Algebraic Foundations for Effect-Dependent Optimisations

Ohad Kammar Gordon D. Plotkin

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh, Scotland
ohad.kammar@ed.ac.uk gdp@ed.ac.uk

Abstract

We present a general theory of Gifford-style type and effect annotations, where effect annotations are sets of effects. Generality is achieved by recourse to the theory of algebraic effects, a development of Moggi's monadic theory of computational effects that emphasises the operations causing the effects at hand and their equational theory. The key observation is that annotation effects can be identified with operation symbols.

We develop an annotated version of Levy's Call-by-Push-Value language with a kind of computations for every effect set; it can be thought of as a sequential, annotated intermediate language. We develop a range of validated optimisations (i.e., equivalences), generalising many existing ones and adding new ones. We classify these optimisations as structural, algebraic, or abstract: structural optimisations always hold; algebraic ones depend on the effect theory at hand; and abstract ones depend on the global nature of that theory (we give modularly-checkable sufficient conditions for their validity).

Categories and Subject Descriptors D.3.4 [*Processors*]: Compilers; Optimization; F.3.1 [*Specifying and Verifying and Reasoning about Programs*]: Logics of programs; F.3.2 [*Semantics of Programming Languages*]: Algebraic approaches to semantics; Denotational semantics; Program analysis; F.3.3 [*Studies of Program Constructs*]: Type structure

General Terms Languages, Theory.

Keywords Call-by-Push-Value, algebraic theory of effects, code transformations, compiler optimisations, computational effects, denotational semantics, domain theory, inequational logic, relevant and affine monads, sum and tensor, type and effect systems, universal algebra.

1. Introduction

In Gifford-style type and effect analysis [27], each term of a programming language is assigned a type and an effect set. The type describes the values the term may evaluate to; the effect set describes the effects the term may cause during its computation, such as memory assignment, exception raising, or I/O.

For example, consider the following term M:

$$\mathbf{if}\ \mathbf{true}\ \mathbf{then}\ \mathtt{x} := \mathtt{1}\ \mathbf{else}\ \mathtt{x} := \mathbf{deref}(\mathtt{y})$$

It has unit type $\mathbf{1}$ as its sole purpose is to cause side effects; it has effect set $\{\mathtt{update}, \mathtt{lookup}\}$, as it might cause memory updates or look-ups. Type and effect systems commonly convey this information via a type and effect judgement:

$$\mathtt{x} : \mathbf{Loc}, \mathtt{y} : \mathbf{Loc} \vdash M : \mathbf{1} \,!\, \{\mathtt{update}, \mathtt{lookup}\}$$

The information gathered by such effect analyses can be used to guarantee implementation correctness[1], to prove authenticity properties [15], to aid resource management [44], or to optimise code using transformations. We focus on the last of these. As an example, purely functional code can be executed out of order:

$$\mathtt{x} \leftarrow M_1;\ \mathtt{y} \leftarrow M_2;\ N \quad = \quad \mathtt{y} \leftarrow M_2;\ \mathtt{x} \leftarrow M_1;\ N$$

This reordering holds more generally, if the terms M_1 and M_2 have non-interfering effects. Such transformations are commonly used in optimising compilers. They are traditionally called *optimisations*, even if neither side is always the more optimal.

In a sequence of papers, Benton et al. [4–8] prove soundness of such optimisations for increasingly complex sets of effects. However, any change in the language requires a complete reformulation of its semantics and so of the soundness proofs, even though the essential reasons for the validity of the optimisations remain the same. Thus, this approach is not robust, as small language changes cause global theory changes.

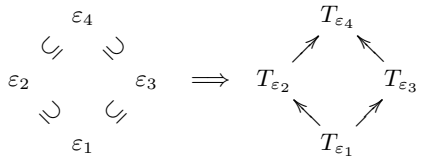
A possible way to obtain robustness is to study effect systems in general. One would hope for a modular approach, seeking to isolate those parts of the theory that change under small language changes, and then recombining them with the unchanging parts. Such a theory may not only be important for compiler optimisations in big, stable languages. It can also be used for effect-dependent equational reasoning. This use may be especially helpful in the case of small, domain-specific languages, as optimising compilers are hardly ever designed for them and their diversity necessitates proceeding modularly.

The only available general work on effect systems seems to be that of Marino and Millstein [28]. They devise a methodology to derive type and effect frameworks which they apply to a call-by-value language with recursion and references; however, their methodology does not account for effect-dependent optimisations.

Fortunately, Wadler and Thiemann [46, 47] had previously made an important connection with the monadic approach to computational effects. They translated judgements of the form $\Gamma \vdash M : A \,!\, \varepsilon$ in a region analysis calculus to judgements of the form $\Gamma' \vdash M' : T_\varepsilon A$ in a multi-monadic calculus. They gave the latter calculus an operational semantics, and conjectured the existence of a corresponding general monadic denotational semantics in which T_ε would denote a monad corresponding to the effects in ε, and in which the partial order of effect sets and inclusions would

[1] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links 0.5, 2009. `http://groups.inf.ed.ac.uk/links` .

induce a hierarchy of monads and monad morphisms, as follows:



Hierarchies of monads in this spirit were identified by both Tolmach [45] and Kieburtz [23]. However, neither hierarchy corresponded fully to their effect set partial order.

Plotkin and Power's algebraic theory of effects [34, 35] provides a semantic foundation for computational effects that focuses on the operations causing the effects. These operations form a single-sorted signature with a natural equational theory. Using tools from universal algebra and category theory, one obtains the standard monadic semantics for such computational effects. The shift in focus from monads to effect operations and their equational theories allows a modular treatment: Hyland et al. [19] show that the theories of combinations of effects can often be built up from those for individual effects by simple operations such as disjoint sum. To model recursion, the algebraic theory of effects turns to domain theory, replacing equational theories by inequational theories.

We argue that the algebraic view of effects provides the missing link needed to develop a general theory of type and effect systems. With the right choice of signature, effects (i.e., the elements of effect sets ε) can be *identified* with the operations. This suggests the way to a full and general account of Gifford-style type and effect systems: effect systems and effect reconstruction arise out of the effect signature; denotational semantics arises out of the (in)equations; and effect-dependent program logic, including optimisations, arises from this semantics.

We concentrate on denotational semantics and optimisations. We present a language, MAIL (Multi-Adjunctive Intermediate Language), which can be thought of as the intermediate language of a compiler. Its terms have both type and effect annotations; these can be thought of as having arisen from a source code syntactic analysis. The denotational semantics of MAIL provides a foundation for optimisations. An optimisation is simply an equation between MAIL terms, and it is *valid* in a MAIL model if both terms have the same denotation.

