order to be able to derive a countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ from the given fibred effect theory $\mathcal{T}_{\text{eff}}$, we require $\mathcal{T}_{\text{eff}}$ to be *countable*. In particular, this means that for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$, we require $[\![x\!:\!I; O]\!]_2$ to be a family of countable sets. Further, we also require $[\![\Gamma; A'_j]\!]_2$ to be a family of countable sets, for all equations $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$ and effect variables $w_j\!:\!A'_j \in \Delta$

We can then construct $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ by expanding $\mathcal{T}_{\text{eff}}$ into a countable equational theory (Grätzer 1979), analogously to how Plotkin and Pretnar (2013) expanded their effect theories. More specifically, $\mathcal{S}_{\text{eff}}$ determines a countable signature consisting of operation symbols $\mathsf{op}_i : |[\![x\!:\!I; O]\!]_2\,(\langle \star, i \rangle)|$, for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$ and $i \in [\![\diamond; I]\!]_2\,(\star)$. Every effect term $\Gamma \mid \Delta \vdash T$ then naturally determines a family of terms $\Delta^\gamma \vdash T^\gamma$ derivable from this countable signature (for all $\gamma \in [\![\Gamma]\!]$), where Δ^γ consists of variables $x^a_{w_j}$ for all $w_j : A_j \in \Delta$ and $a \in [\![\Gamma; A'_j]\!]_2(\gamma)$. In detail, the terms T^γ are defined as follows:

$$
\begin{array}{lll}
(w_j(V))^\gamma & \stackrel{\text{def}}{=} x_{w_j}^{([\![\Gamma;V]\!]_2)_\gamma(\star)} & \\
(\mathsf{op}_V(y.T))^\gamma & \stackrel{\text{def}}{=} \mathsf{op}_{([\![\Gamma;V]\!]_2)_\gamma(\star)}(T^{\langle\gamma,o\rangle})_{1 \leq o \leq |[\![x:I;O]\!]_2(\langle\star,([\![\Gamma;V]\!]_2)_\gamma(\star)\rangle)|} & \\
(\mathsf{pm}\ V\ \mathsf{as}\ (y_1 : B_1, y_2 : B_2)\ \mathsf{in}\ T)^\gamma & \stackrel{\text{def}}{=} T^{\langle\langle\gamma,b_1\rangle,b_2\rangle} & (\textit{when } ([\![\Gamma;V]\!]_2)_\gamma(\star) = \langle b_1, b_2\rangle) \\
(\mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(y_1) \mapsto T_1, \mathsf{inr}(y_2) \mapsto T_2))^\gamma & \stackrel{\text{def}}{=} T_1^{\langle\gamma,b\rangle} & (\textit{when } ([\![\Gamma;V]\!]_2)_\gamma(\star) = \mathsf{inl}\, b) \\
(\mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(y_1) \mapsto T_1, \mathsf{inr}(y_2) \mapsto T_2))^\gamma & \stackrel{\text{def}}{=} T_2^{\langle\gamma,b\rangle} & (\textit{when } ([\![\Gamma;V]\!]_2)_\gamma(\star) = \mathsf{inr}\, b)
\end{array}
$$

Finally, we get a countable equational theory by taking all such equations $\Delta^\gamma \vdash T_1^\gamma = T_2^\gamma$ and close them under the rules of reflexivity, symmetry, transitivity, replacement, and substitution.

The countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ is then given by taking the morphisms $n \longrightarrow m$ in $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ to be m-tuples $(\overrightarrow{x_i} \vdash t_j)_{1 \leq j \leq m}$ of equivalence classes of terms in n variables (in the countable equational theory defined above). The identity morphisms are given by tuples of variables, while the composition of morphisms is given by substitution. We define the functor $I_{\mathcal{T}_{\text{eff}}} : \aleph_1^{\text{op}} \longrightarrow \mathcal{L}_{\mathcal{T}_{\text{eff}}}$ by $I_{\mathcal{T}_{\text{eff}}}(n) \stackrel{\text{def}}{=} n$ and $I_{\mathcal{T}_{\text{eff}}}(f) \stackrel{\text{def}}{=} (\overrightarrow{x_i} \vdash x_{f(j)})_{1 \leq j \leq m} : n \to m$. It is then easy to verify that $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ has countable products (given using the cardinal sum in $\aleph_1$) and $I_{\mathcal{T}_{\text{eff}}}$ strictly preserves them.

Proposition 8.1. *$\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ is a countable Lawvere theory.*

We conclude our discussion about $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ by highlighting that for any set A, one can intuitively view the *free* model $F_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}(A)$ as being given by the set of equivalence classes of terms in the equational theory we derived from $\mathcal{T}_{\text{eff}}$, with the variables of these terms given by elements of the set A.

8.4 Interpretation of the non-pure fragment of eMLTT

We now show how to extend the interpretation of the pure fragment of eMLTT to the rest of our extension of eMLTT with fibred algebraic effects and their handlers, based on the fibred adjunction

$$
\begin{tikzcd}
\mathsf{Fam}(\mathsf{Set}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathsf{fam}_{\mathsf{Set}}"'] & & \mathsf{Fam}(\mathsf{Mod}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{fam}_{\mathsf{Mod}}"] \\
& \mathsf{Set} &
\end{tikzcd}
$$

where the two fibred functors are defined (on objects) as

$$
F(X, A) \stackrel{\text{def}}{=} (X, F_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}} \circ A) \qquad U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}} \circ \underline{C})
$$

That the functors F and U indeed form a fibred adjunction, and that this adjunction is suitable for modelling eMLTT, is an instance of a general result about models of eMLTT (Ahman et al. 2016).

Using this fibred adjunction $F \dashv U$, we can now extend the definition of $[\![-]\!]$ given in §8.2 so that, if defined, it maps a context Γ and a computation type $\underline{C}$ to an object $[\![\Gamma; \underline{C}]\!]$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Mod})$; a context Γ and a computation term M to $[\![\Gamma; M]\!] : 1([\![\Gamma]\!]) \longrightarrow U([\![\Gamma]\!], \underline{C})$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Set})$, for some $\underline{C} : [\![\Gamma]\!] \longrightarrow \mathsf{Set}$; and a context Γ, a variable z, a computation type $\underline{C}$ and a homomorphism term K to $[\![\Gamma; z : \underline{C}; K]\!] : [\![\Gamma; \underline{C}]\!] \longrightarrow ([\![\Gamma]\!], \underline{D})$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Mod})$, for some $\underline{D} : [\![\Gamma]\!] \longrightarrow \mathsf{Set}$.

In particular, the interpretation of the types $\underline{C} \multimap \underline{D}$, FA, and $U\underline{C}$ is defined as follows:

$$
[\![\Gamma; \underline{C} \multimap \underline{D}]\!] \stackrel{\text{def}}{=} \Big([\![\Gamma]\!], \gamma \mapsto \mathsf{Hom}_{\mathsf{Mod}}\big([\![\Gamma; \underline{C}]\!]_2(\gamma), [\![\Gamma; \underline{D}]\!]_2(\gamma)\big)\Big)
$$

$$
[\![\Gamma; FA]\!] \stackrel{\text{def}}{=} F([\![\Gamma; A]\!]) \qquad [\![\Gamma; U\underline{C}]\!] \stackrel{\text{def}}{=} U([\![\Gamma; \underline{C}]\!])
$$

assuming that the objects $[\![\Gamma; A]\!]$, $[\![\Gamma; \underline{C}]\!]$, and $[\![\Gamma; \underline{D}]\!]$ are defined. In the rest of this section, we omit such routine assumptions. We also note that the computational Σ- and Π-types are interpreted similarly to their value counterparts, using the set-indexed coproducts and products in Mod.

We omit the definition of $[\![-]\!]$ for the cases (of computation and homomorphism terms) that are already covered in great detail by Ahman et al. (2016), and instead concentrate on demonstrating how to define the interpretation for algebraic operations, the user-defined algebra type, and the two composition operations. Further, we note that diagrammatic and more detailed definitions of the cases of $[\![-]\!]$ that we discuss below can be found in Appendix C.

