Handling fibred algebraic effects

Danel Ahman

***Abstract*—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 more 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), based on Plotkin and Pretnar's insight that handlers denote algebras for a given algebraic theory. The conventional presentation of handlers can then be routinely derived from our type-based treatment. We also demonstrate that our treatment of handlers provides a useful mechanism for reasoning about effectful computations.**

I. Introduction

An important feature of many widely-used programming languages is their support for computational effects, e.g., raising exceptions, accessing memory, performing input-output, 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, e.g., [3, 4, 8, 9, 14, 26, 29, 31], there is a striking 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 giving a comprehensive account of 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. Their study originated with the pioneering work of Plotkin and Power [32, 33]; they have since been successfully applied to, e.g., modularly combining effects [18] and effect-dependent program optimisations [21]. Examples of algebraic effects include exceptions, state, input-output, nondeterminism, probability, etc. A key insight of Plotkin and Power was that most of Moggi's monads [28, 27] are determined by algebraic effects, with the notable exception of continuations.

A significant role in the recent rise in popularity of algebraic effects can be contributed to their handlers, invented by Plotkin and Pretnar [34]. These are a generalisation of exception handlers, based on the idea that handlers denote 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 languages perspective, a handler $\{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}$ redefines the algebraic operations and the handling construct then recursively traverses a given program, replacing each operation with the corresponding user-defined term, as exemplified by the β-equation:

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

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

Plotkin and Pretnar also showed that handlers can be used to neatly implement timeouts, rollbacks, stream redirection, etc., see [34, §3]. More recently, handlers have 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 state operations (`get` and `put`) using the natural representation of stateful programs as functions $\mathsf{St} \to A \times \mathsf{St}$. To support this style of programming, existing languages have been extended with algebraic effects and their handlers [15, 20, 22], and new languages have been built around them [7, 24]. Algebraic effects and their handlers are also central to the extension of OCaml with shared memory multicore parallelism [1].

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 (§IV-A). Our *other contributions* include: i) a dependently typed generalisation of Plotkin and Pretnar's effect theories (§III-A); ii) demonstrating that these theories can be used to capture more precise notions of computation (§III-B); iii) introducing a new computation type, the user-defined algebra type, to give a type-based treatment of handlers and solve the problem with unsound program equivalences (§IV-B); iv) showing how to derive the conventional term-level definition of handlers from our type-based treatment (§IV-C); v) demonstrating that such handlers provide a useful mechanism for reasoning about effectful computations (§VII); and vi) equipping our language with a sound fibrational denotational semantics (§VIII).

II. EMLTT: AN EFFECTFUL DEPENDENT TYPE THEORY

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. This language is a minor extension of the effectful dependently typed language we developed with Ghani and Plotkin in [4]. It is a natural extension of Martin-Löf's intensional type theory (MLTT) with computational effects, making a clear distinction between values and computations, both at the level of types and terms, analogously to Levy's Call-By-Push-Value (CBPV) [23] and Egger et al.'s Enriched Effect Calculus (EEC) [11]. We refer to it here as EMLTT.

As usual for dependently typed languages, the sets of types and terms of EMLTT are defined mutually inductively.

First, one assumes countable sets of *value variables* $x, y, \ldots$ and *computation variables* $z, \ldots$. Then, the grammar of *value types* $A, B, \ldots$ and *computation types* $\underline{C}, \underline{D}, \ldots$ is given by

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

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. As in [4], we omit general inductive types and use the type of natural numbers as a representative example. However, compared to op. cit., we include the empty type 0, the sum type $A + B$ and the homomorphic function type $\underline{C} \multimap \underline{D}$, which we only sketched in op. cit. We include the first two as to specify signatures of algebraic effects (see §III-B); and the latter as it is generally useful for writing libraries of effectful code without excessive thunking and forcing, compared to using functions $U\underline{C} \to U\underline{D}$. FA is the type of computations that return values of type A.

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

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

In addition to the introduction and elimination forms for the types from MLTT, EMLTT value terms also include thunks of effectful computations and homomorphic lambda abstractions.

As in [4], the terms of EMLTT are decorated with a large number of type annotations. We use them to define the denotational semantics of EMLTT on raw expressions, so as to avoid coherence problems; this is a standard technique in the literature [36, 16]. We omit these annotations in our examples.

Regarding effectful programs, EMLTT makes a further distinction between *computation terms* $M, N, \ldots$ and *homomorphism terms* $K, L, \ldots$. The latter are necessary to define the elimination form for the computational Σ-type. They were also essential to proving a completeness result for EMLTT with respect to a natural class of categorical models, see [4].

The grammar of these two kinds of terms is given by

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

Computation terms include standard combinators for effectful programming, such as the sequential composition of M and N, given by $M\ \mathtt{to}\ x{:}A\ \mathtt{in}_{\underline{C}}\ N$. They also include introduction and elimination forms for computational Σ- and Π-types, forcing of thunked computations and homomorphic function applications. Homomorphism terms are similar, but also include computation variables z, which have to be used linearly and in a way that ensures that the computation bound to z "happens first" in the term containing it. This guarantees that every K denotes an algebra homomorphism in the categorical models we study in §VIII.

As one can use thunking and forcing (resp. the homomorphic function type) to derive elimination forms for value types into computation (resp. homomorphism) terms, these elimination forms are not included primitively in EMLTT.

The *well-formed syntax* 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 $\Gamma \mid z{:}\underline{C} \vdash K : \underline{D}$, where the 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. 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`.

The well-formed syntax is defined mutually with an *equational theory*, consisting of *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 $\Gamma \mid z{:}\underline{C} \vdash K = L : \underline{D}$. We give a selection of these equations in Fig. 2. The definitional equations interact with the well-formed syntax via context and type conversion rules, such as

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

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

Value types:

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

Computation types:

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

Value terms:

$$\frac{\vdash \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \mathtt{zero} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash V : \mathsf{Nat}}{\Gamma \vdash \mathtt{succ}\ V : \mathsf{Nat}}$$

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

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

Computation terms:

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

Homomorphism terms:

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

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

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

Value terms:

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

Computation terms:

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

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

Homomorphism terms:

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

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

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

we 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 $\Sigma\, x{:}A.\underline{C}$. It provides a uniform treatment of type-dependency in sequential composition, by allowing us 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. 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 [4]. Thus, we say that the computational effects in EMLTT are *fibred*.

III. Fibred algebraic effects

In this section we develop a means for formally specifying algebraic computational effects in EMLTT in terms of operations and equations, using a natural dependently typed generalisation of Plotkin and Pretnar's effect theories [34]. In [4], this extension of EMLTT was only sketched informally.

A. Fibred effect theories

We begin by identifying the fragment of EMLTT which we use to define the types of our operation symbols.

A value type is said to be *pure* if it is built up from only Nat, 1, $\Sigma\, x{:}A.B$, $\Pi\, x{:}A.B$, 0, $A + B$ and $V =_A W$, with A pure in propositional equality. This notion of pureness extends straightforwardly to contexts and terms. A value context Γ is *pure* if A_i is pure for every $x_i{:}A_i \in \Gamma$. A value term is

pure if it does not contain thunks and homomorphic lambda abstractions, and all its type annotations are pure. In other words, the pure fragment of EMLTT is precisely MLTT.

Assuming a countable set of *effect variables* $w, \ldots$, we now define fibred effect theories. We start with signatures of operation symbols and then add equations, so as to specify both the computational effects at hand and their behaviour.

Definition III.1. 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, respectively.

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

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

with the value types and value terms all required to be pure.

We follow a convention of omitting V in $\mathsf{op}_V(y.T)$ when the input type of op is 1, and y when the output type is 1.

An *effect context* Δ is a list of distinct effect variables annotated with pure value types. Δ is *well-formed* in a pure 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 $\Gamma \mid \Delta \vdash T$ are given by:

$$\frac{\Gamma \vdash \Delta_1, w : A, \Delta_2 \quad \Gamma \vdash V : A}{\Gamma \mid \Delta_1, w : A, \Delta_2 \vdash w\,(V)}$$

$$\frac{\Gamma \vdash V : I \quad \Gamma \vdash \Delta \quad \Gamma, y : O[V/x] \mid \Delta \vdash T}{\Gamma \mid \Delta \vdash \mathsf{op}_V(y.T)}$$

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

$$\frac{\begin{array}{c}\Gamma \vdash V : A_1 + A_2 \quad \Gamma \vdash \Delta \\ \Gamma, x_1 : A_1 \mid \Delta \vdash T_1 \quad \Gamma, x_2 : A_2 \mid \Delta \vdash T_2\end{array}}{\Gamma \mid \Delta \vdash \mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(x_1 : A_1) \mapsto T_1, \mathsf{inr}(x_2 : A_2) \mapsto T_2)}$$

Definition III.2. A *fibred effect theory* $\mathcal{T}_{\text{eff}}$ is given by a fibred effect signature $\mathcal{S}_{\text{eff}}$ and a finite set $\mathcal{E}_{\text{eff}}$ of equations $\Gamma \mid \Delta \vdash T_1 = T_2$, where $\Gamma \mid \Delta \vdash T_1$ and $\Gamma \mid \Delta \vdash T_2$.