We see optimisations as falling into three classes: *structural*, *algebraic*, and *abstract*. Structural optimisations arise out of the structure of MAIL models, hence hold in all models; η, β and sequencing laws provide typical examples. Algebraic optimisations arise from equations for individual effects in the inequational theories underlying the semantics; a typical example is

$$(\mathtt{x} := 1; \mathtt{x} := 0) = (\mathtt{x} := 0)$$

These are the bread and butter of effectful program optimisation.

Finally, abstract optimisations arise from global properties of effects. The *Discard* optimisation $x \leftarrow M;\ N = N$ is one such example. There is a strong connection to Führmann's work [13]. He had previously considered properties of arrows in Kleisli categories and their dependence on the nature of the monad, and it turns out that these give rise to several abstract optimisations. In particular, as we see below, Discard holds if, and only if, the underlying monad is affine.

Viewing abstract optimisations algebraically reveals a connection to algebraic laws. For example, the underlying monad is affine if, and only if, the *absorption law* $f(\mathtt{x}, \ldots, \mathtt{x}) = \mathtt{x}$ holds generally, i.e., for every operation f. Making use of [19], one can then establish the validity of abstract optimisations modularly. For example, if the absorption law holds generally for each of two theories, then it also holds for their disjoint sum.

Our language arose from a category-theoretic formulation of the semantics, specifically as presheaves of adjunctions over partial orders of sets of effects. In order to maintain an accessible account, we use a minimum of category theory. We hope this will allow informed readers to recognise the inherent categorical structure without excluding readers who are not familiar with category theory.

The syntax and denotational semantics of MAIL are given in Sections 2 and 3. Two classes of semantic models for MAIL are given (Theorems 9 and 12), both based on standard algebraic semantics, settling Wadler and Thiemann's conjecture (while we write in terms of theories and their translations, rather than monads and their morphisms, their relationship is well-known, see, e.g., [18]). The two classes of models are related to the semantics of Alg-CBPV, an algebraic variant of Call-by-Push-Value, justifying the validity of effect dependent optimisations (Theorems 9 and 12).

Next, Section 4 validates, unifies, generalises, and classifies effect dependent optimisations, including new ones (Figure 7). Algebraic conditions for validating these optimisations are presented (Figure 7), and algebraic techniques are given to recognise and modularly combine validity of optimisations (Theorem 13–Proposition 17).

Section 5 considers an example effectful language with recursion, memory regions, exceptions (normal and rollback), terminal I/O and non-determinism, and applies our theory to validate effect dependent optimisations for it. Our modular methodology enables us to significantly cut down on the amount of work involved. We conclude, and discuss related and further work, in Sections 6 and 7.

2. Multi-Adjunctive Intermediate Language

MAIL is based on Levy's Call-by-Push-Value (CBPV) [26] and inspired by Filinski's $\mathrm{M}^3\mathrm{L}$ (MultiMonadic MetaLanguage) [11], whose latest version draws heavily on CBPV [12]. The CBPV paradigm subsumes call-by-name and call-by-value, both syntactically and semantically, and hence is appealing for a general account. Also, in CBPV evaluation order is explicit, as in intermediate languages. Moreover, expressing our optimisations in CBPV decomposes more complicated optimisations into orthogonal ones (see Section 4). Thus the CBPV paradigm seems well-suited for use in intermediate languages for effect-dependent optimisation. We reuse as much of Levy's work as possible, in order to focus on the issues inherent to effect systems.

MAIL is parameterised by *signatures*. An auxiliary notion of pre-signatures makes the definition of signatures more manageable:

Definition 1. *A* MAIL pre-signature σ *is a tuple* $\langle \mathbf{Bsc}, |\sigma|, \Omega, \mathcal{E}, \kappa \rangle$*, where:* $\mathbf{Bsc}$ *is a set of* basic types*, ranged over by* K*;* $|\sigma|$ *is a set of* effect operation symbols*, ranged over by* op *with a distinguished element* Ω*;* $\mathcal{E} \subseteq \mathcal{P}(|\sigma|)$ *is a subset of the powerset of* $|\sigma|$*, the set of* effect sets*, ranged over by* ε*; and* κ *is a set of* built-in constants*, ranged over by* c*.*

For example, $\mathbf{Bsc}$ may include the types: $\mathbf{Word}$ for 64-bit words, $\mathbf{Loc}$ for memory locations, $\mathbf{Str}$ for strings, $\mathbf{Char}$ for characters, or $\mathbf{Exc}$ for exceptions. The effect operation symbol set $|\sigma|$ may be finite:

$$\{\Omega, \mathtt{lookup}, \mathtt{update}, \mathtt{input}, \mathtt{output}, \mathtt{throw}, \mathtt{choose}\}$$

or even countably infinite: $\{\Omega, \mathtt{throw}^n \mid n \in \mathbb{N}\}$. In the latter case, we may want to restrict $\mathcal{E}$ to be a countable set, such as that of all finite subsets $\mathcal{P}_{\mathrm{fin}}(|\sigma|)$, guaranteeing a countable syntax. The distinguished symbol Ω represents non-termination. The set κ typically includes primitives to manipulate basic values, such as: number constants $\mathtt{0}$, $\mathtt{1}$, $\mathtt{2}$; arithmetic operators $+$, $*$, $\mathtt{div}$; boolean operations $=$, $>=$, $<$; string manipulation primitives $''\mathtt{abcd123}''$, $+\!\!+$; and predefined exceptions $\mathtt{ArithmeticOverflow}$, $\mathtt{DivideByZero}$.