First, we define $[\![-]\!]$ on algebraic operations $\mathsf{op}^{\underline{C}}_V(y.M)$ as follows:

$$[\![\Gamma; \mathsf{op}^{\underline{C}}_V(y.M)]\!]_1 \stackrel{\text{def}}{=} \mathsf{id}_{[\![\Gamma]\!]}$$

$$([\![\Gamma; \mathsf{op}^{\underline{C}}_V(y.M)]\!]_2)_\gamma \stackrel{\text{def}}{=} \mathsf{op}^\gamma \circ \iota \circ \textstyle\prod_o \Big(([\![\Gamma, y\!:\!O[V/x]; M]\!]_2)_{\langle \gamma, o\rangle}\Big) \circ \langle \mathsf{id}_1 \rangle_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)} : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))$$

where op^γ is the corresponding *operation* of $[\![\Gamma; \underline{C}]\!]_2(\gamma)$, i.e.,

$$([\![\Gamma; \underline{C}]\!]_2(\gamma))(\overrightarrow{x_o} \vdash \mathsf{op}_{([\![\Gamma; V]\!]_2)_\gamma(\star)}(x_o)_{1 \leq o \leq |[\![\Gamma; O[V/x]]\!]_2(\gamma)|}) : ([\![\Gamma; \underline{C}]\!]_2(\gamma))(|[\![\Gamma; O[V/x]]\!]_2(\gamma)|) \longrightarrow ([\![\Gamma; \underline{C}]\!]_2(\gamma))(1)$$

and where ι is the following countable-product preservation isomorphism:

$$\textstyle\prod_o([\![\Gamma; \underline{C}]\!]_2(\gamma))(1) \cong ([\![\Gamma; \underline{C}]\!]_2(\gamma))(|[\![\Gamma; O[V/x]]\!]_2(\gamma)|)$$

Next, we define $[\![-]\!]$ on the user-defined algebra type $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}\rangle$ as follows:

$$[\![\Gamma; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}\rangle]\!] \stackrel{\text{def}}{=} ([\![\Gamma]\!], \gamma \mapsto \mathcal{M}^\gamma)$$

where the functors $\mathcal{M}^\gamma : \mathcal{L}_{\mathcal{T}_{\text{eff}}} \longrightarrow \mathrm{Set}$ are defined on morphisms of the form $n \longrightarrow 1$ as follows:

$$\mathcal{M}^\gamma(n) \stackrel{\text{def}}{=} \textstyle\prod_{1 \leq j \leq n} [\![\Gamma; A]\!]_2(\gamma) \qquad \mathcal{M}^\gamma(\overrightarrow{x_j} \vdash x_j) \stackrel{\text{def}}{=} \mathsf{proj}_j$$

$$\mathcal{M}^\gamma(\Delta \vdash \mathsf{op}_i(t_o)_{1 \leq o \leq |[\![x:I; O]\!]_2(\langle \star, i\rangle)|}) \stackrel{\text{def}}{=} f^\gamma_{\mathsf{op}_i} \circ \langle \mathcal{M}^\gamma(\Delta \vdash t_o)\rangle_{o \in [\![x:I; O]\!]_2(\langle \star, i\rangle)}$$

where the function $f^\gamma_{\mathsf{op}_i}$ is derived from $[\![\Gamma; V_{\mathsf{op}}]\!]$ as follows:

$$f^\gamma_{\mathsf{op}_i} \stackrel{\text{def}}{=} f \mapsto \mathsf{proj}_{\langle i, f\rangle}\Big(([\![\Gamma; V_{\mathsf{op}}]\!]_2)_\gamma(\star)\Big) : \textstyle\prod_{o \in [\![x:I; O]\!]_2(\langle \star, i\rangle)} [\![\Gamma; A]\!]_2(\gamma) \longrightarrow [\![\Gamma; A]\!]_2(\gamma)$$

It is worth noting that each $\mathcal{M}^\gamma$ extends straightforwardly to m-tuples of terms, so as to account for all morphisms in $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$, i.e., those of the form $n \longrightarrow m$ for a general m. In detail, we have that

$$\mathcal{M}^\gamma((\Delta \vdash t)_{1 \leq j \leq m}) = \langle \mathcal{M}^\gamma(\Delta \vdash t)\rangle_{1 \leq j \leq m}$$

We also note that for $[\![\Gamma; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}\rangle]\!]$ to be defined, we additionally require that $\mathcal{M}^\gamma$ validate the equations given in $\mathcal{E}_{\text{eff}}$. In detail, we require for all $\Gamma' \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$ and $\gamma' \in [\![\Gamma']\!]$ that

$$\mathcal{M}^\gamma(\Delta^{\gamma'} \vdash T_1^{\gamma'}) = \mathcal{M}^\gamma(\Delta^{\gamma'} \vdash T_2^{\gamma'})$$

Finally, we define $[\![-]\!]$ on the two composition operations as follows:

$$[\![\Gamma; M \mathbin{\mathsf{as}} x\!:\!U\underline{C} \mathbin{\mathsf{in}_{\underline{D}}} N]\!]_1 \stackrel{\text{def}}{=} \mathsf{id}_{[\![\Gamma]\!]}$$

$$([\![\Gamma; M \mathbin{\mathsf{as}} x\!:\!U\underline{C} \mathbin{\mathsf{in}_{\underline{D}}} N]\!]_2)_\gamma \stackrel{\text{def}}{=} f^\gamma \circ ([\![\Gamma; M]\!]_2)_\gamma : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{D}]\!]_2(\gamma))$$

$$[\![\Gamma; z\!:\!\underline{C}'; K \mathbin{\mathsf{as}} x\!:\!U\underline{C} \mathbin{\mathsf{in}_{\underline{D}}} N]\!]_1 \stackrel{\text{def}}{=} \mathsf{id}_{[\![\Gamma]\!]}$$

$$([\![\Gamma; z\!:\!\underline{C}'; K \mathbin{\mathsf{as}} x\!:\!U\underline{C} \mathbin{\mathsf{in}_{\underline{D}}} N]\!]_2)_\gamma \stackrel{\text{def}}{=} \mathsf{hom}(f^\gamma) \circ ([\![\Gamma; z\!:\!\underline{C}'; K]\!]_2)_\gamma : [\![\Gamma; \underline{C}]\!]_2(\gamma) \longrightarrow [\![\Gamma; \underline{D}]\!]_2(\gamma)$$

where the function f^γ is derived from $\llbracket \Gamma, x : U\underline{C}; N \rrbracket$ as follows:

$$f^\gamma \stackrel{\text{def}}{=} c \mapsto (\llbracket \Gamma, x : U\underline{C}; N \rrbracket_2)_{\langle \gamma, c\rangle}(\star) : U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma)) \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}(\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))$$

and where $\mathsf{hom}(f^\gamma)$ is a morphism of models of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$, given by components

$$(\mathsf{hom}(f^\gamma))_n \stackrel{\text{def}}{=} \iota_{\llbracket \Gamma; \underline{D} \rrbracket} \circ \textstyle\prod_{1 \leq j \leq n}(f^\gamma) \circ \iota^{-1}_{\llbracket \Gamma; \underline{C} \rrbracket} : (\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(n) \longrightarrow (\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(n)$$

where $\iota_{\llbracket \Gamma; \underline{C} \rrbracket}$ and $\iota_{\llbracket \Gamma; \underline{D} \rrbracket}$ are respectively the following countable-product preservation isomorphisms:

$$\textstyle\prod_{1 \leq j \leq n}(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(1) \cong (\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(n) \qquad \prod_{1 \leq j \leq n}(\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(1) \cong (\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(n)$$

We note that for these two cases of $\llbracket - \rrbracket$ to be defined, we additionally require that the function f^γ commutes with the operations of $\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma)$ and $\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma)$, as depicted in the following diagram:

$$\begin{tikzcd}
\prod_o(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(1) \arrow[rr, "\prod_{o \in \llbracket x:I;O \rrbracket_2(\langle \star, i\rangle)}(f^\gamma)"] \arrow[d, "\cong"'] & & \prod_o(\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(1) \arrow[d, "\cong"] \\
(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(|\llbracket x : I; O \rrbracket_2(\langle \star, i\rangle)|) \arrow[d, "(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"'] & & (\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(|\llbracket x : I; O \rrbracket_2(\langle \star, i\rangle)|) \arrow[d, "(\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"] \\
(\llbracket \Gamma; \underline{C} \rrbracket_2(\gamma))(1) \arrow[rr, "f^\gamma"'] & & (\llbracket \Gamma; \underline{D} \rrbracket_2(\gamma))(1)
\end{tikzcd}$$

8.5 Soundness

In order to establish the soundness of the interpretation we defined for our extension of eMLTT in the previous section, we first prove standard semantic weakening and substitution lemmas. In detail, we begin by defining *a priori* partial *semantic projection* and *substitution* morphisms

$$\mathsf{proj}_{\Gamma_1; x:A; \Gamma_2} : \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \longrightarrow \llbracket \Gamma_1, \Gamma_2 \rrbracket \qquad \mathsf{subst}_{\Gamma_1; x:A; \Gamma_2; V} : \llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket \longrightarrow \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$$

by induction on the size of Γ_2 as follows:

$$\begin{array}{ll}
\mathsf{proj}_{\Gamma_1; x:A; \diamond} \stackrel{\text{def}}{=} \pi_{\llbracket \Gamma_1; A \rrbracket} & \mathsf{proj}_{\Gamma_1; x:A; \Gamma_2, y:B} \stackrel{\text{def}}{=} \overline{\{\mathsf{proj}_{\Gamma_1; x:A; \Gamma_2}}(\llbracket \Gamma_1, \Gamma_2; B \rrbracket)\} \\
\mathsf{subst}_{\Gamma_1; x:A; \diamond; V} \stackrel{\text{def}}{=} \llbracket \Gamma; V \rrbracket & \mathsf{subst}_{\Gamma_1; x:A; \Gamma_2, y:B; V} \stackrel{\text{def}}{=} \overline{\{\mathsf{subst}_{\Gamma_1; x:A; \Gamma_2; V}}(\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket)\}
\end{array}$$

We then show that both morphisms are defined if the interpretations of the involved contexts and terms are defined, and that reindexing along them indeed models weakening and substitution.