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

B. Examples of fibred effect theories

As our fibred effect theories are a natural dependently typed generalisation of Plotkin and Pretnar's effect theories [34], we can capture all the effects they can. For example, 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 (binary) nondeterminism*, which is given by one operation symbol $\mathsf{or} : 1 \longrightarrow 1 + 1$ and three equations that make or into a semi-lattice operation. See op. cit. for more examples.

Compared to Plotkin and Pretnar's work, our dependently typed operations further allow us to capture more precise notions of computation. We discuss two such examples: 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 [6].

Global state. Assuming given well-formed pure value types

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

of memory locations and values stored at them, 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 is that get denotes an effectful command that returns the current value of the store at the given location; and put sets the store at the given location to the given value. Importantly, the types of values stored can vary depending on locations.

The corresponding fibred effect theory $\mathcal{T}_{\text{GS}}$ is then given by

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

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

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

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

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

where the last two equations both include a side-condition requiring the locations denoted by x_1 and x_2 to be different.

Similarly to [34], this notation for side-conditions is an informal shorthand. Formally, we assume a decidable equality on locations (for simplicity, boolean-valued), given by $\diamond \vdash \mathsf{eq} : \mathsf{Loc} \to \mathsf{Loc} \to 1 + 1$, and then write the right-hand sides of these two equations using case analysis, e.g., the right-hand side of the last equation would be formally written as

$$\begin{array}{l} \mathsf{case}\ (\mathsf{eq}\, x_1\, x_2)\ \mathsf{of}\ (\mathsf{inl}(x'_1 : 1) \mapsto \mathsf{put}_{\langle x_1,y_1\rangle}(\mathsf{put}_{\langle x_2,y_2\rangle}(w\,(\star))), \\ \qquad\qquad\qquad\qquad \mathsf{inr}(x'_2 : 1) \mapsto \mathsf{put}_{\langle x_2,y_2\rangle}(\mathsf{put}_{\langle x_1,y_1\rangle}(w\,(\star)))) \end{array}$$

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 (4th and 5th equation).

Dependently typed update monads. We assume given two well-formed pure value types

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

of store values and store updates, together with well-typed closed pure value terms (omitting the empty value contexts)

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

satisfying the following five closed equations (in the pure fragment of the equational theory of EMLTT; we omit contexts and types, and write the first argument to $\oplus$ as a subscript):

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

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

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

The 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 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. The 5-tuple $(\mathsf{St}, \mathsf{Upd}, \downarrow, \mathsf{o}, \oplus)$ is known in the literature under the name of *directed containers* [2].

The corresponding fibred effect theory $\mathcal{T}_{\text{UPD}}$ is then given by

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

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

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

These equations are similar to the first three equations of the global state theory $\mathcal{T}_{\text{GS}}$, but instead of an overwriting behaviour, they describe how updates are applied to the store. Observe how subsequent updates are combined using $\oplus$.

C. Extending EMLTT with fibred algebraic effects

We now show how to extend EMLTT with fibred algebraic effects given by a fibred effect theory $\mathcal{T}_{\text{eff}} = (\mathcal{S}_{\text{eff}}, \mathcal{E}_{\text{eff}})$. First, we extend the computation terms with *algebraic operations*:

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

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

To extend the well-formed syntax with corresponding typing rule and definitional equations, we define a translation of effect terms into value terms. While it might be more intuitive to translate effect terms into computation terms, giving the translation into value terms allows us to reuse it in §IV-B for extending EMLTT with handlers of fibred algebraic effects.

Using this translation, we define the typing rule and definitional equations for algebraic operations in Fig. 3.

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. In particular, various subsequent results refer to value terms substituted for all the effect variables in Δ, not just for the free effect variables appearing in T. As a convention, we use $\overrightarrow{V_i}$ for a set of value terms $\{V_1, \ldots, V_n\}$.

Definition III.3. Given $\Gamma \mid \Delta \vdash T$, 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 $\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$), the *translation* of the effect term T to a value term $(\!|\, T \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{W_{\mathsf{op}}}}$ is defined by recursion on the structure of T as follows:

$$(\!|\, w_j\,(V) \,|\!) \quad \stackrel{\text{def}}{=} V_j'\ (V[\overrightarrow{V_i}/\overrightarrow{x_i}])$$

$$(\!|\, \mathsf{op}_V(y.T) \,|\!) \stackrel{\text{def}}{=} W_{\mathsf{op}}\ \langle V[\overrightarrow{V_i}/\overrightarrow{x_i}], \lambda\, y{:}O[V[\overrightarrow{V_i}/\overrightarrow{x_i}]/x].(\!|\, T \,|\!)\rangle$$

$$(\!|\, \mathtt{pm}\ V\ \mathtt{as}\ (y_1{:}B_1, y_2{:}B_2)\ \mathtt{in}\ T \,|\!) \stackrel{\text{def}}{=} \mathtt{pm}\ V[\overrightarrow{V_i}/\overrightarrow{x_i}]\ \mathtt{as}\ (y_1{:}B_1[\overrightarrow{V_i}/\overrightarrow{x_i}], y_2{:}B_2[\overrightarrow{V_i}/\overrightarrow{x_i}])\ \mathtt{in}\ (\!|\, T \,|\!)$$

$$(\!|\, \mathsf{case}\ V\ \mathsf{of}\ (\mathsf{inl}(y_1{:}B_1) \mapsto T_1, \mathsf{inr}(y_2{:}B_2) \mapsto T_2) \,|\!) \stackrel{\text{def}}{=} \mathtt{case}\ V[\overrightarrow{V_i}/\overrightarrow{x_i}]\ \mathtt{of}\ (\mathtt{inl}(y_1{:}B_1[\overrightarrow{V_i}/\overrightarrow{x_i}]) \mapsto (\!|\, T_1 \,|\!),\ \mathtt{inr}(y_2{:}B_2[\overrightarrow{V_i}/\overrightarrow{x_i}]) \mapsto (\!|\, T_2 \,|\!))$$

where we omit the subscripts on the translation to improve readability. However, it is still worthwhile to note that for the cases where the given effect term involves variable bindings, the set of value terms $\overrightarrow{V_i}$ is extended with the bound value variables, e.g., the second case is formally defined as

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

For convenience, we include the equations given in $\mathcal{E}_{\text{eff}}$ as definitional equations between value terms. The corresponding equations between computation terms are 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 easily derived from the translation of the equation

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

given in the global state theory $\mathcal{T}_{\text{GS}}$ discussed earlier.

Further, it is also worthwhile to note that the definitional equations corresponding to the equations given in $\mathcal{E}_{\text{eff}}$ can be instantiated with any suitable value context Γ'. We do so to ensure that the weakening and substitution theorems remain derivable for this extension of EMLTT. We also note that the disjointness requirement on value contexts imposed in these equations is important for the substitution theorem (Thm. V.3) to go through. While at first sight this requirement might seem limiting on the number of definitional equations we can derive, it turns out that the corresponding definitional equation without the disjointness requirement is derivable, as shown in Prop. VI.1, which we postpone to §VI because its proof crucially relies on the meta-theory we establish in §V.

IV. Handlers via the user-defined algebra type

A. Problem with adding conventional handlers to EMLTT

Before we show how to extend EMLTT with handlers of fibred algebraic effects using a novel computation type, called

Typing rule for algebraic operations:

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

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

Congruence equations:

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

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

General algebraicity equations:

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

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

Equations of the given fibred effect theory:

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

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

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

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

Fig. 3. Rules for extending EMLTT with fibred algebraic effects.

the user-defined algebra type, we first explain how taking the conventional term-level definition of handlers as primitive leads to unsound program equivalences becoming derivable.

Recall from the Introduction that Plotkin and Pretnar (and others since) include handlers in effectful languages by extending the syntax of computation terms with a handling construct

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

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

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

to EMLTT, the combination of the general algebraicity equation and the definitional equations associated with the handling construct gives rise to a non-convergent critical pair.

To explain this problem in more detail, let us first assume the *theory $\mathcal{T}_{\text{I/O}}$ of interactive input-output of bits*, given by two operation symbols $\mathtt{read} : 1 \longrightarrow 1+1$ and $\mathtt{write} : 1+1 \longrightarrow 1$, and no equations. Next, recall from the Introduction that a handler redefines the operations of a given effect theory, e.g., by flipping the bits written to the output, as given by

$$\begin{array}{l}\{\ \mathtt{read}(x') \quad \mapsto \mathtt{read}^{F1}(y.\mathtt{force}\,(x'\,y)) \\ \quad\ \mathtt{write}_x(x') \mapsto \mathtt{write}^{F1}_{\neg x}(\mathtt{force}\,(x'\,\star))\ \}\end{array}$$

where $\neg : 1+1 \to 1+1$ swaps the left and right injections.