$$
\begin{array}{lrcl}
\text{Kinds:} & \mathcal{K} & ::= & \mathsf{Val} \mid \mathsf{Comp}(\varepsilon) \\
\text{Value Types:} & A, B, \ldots & ::= & K \mid \mathbf{1} \mid A_1 \times A_2 \mid \mathbf{0} \mid A_1 + A_2 \mid \mathbf{U}_\varepsilon \underline{B} \\
\text{Computation Types:} & \underline{A}, \underline{B}, \ldots & ::= & \mathbf{F}_\varepsilon A \mid \underline{B}_1 \times \underline{B}_2 \mid A \to \underline{B} \\
\text{Ground Value Types:} & G & ::= & K \mid \mathbf{1} \mid G_1 \times G_2 \mid \mathbf{0} \mid G_1 + G_2 \\
\text{Value Terms:} & V & ::= & c \mid x \mid \star \mid (V_1, V_2) \mid \mathbf{inj}_{\mathbf{1}}^{A_1+A_2} V \\
 & & & \mid \mathbf{inj}_{\mathbf{2}}^{A_1+A_2} V \mid \mathbf{thunk}\ M \\
\text{Computation Terms:} & M, N, \ldots & ::= & \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M \mid \mathbf{return}_\varepsilon V \mid M\ \mathbf{to}\ x : A.\ N \\
 & & & \mid \lambda\{\mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2\} \mid \mathbf{1}`M \mid \mathbf{2}`M \mid \lambda x : A.\ M \\
 & & & \mid V`M \mid \mathbf{match}\ V\ \mathbf{as}\ (x_1 : A_1, x_2 : A_2).\ M \\
 & & & \mid \mathbf{match}\ V : \mathbf{0}\ \mathbf{as}\ \{\}^{\underline{B}} \\
 & & & \mid \mathbf{match}\ V\ \mathbf{as}\ \{\mathbf{inj_1} x_1 : A_1.\ M_1, \mathbf{inj_2} x_2 : A_2.\ M_2\} \\
 & & & \mid \mathbf{force}\ V \mid \mu x : \mathbf{U}_\varepsilon \underline{B}.\ M \mid \mathrm{op}_V^{\underline{B}} M
\end{array}
$$

Figure 1. MAIL Syntax

$$
\vdash_{\mathrm{k}} K : \mathsf{Val} \quad \vdash_{\mathrm{k}} \mathbf{1} : \mathsf{Val} \quad \frac{\vdash_{\mathrm{k}} A_1 : \mathsf{Val} \quad \vdash_{\mathrm{k}} A_2 : \mathsf{Val}}{\vdash_{\mathrm{k}} A_1 \times A_2 : \mathsf{Val}}
$$

$$
\vdash_{\mathrm{k}} \mathbf{0} : \mathsf{Val} \quad \frac{\vdash_{\mathrm{k}} A_1 : \mathsf{Val} \quad \vdash_{\mathrm{k}} A_2 : \mathsf{Val}}{\vdash_{\mathrm{k}} A_1 + A_2 : \mathsf{Val}} \quad \frac{\vdash_{\mathrm{k}} \underline{B} : \mathsf{Comp}(\varepsilon)}{\vdash_{\mathrm{k}} \mathbf{U}_\varepsilon \underline{B} : \mathsf{Val}}
$$

$$
\frac{\vdash_{\mathrm{k}} A : \mathsf{Val}}{\vdash_{\mathrm{k}} \mathbf{F}_\varepsilon A : \mathsf{Comp}(\varepsilon)} \quad \frac{\vdash_{\mathrm{k}} \underline{B}_1 : \mathsf{Comp}(\varepsilon) \quad \vdash_{\mathrm{k}} \underline{B}_2 : \mathsf{Comp}(\varepsilon)}{\vdash_{\mathrm{k}} \underline{B}_1 \times \underline{B}_2 : \mathsf{Comp}(\varepsilon)}
$$

$$
\frac{\vdash_{\mathrm{k}} A : \mathsf{Val} \quad \vdash_{\mathrm{k}} \underline{B} : \mathsf{Comp}(\varepsilon)}{\vdash_{\mathrm{k}} A \to \underline{B} : \mathsf{Comp}(\varepsilon)}
$$

Figure 2. MAIL Kind System

The syntax of MAIL, for a given pre-signature σ, is displayed in Figure 1. It refines the CBPV dichotomy between values and computations. Instead of one kind of computation, for each effect set $\varepsilon \in \mathcal{E}$, we have ε-computations $\mathsf{Comp}(\varepsilon)$ that can cause effects in ε. We view MAIL as multiple copies of CBPV, one for each ε, sharing the same values. One can translate between these different CBPVs by means of coercion (see below).

Our types are a slight variation on CBPV types. Note that basic types are always value types. The unit, product, zero, and sum value types are standard. We have modified the CBPV thunk type to thunks $\mathbf{U}_\varepsilon \underline{B}$ of ε-computations of type $\underline{B}$. For each $\varepsilon \in \mathcal{E}$, the *returner type* $\mathbf{F}_\varepsilon A$ is the type of ε-computations that return a value of type A. It plays a similar rôle to that of the monadic type TA (where T is a monad) in Haskell. Levy's CBPV also has products of computations, which we include, and functions are computations that depend on a value. The *ground value types* $G \in \mathbf{Gnd}$ are those value types which do not include thunks. We call types of the form $\mathbf{F}_\varepsilon G$ *ground returner types*.

All of the built-in constants of σ-MAIL are value terms. The variables, unit value, and pairing construct are standard. Injections are annotated with their sum type. Computations M are thunked into values $\mathbf{thunk}\ M$, just as in ordinary CBPV.

Many type and effect systems contain a sub-effecting rule: if M is an ε_1-computation and $\varepsilon_1 \subseteq \varepsilon_2$ then M is an ε_2-computation. One implication of such a rule is that well-typed terms can have multiple types, and even multiple type derivations. As a consequence, the denotational semantics can no longer be defined directly on terms. Instead, it needs to be defined on the proofs that these terms are well-typed, and then *coherence* results are needed to show that the different semantics are compatible with each other.

This issue is familiar from languages that support *subtyping*. One standard way to circumvent it, followed here, is to use explicit coercion between a subtype and its supertype [43]. The terms $\mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M$ explicitly coerce ε_1-computations to ε_2-computations.

As usual in CBPV, we can turn any value into an ε-computation by returning it. The $M\ \mathbf{to}\ x : A.\ N$ construct sequences computations; it is analogous to $x \leftarrow M; N$ in Haskell. Note the intrinsic typing (a.k.a. Church-style typing).

The standard operational semantics of CBPV uses a stack machine. The two λ terms pop from the stack while the application terms $-`M$ push onto it. Thus, $\lambda\{\mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2\}$ pops a *tag* off the stack and executes $M_1$ or $M_2$ accordingly. In turn, $\mathbf{1}`M$ pushes the tag $\mathbf{1}$ onto the stack and continues to execute M. Similarly, $\lambda x : A.\ M$ pops a value of type A and binds x to it, while $V`M$ pushes V onto the stack.

The pattern-matching terms eliminating products, zero and sum values are standard. Thunked computations are eliminated by forcing. Recursion is expressed by least fixed-points, as usual.

Importantly, computational effects are caused by the $\mathrm{op}_V^{\underline{B}} M$ terms. As an example, the following computation consists of a memory lookup operation, dereferencing memory location ℓ, followed by returning the memory word $\mathtt{w}$ stored there:

$$
\mathtt{deref}^\varepsilon\ (\ell) \stackrel{\text{def}}{=} \mathtt{lookup}_\ell^{\mathbf{F}_\varepsilon \mathbf{Word}}\ (\lambda \mathtt{w} : \mathbf{Word}.\ \mathbf{return}_\varepsilon \mathtt{w})
$$

The *effect operation symbol* `lookup` takes as a *parameter* the location ℓ to be dereferenced, and requires as *argument* a computation depending on the dereferenced memory word.