Proposition 8.2. *Given value contexts Γ_1 and Γ_2, a value type A, and a value variable x such that $x \notin Vars(\Gamma_1)$, $x \notin Vars(\Gamma_2)$, $\llbracket \Gamma_1, \Gamma_2 \rrbracket \in \mathcal{B}$, and $\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \in \mathcal{B}$, then i) the semantic projection morphism $\mathsf{proj}_{\Gamma_1; x:A; \Gamma_2}$ is defined; and ii) given a value type B such that $\llbracket \Gamma_1, \Gamma_2; B \rrbracket \in \mathcal{V}_{\llbracket \Gamma_1, \Gamma_2 \rrbracket}$, then*

$$\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket = \mathsf{proj}^*_{\Gamma_1; x:A; \Gamma_2}(\llbracket \Gamma_1, \Gamma_2; B \rrbracket)$$

and similarly for computation types, and value, computation, and homomorphism terms.

Proposition 8.3. *Given value contexts Γ_1 and Γ_2, a value type A, a value variable x, and a value term V such that $x \notin Vars(\Gamma_1)$, $x \notin Vars(\Gamma_2)$, $\llbracket \Gamma_1; V \rrbracket : 1(\llbracket \Gamma_1 \rrbracket) \longrightarrow \llbracket \Gamma_1; A \rrbracket$, $\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \in \mathcal{B}$, and $\llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket \in \mathcal{B}$, then i) the semantic substitution morphism $\mathsf{subst}_{\Gamma_1; x:A; \Gamma_2; V}$ is defined; and ii) given a value type B such that $\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket \in \mathcal{V}_{\llbracket \Gamma_1, x:A, \Gamma_2 \rrbracket}$, then*

$$\llbracket \Gamma_1, \Gamma_2[V/x]; B[V/x] \rrbracket = \mathsf{subst}^*_{\Gamma_1; x:A; \Gamma_2; V}(\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket)$$

and similarly for computation types, and value, computation, and homomorphism terms.

In addition, we show that substituting computation and homomorphism terms for computation variables corresponds to composition of the morphisms that the given terms denote.

Proposition 8.4. *Given a value context* Γ, *a computation variable* z, *a computation type* $\underline{C}$, *a computation term* M, *and a homomorphism term* K *such that* $[\![\Gamma; M]\!] : 1([\![\Gamma]\!]) \longrightarrow U([\![\Gamma; \underline{C}]\!])$ *and* $[\![\Gamma; z\!:\!\underline{C}; K]\!] : [\![\Gamma; \underline{C}]\!] \longrightarrow \underline{D}$, *then we have*

$$[\![\Gamma; K[M/z]]\!] = U([\![\Gamma; z : \underline{C}; K]\!]) \circ [\![\Gamma; M]\!] : 1([\![\Gamma]\!]) \longrightarrow U(\underline{D})$$

in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Mod})$, *and similarly for homomorphism terms.*

Finally, we prove the soundness of the interpretation of our extension of eMLTT.

Theorem 8.5 (Soundness). $[\![-]\!]$ *is defined on all well-formed contexts, well-formed types, and well-typed terms, and it further identifies definitionally equal contexts, types, and terms.*

Proof. We prove this theorem by induction on the given derivations. In particular, for the definitional equations that correspond to the equations given in $\mathcal{E}_{\mathrm{eff}}$ (see Fig. 4), we recall that these equations hold in $\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}$ by construction. Therefore, all models of the countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}$ validate them, including those modelling our computation types. For computation types, we prove that $[\![\Gamma; \langle A, \{V_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle]\!]$ is defined by observing that the equational proof obligations included in $\Gamma \vdash \{V_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$ on A ensure that each $\mathcal{M}^{\gamma}$ validates the equations given in $\mathcal{E}_{\mathrm{eff}}$, as required in §8.4. For computation terms, we prove that $[\![\Gamma; M \,\mathtt{as}\, x\!:\!U\underline{C} \,\mathtt{in}_{\underline{D}}\, N]\!]$ is defined by observing that the equational proof obligations included in $\Gamma, x\!:\!U\underline{C} \vdash_{\mathrm{hom}} N : \underline{D}$ ensure that the functions f^{γ} we derive from $[\![\Gamma, x\!:\!U\underline{C}; N]\!]$ commute with the operations of $[\![\Gamma; \underline{C}]\!]_2(\gamma)$ and $[\![\Gamma; \underline{D}]\!]_2(\gamma)$; the case for the other composition operation, namely, $K \,\mathtt{as}\, x\!:\!U\underline{C} \,\mathtt{in}_{\underline{D}}\, N$, is proved analogously. ◻

9 CONCLUSIONS AND FUTURE WORK

In this paper we have given a comprehensive account of algebraic effects and their handlers in the dependently typed setting. In detail, we gave handlers a novel type-based treatment and demonstrated that being able to handle computations into values provides a useful mechanism for reasoning about effectful computations. We also showed how to equip the resulting language with a denotational semantics, based on families fibrations and models of countable Lawvere theories.

In future, we plan to combine our treatment of handlers with effect-typing (Kammar et al. 2013) and multi-handlers (Lindley et al. 2017). We also plan to compare our handler-based definition of Dijkstra's predicate transformers with their CPS-based definition used in the F* language (Ahman et al. 2017). Further, we plan to extend computation terms with recursion following the analyses of Ahman et al. (2016); Plotkin and Pretnar (2013). Specifically, we plan to generalise from an equational presentation of effects to an inequational presentation, and develop the corresponding denotational semantics using fibrations of continuous families of ω-cpos and models of countable discrete CPO-enriched Lawvere theories (Hyland and Power 2006). More generally, we plan to extend our work from families fibrations to more general fibrational models of dependent types, where definitional and propositional proof obligations (see discussion in §4.3) might not coincide.

ACKNOWLEDGMENTS

The author is thankful to Sam Lindley, Gordon Plotkin, and Tarmo Uustalu for useful discussions.

REFERENCES

J. Adamek and J. Rosicky. 1994. *Locally Presentable and Accessible Categories.* Number 189 in London Mathematical Society Lecture Note Series. Cambridge Univ. Press.

Danel Ahman, James Chapman, and Tarmo Uustalu. 2014. When is a container a comonad? *Logical Methods in Computer Science* 10, 3 (2014).