Now, let us consider handling a simple program, given by $\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\ \star)$, using the handler we defined above. Using the β-equations for handling (see §IV-C), we can derive

$$\begin{array}{l}\Gamma \vdash (\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\ \star))\ \mathtt{handled\ with} \\ \qquad\qquad \{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op}\in\mathcal{S}_{\text{I/O}}}\ \mathtt{to}\ y{:}1\ \mathtt{in}\ \mathtt{return}\ \star \\ = \mathtt{write}^{F1}_{\mathtt{inr}\,\star}(\mathtt{return}\ \star) : F1\end{array}$$

On the other hand, using the general algebraicity equation from Fig. 3, which ensures that homomorphism terms indeed behave as if they were homomorphisms, we can derive

$$\begin{array}{l}\Gamma \vdash (\mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\ \star))\ \mathtt{handled\ with} \\ \qquad\qquad \{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op}\in\mathcal{S}_{\text{I/O}}}\ \mathtt{to}\ y{:}1\ \mathtt{in}\ \mathtt{return}\ \star \\ = \mathtt{write}^{F1}_{\mathtt{inl}\,\star}(\mathtt{return}\ \star) : F1\end{array}$$

Clearly, if we want to equip EMLTT with a sound denotational semantics based on the models of $\mathcal{T}_{\text{I/O}}$, these two equations had better not be derivable at the same time.

The reason for this discrepancy lies in the term-level definition of this handler. In particular, while the homomorphic behaviour of homomorphism terms is determined purely 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 $\{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op}\in\mathcal{S}_{\text{I/O}}}$.

It is worthwhile to note that this problem is not inherent to EMLTT, but would also arise in the simply typed setting, when combining handlers with CBPV and its stack terms, and with EEC and its linear terms. The reason why Plotkin and Pretnar could give a sound denotational semantics to their language was exactly due to their choice of using CBPV *without* stack terms, i.e., with only value and computation terms.

B. Extending EMLTT *with the user-defined algebra type*

In this section we solve the problem discussed above by giving handlers a novel type-based treatment, internalising the idea that they denote algebras for the given algebraic theory.

Given a fibred effect theory $\mathcal{T}_{\text{eff}} = (\mathcal{S}_{\text{eff}}, \mathcal{E}_{\text{eff}})$, we extend EMLTT computation types with the *user-defined algebra type*:

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

which pairs a value type (the carrier) with a family of value terms (the operations). In addition, we extend the computation and homomorphism terms with *composition operations*:

$$\begin{array}{rcl} M & ::= & \ldots \ \mid\ M\ \mathtt{as}\ x{:}U\underline{C}\ \mathtt{in}_{\underline{D}}\ N \\ K & ::= & \ldots \ \mid\ K\ \mathtt{as}\ x{:}U\underline{C}\ \mathtt{in}_{\underline{D}}\ N \end{array}$$

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

In principle, we could have restricted these composition operations to only the user-defined algebra type, but then we

would not be able to derive a useful type isomorphism to coerce computations between a general $\underline{C}$ and its canonical representation as a user-defined algebra type (see Prop. IV.1).

Conceptually, these composition operations are a form of explicit substitution of thunked computations for value variables, e.g., M as $x{:}U\underline{C}$ in$_{\underline{D}}$ N is definitionally equal to $N[\texttt{thunk}\ M/x]$. As such, the value variable x refers to the whole of (the thunk of) M, compared to sequential composition M to $x{:}A$ in N, where x refers to the value computed by M. Thus, we use as (running M *as* if it was x), compared to to (running M *to* produce a value for x).

Importantly, the typing rules for M as $x{:}U\underline{C}$ in$_{\underline{D}}$ N and K as $x{:}U\underline{C}$ 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 to ensure that N behaves as if it was a homomorphism term, meaning that the effects in M are guaranteed to "happen before" those in N. However, rather than extending EMLTT further with a 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. 4, where we give the rules for extending the well-formed syntax and equational theory of EMLTT with the user-defined algebra type and the composition operations.

In the rules concerning the user-defined algebra type, we use the judgment $\Gamma' \vdash \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}$ on A, which holds iff the value terms V_{op} form an algebra *on* the value type A, i.e., iff $\Gamma' \vdash A$, $\Gamma' \vdash V_{\mathsf{op}} : (\Sigma\, x{:}I.O \to A) \to A$ (for all $\mathsf{op} : (x{:}I) \longrightarrow O$ in $\mathcal{S}_{\text{eff}}$) and for all $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$, we have

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

where we write $\overrightarrow{\lambda\, x_{w_j} : A'_j \to A.}$ for the sequence of 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.

In the rules concerning the composition operations, we use the 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 have

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

for all $\mathsf{op} : (x{:}I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$, where the omitted type annotations on x and x' are given by I and $O \to U\underline{C}$, respectively.

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 the user-defined algebra type $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle$. Analogously, the η-equation for algebraic operations captures the intuition that the value terms V_{op} are precisely (the thunked versions of) the algebraic operations at the user-defined algebra type $\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle$.

It is also worthwhile to note that we have not included an η-equation for the user defined algebra type because it does not hold in the natural denotational semantics we develop for this extension of EMLTT in §VIII. Instead, as promised earlier, we can construct a corresponding type isomorphism.

Formation rule for the user-defined algebra type:

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

Typing rules for the composition operations:

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \vdash \underline{D} \quad \Gamma, x{:}U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}}{\Gamma \vdash M\ \texttt{as}\ x{:}U\underline{C}\ \texttt{in}_{\underline{D}}\ N : \underline{D}}$$

$$\frac{\Gamma \mid z{:}\underline{C} \vdash K : \underline{D}_1 \quad \Gamma \vdash \underline{D}_2 \quad \Gamma, x{:}U\underline{D}_1 \vdash_{\mathsf{hom}} M : \underline{D}_2}{\Gamma \mid z{:}\underline{C} \vdash K\ \texttt{as}\ x : U\underline{D}_1\ \texttt{in}_{\underline{D}_2}\ M : \underline{D}_2}$$

Congruence equations:

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

plus similar two equations for composition operations.

β-equation for the user-defined algebra type:

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

β-equation for the composition operations:

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

η-equations for the composition operations:

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \mid z{:}\underline{C} \vdash K : \underline{D}}{\Gamma \vdash M\ \texttt{as}\ x{:}U\underline{C}\ \texttt{in}_{\underline{D}}\ K[\texttt{force}_{\underline{C}}\ x/z] = K[M/z] : \underline{D}}$$

$$\frac{\Gamma \mid z_1{:}\underline{C} \vdash K : \underline{D}_1 \quad \Gamma \mid z_2{:}\underline{D}_1 \vdash L : \underline{D}_2}{\begin{array}{l}\Gamma \mid z_1{:}\underline{C} \vdash K\ \texttt{as}\ x{:}U\underline{D}_1\ \texttt{in}_{\underline{D}_2}\ L[\texttt{force}_{\underline{D}_1}\ x/z_2] \\ \qquad = L[K/z_2] : \underline{D}_2\end{array}}$$

η-equation for algebraic operations:

$$\frac{\begin{array}{c}\Gamma \vdash V : I \quad \Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle \\ \Gamma, y{:}O[V/x] \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle\end{array}}{\begin{array}{l}\Gamma \vdash \mathsf{op}_V^{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle}(y.M) \\ \quad = \texttt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle}\,(V_{\mathsf{op}}\,\langle V, \lambda\, y{:}O[V/x].\texttt{thunk}\ M\rangle) \\ \qquad\qquad : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle\end{array}}$$

Fig. 4. Rules for extending EMLTT with the user-defined algebra type.

Proposition IV.1. *Given $\Gamma \vdash \underline{C}$, we have a type isomorphism*

$$\Gamma \vdash \underline{C} \cong \langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathit{eff}}}\rangle$$

where each $\Gamma \vdash V_{\mathsf{op}} : (\Sigma\, x{:}I.O \to U\underline{C}) \to U\underline{C}$ is defined as

$$\begin{array}{r}\lambda\, y{:}(\Sigma\, x{:}I.O \to U\underline{C}).\texttt{pm}\ y\ \texttt{as}\ (x{:}I, x'{:}O \to U\underline{C})\ \texttt{in} \\ \texttt{thunk}\,(\mathsf{op}^{\underline{C}}_x(y'.\texttt{force}_{\underline{C}}\,(x'\ y')))\end{array}$$

Proof sketch. The witness for the left-to-right direction is given by the function $\lambda\, z{:}\underline{C}.z$ as $x{:}U\underline{C}$ in $\texttt{force}_{\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\text{eff}}}\rangle}\ x$. The witness for the other direction is defined analogously. □

C. Deriving the term-level definition of handlers

We now show how to derive the conventional term-level definition of handlers from our type-based treatment, by defining the handling construct using sequential composition:

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

where each $\Gamma \vdash V_{\mathsf{op}} : (\Sigma\, x{:}I.O \to U\underline{C}) \to U\underline{C}$ is defined as

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

The expected typing rule and β-equations are then derivable.