As another example, consider a non-deterministic choice operator `choose`, and a computation for non-deterministic coin tossing:

$$
\mathtt{toss}^\varepsilon \stackrel{\text{def}}{=} \mathtt{choose}_\star^{\mathbf{F}_\varepsilon \mathbf{1}+\mathbf{1}}\ (\lambda \mathtt{v} : \mathbf{1} + \mathbf{1}.\ \mathbf{return}_\varepsilon\ \mathtt{v})
$$

In this case the parameter is the unit value $\star$.

In general, $\mathrm{op}_V^{\underline{B}} M$ is an effect operation term with parameter V and argument M. Terms of the form

$$
\mathrm{gen}_{\mathrm{op}}^\varepsilon\ (V) \stackrel{\text{def}}{=} \mathrm{op}_V^{\mathbf{F}_\varepsilon A}\ (\lambda x : A.\ \mathbf{return}_\varepsilon x)
$$

are called *generic effects*. Thus $\mathtt{deref}^\varepsilon$ and $\mathtt{toss}^\varepsilon$ are the generic effects corresponding to `lookup` and `choose` respectively.

Our combination of CBPV syntax, intrinsic typing and explicit coercion leads to a verbose and cumbersome syntax. However, considering our setting as an *intermediate representation* used by an optimising compiler, this issue becomes irrelevant, as the syntax is generated automatically, being seen only by the compiler.

The kind system for MAIL, given a pre-signature σ, is displayed in Figure 2; it consists of a kind judgement relation $\vdash_{\mathrm{k}}$ between types and kinds. We denote by $\mathbf{Val}$ the set of well-kinded value types $\{V \mid \vdash_{\mathrm{k}} V : \mathsf{Val}\}$. Similarly, we write $\mathbf{Comp}(\varepsilon)$ for the set of well-kinded ε-computation types. A *well-kinded context* $\Gamma : \mathrm{Dom}\,(\Gamma) \to \mathbf{Val}$ is a function from a *finite* set of variables to $\mathbf{Val}$. We write $\Gamma, x : A$ for the extension $\Gamma[x \mapsto A]$.

Given a pre-signature σ, an *arity assignment* $\mathrm{ar} : |\sigma| \to \mathbf{Gnd}^2$ sends elements of $|\sigma|$ to pairs of ground types such that $\mathrm{ar}(\Omega) = \langle \mathbf{0}, \mathbf{1} \rangle$. When $\mathrm{ar}(\mathrm{op}) = \langle A, P \rangle$ we write instead $\mathrm{op} : A \to P$. The first component A is called the *arity type* and the second component P is called the *parameter type*. When the parameter type is $P = \mathbf{1}$ we simply write $\mathrm{op} : A$. When $\mathrm{op} : \mathbf{0}$ we call op an *(effect) constant symbol*. Thus, Ω is a constant symbol.

For example, one would have $\mathtt{lookup} : \mathbf{Word} \to \mathbf{Loc}$ as `lookup` has a location parameter and its argument expects the cor-

$$\Gamma \vdash_{\mathrm{v}} c : A_c \quad (c : \mathrm{ar}) \qquad \frac{\Gamma(x) = A}{\Gamma \vdash_{\mathrm{v}} x : A} \qquad \Gamma \vdash_{\mathrm{v}} \star : \mathbf{1} \qquad \frac{\Gamma \vdash_{\mathrm{v}} V_1 : A_1 \quad \Gamma \vdash_{\mathrm{v}} V_2 : A_2}{\Gamma \vdash_{\mathrm{v}} (V_1, V_2) : A_1 \times A_2}$$

$$\frac{\Gamma \vdash_{\mathrm{v}} V : A_i}{\Gamma \vdash_{\mathrm{v}} \mathbf{inj}_{\mathbf{i}}^{A_1+A_2} V : A_1 + A_2} \qquad \frac{\Gamma \vdash_{\varepsilon} M : \underline{B}}{\Gamma \vdash_{\mathrm{v}} \mathbf{thunk}\, M : \mathbf{U}_{\varepsilon}\underline{B}}$$

$$\frac{\Gamma \vdash_{\varepsilon_1} M : \mathbf{F}_{\varepsilon_1} A}{\Gamma \vdash_{\varepsilon_2} \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M : \mathbf{F}_{\varepsilon_2} A} \qquad \frac{\Gamma \vdash_{\mathrm{v}} V : A}{\Gamma \vdash_{\varepsilon} \mathbf{return}_{\varepsilon} V : \mathbf{F}_{\varepsilon} A}$$

$$\frac{\Gamma \vdash_{\varepsilon} M : \mathbf{F}_{\varepsilon} A \quad \Gamma, x : A \vdash_{\varepsilon} N : \underline{B}}{\Gamma \vdash_{\varepsilon} M \,\mathbf{to}\, x : A.\, N : \underline{B}}$$

$$\frac{\Gamma \vdash_{\varepsilon} M_1 : \underline{B}_1 \quad \Gamma \vdash_{\varepsilon} M_2 : \underline{B}_2}{\Gamma \vdash_{\varepsilon} \lambda\{\mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2\} : \underline{B}_1 \times \underline{B}_2} \qquad \frac{\Gamma \vdash_{\varepsilon} M : \underline{B}_1 \times \underline{B}_2}{\Gamma \vdash_{\varepsilon} \mathbf{i}\text{`}M : \underline{B}_i}$$

$$\frac{\Gamma, x : A \vdash_{\varepsilon} M : \underline{B}}{\Gamma \vdash_{\varepsilon} \lambda x : A.\, M : A \to \underline{B}} \qquad \frac{\Gamma \vdash_{\mathrm{v}} V : A \quad \Gamma \vdash_{\varepsilon} M : A \to \underline{B}}{\Gamma \vdash_{\varepsilon} V\text{`}M : \underline{B}}$$

$$\frac{\Gamma \vdash_{\mathrm{v}} V : A_1 \times A_2 \quad \Gamma, x_1 : A_2, y : A_2 \vdash_{\varepsilon} M : \underline{B}}{\Gamma \vdash_{\varepsilon} \mathbf{match}\, V \,\mathbf{as}\, (x_1 : A_1, x_2 : A_2)\,.\, M : \underline{B}}$$

$$\frac{\Gamma \vdash_{\mathrm{v}} V : \mathbf{0}}{\Gamma \vdash_{\varepsilon} \mathbf{match}\, V : \mathbf{0} \,\mathbf{as}\, \{\}^{\underline{B}} : \underline{B}}$$