Danel Ahman, Neil Ghani, and Gordon D. Plotkin. 2016. Dependent Types and Fibred Computational Effects. In *Proc. of 19th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS 2016 (LNCS)*, Vol. 9634. Springer, 1–19.
Danel Ahman, Cătălin Hriţcu, Kenji Maillard, Guido Martínez, Gordon Plotkin, Jonathan Protzenko, Aseem Rastogi, and Nikhil Swamy. 2017. Dijkstra Monads for Free. In *Proc. of 44th ACM SIGPLAN Symp. on Principles of Programming Languages, POPL 2017*. ACM, 515–529.
Danel Ahman and Sam Staton. 2013. Normalization by Evaluation and Algebraic Effects. In *Proc. of 29th Conf. on the Mathematical Foundations of Programming Semantics, MFPS XXIX (ENTCS)*, Vol. 298. Elsevier, 51–69.
Danel Ahman and Tarmo Uustalu. 2014. Update Monads: Cointerpreting Directed Containers. In *Post-proc. of the 19th Meeting "Types for Proofs and Programs", TYPES 2013 (LIPIcs)*, Vol. 26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 1–23.
Andrej Bauer and Matija Pretnar. 2015. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.* 84, 1 (2015), 108–123.
Edwin Brady. 2013. Programming and reasoning with algebraic effects and dependent types. In *Proc. of 18th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP 2013*. ACM, 133–144.
Chris Casinghino. 2014. *Combining Proofs and Programs*. Ph.D. Dissertation. University of Pennsylvania.
Edsger W. Dijkstra. 1975. Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *CACM* 18, 8 (1975), 5.
Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. 2014. The enriched effect calculus: syntax and semantics. *J. Log. Comput.* 24, 3 (2014), 615–654.
Neil Ghani, Lorenzo Malatesta, Fredrik Nordvall Forsberg, and Anton Setzer. 2013. Fibred Data Types. In *Proc. of 28th Ann. Symp on Logic in Computer Science, LICS 2013*. IEEE Computer Society, 243–252.
George A. Grätzer. 1979. *Universal Algebra* (2nd ed.). Springer.
Peter Hancock and Anton Setzer. 2000. Interactive programs in dependent type theory. In *Proc. of 14th Ann. Conf. of the EACSL on Computer Science Logic, CSL 2000 (LNCS)*, Vol. 1862. Springer, 317–331.
Daniel Hillerström and Sam Lindley. 2016. Liberating Effects with Rows and Handlers. In *Proc. of 1st Wksh. on Type-Driven Development, TyDe 2016*. ACM, 15–27.
Martin Hofmann. 1995. *Extensional concepts in intensional type theory*. Ph.D. Dissertation. Laboratory for Foundations in Computer Science, University of Edinburgh.
Martin Hofmann. 1997. Syntax and Semantics of Dependent Types. In *Semantics and Logics of Computation*, Andrew M. Pitts and P. Dybjer (Eds.). Cambridge Univ. Press, 79–130.
Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *Proc. of 7th European Symp. on Programming, ESOP 1998 (LNCS)*, Vol. 1381. Springer, 122–138.
Martin Hyland, Gordon Plotkin, and John Power. 2006. Combining effects: Sum and tensor. *Theor. Comput. Sci.* 357, 1–3 (2006), 70–99.
Martin Hyland and John Power. 2006. Discrete Lawvere theories and computational effects. *Theor. Comput. Sci.* 366, 1-2 (2006), 144–162.
Bart Jacobs. 1999. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Elsevier.
Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in Action. In *Proc. of 18th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP 2013*. ACM, 145–158.
Ohad Kammar and Gordon D. Plotkin. 2012. Algebraic Foundations for Effect-dependent Optimisations. In *Proc. of 39th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL 2012*. ACM, 349–360.
Daan Leijen. 2017. Type Directed Compilation of Row-Typed Algebraic Effects. In *Proc. of 44th ACM SIGPLAN Symp. on Principles of Programming Languages, POPL 2017*. ACM, 486–499.
Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer.
Sam Lindley, Conor McBride, and Craig McLaughlin. 2017. Do Be Do Be Do. In *Proc. of 44th ACM SIGPLAN Symp. on Principles of Programming Languages, POPL 2017*. ACM, 500–514.
Per Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis.
Conor McBride. 2011. Functional Pearl: Kleisli arrows of outrageous fortune. *J. Funct. Program.* (2011). (To appear).
Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proc. of 4th Ann. Symp. on Logic in Computer Science, LICS 1989*, Rohit Parikh (Ed.). IEEE, 14–23.
Eugenio Moggi. 1991. Notions of Computation and Monads. *Inf. Comput.* 93, 1 (1991), 55–92.
Aleksandar Nanevski, Greg Morrisett, and Lars Birkedal. 2008. Hoare Type Theory, polymorphism and separation. *J. Funct. Program.* 18, 5-6 (2008), 865–911.
M. Okada and P. J. Scott. 1999. A Note on Rewriting Theory for Uniqueness of Iteration. *Theory Appl. Categ.* 6, 4 (1999), 47–64.

Pierre-Marie Pédrot and Nicolas Tabareau. 2017. An Effectful Way to Eliminate Addiction to Dependence. In *Proc. of 32nd Ann. Symp on Logic in Computer Science, LICS 2017.* To appear.

A. M. Pitts. 1991. Evaluation Logic. In *Proc. IVth Higher Order Workshop (Workshops in Computing)*. Springer, 162–189.

A. M. Pitts, J. Matthiesen, and J. Derikx. 2015. A Dependent Type Theory with Abstractable Names. In *Proc. of 9th Wksh. on Logical and Semantic Frameworks, with Applications, LSFA 2014 (ENTCS)*, Vol. 312. Elsevier, 19–50.

Gordon Plotkin and John Power. 2001. Semantics for Algebraic Operations. In *Proc. of 17th Conf. on the Mathematical Foundations of Programming Semantics, MFPS XVII (ENTCS)*, Vol. 45. Elsevier, 332–345.

Gordon D. Plotkin and John Power. 2002. Notions of Computation Determine Monads. In *Proc. of 5th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS 2002 (LNCS)*, Vol. 2303. Springer, 342–356.

Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *Logical Methods in Computer Science* 9, 4:23 (2013).

John Power. 2006. Countable Lawvere Theories and Computational Effects. In *Proc. of 3rd Irish Conf. on the Mathematical Foundations of Computer Science and Information Technology, MFCSIT 2004 (ENTCS)*, Vol. 161. Elsevier, 59–71.

Thomas Streicher. 1991. *Semantics of Type Theory. Correctness, Completeness and Independence Results.* Birkhäuser Boston.

A TYPING RULES FOR THE CORE OF EMLTT

Value contexts:

$$\frac{}{\vdash \diamond} \qquad \frac{\vdash \Gamma \quad \Gamma \vdash A \quad x \notin \mathit{Vars}(\Gamma)}{\vdash \Gamma, x : A}$$

Value and computation variables:

$$\frac{\vdash \Gamma_1, x : A, \Gamma_2}{\Gamma_1, x : A, \Gamma_2 \vdash x : A} \qquad \frac{\Gamma \vdash \underline{C}}{\Gamma \mid z : \underline{C} \vdash z : \underline{C}}$$

Type of natural numbers:

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash V : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\ V : \mathsf{Nat}}$$

$$\frac{\Gamma, x : \mathsf{Nat} \vdash A \quad \Gamma \vdash V : \mathsf{Nat} \quad \Gamma \vdash V_z : A[\mathsf{zero}/x] \quad \Gamma, y_1 : \mathsf{Nat}, y_2 : A[y_1/x] \vdash V_s : A[\mathsf{succ}\ y_1/x]}{\Gamma \vdash \mathsf{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, V) : A[V/x]}$$

Unit type:

$$\frac{\vdash \Gamma}{\Gamma \vdash \star : 1}$$

Empty type:

$$\frac{\Gamma, x : 0 \vdash A \quad \Gamma \vdash V : 0}{\Gamma \vdash \mathsf{case}\ V\ \mathsf{of}_{x.A}\ () : A[V/x]}$$

Coproduct type:

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{inl}_{A+B}\ V : A + B} \qquad \frac{\Gamma \vdash V : B}{\Gamma \vdash \mathsf{inr}_{A+B}\ V : A + B}$$

$$\frac{\Gamma, x : A_1 + A_2 \vdash B \quad \Gamma \vdash V : A_1 + A_2 \quad \Gamma, y_1 : A_1 \vdash W_1 : B[\mathsf{inl}_{A_1+A_2}\ y_1/x] \quad \Gamma, y_2 : A_2 \vdash W_2 : B[\mathsf{inr}_{A_1+A_2}\ y_2/x]}{\Gamma \vdash \mathsf{case}\ V\ \mathsf{of}_{x.B}\ (\mathsf{inl}(y_1 : A_1) \mapsto W_1, \mathsf{inr}(y_2 : A_2) \mapsto W_2) : B[V/x]}$$

Value Σ-type:

$$\frac{\Gamma \vdash V : A \quad \Gamma, x : A \vdash B \quad \Gamma \vdash W : B[V/x]}{\Gamma \vdash \langle V, W \rangle_{(x:A).B} : \Sigma x : A.B}$$

$$\frac{\Gamma, y : \Sigma x_1 : A_1.A_2 \vdash B \quad \Gamma \vdash V : \Sigma x_1 : A_1.A_2 \quad \Gamma, x_1 : A_1, x_2 : A_2 \vdash W : B[\langle x_1, x_2 \rangle_{(x_1:A_1).A_2}/y]}{\Gamma \vdash \mathsf{pm}\ V\ \mathsf{as}\ (x_1 : A_1, x_2 : A_2)\ \mathsf{in}_{y.B}\ W : B[V/y]}$$

Value Π-type:

$$\frac{\Gamma, x : A \vdash V : B}{\Gamma \vdash \lambda x : A.V : \Pi x : A.B} \qquad \frac{\Gamma, x : A \vdash B \quad \Gamma \vdash V : \Pi x : A.B \quad \Gamma \vdash W : A}{\Gamma \vdash V(W)_{(x:A).B} : B[W/x]}$$

Propositional equality:

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{refl}\ V : V =_A V} \qquad \frac{\begin{array}{c}\Gamma \vdash A \quad \Gamma, x_1 : A, x_2 : A, x_3 : x_1 =_A x_2 \vdash B \quad \Gamma \vdash V_1 : A \quad \Gamma \vdash V_2 : A \\ \Gamma \vdash V_p : V_1 =_A V_2 \quad \Gamma, y : A \vdash W : B[y/x_1][y/x_2][\mathsf{refl}\ y/x_3]\end{array}}{\Gamma \vdash \mathsf{eq\text{-}elim}_A(x_1.x_2.x_3.B, y.W, V_1, V_2, V_p) : B[V_1/x_1][V_2/x_2][V_p/x_3]}$$

Thunking and forcing:

$$\frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash \mathsf{thunk}\ M : U\underline{C}} \qquad \frac{\Gamma \vdash V : U\underline{C}}{\Gamma \vdash \mathsf{force}_{\underline{C}}\ V : \underline{C}}$$