Proposition IV.2. *The following typing rule is derivable:*

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

where each V_{op} is derived from N_{op}, as defined above.

Proposition IV.3. *The following β-equations are derivable:*

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

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

It is worthwhile to recall that Plotkin and Pretnar do not enforce the correctness of their handlers during typechecking as it is in general undecidable [34, §6] (i.e., they do not require the 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, we could develop a normaliser that is optimised for important fibred effect theories (e.g., for state, as in [5, §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.

V. Basic meta-theory of EMLTT

We now discuss some basic meta-theoretic properties of the extension of EMLTT with fibred algebraic effects and their handlers, given by $\mathcal{T}_{\mathsf{eff}} = (\mathcal{S}_{\mathsf{eff}}, \mathcal{E}_{\mathsf{eff}})$. As various typing rules and definitional equations include (translations of) effect terms, we also prove corresponding meta-theoretic properties for effect terms, which the results for EMLTT then use.

First, we show that weakening of value variables is admissible. Due to the disjointness requirement on value contexts in the equations given in Fig. 3, we also require the given value variable to be fresh with respect to the equations given in $\mathcal{E}_{\mathsf{eff}}$.

Theorem V.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 equations $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\mathit{eff}}$), then $\Gamma_1, x{:}A, \Gamma_2 \vdash B$, and similarly for other types, terms and definitional equations.*

Proof sketch. By induction on the given derivations. □

Next, we show that substitution is admissible.

Proposition V.2. *Given $\Gamma \mid \Delta \vdash T$, y such that $y \notin Vars(\Gamma)$, and W such that $FVV(W) \cap Vars(\Gamma) = \emptyset$, then*

$$(\!|\, T \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{W_{\mathsf{op}}}}[W/y] = (\!|\, T \,|\!)_{A[W/y];\overrightarrow{V_i[W/y]};\overrightarrow{V_j'[W/y]};\overrightarrow{W_{\mathsf{op}}[W/y]}}$$

Theorem V.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.*

Proof sketch. We first prove Prop. V.2 and then Thm. V.3, both by induction on the derivations of the given judgments. □

Theorem V.4. *Given $\Gamma \mid z{:}\underline{C} \vdash K : \underline{D}$ and $\Gamma \vdash M : \underline{C}$, then $\Gamma \vdash K[M/z] : \underline{D}$, and similarly for substituting an L for z.*

Proof sketch. By induction on the given derivations. □

Finally, we show that judgments of well-formed types, etc. only refer to well-formed contexts, etc.

Proposition V.5. *Given $\Gamma \mid \Delta \vdash T$ and Γ' such that Γ' and Γ are disjoint, $\Gamma' \vdash A$, and the value terms in the subscripts are well-typed in Γ' (as in Fig. 3), then $\Gamma' \vdash (\!|\, T \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{W_{\mathsf{op}}}} : A$.*

Proposition V.6. *Given $\Gamma \mid \Delta \vdash T$ and Γ' such that Γ' and Γ are disjoint, $\Gamma' \vdash A = B$, and the value terms are definitionally equal, then $\Gamma' \vdash (\!|\, T \,|\!)_{A;\overrightarrow{V_i};\overrightarrow{V_j'};\overrightarrow{V_{\mathsf{op}}}} = (\!|\, T \,|\!)_{B;\overrightarrow{W_i};\overrightarrow{W_j'};\overrightarrow{W_{\mathsf{op}}}} : A$.*

Theorem V.7. *Given $\Gamma \vdash V : A$, then $\vdash \Gamma$ and $\Gamma \vdash A$, and similarly for other judgments of types, terms and equations.*

Proof sketch. We first prove Prop. V.5, and then Prop. V.6 and Thm. V.7, all by induction on the given derivations. □

VI. Derivable equations

We begin by showing that the disjointness requirement on Γ' and Γ can be omitted in the equations given in Fig. 3.

Proposition VI.1. *Given $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\mathit{eff}}$, then*

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

Proof sketch. If Γ' and Γ are disjoint, we use the corresponding restricted rule from Fig. 3. When Γ' and Γ are not disjoint, we

first systematically replace the overlapping variables with fresh ones using the weakening and substitution theorems from §V, and then use the corresponding restricted rule from Fig. 3. □

Next, we can derive more specialised algebraicity equations.

Proposition VI.2. *We can derive the algebraicity equation*

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

and similarly for other computation term formers.

We also have useful equations for composition operations.

Proposition VI.3. *The following unit and associativity equations are derivable for the composition operations:*

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

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

and similarly for the corresponding homomorphism terms.

Composition operations commute with other term formers.

Proposition VI.4. *The following equations are derivable:*

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

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

and similarly for computational pattern-matching and the corresponding homomorphism terms.

Proposition VI.5. *The composition operations commute with all other computation and homomorphism term formers from the left, e.g., for computational pairing we can derive*

$$\frac{\Gamma \vdash M : \underline{C} \quad \Gamma \vdash V : A \quad \Gamma, y{:}A \vdash \underline{D} \quad \Gamma, x{:}U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}[V/y]}{\begin{array}{l}\Gamma \vdash M\ \mathtt{as}\ x{:}U\underline{C}\ \mathtt{in}\ \langle V, N\rangle \\ \quad = \langle V, M\ \mathtt{as}\ x{:}U\underline{C}\ \mathtt{in}\ N\rangle : \Sigma\, y{:}A.\underline{D}\end{array}}$$

VII. Using handlers to reason about effects

We now show that our type-based treatment of handlers provides a useful and convenient mechanism for reasoning about effectful computations, giving us an alternative to defining predicates on computations using propositional equality.

To facilitate such reasoning, we introduce *universes* à la Tarski [25], by extending value and computation types with

$$A ::= \ldots \mid \mathsf{VU} \mid \mathsf{CU} \mid \mathsf{El}(V) \qquad \underline{C} ::= \ldots \mid \mathsf{El}(V)$$

and value terms with *codes of types*:

$$V ::= \ldots \mid \mathtt{empty\text{-}c} \mid \mathtt{v\text{-}pi\text{-}c}(V, x.W) \mid \ldots$$
$$\ldots \mid \mathtt{f\text{-}c}(V) \mid \mathtt{c\text{-}pi\text{-}c}(V, x.W) \mid \ldots$$

The corresponding typing rules and equations are of the form:

$$\frac{\Gamma \vdash V : \mathsf{VU} \quad \Gamma, x{:}\mathsf{El}(V) \vdash W : \mathsf{VU}}{\Gamma \vdash \mathtt{v\text{-}pi\text{-}c}(V, x.W) : \mathsf{VU}}$$

$$\Gamma \vdash \mathsf{El}(\mathtt{v\text{-}pi\text{-}c}(V, x.W)) = \Pi\, x{:}\mathsf{El}(V).\mathsf{El}(W)$$

The predicates we consider are defined as value terms of the form $\Gamma \vdash V : UFA \to \mathsf{VU}$, with the aim of using them to refine computations using Σ-types, i.e., as $\Sigma\, x{:}UFA.\mathsf{El}(x)$.

A. Lifting predicates from return values to computations

Lifting predicates given on return values to computations is easiest when the effect theory does not contain equations. Thus, let us consider the theory $\mathcal{T}_{\mathsf{I/O}}$ of *input-output* from §IV-A; other equation-free effects can be reasoned about similarly, e.g., exceptions. Assuming a predicate $\Gamma \vdash V_P : A \to \mathsf{VU}$ on return values, we lift V_P to a predicate $V_{\widehat{P}}$ on UFA by

$$V_{\widehat{P}} \stackrel{\text{def}}{=} \lambda y{:}UFA.\mathtt{thunk}\,\big((\mathtt{force}_{FA}\, y)\ \mathtt{to}\ y'{:}A\ \mathtt{in}\ \mathtt{force}_{\langle \mathsf{VU}, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{I/O}}}\rangle}\,(V_P\, y')\big)$$

where we let $\mathtt{bit\text{-}c} \stackrel{\text{def}}{=} \mathtt{sum\text{-}c}(\mathtt{unit\text{-}c}, \mathtt{unit\text{-}c})$ and define

$$V_{\mathsf{read}} \stackrel{\text{def}}{=} \lambda\, y{:}(\Sigma\, x{:}1.1{+}1 \to \mathsf{VU}).\mathtt{v\text{-}sig\text{-}c}(\mathtt{bit\text{-}c}, y'.(\mathtt{snd}\, y)\, y')$$
$$V_{\mathsf{write}} \stackrel{\text{def}}{=} \lambda\, y{:}(\Sigma\, x{:}1{+}1.\, 1 \to \mathsf{VU}).(\mathtt{snd}\, y)\, \star$$

On closer inspection, we see that $V_{\widehat{P}}$ agrees with the possibility modality from Evaluation Logic [30], in that a computation satisfies $V_{\widehat{P}}$ if there exists a return value that satisfies V_P. If we replace `v-sig-c` (code for value Σ-type) with `v-pi-c`, we get a predicate that holds if all the return values satisfy V_P.