$$\frac{\Gamma \vdash_{\mathrm{v}} V : A_1 + A_2 \quad \Gamma, x_1 : A_1 \vdash_{\varepsilon} M_1 : \underline{B} \quad \Gamma, x_2 : A_2 \vdash_{\varepsilon} M_2 : \underline{B}}{\Gamma \vdash_{\varepsilon} \mathbf{match}\, V \,\mathbf{as}\, \{\mathbf{inj_1} x_1 : A_1.\, M_1, \mathbf{inj_2} x_2 : A_2.\, M_2\} : \underline{B}}$$

$$\frac{\Gamma \vdash_{\mathrm{v}} V : \mathbf{U}_{\varepsilon}\underline{B}}{\Gamma \vdash_{\varepsilon} \mathbf{force}\, V : \underline{B}} \qquad \frac{\Gamma, x : \mathbf{U}_{\varepsilon}\underline{B} \vdash_{\varepsilon} M : \underline{B}}{\Gamma \vdash_{\varepsilon} \mu x : \mathbf{U}_{\varepsilon}\underline{B}.\, M : \underline{B}}\ (\Omega \in \varepsilon)$$

$$\frac{\Gamma \vdash_{\varepsilon} V : P \quad \Gamma \vdash_{\varepsilon} M : A \to \underline{B}}{\Gamma \vdash_{\varepsilon} \mathrm{op}_V^{\underline{B}} M : \underline{B}}\ (\mathrm{op} : A \to P, \mathrm{op} \in \varepsilon)$$

Figure 3. MAIL Type System

responding memory word. Similarly, one would have `choose` : $\mathbf{2}$ where, $\mathbf{2} = \mathbf{1} + \mathbf{1}$, as `choose` has a trivial parameter. We say that `choose` is a *binary* operation symbol. Example constant type assignments are: $'\mathtt{a}' : \mathbf{Char}$, $+\!\!+ : \mathbf{U}_{\emptyset}((\mathbf{Str} \times \mathbf{Str}) \to \mathbf{F}_{\emptyset}\mathbf{Str})$, and, for $\varepsilon \stackrel{\text{def}}{=} \{\mathtt{ArithmeticOverflow}\}$:

$$+ : \mathbf{U}_{\varepsilon}(\mathbf{Word} \times \mathbf{Word} \to \mathbf{F}_{\varepsilon}\mathbf{Word})$$

Definition 2. *A* MAIL signature *is a triple* $\Sigma = \langle \sigma, \mathrm{ar}, A_- \rangle$ *where:* σ *is a pre-signature;* ar *is an arity assignment; and* $A_- : \kappa \to \mathbf{Val}$ *is a* type assignment *for the built-in constant symbols.*

The type system of MAIL, given a signature Σ, is displayed in Figure 3; it is given by type judgement relations $\Gamma \vdash_{\mathrm{v}} V : A$ and $\Gamma \vdash_{\varepsilon} M : \underline{B}$, where Γ, A, $\underline{B}$ are well-kinded contexts, value types and ε-computation kinds, respectively, and where V, M are value and computation terms, respectively. Signatures Σ determine the language of MAIL, meaning its syntax and kind and typing relations; when we need to mention this dependence we write Σ-MAIL. We call closed ground returners $\vdash_{\varepsilon} P : \mathbf{F}_{\varepsilon} G$ *programs*.

The type system is straightforward, apart from the rules for coercion, recursion and effect operation symbols. The coercion rule may seem surprising — we only allow coercion of returners. However this restriction, which is semantically natural, allows sufficient generality. For if $\varepsilon \subseteq \varepsilon'$, then for any $\underline{B} : \mathsf{Comp}(\varepsilon)$ one can inductively define a type $\underline{B}' : \mathsf{Comp}(\varepsilon')$ by: $(\mathbf{F}_{\varepsilon} A)' = \mathbf{F}_{\varepsilon'} A$, $(\underline{B}_1 \times \underline{B}_2)' = (\underline{B}_1)' \times (\underline{B}_2)'$, and $(A \to \underline{B})' = A \to \underline{B}'$; there is then an evident coercion from $\underline{B}$ to $\underline{B}'$.

We note too that, from a CBPV perspective, call-by-value types are always translated into returners [26], hence the coercion rule immediately suffices to subsume existing effect systems. We do not know of any call-by-name effect systems (see also Section 7).

Next, the recursion rule only allows recursive calls when the effect set includes possible divergence. Finally, the effect operation rule formalises the informal explanation given earlier. Another way to view this rule is via continuation passing — we pass a continuation M that, depending on the effect's result, proceeds after the effect has been caused. For example, the rules for the I/O effects `input` : $\mathbf{Char}$ and `output` : $\mathbf{1} \to \mathbf{Char}$ are, for suitable ε's:

$$\frac{\Gamma \vdash_{\varepsilon} M : \mathbf{Char} \to \underline{B}}{\Gamma \vdash_{\varepsilon} \mathtt{input}^{\underline{B}} M : \underline{B}} \qquad \frac{\Gamma \vdash_{\mathrm{v}} V : \mathbf{Char} \quad \Gamma \vdash_{\varepsilon} M : \mathbf{1} \to \underline{B}}{\Gamma \vdash_{\varepsilon} \mathtt{output}_V^{\underline{B}} M : \underline{B}}$$

The latter is analogous to Levy's `print` statement [26]. For the corresponding generic effects we derive the familiar $\mathtt{get}^{\varepsilon}$, $\mathtt{put}^{\varepsilon}$:

$$\Gamma \vdash_{\varepsilon} \mathtt{get}^{\varepsilon} : \mathbf{F}_{\varepsilon}\mathbf{Char} \qquad \frac{\Gamma \vdash_{\mathrm{v}} V : \mathbf{Char}}{\Gamma \vdash_{\varepsilon} \mathtt{put}^{\varepsilon} : \mathbf{F}_{\varepsilon}\mathbf{1}}$$

3. Semantics

We begin with some preliminary material on domain theory in Section 3.1 concerning ω-cpos, and then consider inequational theories and their models in ω-cpos in Section 3.2. We can then define models of MAIL and its denotational semantics in Section 3.3. In Section 3.4 we consider the validity of optimisations and construct our two main MAIL models — the conservative and axiomatic restriction models.