Homomorphic function type:

$$\frac{\Gamma \mid z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \vdash \lambda z\!:\!\underline{C}.K : \underline{C} \multimap \underline{D}} \qquad \frac{\Gamma \vdash V : \underline{C} \multimap \underline{D} \quad \Gamma \vdash M : \underline{C}}{\Gamma \vdash V(M)_{\underline{C},\underline{D}} : \underline{D}} \qquad \frac{\Gamma \vdash V : \underline{D}_1 \multimap \underline{D}_2 \quad \Gamma \mid z\!:\!\underline{C} \vdash K : \underline{D}_1}{\Gamma \mid z\!:\!\underline{C} \vdash V(K)_{\underline{D}_1,\underline{D}_2} : \underline{D}_2}$$

Sequential composition:

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, x\!:\!A \vdash N : \underline{C}}{\Gamma \vdash M\ \mathtt{to}\ x\!:\!A\ \mathtt{in}_{\underline{C}}\ N : \underline{C}} \qquad \frac{\Gamma \mid z\!:\!\underline{C} \vdash K : FA \quad \Gamma \vdash \underline{D} \quad \Gamma, x\!:\!A \vdash M : \underline{D}}{\Gamma \mid z\!:\!\underline{C} \vdash K\ \mathtt{to}\ x\!:\!A\ \mathtt{in}_{\underline{D}}\ M : \underline{D}}$$

Computational Σ-type:

$$\frac{\Gamma \vdash V : A \quad \Gamma, x\!:\!A \vdash \underline{C} \quad \Gamma \vdash M : \underline{C}[V/x]}{\Gamma \vdash \langle V, M\rangle_{(x:A).\underline{C}} : \Sigma x\!:\!A.\underline{C}} \qquad \frac{\Gamma \vdash M : \Sigma x\!:\!A.\underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x\!:\!A \mid z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \vdash M\ \mathtt{to}\ (x\!:\!A, z\!:\!\underline{C})\ \mathtt{in}_{\underline{D}}\ K : \underline{D}}$$

$$\frac{\Gamma \vdash V : A \quad \Gamma, x\!:\!A \vdash \underline{D} \quad \Gamma \mid z\!:\!\underline{C} \vdash K : \underline{D}[V/x]}{\Gamma \mid z\!:\!\underline{C} \vdash \langle V, K\rangle_{(x:A).\underline{D}} : \Sigma x\!:\!A.\underline{D}} \qquad \frac{\Gamma \mid z_1\!:\!\underline{C} \vdash K : \Sigma x\!:\!A.\underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x\!:\!A \mid z_2\!:\!\underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \mid z_1\!:\!\underline{C} \vdash K\ \mathtt{to}\ (x\!:\!A, z_2\!:\!\underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L : \underline{D}_2}$$

Computational Π-type:

$$\frac{\Gamma, x\!:\!A \vdash M : \underline{C}}{\Gamma \vdash \lambda x\!:\!A.M : \Pi x\!:\!A.\underline{C}} \qquad \frac{\Gamma, x\!:\!A \vdash \underline{C} \quad \Gamma \vdash M : \Pi x\!:\!A.\underline{C} \quad \Gamma \vdash V : A}{\Gamma \vdash M(V)_{(x:A).\underline{C}} : \underline{C}[V/x]}$$

$$\frac{\Gamma \vdash \underline{C} \quad \Gamma, x\!:\!A \mid z\!:\!\underline{C} \vdash K : \underline{D}}{\Gamma \mid z\!:\!\underline{C} \vdash \lambda x\!:\!A.K : \Pi x\!:\!A.\underline{D}} \qquad \frac{\Gamma, x\!:\!A \vdash \underline{D} \quad \Gamma \mid z\!:\!\underline{C} \vdash K : \Pi x\!:\!A.\underline{D} \quad \Gamma \vdash V : A}{\Gamma \mid z\!:\!\underline{C} \vdash K(V)_{(x:A).\underline{D}} : \underline{D}[V/x]}$$

Context and type conversion rules:

$$\frac{\Gamma_1 \vdash A \quad \vdash \Gamma_1 = \Gamma_2}{\Gamma_2 \vdash A} \qquad \frac{\Gamma_1 \vdash \underline{C} \quad \vdash \Gamma_1 = \Gamma_2}{\Gamma_2 \vdash \underline{C}}$$

$$\frac{\Gamma_1 \vdash V : A \quad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash A = B}{\Gamma_2 \vdash V : B} \qquad \frac{\Gamma_1 \vdash M : \underline{C} \quad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash \underline{C} = \underline{D}}{\Gamma_2 \vdash M : \underline{D}}$$

$$\frac{\Gamma_1 \mid z\!:\!\underline{C}_1 \vdash K : \underline{D}_1 \quad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash \underline{C}_1 = \underline{C}_2 \quad \Gamma_1 \vdash \underline{D}_1 = \underline{D}_2}{\Gamma_2 \mid z\!:\!\underline{C}_2 \vdash K : \underline{D}_2}$$

B DEFINITIONAL EQUATIONS FOR THE CORE OF EMLTT

We omit the rules for the standard equations of reflexivity, symmetry, transitivity, replacement, and congruence.

Value contexts:

$$\frac{}{\vdash \diamond = \diamond} \qquad \frac{\vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash A = B \quad x \notin \mathit{Vars}(\Gamma_1) \quad x \notin \mathit{Vars}(\Gamma_2)}{\vdash \Gamma_1, x\!:\!A = \Gamma_2, x\!:\!B}$$

Type of natural numbers:

$$\frac{\Gamma, x\!:\!\mathsf{Nat} \vdash A \quad \Gamma \vdash V_z : A[\mathsf{zero}/x] \quad \Gamma, y_1\!:\!\mathsf{Nat}, y_2\!:\!A[y_1/x] \vdash V_s : A[\mathsf{succ}\ y_1/x]}{\Gamma \vdash \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, \mathsf{zero}) = V_z : A[\mathsf{zero}/x]}$$

$$\frac{\Gamma, x\!:\!\mathsf{Nat} \vdash A \quad \Gamma \vdash V : \mathsf{Nat} \quad \Gamma \vdash V_z : A[\mathsf{zero}/x] \quad \Gamma, y_1\!:\!\mathsf{Nat}, y_2\!:\!A[y_1/x] \vdash V_s : A[\mathsf{succ}\ y_1/x]}{\Gamma \vdash \mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, \mathsf{succ}\ V) = V_s[V/y_1][\mathtt{nat\text{-}elim}_{x.A}(V_z, y_1.y_2.V_s, V)/y_2] : A[\mathsf{succ}\ V/x]}$$

Unit type:

$$\frac{\Gamma \vdash V : 1}{\Gamma \vdash V = \star : 1}$$

Empty type:

$$\frac{\Gamma, x\!:\!0 \vdash A \quad \Gamma \vdash V : 0 \quad \Gamma, x\!:\!0 \vdash W : A}{\Gamma \vdash \mathsf{case}\ V\ \mathsf{of}_{x.A}\ () = W[V/x] : A[V/x]}$$

Coproduct type:

$$\frac{\Gamma, x\!:\!A_1 + A_2 \vdash B \quad \Gamma \vdash V : A_1 \quad \Gamma, y_1\!:\!A_1 \vdash W_1 : B[\mathtt{inl}_{A_1+A_2}\ y_1/x] \quad \Gamma, y_2\!:\!A_2 \vdash W_2 : B[\mathtt{inr}_{A_1+A_2}\ y_2/x]}{\Gamma \vdash \mathsf{case}\ (\mathtt{inl}_{A_1+A_2}\ V)\ \mathsf{of}_{x.B}\ (\mathtt{inl}(y_1\!:\!A_1) \mapsto W_1, \mathtt{inr}(y_2\!:\!A_2) \mapsto W_2) = W_1[V/y_1] : B[\mathtt{inl}_{A_1+A_2}\ V/x]}$$

$$\frac{\Gamma, x\!:\!A_1 + A_2 \vdash B \quad \Gamma \vdash V : A_2 \quad \Gamma, y_1\!:\!A_1 \vdash W_1 : B[\mathtt{inl}_{A_1+A_2}\ y_1/x] \quad \Gamma, y_2\!:\!A_2 \vdash W_2 : B[\mathtt{inr}_{A_1+A_2}\ y_2/x]}{\Gamma \vdash \mathsf{case}\ (\mathtt{inr}_{A_1+A_2}\ V)\ \mathsf{of}_{x.B}\ (\mathtt{inl}(y_1\!:\!A_1) \mapsto W_1, \mathtt{inr}(y_2\!:\!A_2) \mapsto W_2) = W_2[V/y_2] : B[\mathtt{inr}_{A_1+A_2}\ V/x]}$$

$$\frac{\begin{array}{c} y_1 \notin \mathit{Vars}(\Gamma) \cup \{x_1\} \quad y_2 \notin \mathit{Vars}(\Gamma) \cup \{x_2\} \quad x_2 \notin \{y_1, y_2\} \\ \Gamma, x_1\!:\!A_1 + A_2 \vdash B \quad \Gamma \vdash V : A_1 + A_2 \quad \Gamma, x_2\!:\!A_1 + A_2 \vdash W : B \end{array}}{\begin{array}{l} \Gamma \vdash \mathsf{case}\ V\ \mathsf{of}_{x_1.B}\ (\mathtt{inl}(y_1\!:\!A_1) \mapsto W[\mathtt{inl}_{A_1+A_2}\ y_1/x_2], \\ \qquad\qquad \mathtt{inr}(y_2\!:\!A_2) \mapsto W[\mathtt{inr}_{A_1+A_2}\ y_2/x_2]) = W[V/x_2] : B[V/x_1] \end{array}}$$