As a second example, we consider a fibred effect theory with equations, namely, the theory $\mathcal{T}_{\mathsf{GS}}$ of *global state*. Assuming a predicate $\Gamma \vdash V_Q : A \to \mathsf{S} \to \mathsf{VU}$ on return values and *final states*, where $\mathsf{S} \stackrel{\text{def}}{=} \Pi\, x{:}\mathsf{Loc}.\mathsf{Val}$, we define a predicate

$$V_{\widehat{Q}} \stackrel{\text{def}}{=} \lambda y{:}UFA.\lambda x_S{:}\mathsf{S}.\mathtt{fst}\big(\big(\mathtt{thunk}\,\big((\mathtt{force}_{FA}\, y)\ \mathtt{to}\ y'{:}A\ \mathtt{in}\ \mathtt{force}_{\langle \mathsf{S} \to \mathsf{VU} \times \mathsf{S}, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{GS}}}\rangle}\,(\lambda x'_S{:}\mathsf{S}.\langle V_Q\, y'\, x'_S, x'_S\rangle)\big)\big)\, x_S\big)$$

on (thunks of) computations and *initial states*, where V_{get} and V_{put} are defined using the natural representation of stateful programs as functions $\mathsf{S} \to \mathsf{VU} \times \mathsf{S}$. 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 state corresponding to $\mathcal{T}_{\mathsf{GS}}$.

On closer inspection, $V_{\widehat{Q}}$ corresponds to Dijkstra's weakest precondition semantics of stateful programs [10], e.g., taking $\mathsf{Loc} \stackrel{\text{def}}{=} 1$ and omitting location arguments, we have

$$\Gamma \vdash V_{\widehat{Q}}\ (\mathtt{thunk}\,(\mathtt{return}\ V))\ V_S = V_Q\, V\, V_S : \mathsf{VU}$$
$$\Gamma \vdash V_{\widehat{Q}}\ (\mathtt{thunk}\,(\mathtt{get}^{FA}(y.M)))\ V_S = V_{\widehat{Q}}\ (\mathtt{thunk}\, M[V_S/y])\ V_S : \mathsf{VU}$$
$$\Gamma \vdash V_{\widehat{Q}}\ (\mathtt{thunk}\,(\mathtt{put}_{V'_S}^{FA}(M)))\ V_S = V_{\widehat{Q}}\ (\mathtt{thunk}\, M)\ V'_S : \mathsf{VU}$$

B. Specifying patterns of allowed effects in computations

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

As a first example, we consider a coarse grained specification on I/O computations, namely, disallowing all writes:

$$V_{\mathsf{no\text{-}w}} \stackrel{\text{def}}{=} \lambda y{:}UFA.\mathtt{thunk}\,\big((\mathtt{force}_{FA}\, y)\ \mathtt{to}\ y'{:}A\ \mathtt{in}\ \mathtt{force}_{\langle \mathsf{VU}, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{I/O}}}\rangle}\ \mathtt{unit\text{-}c}\big)$$

where

$$V_{\mathsf{read}} \stackrel{\text{def}}{=} \lambda\, y{:}(\Sigma\, x{:}1.1{+}1 \to \mathsf{VU}).\texttt{v-pi-c}(\texttt{bit-c}, y'.(\texttt{snd}\, y)\, y')$$
$$V_{\mathsf{write}} \stackrel{\text{def}}{=} \lambda\, y{:}(\Sigma\, x{:}1 + 1.\, 1 \to \mathsf{VU}).\texttt{empty-c}$$

E.g., $\texttt{read}^{FA}(x.\texttt{write}^{FA}_{V}(M))$ does not satisfy $V_{\mathsf{no\text{-}w}}$ because

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

As a more involved example, we consider specifications on I/O computations in the style of session types [17]. First, we assume an inductive type $\diamond \vdash \mathsf{Protocol}$ with three constructors:

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

describing patterns of allowed I/O effects. Then, given some particular protocol $\Gamma \vdash V_{\mathsf{pr}} : \mathsf{Protocol}$, we define a predicate

$$V_{\widehat{\mathsf{pr}}} \stackrel{\text{def}}{=} \lambda y{:}UFA.\big(\texttt{thunk}\,\big((\texttt{force}_{FA}\, y)\ \texttt{to}\ y'{:}A\ \texttt{in}\ \texttt{force}_{\langle \mathsf{Protocol}\to\mathsf{VU}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathrm{I/O}}}\rangle}\, V_{\mathsf{ret}}\big)\big)\, V_{\mathsf{pr}}$$

where the value terms are defined as follows (for simplicity, we give their definitions by pattern-matching on their arguments):

$$V_{\mathsf{ret}} \quad \mathtt{e} \quad \stackrel{\text{def}}{=} \quad \texttt{unit-c}$$
$$V_{\mathsf{read}} \quad \langle V, V_{\mathsf{rk}}\rangle \quad (\mathtt{r}\ V'_{\mathsf{pr}}) \quad \stackrel{\text{def}}{=} \quad \texttt{v-pi-c}(\texttt{bit-c}, y.(V_{\mathsf{rk}}\, y)\, (V'_{\mathsf{pr}}\, y))$$
$$V_{\mathsf{write}} \quad \langle V, V_{\mathsf{wk}}\rangle \quad (\mathtt{w}\ \langle V_P, V'_{\mathsf{pr}}\rangle) \quad \stackrel{\text{def}}{=} \quad \texttt{v-sig-c}(V_P\, V, y.V_{\mathsf{wk}}\ \star\ V'_{\mathsf{pr}})$$

with all other cases defined to be equal to `empty-c`, and where

$$\Gamma \vdash V_{\mathsf{rk}} : 1{+}1 \to \mathsf{Protocol} \to \mathsf{VU} \quad \Gamma \vdash V_{\mathsf{wk}} : 1 \to \mathsf{Protocol} \to \mathsf{VU}$$

We can also easily combine these predicates with those from §VII-A by replacing `unit-c` with a predicate on return values.

VIII. Fibrational semantics

We conclude this paper by giving a sound denotational semantics to the extension of EMLTT with fibred algebraic effects and their handlers, given by $\mathcal{T}_{\text{eff}} = (\mathcal{S}_{\text{eff}}, \mathcal{E}_{\text{eff}})$. The semantics we develop is an instance of a more general class of models of EMLTT, based on *fibrations* (functors with extra structure) and *adjunctions* between them, see [4] for details.

We proceed in three steps. First, we define the interpretation of the pure fragment of EMLTT. Next, we derive a countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ from $\mathcal{T}_{\text{eff}}$. Finally, we define the interpretation of the rest of EMLTT using the models of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$.

We leave the semantics of the extension with universes (see §VII) for future work, expecting it to closely follow the standard fibrational treatment of induction-recursion [12].

A. Families fibrations

We begin with an overview of the fibrations we use in our denotational semantics. For a general treatment of fibrations and their use in modelling dependent types, we suggest [19].

Given a category $\mathcal{C}$, we can define a new category $\mathsf{Fam}(\mathcal{C})$, whose objects are pairs (X, A) of a set X and a functor $A : X \longrightarrow \mathcal{C}$ (treating X as a discrete category); 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 $\mathcal{C}$-valued *families fibration* $\mathsf{fam}_{\mathcal{C}} : \mathsf{Fam}(\mathcal{C}) \longrightarrow \mathsf{Set}$ is defined as $\mathsf{fam}_{\mathcal{C}}(X, A) \stackrel{\text{def}}{=} X$ and $\mathsf{fam}_{\mathcal{C}}(f, g) \stackrel{\text{def}}{=} f$.

For any set X, $\mathsf{Fam}_X(\mathcal{C})$ is called the *fibre* over X, i.e., the subcategory of $\mathsf{Fam}(\mathcal{C})$ whose objects and morphisms are of the form (X, A) and (id_X, g). Given a function $f : X \longrightarrow Y$, the *reindexing functor* $f^* : \mathsf{Fam}_Y(\mathcal{C}) \longrightarrow \mathsf{Fam}_X(\mathcal{C})$ is given by $f^*(Y, A) \stackrel{\text{def}}{=} (X, A \circ f)$ and analogously on morphisms.

We get a prototypical model of dependent types when we take $\mathcal{C} \stackrel{\text{def}}{=} \mathsf{Set}$. 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 $\{(X, A)\} \stackrel{\text{def}}{=} \coprod_{x\in X} A(x)$. The latter provides semantics to context extensions $\Gamma, x{:}A$. There also exist canonical *projection maps* $\pi_{(X,A)} : \{(X, A)\} \longrightarrow X$, given by $\langle x, a\rangle \mapsto x$.