3.1 ω-cpos

Domain theory provides the mathematical machinery needed to model recursion; here we review terminology and relevant notation. The simplest type of domains sufficient for our needs are *ω-complete partial orders (ω-cpos)*, i.e., partial orders $W = \langle |W|, \leq \rangle$ closed under suprema $\bigvee a_n$ of increasing ω-chains $\langle a_n \rangle$. A *(Scott) continuous function* $f : W_1 \to W_2$ between ω-cpos is a monotone function preserving such suprema (i.e., $f(\bigvee a_n) = \bigvee f(a_n)$ for any increasing ω-chain $\langle a_n \rangle$).

A *discrete* ω-cpo is a set ordered by equality $\langle A, = \rangle$. The *empty ω-cpo* $\mathbb{0}$ is $\langle \emptyset, = \rangle$; we write $?_W : \emptyset \to |W|$ for the empty map. The singleton ω-cpo $\mathbb{1}$ is $\langle \{\star\}, = \rangle$; we write $!_W : |W| \to \{\star\}$ for the constantly-$\star$ map.

The *sum* $W_1 + W_2$ of two ω-cpos is the evident ω-cpo over their disjoint union $|W_1| + |W_2|$. We write $\iota_i : W_i \to W_1 + W_2$ for the evident injections. Given two continuous functions $f_i : W_i \to W'$, $i = 1, 2$, the map $[f_1, f_2] : W_1 + W_2 \to W'$, where:

$$[f_1, f_2]\,(w) \stackrel{\text{def}}{=} \begin{cases} f_1(w_1) & w = \iota_1 w_1 \\ f_2(w_2) & w = \iota_2 w_2 \end{cases}$$

is continuous. We write $\mathbb{n}$ for the discrete domain over n elements, namely $\mathbb{1} + \ldots + \mathbb{1}$.

The *product* $W_1 \times W_2$ of two ω-cpos is the componentwise partial order over their Cartesian product $|W_1| \times |W_2|$. We write $\pi_i : W_1 \times W_2 \to W_i$ for the evident projections. Given two continuous functions $f_i : W' \to W_i$, $i = 1, 2$, the map $\langle f_1, f_2 \rangle : W' \to W_1 \times W_2$, where:

$$\langle f_1, f_2 \rangle\,(w) \stackrel{\text{def}}{=} \langle f_1(w), f_2(w) \rangle$$

is continuous. The product $\prod_{i=1}^{n} W_i$ of any finite number of ω-cpos is defined similarly.

The *function space* $W_2^{W_1}$ of two ω-cpos consists of all continuous functions from W_1 to W_2, ordered pointwise: $f \leq g$ iff for all $w \in W_1$, $f(w) \leq g(w)$; its lubs are also given pointwise. We write $\mathrm{eval} : W_2^{W_1} \times W_1 \to W_2$ for the evaluation map given by $\langle f, d \rangle \mapsto f(w)$. If $f : \prod_i W_i \to W$ is continuous then *currying*

the j-th component $\langle d_i \rangle_{i \neq j} \mapsto (w_j \mapsto f \langle d_i \rangle)$ yields a continuous map $\mathrm{curry}_j\, f : \left(\prod_{i \neq j} W_i\right) \to W^{W_j}$.

An ω-cpo W is *pointed* if it has a least element $\bot$. Continuous functions over pointed ω-cpos have *least* fixed-points and the least fixed-point operation $\mu : W^W \to W$ is continuous.

3.2 Inequational Theories

A full account of Plotkin and Power's algebraic theory of effects requires the notion of (discrete) countable $\omega\mathbf{CPO}$-enriched Lawvere theories [17]. Here we restrict to the simpler, and more elementary, *(finitary) inequational theories*.

An *(equational) signature* is a pair $\Sigma = \langle |\Sigma|, \mathrm{ar}_\Sigma \rangle$ where $|\Sigma|$ is a set of *operation symbols* and the *arity function* ar_Σ assigns to each operation $f \in \Sigma$ a finite set $\mathrm{ar}_\Sigma(f)$ called its *arity*. We adopt the notation $f : A$ for $A = \mathrm{ar}(f)$. When $f : \mathbb{n}$ we say that f is *n-ary*. Nullary operations are also called *constants*. We will write $\{f_1 : \mathrm{ar}(f_1), \ldots, f_n : \mathrm{ar}(f_n)\}$ for the signature $\langle \{f_1, \ldots, f_n\}, \mathrm{ar} \rangle$.

For example, the *non-determinism* signature, Σ_{ND}, is $\{\vee : \mathbb{2}\}$. It consists of exactly one binary operation $\vee : \mathbb{2}$. Again, given a finite set $\mathbb{V}$ of *storable values*, the global $\mathbb{V}$-state signature, $\Sigma_{\mathrm{GS}(\mathbb{V})}$ is given by a look-up operation $\mathtt{lookup} : \mathbb{V}$ and $\mathbb{V}$-many unary operations for updating the state $\mathtt{update}_v : \mathbb{1}$ $(v \in \mathbb{V})$.

Fix a countable set of *variables* $\mathbf{Var}$ ranged over by $\mathbf{x}$, $\mathbf{y}$, etc. Given a signature Σ, we define the set Terms^Σ of Σ-terms inductively: a term is either a variable $\mathbf{x}$, or of the form $f(\tau)$ where $f : A$ and $\tau : A \to \mathrm{Terms}^\Sigma$. When f is n-ary we may use the usual positional notation $f(t_1, \ldots, t_n)$ to write such terms, and, in the binary case, we may use infix notation. It is routine to define substitutions $\theta : \mathbf{Var} \to \mathrm{Terms}$, and their applications $t\theta$ to terms.

Recall that a preorder is a reflexive, transitive binary relation. An *inequational theory* $\mathcal{T}$ is a pair $\langle \Sigma, \leq \rangle$ where Σ is a signature and $\leq$ is a preorder over terms, which is a *substitutive congruence*, meaning that it satisfies the following two properties:

Congruence For all substitutions θ_1, θ_2 satisfying, for all $\mathbf{x}$ in $\mathbf{Var}$, $\theta_1(\mathbf{x}) \leq \theta_2(\mathbf{x})$, and for all terms t, we have $t\theta_1 \leq t\theta_2$.

Substitution For all terms t_1,t_2 for which $t_1 \leq t_2$, and for all substitutions θ, we have $t_1\theta \leq t_2\theta$.

We write $t = s$ iff $t \leq s$ and $s \leq t$.

Let Σ be a signature. The *free* or *empty* theory for Σ is given by syntactic equality, $\langle \Sigma, \equiv \rangle$. The *inconsistent* theory for Σ is given by the full relation $\left\langle \Sigma, \mathrm{Terms}^2 \right\rangle$. Note that all theories $\langle \Sigma, \leq \rangle$ lie between these two theories, i.e., $\equiv \subseteq \leq \subseteq \mathrm{Terms}^2$.