Value Σ-type:

$$\frac{\Gamma, y\!:\!\Sigma x_1\!:\!A_1.A_2 \vdash B \quad \Gamma \vdash V_1 : A_1 \quad \Gamma \vdash V_2 : A_2[V_1/x_1] \quad \Gamma, x_1\!:\!A_1, x_2\!:\!A_2 \vdash W : B[\langle x_1, x_2\rangle_{(x_1:A_1).A_2}/y]}{\Gamma \vdash \mathsf{pm}\ \langle V_1, V_2\rangle_{(x_1:A_1).A_2}\ \mathsf{as}\ (x_1\!:\!A_1, x_2\!:\!A_2)\ \mathsf{in}_{y.B}\ W = W[V_1/x_1][V_2/x_2] : B[\langle V_1, V_2\rangle_{(x_1:A_1).A_2}/y]}$$

$$\frac{\begin{array}{c} \Gamma \vdash A_1 \quad \Gamma, x_1\!:\!A_1 \vdash A_2 \quad x_2 \notin \mathit{Vars}(\Gamma) \cup \{x_1\} \quad y_2 \notin \{x_1, x_2\} \\ \Gamma \vdash V : \Sigma x_1\!:\!A_1.A_2 \quad \Gamma, y_1\!:\!\Sigma x_1\!:\!A_1.A_2 \vdash B \quad \Gamma, y_2\!:\!\Sigma x_1\!:\!A_1.A_2 \vdash W : B[y_2/y_1] \end{array}}{\Gamma \vdash \mathsf{pm}\ V\ \mathsf{as}\ (x_1\!:\!A_1, x_2\!:\!A_2)\ \mathsf{in}_{y_1.B}\ W[\langle x_1, x_2\rangle_{(x_1:A_1).A_2}/y_2] = W[V/y_2] : B[V/y_1]}$$

Value Π-type:

$$\frac{\Gamma, x\!:\!A \vdash V : B \quad \Gamma \vdash W : A}{\Gamma \vdash (\lambda x\!:\!A.V)(W)_{(x:A).B} = V[W/x] : B[W/x]} \qquad \frac{\Gamma, x\!:\!A \vdash B \quad \Gamma \vdash V : \Pi x\!:\!A.B}{\Gamma \vdash V = \lambda x\!:\!A.V(x)_{(x:A).B} : \Pi x\!:\!A.B}$$

Propositional equality:

$$\frac{\Gamma \vdash A \quad \Gamma, x_1 \!:\! A, x_2 \!:\! A, x_3 \!:\! x_1 =_A x_2 \vdash B \quad \Gamma \vdash V : A \quad \Gamma, y \!:\! A \vdash W : B[y/x_1][y/x_2][\mathtt{refl}\ y/x_3]}{\Gamma \vdash \mathtt{eq\text{-}elim}_A(x_1.x_2.x_3.B, y.W, V, V, \mathtt{refl}\ V) = W[V/y] : B[V/x_1][V/x_2][\mathtt{refl}\ V/x_3]}$$

Thunking and forcing:

$$\frac{\Gamma \vdash V : U\underline{C}}{\Gamma \vdash \mathtt{thunk}\ (\mathtt{force}_{\underline{C}}\ V) = V : U\underline{C}} \qquad \frac{\Gamma \vdash M : \underline{C}}{\Gamma \vdash \mathtt{force}_{\underline{C}}\ (\mathtt{thunk}\ M) = M : \underline{C}}$$

Homomorphic function type:

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \,|\, z \!:\! \underline{C} \vdash K : \underline{D}}{\Gamma \vdash (\lambda z \!:\! \underline{C}.K)(M)_{\underline{C},\underline{D}} = K[M/z] : \underline{D}} \qquad \frac{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash K : \underline{D}_1 \quad \Gamma \,|\, z_2 \!:\! \underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash (\lambda z_2 \!:\! \underline{D}_1.L)(K)_{\underline{D}_1,\underline{D}_2} = L[K/z_2] : \underline{D}_2}$$

$$\frac{\Gamma \vdash V : \underline{C} \multimap \underline{D}}{\Gamma \vdash V = \lambda z \!:\! \underline{C}.V(z)_{\underline{C},\underline{D}} : \underline{C} \multimap \underline{D}}$$

Sequential composition:

$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash \underline{C} \quad \Gamma, x \!:\! A \vdash M : \underline{C}}{\Gamma \vdash \mathtt{return}\ V\ \mathtt{to}\ x \!:\! A\ \mathtt{in}_{\underline{C}}\ M = M[V/x] : \underline{C}}$$

$$\frac{\Gamma \vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma \,|\, z \!:\! FA \vdash K : \underline{C}}{\Gamma \vdash M\ \mathtt{to}\ x \!:\! A\ \mathtt{in}_{\underline{C}}\ K[\mathtt{return}\ x/z] = K[M/z] : \underline{C}} \qquad \frac{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash K : FA \quad \Gamma \vdash \underline{D} \quad \Gamma \,|\, z_2 \!:\! FA \vdash L : \underline{D}}{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash K\ \mathtt{to}\ x \!:\! A\ \mathtt{in}_{\underline{D}}\ L[\mathtt{return}\ x/z_2] = L[K/z_2] : \underline{D}}$$

Computational Σ-type:

$$\frac{\Gamma \vdash V : A \quad \Gamma \vdash M : \underline{C}[V/x] \quad \Gamma \vdash \underline{D} \quad \Gamma, x \!:\! A \,|\, z \!:\! \underline{C} \vdash K : \underline{D}}{\Gamma \vdash \langle V, M \rangle_{(x:A).\underline{C}}\ \mathtt{to}\ (x \!:\! A, z \!:\! \underline{C})\ \mathtt{in}_{\underline{D}}\ K = K[V/x][M/z] : \underline{D}}$$

$$\frac{\Gamma, x \!:\! A \vdash \underline{C} \quad \Gamma \vdash M : \Sigma x \!:\! A.\underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma \,|\, z_2 \!:\! \Sigma x \!:\! A.\underline{C} \vdash K : \underline{D}}{\Gamma \vdash M\ \mathtt{to}\ (x \!:\! A, z_1 \!:\! \underline{C})\ \mathtt{in}_{\underline{D}}\ K[\langle x, z_1 \rangle_{(x:A).\underline{C}}/z_2] = K[M/z_2] : \underline{D}}$$

$$\frac{\Gamma \vdash V : A \quad \Gamma \,|\, z_1 \!:\! \underline{C} \vdash K : \underline{D}_1[V/x] \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x \!:\! A \,|\, z_2 \!:\! \underline{D}_1 \vdash L : \underline{D}_2}{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash \langle V, K \rangle_{(x:A).\underline{D}_1}\ \mathtt{to}\ (x \!:\! A, z_2 \!:\! \underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L = L[V/x][K/z_2] : \underline{D}_2}$$

$$\frac{\Gamma, x \!:\! A \vdash \underline{D}_1 \quad \Gamma \,|\, z_1 \!:\! \underline{C} \vdash K : \Sigma x \!:\! A.\underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma \,|\, z_3 \!:\! \Sigma x \!:\! A.\underline{D}_1 \vdash K : \underline{D}_2}{\Gamma \,|\, z_1 \!:\! \underline{C} \vdash K\ \mathtt{to}\ (x \!:\! A, z_2 \!:\! \underline{D}_1)\ \mathtt{in}_{\underline{D}_2}\ L[\langle x, z_2 \rangle_{(x:A).\underline{D}_1}/z_3] = L[K/z_3] : \underline{D}_2}$$