B. Interpretation of the pure fragment of EMLTT

Next, we recall from [4] how the pure fragment of EMLTT is interpreted in the fibration $\mathsf{fam}_{\mathsf{Set}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}$.

More specifically, we define 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 functor $A : [\![\Gamma]\!] \longrightarrow \mathsf{Set}$.

For better readability, we denote the first and second components of $[\![\Gamma; A]\!]$ and $[\![\Gamma; V]\!]$ using subscripts $1, 2$.

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

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

$\mathsf{Nat}, 1, 0$ and $A + B$ are interpreted using the corresponding categorical structure in the fibres of $\mathsf{Fam}(\mathsf{Set})$, given by extending the corresponding sets pointwise to families of sets, e.g., $(X, A) + (X, B) \stackrel{\text{def}}{=} (X, x \mapsto A(x) + B(x))$. Finally,

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

For terms, one then has, e.g., that $[\![\Gamma; \mathtt{inl}_{A+B}\ V]\!]_1 \stackrel{\text{def}}{=} \mathsf{id}_{[\![\Gamma]\!]}$ and $([\![\Gamma; \mathtt{inl}_{A+B}\ V]\!]_2)_\gamma \stackrel{\text{def}}{=} \star \mapsto \mathsf{inl}\,(([\![\Gamma; V]\!]_2)_\gamma(\star))$, assuming that $[\![\Gamma; V]\!] : 1([\![\Gamma]\!]) \longrightarrow [\![\Gamma; A]\!]$ and $[\![\Gamma; B]\!]$ are defined.

From the soundness theorem for EMLTT [4, Thm. 1], we get that $[\![-]\!]$ is in fact defined on all well-formed types and well-typed terms, and it validates pure definitional equations.

C. Countable Lawvere theories from fibred effect theories

We now derive a countable Lawvere theory from $\mathcal{T}_{\text{eff}}$. We begin by recalling some basic definitions and results from [35].

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 a skeleton of the category of countable sets.

A *model* of a countable Lawvere theory $\mathcal{L}$ in a category $\mathcal{C}$ with countable products is given by a countable-product preserving functor $\mathcal{M} : \mathcal{L} \longrightarrow \mathcal{C}$. A *morphism* of models from $\mathcal{M}_1 : \mathcal{L} \longrightarrow \mathcal{C}$ to $\mathcal{M}_2 : \mathcal{L} \longrightarrow \mathcal{C}$ is given by a natural transformation $\mathcal{M}_1 \longrightarrow \mathcal{M}_2$. This gives us a category $\mathsf{Mod}(\mathcal{L}, \mathcal{C})$.

There is a forgetful functor $U_{\mathcal{L}} : \mathsf{Mod}(\mathcal{L}, \mathcal{C}) \longrightarrow \mathcal{C}$, given by $U_{\mathcal{L}}(\mathcal{M}) \stackrel{\text{def}}{=} \mathcal{M}(1)$. If $\mathcal{C}$ is locally countably presentable, $U_{\mathcal{L}}$

has a left adjoint $F_{\mathcal{L}} : \mathcal{C} \longrightarrow \mathsf{Mod}(\mathcal{L}, \mathcal{C})$. Importantly for the purposes of this paper, Set is locally countably presentable. Furthermore, $\mathsf{Mod}(\mathcal{L}, \mathsf{Set})$ is both complete and cocomplete.

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

We construct $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ by first expanding $\mathcal{T}_{\text{eff}}$ into a countable equational theory [13], analogously to how Plotkin and Pretnar expanded their effect theories [34]. 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 $\Gamma \mid \Delta \vdash T$ 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 $a \in [\![\Gamma; A'_j]\!]_2(\gamma)$. For example, we have

$$(w_j(V))^\gamma \quad \stackrel{\text{def}}{=} x^{([\![\Gamma; V]\!]_2)_\gamma(\star)}_{w_j}$$
$$(\mathsf{op}_V(y.\,T))^\gamma \stackrel{\text{def}}{=} \mathsf{op}_i(T^{\langle \gamma, o \rangle})_{1 \le o \le |[\![x:I;O]\!]_2\,(\langle \star, i \rangle)|}$$

where $i \stackrel{\text{def}}{=} ([\![\Gamma; V]\!]_2)_\gamma(\star)$. We omit the cases for pattern-matching and case analysis. We then get a countable equational theory by taking equations $\Delta^\gamma \vdash T_1^\gamma = T_2^\gamma$, for all $\gamma \in [\![\Gamma]\!]$ and $\Gamma \mid \Delta \vdash T_1 = T_2 \in \mathcal{E}_{\text{eff}}$, and closing them under rules of reflexivity, symmetry, transitivity, replacement and substitution.

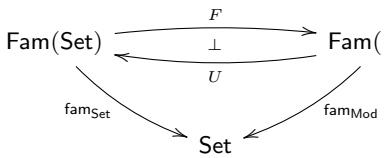
The countable Lawvere theory $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ is then given by taking the morphisms $n \longrightarrow m$ in $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ to be m-tuples $(\overrightarrow{x_i} \vdash t_j)_{1 \le j \le m}$ of equivalence classes of terms in n variables (in the countable 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 $I_{\mathcal{T}_{\text{eff}}}(f) \stackrel{\text{def}}{=} (\overrightarrow{x_i} \vdash x_{f(j)})_{1 \le j \le m} : n \to m$. It is easy to verify that $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ has countable products (given using the cardinal sums in $\aleph_1$) and $I_{\mathcal{T}_{\text{eff}}}$ strictly preserves them.

Proposition VIII.1. *$\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ is a countable Lawvere theory.*

For better readability, we write Mod for $\mathsf{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \mathsf{Set})$.

D. Interpretation of the non-pure fragment of EMLTT

We now extend the interpretation of the pure fragment of EMLTT to the rest of EMLTT, based on the fibred adjunction



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

This is an instance of a general result about defining models of EMLTT by lifting adjunctions to families, see [4, Thm. 3].

We extend the definition of $[\![-]\!]$ so that, if defined, it maps a context Γ and computation type $\underline{C}$ to an object $[\![\Gamma; \underline{C}]\!]$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Mod})$; a context Γ and computation term M to $[\![\Gamma; M]\!] : 1([\![\Gamma]\!]) \longrightarrow U([\![\Gamma]\!], \underline{C})$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Set})$, for some functor $\underline{C}$; and a context Γ, variable z, computation type $\underline{C}$ and homomorphism term K to $[\![\Gamma; z : \underline{C}; K]\!] : [\![\Gamma; \underline{C}]\!] \longrightarrow ([\![\Gamma]\!], \underline{D})$ in $\mathsf{Fam}_{[\![\Gamma]\!]}(\mathsf{Mod})$, for some functor $\underline{D}$. In particular, we define

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

assuming that $[\![\Gamma; A]\!]$, $[\![\Gamma; \underline{C}]\!]$ and $[\![\Gamma; \underline{D}]\!]$ are defined. Below, we omit such routine assumptions. 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 most computation and homomorphism terms (see [4]). The cases for algebraic operations, the user-defined algebra type and the composition operations are defined as (see Appendix for more details)

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

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

$$([\![\Gamma; M \ \mathtt{as}\ x : U\underline{C}\ \mathtt{in}_{\underline{D}}\ N]\!]_2)_\gamma \stackrel{\text{def}}{=} f^\gamma \circ ([\![\Gamma; M]\!]_2)_\gamma$$

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

where, for terms, the first components are given by $\mathsf{id}_{[\![\Gamma]\!]}$.

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

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

and ι is the countable-product preservation isomorphism

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

Further, the functor $\mathcal{M}^\gamma : \mathcal{L}_{\mathcal{T}_{\text{eff}}} \longrightarrow \mathsf{Set}$ is defined as

$$\mathcal{M}^\gamma(n) \stackrel{\text{def}}{=} \textstyle\prod_{1 \le j \le n} [\![\Gamma; A]\!]_2(\gamma) \qquad \mathcal{M}^\gamma(\overrightarrow{x_j} \vdash x_j) \stackrel{\text{def}}{=} \mathsf{proj}_j$$
$$\mathcal{M}^\gamma(\Delta \vdash \mathsf{op}_i(t_o)_{1 \le o \le |[\![x:I;O]\!]_2\,(\langle \star, i \rangle)|}) \stackrel{\text{def}}{=}$$
$$f^\gamma_{\mathsf{op}_i} \circ \langle \mathcal{M}^\gamma(\Delta \vdash t_o) \rangle_{1 \le o \le |[\![x:I;O]\!]_2\,(\langle \star, i \rangle)|}$$

where $f^\gamma_{\mathsf{op}_i} : \prod_{o \in [\![x:I;O]\!]_2\,(\langle \star, i \rangle)} [\![\Gamma; A]\!]_2(\gamma) \longrightarrow [\![\Gamma; A]\!]_2(\gamma)$ is given by $f \mapsto \mathsf{proj}_{\langle i, f \rangle}(([\![\Gamma; V_{\mathsf{op}}]\!]_2)_\gamma(\star))$. Moreover, $\mathcal{M}^\gamma$ extends straightforwardly to tuples of terms, so as to account for all morphisms in $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$, i.e., those of the form $n \longrightarrow m$.