Let $\mathrm{Ax} \subseteq \mathrm{Terms} \times \mathrm{Terms}$ be any set of *axioms*. The *theory generated from* Ax is $\mathrm{Th}^\Sigma \mathrm{Ax} = \langle \Sigma, \leq \rangle$ where $\leq$ is the least preorder over terms that contains Ax and which is a substitutive congruence. Note that the empty theory is $\mathrm{Th}^\Sigma \emptyset$ and the inconsistent theory is $\mathrm{Th}^\Sigma \{\mathbf{x} \leq \mathbf{y}\} = \mathrm{Th}^\Sigma \{\mathbf{x} = \mathbf{y}\}$. The *lifting* theory $\mathcal{T}_\bot$ is given by $\mathrm{Th}^{\{\bot : \mathbb{0}\}} \{\bot \leq \mathbf{x}\}$. A more interesting example is the theory of *semilattices*, which consists of commutativity, associativity and absorption equations for the non-determinism signature Σ_{ND}:

$$\mathbf{x} \vee \mathbf{y} = \mathbf{y} \vee \mathbf{x}, \quad (\mathbf{x} \vee \mathbf{y}) \vee \mathbf{z} = \mathbf{x} \vee (\mathbf{y} \vee \mathbf{z}), \quad \mathbf{x} \vee \mathbf{x} = \mathbf{x}$$

The theory of *lower* semilattices is obtained by adding the axiom $\mathbf{x} \leq \mathbf{x} \vee \mathbf{y}$; the theory of *upper* semilattices is obtained by instead adding $\mathbf{x} \geq \mathbf{x} \vee \mathbf{y}$. As a final example, let $\mathbb{V}$ be a finite set of storable values. The corresponding theory of global state arises out of the following three equations [29, 34] over the global state signature $\Sigma_{\mathrm{GS}(\mathbb{V})}$:

$$\mathtt{lookup}\,(\lambda v : \mathbb{V}.\mathtt{update}_v(\mathbf{x})) = \mathbf{x}$$
$$\mathtt{update}_{v_0}(\mathtt{lookup}\,(\lambda v : \mathbb{V}.\mathbf{x}_v)) = \mathtt{update}_{v_0}(\mathbf{x}_{v_0})$$
$$\mathtt{update}_{v_1}(\mathtt{update}_{v_2}(\mathbf{x})) = \mathtt{update}_{v_2}(\mathbf{x})$$

Given a signature Σ, an *ω-cpo Σ-algebra* $\mathcal{A}$ consists of a pair $\left\langle |\mathcal{A}|, [\![-]\!]_{\mathcal{A}} \right\rangle$ where $|\mathcal{A}|$ is an ω-cpo and $[\![-]\!]_{\mathcal{A}}$ assigns to each operation $f : A$ a continuous function $[\![f]\!] : |\mathcal{A}|^A \to |\mathcal{A}|$. Every ω-cpo Σ-algebra $\mathcal{A}$ induces an *interpretation function* $\mathcal{I}_{\mathcal{A}}\, [\![-]\!]_{\_} : \mathrm{Terms} \times |\mathcal{A}|^{\mathbf{Var}} \to |\mathcal{A}|$, given for every variable assignment $\delta \in |\mathcal{A}|^{\mathbf{Var}}$ inductively over terms t by:

$$\mathcal{I}_{\mathcal{A}}\, [\![t]\!]_\delta \stackrel{\mathrm{def}}{=} \begin{cases} \delta(\mathbf{x}) & t = \mathbf{x} \\ \left([\![f]\!]_{\mathcal{A}}\, (\lambda a : A.[\![\tau(a)]\!]_\delta)\right) & t = f(\tau), f : A \end{cases}$$

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two such Σ-algebras. Their *product* $\mathcal{A}_1 \times \mathcal{A}_2$ is given by $\left\langle |\mathcal{A}|_1 \times |\mathcal{A}|_2, [\![-]\!]_\times \right\rangle$, where, for all $f : A$, $[\![f]\!]_\times\,(\tau)$ is given pointwise by $\left\langle [\![f]\!]_{\mathcal{A}_1}\,(\pi_1 \circ \tau), [\![f]\!]_{\mathcal{A}_2}\,(\pi_2 \circ \tau) \right\rangle$. Let $\mathcal{A}$ be a Σ-algebra and W an ω-cpo. The *power algebra* $\mathcal{A}^W$ is given by $\left\langle |\mathcal{A}|^W, [\![-]\!] \right\rangle$, where, for all $f : A$, $[\![f]\!]\,(\tau)$ is given pointwise by $\lambda w.[\![f]\!]_{\mathcal{A}}\,(\lambda a.\tau(a)(w))$.

Given an inequational theory $\mathcal{T} = \langle \Sigma, \leq \rangle$, an *$\omega$-cpo $\mathcal{T}$-model* is a Σ-algebra $\mathcal{A}$ such that, for all $t_1 \leq t_2$, and for all $\delta \in |\mathcal{A}|^{\mathbf{Var}}$, we have $[\![t_1]\!]_\delta \leq [\![t_2]\!]_\delta$. For example, the discretely-ordered collection of all non-empty finite subsets of a set with the union operation is a model of the theory of semilattices. As another example, let $\mathbb{V}$ be a finite discrete ω-cpo of storable values and let W be any ω-cpo. Then $(\mathbb{V} \times W)^{\mathbb{V}}$ is the carrier set for a $\Sigma_{\mathrm{GS}(\mathbb{V})}$-algebra with the obvious operations. The product of two $\mathcal{T}$-models is again a $\mathcal{T}$-model. If $\mathcal{A}$ is a $\mathcal{T}$ model and W an ω-cpo, then $\mathcal{A}^W$ is also a $\mathcal{T}$-model.

Given two Σ-algebras $\mathcal{A}_1$, $\mathcal{A}_2$, a *Σ-algebra homomorphism* $h : \mathcal{A}_1 \to \mathcal{A}_2$ is a continuous function $h : |\mathcal{A}_1| \to |\mathcal{A}_2|$ that satisfies, for all $f : A$ and $\tau \in |\mathcal{A}_1|^A$:

$$h\left([\![f]\!]_1\,(\tau)\right) = [\![f]\!]_2\,(h \circ \tau)$$

Homomorphisms between two $\mathcal{T}$-*models* are Σ-algebra homomorphisms between them as algebras. For example, the projections $\pi_i : |\mathcal{A}_1| \times |\mathcal{A}_2| \to |\mathcal{A}_i|$ are Σ-algebra homomorphisms from the product $\mathcal{A}_1 \times \mathcal{A}_2$, for all Σ-algebras $\mathcal{A}_1$, $\mathcal{A}_2$.