Computational Π-type:

$$\frac{\Gamma, x \!:\! A \vdash M : \underline{C} \quad \Gamma \vdash V : A}{\Gamma \vdash (\lambda x \!:\! A.M)(V)_{(x:A).\underline{C}} = M[V/x] : \underline{C}[V/x]} \qquad \frac{\Gamma, x \!:\! A \vdash \underline{C} \quad \Gamma \vdash M : \Pi x \!:\! A.\underline{C}}{\Gamma \vdash M = \lambda x \!:\! A.M(x)_{(x:A).\underline{C}} : \Pi x \!:\! A.\underline{C}}$$

$$\frac{\Gamma \vdash \underline{C} \quad \Gamma, x \!:\! A \,|\, z \!:\! \underline{C} \vdash K : \underline{D} \quad \Gamma \vdash V : A}{\Gamma \,|\, z \!:\! \underline{C} \vdash (\lambda x \!:\! A.K)(V)_{(x:A).\underline{D}} = K[V/x] : \underline{D}[V/x]} \qquad \frac{\Gamma, x \!:\! A \vdash \underline{D} \quad \Gamma \,|\, z \!:\! \underline{C} \vdash K : \Pi x \!:\! A.\underline{D}}{\Gamma \,|\, z \!:\! \underline{C} \vdash K = \lambda x \!:\! A.K(x)_{(x:A).\underline{D}} : \Pi x \!:\! A.\underline{D}}$$

Context and type conversion rules:

$$\frac{\Gamma_1 \vdash A = B \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_2 \vdash A = B} \qquad \frac{\Gamma_1 \vdash \underline{C} = \underline{D} \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_2 \vdash \underline{C} = \underline{D}}$$

$$\frac{\Gamma_1 \vdash V = W : A \qquad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash A = B}{\Gamma_2 \vdash V = W : B} \qquad \frac{\Gamma_1 \vdash M = N : \underline{C} \qquad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash \underline{C} = \underline{D}}{\Gamma_2 \vdash M = N : \underline{D}}$$

$$\frac{\Gamma_1 \,|\, z \colon \underline{C}_1 \vdash K = L : \underline{D}_1 \qquad \vdash \Gamma_1 = \Gamma_2 \quad \Gamma_1 \vdash \underline{C}_1 = \underline{C}_2 \quad \Gamma_1 \vdash \underline{D}_1 = \underline{D}_2}{\Gamma_2 \,|\, z \colon \underline{C}_2 \vdash K = L : \underline{D}_2}$$

C DIAGRAMMATIC AND MORE DETAILED DEFINITION OF $[\![-]\!]$

In this appendix we give diagrammatic and more detailed definition of $[\![-]\!]$ for algebraic operations, the user-defined algebra type, and the two composition operations. We present the partial definition in a natural deduction style, where the premises of the rule are assumed to hold for the conclusion to be defined. Further, we write

$$[\![\Gamma; A]\!]_1 = [\![\Gamma]\!] \in \mathsf{Set} \qquad [\![\Gamma; A]\!]_2 : [\![\Gamma]\!] \longrightarrow \mathsf{Set}$$

to mean that: i) $[\![\Gamma; A]\!]$ is defined; ii) its first component is equal to the set $[\![\Gamma]\!]$; and iii) its second component is given by a functor $[\![\Gamma]\!] \longrightarrow \mathsf{Set}$. The notation we use for the interpretation of terms is analogous.

C.1 Algebraic operations:

Given $\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$, we give the definition of $[\![\Gamma; \mathsf{op}^{\underline{C}}_V(y.M)]\!]$ in Fig. 6, where the morphism

$$\mathsf{op}^{[\![\Gamma;\underline{C}]\!]_2(\gamma)}_{([\![\Gamma;V]\!]_2)_\gamma(\star)}$$

is defined using the countable-product preservation property of $[\![\Gamma; \underline{C}]\!]_2(\gamma)$, namely, as the following composite:

$$\begin{array}{c}
\prod_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)} U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma)) \\
\Big\downarrow {=} \\
\prod_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)} ([\![\Gamma; \underline{C}]\!]_2(\gamma))(1) \\
\Big\downarrow {\cong} \\
([\![\Gamma; \underline{C}]\!]_2(\gamma))(+_{1 \leq o \leq |[\![\Gamma; O[V/x]]\!]_2(\gamma)|}\, 1) \\
\Big\downarrow {=} \\
([\![\Gamma; \underline{C}]\!]_2(\gamma))(|[\![\Gamma; O[V/x]]\!]_2(\gamma)|) \\
\Big\downarrow {(*)} \\
([\![\Gamma; \underline{C}]\!]_2(\gamma))(1) \\
\Big\downarrow {=} \\
U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))
\end{array}$$

$$
\frac{
\begin{array}{c}
[\![\Gamma; V]\!]_1 = \mathsf{id}_{[\![\Gamma]\!]} : [\![\Gamma]\!] \longrightarrow [\![\Gamma]\!] \qquad ([\![\Gamma; V]\!]_2)_\gamma : 1 \longrightarrow [\![\diamond; I]\!]_2(\star) \\
[\![\Gamma; O[V/x]]\!]_1 = [\![\Gamma]\!] \in \mathsf{Set} \qquad [\![\Gamma; O[V/x]]\!]_2(\gamma) = [\![\Gamma, x\!:\!I; O]\!]_2(\langle \gamma, ([\![\Gamma; V]\!]_2)_\gamma(\star)\rangle) \in \mathsf{Set} \\
[\![\Gamma, y\!:\!O[V/x]; M]\!]_1 = \mathsf{id}_{\coprod_{\gamma \in [\![\Gamma]\!]} [\![\Gamma; O[V/x]]\!]_2(\gamma)} \\
([\![\Gamma, y\!:\!O[V/x]; M]\!]_2)_{\langle \gamma, o\rangle} : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))
\end{array}
}{
\begin{tikzcd}
{[\![\Gamma; \mathsf{op}^{\underline{C}}_V(y.M)]\!]_1} \arrow[d, phantom, "\overset{\mathrm{def}}{=}"] & {([\![\Gamma; \mathsf{op}^{\underline{C}}_V(y.M)]\!]_2)_\gamma} \arrow[d, phantom, "\overset{\mathrm{def}}{=}"] \\
{[\![\Gamma]\!]} \arrow[ddd, "\mathsf{id}_{[\![\Gamma]\!]}"'] & 1 \arrow[d, "\langle \mathsf{id}_1\rangle_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)}"'] \\
& \prod_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)} 1 \arrow[d, "\prod_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)}(([\![\Gamma, y:O[V/x]; M]\!]_2)_{\langle \gamma, o\rangle})"'] \\
& \prod_{o \in [\![\Gamma; O[V/x]]\!]_2(\gamma)} U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma)) \arrow[d, "\mathsf{op}^{[\![\Gamma; \underline{C}]\!]_2(\gamma)}_{([\![\Gamma; V]\!]_2)_\gamma(\star)}"'] \\
{[\![\Gamma]\!]} & U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))
\end{tikzcd}
}
$$

Fig. 6. Appendix: Interpretation of algebraic operations.

where $(*)$ denotes the following function:

$$([\![\Gamma; \underline{C}]\!]_2(\gamma))(\overrightarrow{x_o} \vdash \mathsf{op}_{([\![\Gamma; V]\!]_2)_\gamma(\star)}(x_o)_{1 \le o \le |[\![\Gamma; O[V/x]]\!]_2(\gamma)|})$$

C.2 User-defined algebra type:

We give the definition of $[\![\Gamma'; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathrm{eff}}}\rangle]\!]$ in Fig. 7. The models $\mathcal{M}^{\gamma'}$ of $\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}$ are defined as in §8.4.

C.3 Composition operations:

We define $[\![-]\!]$ for the two composition operations in Fig. 8 and Fig. 9, respectively.

For better readability, we present the assumptions about the functions f^γ diagrammatically in Fig. 8 and Fig. 9.

$$\dfrac{\begin{array}{c}\llbracket \Gamma'; A \rrbracket_1 = \llbracket \Gamma' \rrbracket \in \mathsf{Set} \qquad \llbracket \Gamma'; A \rrbracket_2 : \llbracket \Gamma' \rrbracket \longrightarrow \mathsf{Set} \\ \llbracket \Gamma'; V_{\mathrm{op}} \rrbracket_1 = \mathsf{id}_{\llbracket \Gamma' \rrbracket} : \llbracket \Gamma' \rrbracket \longrightarrow \llbracket \Gamma' \rrbracket \\ (\llbracket \Gamma'; V_{\mathrm{op}} \rrbracket_2)_{\gamma'} : 1 \longrightarrow \prod_{\langle i, f \rangle \in \coprod_{i \in \llbracket \diamond; I \rrbracket_2(\star)} \prod_{o \in \llbracket x:I; O \rrbracket_2 (\langle \star, i \rangle)} \llbracket \Gamma'; A \rrbracket_2(\gamma')} \llbracket \Gamma'; A \rrbracket_2(\gamma') \\ \mathcal{M}^{\gamma'}(\Delta^{\gamma} \vdash T_1^{\gamma}) = \mathcal{M}^{\gamma'}(\Delta^{\gamma} \vdash T_2^{\gamma}) \\ (\text{for all } \Gamma \mid \Delta \vdash T_1 = T_2 \text{ in } \mathcal{E}_{\mathrm{eff}} \text{ and } \gamma \text{ in } \llbracket \Gamma \rrbracket)\end{array}}{\begin{array}{rcl}\llbracket \Gamma'; \langle A, \{V_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle \rrbracket_1 & \overset{\mathrm{def}}{=} & \llbracket \Gamma' \rrbracket \\ \llbracket \Gamma'; \langle A, \{V_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \rangle \rrbracket_2(\gamma') & \overset{\mathrm{def}}{=} & \mathcal{M}^{\gamma'}\end{array}}$$

Fig. 7. Appendix: Interpretation of the user-defined algebra type.