In the interpretation of the two composition operations, the function $f^\gamma : U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma)) \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{D}]\!]_2(\gamma))$ is given by $c \mapsto ([\![\Gamma, x : U\underline{C}; N]\!]_2)_{\langle \gamma, c \rangle}(\star)$, and $\mathsf{hom}(f^\gamma)$ is the corresponding morphism of models of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$, given by

$$(\mathsf{hom}(f^\gamma))_n \stackrel{\text{def}}{=} \iota_{[\![\Gamma; \underline{D}]\!]} \circ \textstyle\prod_{1 \le j \le n}(f^\gamma) \circ \iota^{-1}_{[\![\Gamma; \underline{C}]\!]}$$

where $\iota_{[\![\Gamma; \underline{C}]\!]} : \prod_{1 \le j \le n}([\![\Gamma; \underline{C}]\!]_2(\gamma))(1) \cong ([\![\Gamma; \underline{C}]\!]_2(\gamma))(n)$.

For these cases of $[\![-]\!]$ to be defined, we additionally assume that $\mathcal{M}^\gamma$ validates the equations given in $\mathcal{E}_{\text{eff}}$, and f^γ commutes with the operations of $[\![\Gamma; \underline{C}]\!]_2(\gamma)$ and $[\![\Gamma; \underline{D}]\!]_2(\gamma)$.

We then prove standard semantic weakening and substitution lemmas, showing that weakening and substitution for value variables correspond to reindexing, e.g., the former along projection maps $\pi_{(X,A)} : \{(X, A)\} \longrightarrow X$. We also prove that substitution for computation variables corresponds to composition, e.g., $[\![\Gamma; K[M/z]]\!] = U([\![\Gamma; z : \underline{C}; K]\!]) \circ [\![\Gamma; M]\!]$.

Theorem VIII.2 (Soundness)**.** $[\![-]\!]$ *is defined on all well-formed contexts, well-formed types and well-typed terms, and it identifies all definitionally equal contexts, types and terms.*

Proof sketch. We prove this theorem by induction on the given derivations. In particular, for the definitional equations that correspond to the equations given in $\mathcal{E}_{\text{eff}}$ (see Fig. 3), we recall that these equations hold in $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ by construction, thus all models of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ validate them, including those modelling our computation types. For computation types, we prove that $[\![\Gamma; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}} \rangle]\!]$ is defined by observing that the equational proof obligations included in $\Gamma \vdash \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}$ on A ensure that each $\mathcal{M}^\gamma$ validates the equations given in $\mathcal{E}_{\text{eff}}$. For computation terms, we prove that $[\![\Gamma; M \texttt{ as } x : U\underline{C} \texttt{ in}_{\underline{D}} N]\!]$ is defined by observing that the assumption $\Gamma, x : U\underline{C} \vdash_{\mathsf{hom}} N : \underline{D}$ ensures 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 is analogous. □

IX. Conclusion and future work

In this paper we have given a comprehensive account of algebraic effects and their handlers in the dependently typed setting. We gave handlers a novel type-based treatment and demonstrated that they provide a useful mechanism for reasoning about effectful computations. We also equip the resulting language with a sound fibrational semantics.

In future, we plan to combine our treatment of handlers with effect-typing [20] and multi-handlers [24]. We also plan to compare our handler-based definition of predicate transformers with their CPS-translation based definition in F* [3]. Further, we plan to extend computation terms with recursion following the analyses in [34, 4], by generalising from equations to inequations, and by using fibrations of continuous families of ω-cpos and models of countable discrete CPO-enriched Lawvere theories. More generally, we plan to extend this work from families (of sets) fibrations to more general fibrational models of dependent types, where definitional and propositional proof obligations (see discussion in §IV-C) might not coincide.

Acknowledgments

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

References

[1] The multicore OCaml project. Available: https://github.com/ocamllabs/ocaml-multicore/.
[2] D. Ahman, J. Chapman, and T. Uustalu. When is a container a comonad? *LMCS*, 10(3), 2014.
[3] D. Ahman et al. Dijkstra monads for free. In *POPL'17*.
[4] D. Ahman, N. Ghani, and G. D. Plotkin. Dependent types and fibred computational effects. In *FoSSaCS 2016*.
[5] D. Ahman and S. Staton. Normalization by evaluation and algebraic effects. In *MFPS XXIX*.
[6] D. Ahman and T. Uustalu. Update monads: Cointerpreting directed containers. In *TYPES 2013*.
[7] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. *JLAMP*, 84(1):108–123, 2015.
[8] E. Brady. Programming and reasoning with algebraic effects and dependent types. In *ICFP 2013*.
[9] C. Casinghino. *Combining Proofs and Programs*. PhD thesis, University of Pennsylvania, 2014.
[10] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8), 1975.
[11] J. Egger, R. E. Møgelberg, and A. Simpson. The enriched effect calculus: syntax and semantics. *J. Log. Comput.*, 24(3):615–654, 2014.
[12] N. Ghani, L. Malatesta, F. N. Forsberg, and A. Setzer. Fibred data types. In *LICS 2013*.
[13] G. A. Grätzer. *Universal Algebra*. Springer, 1979.
[14] P. Hancock and A. Setzer. Interactive programs in dependent type theory. In *CSL 2000*.
[15] D. Hillerström and S. Lindley. Liberating effects with rows and handlers. In *TyDe 2016*.
[16] M. Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*. CUP, 1997.
[17] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP 1998*.
[18] M. Hyland, G. Plotkin, and J. Power. Combining effects: Sum and tensor. *TCS*, 357(1–3):70–99, 2006.
[19] B. Jacobs. *Categorical Logic and Type Theory*. 1999.
[20] O. Kammar, S. Lindley, and N. Oury. Handlers in action. In *ICFP 2013*.
[21] O. Kammar and G. D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *POPL 2012*.
[22] D. Leijen. Type directed compilation of row-typed algebraic effects. In *POPL 2017*.
[23] P. B. Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*. 2004.
[24] S. Lindley, C. McBride, and C. McLaughlin. Do be do be do. In *POPL 2017*.
[25] P. Martin-Löf. *Intuitionistic Type Theory*. 1984.
[26] C. McBride. Functional pearl: Kleisli arrows of outrageous fortune. *J. Funct. Program.* (To appear).
[27] E. Moggi. Computational lambda-calculus and monads. In *LICS 1989*.
[28] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
[29] A. Nanevski, G. Morrisett, and L. Birkedal. Hoare type theory, polymorphism and separation. *J. Funct. Program.*, 18(5-6):865–911, 2008.
[30] A. M. Pitts. Evaluation logic. In *4th HOW*, 1991.
[31] A. M. Pitts, J. Matthiesen, and J. Derikx. A dependent type theory with abstractable names. In *LSFA 2014*.
[32] G. Plotkin and J. Power. Semantics for algebraic operations. In *MFPS XVII*.
[33] G. D. Plotkin and J. Power. Notions of computation determine monads. In *FOSSACS 2002*.
[34] G. D. Plotkin and M. Pretnar. Handling algebraic effects. *LMCS*, 9(4:23), 2013.
[35] J. Power. Countable Lawvere theories and computational effects. In *MFCSIT 2004*.
[36] T. Streicher. *Semantics of Type Theory*. 1991.

APPENDIX

In this appendix we give more details about the definition of $[\![-]\!]$ for algebraic operations, the user-defined algebra type and the 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. 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 for terms is analogous.

Algebraic operations:

Given $\mathsf{op} : (x : I) \longrightarrow O \in \mathcal{S}_{\text{eff}}$, we give the definition of $[\![\Gamma; \mathsf{op}^{\underline{C}}_{V}(y.M)]\!]$ in Fig. 5, 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:

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

where $(*)$ is

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

User-defined algebra type:

We give the definition of $[\![-]\!]$ for the user-defined algebra type in Fig. 6.

The models $\mathcal{M}^{\gamma'}$ of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ are defined as in §VIII-D.

Composition operations:

We give the definition of $[\![-]\!]$ for the composition operations in Fig. 7 and Fig. 8.

For better readability, we present the assumptions about the functions f^γ diagrammatically in Fig. 7 and Fig. 8.

For Fig. 8, we define the morphism $\mathsf{hom}(f^\gamma)$ of models of $\mathcal{L}_{\mathcal{T}_{\text{eff}}}$ from $[\![\Gamma; \underline{D}_1]\!]_2(\gamma)$ to $[\![\Gamma; \underline{D}_2]\!]_2(\gamma)$ (i.e., a natural transformation) using components $(\mathsf{hom}(f^\gamma))_n$ given by

$$\begin{array}{c}
([\![\Gamma; \underline{D}_1]\!]_2(\gamma))(n) \\
\downarrow{\cong} \\
\prod_{1 \leq j \leq n} ([\![\Gamma; \underline{D}_1]\!]_2(\gamma))(1) \\
\downarrow{\prod_j (f^\gamma)} \\
\prod_{1 \leq j \leq n} ([\![\Gamma; \underline{D}_2]\!]_2(\gamma))(1) \\
\downarrow{\cong} \\
([\![\Gamma; \underline{D}_2]\!]_2(\gamma))(n)
\end{array}$$

$$[\![\Gamma; V]\!]_1 = \mathsf{id}_{[\![\Gamma]\!]} : [\![\Gamma]\!] \longrightarrow [\![\Gamma]\!] \qquad ([\![\Gamma; V]\!]_2)_\gamma : 1 \longrightarrow [\![\diamond; I]\!]_2(\star)$$
$$[\![\Gamma; O[V/x]]\!]_1 = [\![\Gamma]\!] \in \mathsf{Set} \qquad [\![\Gamma; O[V/x]]\!]_2(\gamma) = [\![\Gamma, x:I; O]\!]_2(\langle \gamma, ([\![\Gamma; V]\!]_2)_\gamma(\star)\rangle) \in \mathsf{Set}$$
$$[\![\Gamma, y:O[V/x]; M]\!]_1 = \mathsf{id}_{\coprod_{\gamma \in [\![\Gamma]\!]} [\![\Gamma; O[V/x]]\!]_2(\gamma)}$$
$$([\![\Gamma, y:O[V/x]; M]\!]_2)_{\langle \gamma, o\rangle} : 1 \longrightarrow U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma))$$

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

$$([\![\Gamma; \mathsf{op}_V^C(y.M)]\!]_2)_\gamma \stackrel{\text{def}}{=} 1$$

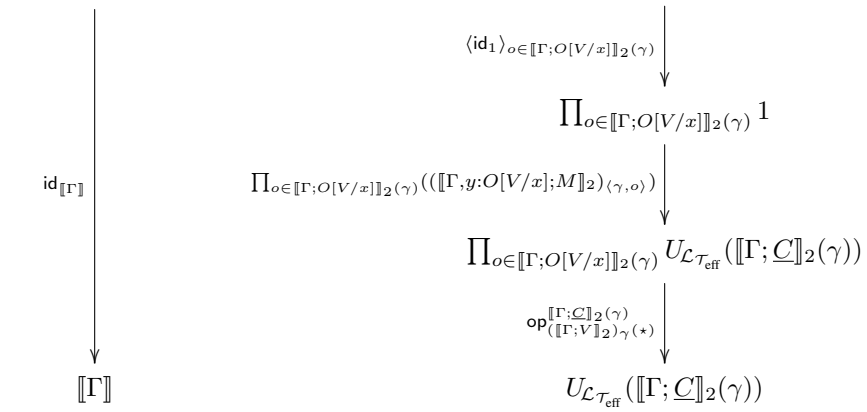


Fig. 5. Appendix: Interpretation of algebraic operations.

$$[\![\Gamma'; A]\!]_1 = [\![\Gamma']\!] \in \mathsf{Set} \qquad [\![\Gamma'; A]\!]_2 : [\![\Gamma']\!] \longrightarrow \mathsf{Set}$$
$$[\![\Gamma'; V_{\mathsf{op}}]\!]_1 = \mathsf{id}_{[\![\Gamma']\!]} : [\![\Gamma']\!] \longrightarrow [\![\Gamma']\!]$$
$$([\![\Gamma'; V_{\mathsf{op}}]\!]_2)_{\gamma'} : 1 \longrightarrow \prod_{\langle i, f\rangle \in \coprod_{i \in [\![\diamond; I]\!]_2(\star)} \prod_{o \in [\![x:I; O]\!]_2\,(\langle \star, i\rangle)} [\![\Gamma'; A]\!]_2(\gamma')} [\![\Gamma'; A]\!]_2(\gamma')$$
$$\mathcal{M}^{\gamma'}(\Delta^\gamma \vdash T_1^\gamma) = \mathcal{M}^{\gamma'}(\Delta^\gamma \vdash T_2^\gamma)$$
$$(\text{for all } \Gamma \mid \Delta \vdash T_1 = T_2 \text{ in } \mathcal{E}_{\text{eff}} \text{ and } \gamma \text{ in } [\![\Gamma]\!])$$

$$[\![\Gamma'; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}\rangle]\!]_1 \stackrel{\text{def}}{=} [\![\Gamma']\!]$$
$$[\![\Gamma'; \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\text{eff}}}\rangle]\!]_2(\gamma') \stackrel{\text{def}}{=} \mathcal{M}^{\gamma'}$$

Fig. 6. Appendix: Interpretation of the user-defined algebra type.

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

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

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

$$\begin{array}{ccc}
\prod_o \mathcal{M}_1^\gamma(1) & \xrightarrow{\prod_{o \in [\![x:I;O]\!]_2\,(\langle \star, i\rangle)}(f^\gamma)} & \prod_o \mathcal{M}_2^\gamma(1) \\
\cong \downarrow & & \downarrow \cong \\
\mathcal{M}_1^\gamma(|[\![x:I;O]\!]_2\,(\langle \star, i\rangle)|) & & \mathcal{M}_2^\gamma(|[\![x:I;O]\!]_2\,(\langle \star, i\rangle)|) \\
{\scriptstyle \mathcal{M}_1^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)}\downarrow & & \downarrow{\scriptstyle \mathcal{M}_2^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)} \\
\mathcal{M}_1^\gamma(1) & \xrightarrow[f^\gamma]{} & \mathcal{M}_2^\gamma(1)
\end{array}$$

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

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

$$\begin{array}{ccc}
[\![\Gamma; M \ \mathtt{as}\ y{:}U\underline{C}\ \mathtt{in}_{\underline{D}}\ N]\!]_1 & & ([\![\Gamma; M \ \mathtt{as}\ y{:}U\underline{C}\ \mathtt{in}_{\underline{D}}\ N]\!]_2)_\gamma \\
\stackrel{\text{def}}{=} & & \stackrel{\text{def}}{=} \\
[\![\Gamma]\!] & & 1 \\
\downarrow & & \downarrow{\scriptstyle ([\![\Gamma; M]\!]_2)_\gamma} \\
{\scriptstyle \mathsf{id}_{[\![\Gamma]\!]}}\downarrow & & U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{C}]\!]_2(\gamma)) \\
 & & \downarrow{\scriptstyle f^\gamma} \\
[\![\Gamma]\!] & & U_{\mathcal{L}_{\mathcal{T}_{\text{eff}}}}([\![\Gamma; \underline{D}]\!]_2(\gamma))
\end{array}$$

Fig. 7. Appendix: Interpretation of the composition operation for computation terms.

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

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

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

$$\begin{array}{ccc}
\prod_o \mathcal{M}_1^\gamma(1) & \xrightarrow{\prod_{o \in [\![x:I;O]\!]_2\,(\langle\star,i\rangle)}(f^\gamma)} & \prod_o \mathcal{M}_2^\gamma(1) \\
\cong \downarrow & & \downarrow \cong \\
\mathcal{M}_1^\gamma(|[\![x:I;O]\!]_2\,(\langle\star,i\rangle)|) & & \mathcal{M}_2^\gamma(|[\![x:I;O]\!]_2\,(\langle\star,i\rangle)|) \\
{\scriptstyle \mathcal{M}_1^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)}\downarrow & & \downarrow{\scriptstyle \mathcal{M}_2^\gamma(\overrightarrow{x_o} \vdash \mathsf{op}_i(x_o)_o)} \\
\mathcal{M}_1^\gamma(1) & \xrightarrow[f^\gamma]{} & \mathcal{M}_2^\gamma(1)
\end{array}$$

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

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

$$\begin{array}{ccc}
[\![\Gamma; z{:}\underline{C}; K \ \mathtt{as}\ y{:}U\underline{D}_1\ \mathtt{in}_{\underline{D}_2}\ M]\!]_1 & & ([\![\Gamma; z{:}\underline{C}; K \ \mathtt{as}\ y{:}U\underline{D}_1\ \mathtt{in}_{\underline{D}_2}\ M]\!]_2)_\gamma \\
\stackrel{\text{def}}{=} & & \stackrel{\text{def}}{=} \\
[\![\Gamma]\!] & & [\![\Gamma; \underline{C}]\!]_2(\gamma) \\
\downarrow & & \downarrow{\scriptstyle ([\![\Gamma; z:\underline{C}; K]\!]_2)_\gamma} \\
{\scriptstyle \mathsf{id}_{[\![\Gamma]\!]}}\downarrow & & [\![\Gamma; \underline{D}_1]\!]_2(\gamma) \\
 & & \downarrow{\scriptstyle \mathsf{hom}(f^\gamma)} \\
[\![\Gamma]\!] & & [\![\Gamma; \underline{D}_2]\!]_2(\gamma)
\end{array}$$

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