For Fig. 9, we define the morphism $\mathsf{hom}(f^{\gamma})$ of models of $\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}$ from $\llbracket \Gamma; \underline{D}_1 \rrbracket_2(\gamma)$ to $\llbracket \Gamma; \underline{D}_2 \rrbracket_2(\gamma)$ (i.e., a natural transformation) using components $(\mathsf{hom}(f^{\gamma}))_n$ given by

$$\begin{tikzcd}
(\llbracket \Gamma; \underline{D}_1 \rrbracket_2(\gamma))(n) \arrow[d, "\cong"'] \\
\prod_{1 \leq j \leq n} (\llbracket \Gamma; \underline{D}_1 \rrbracket_2(\gamma))(1) \arrow[d, "\prod_j (f^{\gamma})"'] \\
\prod_{1 \leq j \leq n} (\llbracket \Gamma; \underline{D}_2 \rrbracket_2(\gamma))(1) \arrow[d, "\cong"'] \\
(\llbracket \Gamma; \underline{D}_2 \rrbracket_2(\gamma))(n)
\end{tikzcd}$$

$$[\![\Gamma; M]\!]_1 = \mathsf{id}_{[\![\Gamma]\!]} : [\![\Gamma]\!] \longrightarrow [\![\Gamma]\!] \qquad ([\![\Gamma; M]\!]_2)_\gamma : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))$$

$$[\![\Gamma, y\!:\!U\underline{C}; N]\!]_1 = \mathsf{id}_{\coprod_{\gamma \in [\![\Gamma]\!]} U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma;\underline{C}]\!]_2(\gamma))} : [\![\Gamma, y\!:\!U\underline{C}]\!] \longrightarrow [\![\Gamma, y\!:\!U\underline{C}]\!]$$

$$([\![\Gamma, y\!:\!U\underline{C}; N]\!]_2)_{\langle \gamma, c\rangle} : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{D}]\!]_2(\gamma))$$

$$\begin{tikzcd}
\prod_o \mathcal{M}_1^\gamma(1) \arrow[rr, "\prod_{o \in [\![x:I;O]\!]_2(\langle\star, i\rangle)}(f^\gamma)"] \arrow[d, "\cong"'] & & \prod_o \mathcal{M}_2^\gamma(1) \arrow[d, "\cong"] \\
\mathcal{M}_1^\gamma(|[\![x\!:\!I;O]\!]_2(\langle\star, i\rangle)|) \arrow[d, "\mathcal{M}_1^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"'] & & \mathcal{M}_2^\gamma(|[\![x\!:\!I;O]\!]_2(\langle\star, i\rangle)|) \arrow[d, "\mathcal{M}_2^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"] \\
\mathcal{M}_1^\gamma(1) \arrow[rr, "f^\gamma"'] & & \mathcal{M}_2^\gamma(1)
\end{tikzcd}$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$, $\gamma \in [\![\Gamma]\!]$ and $i \in [\![\diamond; I]\!]_2(\star)$, and where

$$\mathcal{M}_1^\gamma \stackrel{\mathrm{def}}{=} [\![\Gamma; \underline{C}]\!]_2(\gamma) \qquad \mathcal{M}_2^\gamma \stackrel{\mathrm{def}}{=} [\![\Gamma; \underline{D}]\!]_2(\gamma) \qquad f^\gamma \stackrel{\mathrm{def}}{=} c \mapsto ([\![\Gamma, y\!:\!U\underline{C}; N]\!]_2)_{\langle\gamma, c\rangle}(\star)$$

$$\begin{tikzcd}
{[\![\Gamma; M \;\mathsf{as}\; y\!:\!U\underline{C} \;\mathtt{in}_{\underline{D}}\; N]\!]_1 \stackrel{\mathrm{def}}{=} [\![\Gamma]\!]} \arrow[dd, "\mathsf{id}_{[\![\Gamma]\!]}"'] & {([\![\Gamma; M \;\mathsf{as}\; y\!:\!U\underline{C} \;\mathtt{in}_{\underline{D}}\; N]\!]_2)_\gamma \stackrel{\mathrm{def}}{=} 1} \arrow[d, "([\![\Gamma;M]\!]_2)_\gamma"] \\
 & U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma)) \arrow[d, "f^\gamma"] \\
{[\![\Gamma]\!]} & U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{D}]\!]_2(\gamma))
\end{tikzcd}$$

Fig. 8. Appendix: Interpretation of the composition operation for computation terms.

$$[\![\Gamma; z\!:\!\underline{C}; K]\!]_1 = \mathsf{id}_{[\![\Gamma]\!]} : [\![\Gamma]\!] \longrightarrow [\![\Gamma]\!] \qquad ([\![\Gamma; z\!:\!\underline{C}; K]\!]_2)_\gamma : [\![\Gamma; \underline{C}]\!]_2(\gamma) \longrightarrow [\![\Gamma; \underline{D}_1]\!]_2(\gamma)$$

$$[\![\Gamma, y\!:\!U\underline{D}_1; M]\!]_1 = \mathsf{id}_{\coprod_{\gamma \in [\![\Gamma]\!]} U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{D}_1]\!]_2(\gamma))} : [\![\Gamma, y\!:\!U\underline{D}_1]\!] \longrightarrow [\![\Gamma, y\!:\!U\underline{D}_1]\!]$$

$$([\![\Gamma, y\!:\!U\underline{D}_1; M]\!]_2)_{\langle \gamma, d \rangle} : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}}([\![\Gamma; \underline{D}_2]\!]_2(\gamma))$$

$$\begin{tikzcd}
\prod_o \mathcal{M}_1^\gamma(1) \arrow[rr, "\prod_{o \in [\![x:I;O]\!]_2\,(\langle \star, i \rangle)}(f^\gamma)"] \arrow[d, "\cong"'] & & \prod_o \mathcal{M}_2^\gamma(1) \arrow[d, "\cong"] \\
\mathcal{M}_1^\gamma(|[\![x\!:\!I; O]\!]_2\,(\langle \star, i \rangle)|) \arrow[d, "\mathcal{M}_1^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"'] & & \mathcal{M}_2^\gamma(|[\![x\!:\!I; O]\!]_2\,(\langle \star, i \rangle)|) \arrow[d, "\mathcal{M}_2^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)"] \\
\mathcal{M}_1^\gamma(1) \arrow[rr, "f^\gamma"'] & & \mathcal{M}_2^\gamma(1)
\end{tikzcd}$$

for all $\mathsf{op} : (x\!:\!I) \longrightarrow O \in \mathcal{S}_{\mathrm{eff}}$, $\gamma \in [\![\Gamma]\!]$ and $i \in [\![\diamond; I]\!]_2(\star)$, and where

$$\mathcal{M}_1^\gamma \overset{\mathrm{def}}{=} [\![\Gamma; \underline{D}_1]\!]_2(\gamma) \qquad \mathcal{M}_2^\gamma \overset{\mathrm{def}}{=} [\![\Gamma; \underline{D}_2]\!]_2(\gamma) \qquad f^\gamma \overset{\mathrm{def}}{=} d \mapsto ([\![\Gamma, y\!:\!U\underline{D}_1; M]\!]_2)_{\langle \gamma, d \rangle}(\star)$$

$$\begin{tikzcd}
{[\![\Gamma; z\!:\!\underline{C}; K \ \mathtt{as}\ y\!:\!U\underline{D}_1 \ \mathtt{in}_{\underline{D}_2}\ M]\!]_1} \arrow[d, phantom, "\overset{\mathrm{def}}{=}"] & ([\![\Gamma; z\!:\!\underline{C}; K \ \mathtt{as}\ y\!:\!U\underline{D}_1 \ \mathtt{in}_{\underline{D}_2}\ M]\!]_2)_\gamma \arrow[d, phantom, "\overset{\mathrm{def}}{=}"] \\
{[\![\Gamma]\!]} \arrow[dd, "\mathsf{id}_{[\![\Gamma]\!]}"'] & {[\![\Gamma; \underline{C}]\!]_2(\gamma)} \arrow[d, "([\![\Gamma; z:\underline{C}; K]\!]_2)_\gamma"] \\
 & {[\![\Gamma; \underline{D}_1]\!]_2(\gamma)} \arrow[d, "\mathsf{hom}(f^\gamma)"] \\
{[\![\Gamma]\!]} & {[\![\Gamma; \underline{D}_2]\!]_2(\gamma)}
\end{tikzcd}$$

Fig. 9. Appendix: Interpretation of the composition operation for homomorphism terms.