Let $\mathcal{T}$ be a theory and W an ω-cpo. The *free $\mathcal{T}$-model over* W is a $\mathcal{T}$-model $F_{\mathcal{T}}W$ for which exists a continuous function $\eta^{\mathcal{T}}_W : W \to |F_{\mathcal{T}}W|$, called the *unit* of the free model, such that for all $\mathcal{T}$-models $\mathcal{A}$ and continuous functions $f : W \to |\mathcal{A}|$ there exists a unique $\mathcal{T}$-algebra homomorphism $f^\dagger_{\mathcal{T}} : F_{\mathcal{T}}W \to \mathcal{A}$ satisfying $f = f^\dagger \circ \eta$.

For example, the free model of the *lifting theory*, defined by $\mathrm{Th}^{\{\bot : \mathbb{0}\}}\{\Omega \leq \mathbf{x}\}$, over an ω-cpo W is given by $|W| + \{\bot\}$ with $w \leq v$ iff $w = \bot$ or $w \leq_W v$. As another example, the $\mathcal{T}_{\mathrm{GS}(\mathbb{V})}$-algebra on $(\mathbb{V} \times W)^{\mathbb{V}}$ described above is the free model over W [19, 34]. The free model for any theory over any ω-cpo always exists, by appeal to Freyd's adjoint functor theorem, cf. Abramsky and Jung [1]. However, its structure may be non-trivial [16, 31].

Let Σ_1, Σ_2 be signatures. A *translation* $\mathfrak{T} : \Sigma_1 \to \Sigma_2$ is a pair $\left\langle \mathbf{x}^-_-, \mathfrak{T}\,(-) \right\rangle$ where $\mathbf{x}^f_- : A \hookrightarrow \mathbf{Var}$ is an injection, for each Σ_1-operation $f : A$, and $\mathfrak{T}\,(-) : \Sigma_1 \to \mathrm{Terms}^{\Sigma_2}$ such that, for each Σ_1-operation f, all the variables in $\mathfrak{T}\,(f)$ are in the image of $\mathbf{x}^f_-$. I.e., for each a in A there is a distinct variable $\mathbf{x}^f_a$, and the variables in $\mathfrak{T}\,(f)$ may only involve these variables. Each translation $\mathfrak{T} : \Sigma_1 \to \Sigma_2$ induces a *translation function* $\mathfrak{T} : \mathrm{Terms}^{\Sigma_1} \to \mathrm{Terms}^{\Sigma_2}$ where $\mathfrak{T}(\mathbf{x}) = \mathbf{x}$ and where $\mathfrak{T}(f(\tau))$ is given by substituting $\mathfrak{T}(\tau a)$ for $\mathbf{x}^f_a$ in $\mathfrak{T}(f)$ simultaneously for all $a \in A$, i.e., $\mathfrak{T}(f(\tau)) = \mathfrak{T}(f)\left[\mathfrak{T}(\tau a)/\mathbf{x}^f_a\right]_{a \in A}$.

Let $\mathcal{T}_1$, $\mathcal{T}_2$ be theories. A *translation* $\mathfrak{T} : \mathcal{T}_1 \to \mathcal{T}_2$ is a translation $\mathfrak{T} : \Sigma_1 \to \Sigma_2$ that respects the inequalities:

$$t_1 \leq_1 t_2 \quad \Longrightarrow \quad \mathfrak{T}(t_1) \leq_2 \mathfrak{T}(t_2)$$

For example, given a finite set $\mathbb{V}$, consider the overwrite theory $\mathcal{T}_{\mathrm{OW}(\mathbb{V})}$ given by the signature $\{\mathtt{update}_v : \mathbb{1} \mid v \in \mathbb{V}\}$ and the axioms $\left\{\mathtt{update}_{v_1}(\mathtt{update}_{v_2}\mathbf{x}) = \mathtt{update}_{v_2}\mathbf{x} \;\middle|\; v_1, v_2 \in \mathbb{V}\right\}$. The map $\mathtt{update}_v \mapsto \mathtt{update}_v\mathbf{x}_\star$ induces a translation from $\mathcal{T}_{\mathrm{OW}(\mathbb{V})}$ to $\mathcal{T}_{\mathrm{GS}(\mathbb{V})}$. We will only use such trivial translations, which have the form $\langle\langle f, a\rangle \mapsto \mathbf{x}_a, f \mapsto f(\lambda a.\mathbf{x}_a)\rangle$, i.e., they translate operation symbols to themselves. They are translations from $\mathcal{T}_1$ to $\mathcal{T}_2$ iff $\Sigma_1 \subseteq \Sigma_2$ and $\leq_1 \;\subseteq\; \leq_2$.

Let $\mathfrak{T} : \mathcal{T}_1 \to \mathcal{T}_2$ be a translation. Every $\mathcal{T}_2$-model $\mathcal{A}$ induces a $\mathcal{T}_1$-model $\mathfrak{T}^*(\mathcal{A})$ as follows. The carrier ω-cpo $|\mathfrak{T}^*(\mathcal{A})|$ is $|\mathcal{A}|$. For each operation $f : A$ in Σ_1, we have a term $\mathfrak{T}(f) \in \mathrm{Terms}^{\Sigma_2}$. Recall the interpretation function $\mathcal{I}_{\mathcal{A}}\,[\![-]\!]_-$, and set, for all δ in $|\mathcal{A}|^{\mathbf{Var}}$:

$$[\![f]\!]_{\mathfrak{T}^*(\mathcal{A})}(\delta) \stackrel{\mathrm{def}}{=} \mathcal{I}_A\,[\![\mathfrak{T}(f)]\!]_\delta$$

We obtain a continuous function $|\mathfrak{T}^*(\mathcal{A})|^A \to |\mathfrak{T}^*(\mathcal{A})|$. Thus $\mathfrak{T}^*(\mathcal{A})$ is a Σ_1-algebra, and because $\mathfrak{T}$ preserves $\mathcal{T}_1$-inequalities and $\mathcal{A}$ is a $\mathcal{T}_2$-model, $\mathfrak{T}^*(\mathcal{A})$ is a $\mathcal{T}_1$-model. Given an ω-cpo W, the $\mathcal{T}_2$ unit over W is a map $\eta^{\mathcal{T}_2}_W : W \to |\mathfrak{T}^*(F_{\mathcal{T}_2}W)|$. Therefore, by the free $\mathcal{T}_1$-model definition, there exists a (unique) homomorphism

$$m^{\mathfrak{T}}_W \stackrel{\mathrm{def}}{=} \left(\eta^{\mathcal{T}_2}_W\right)^\dagger_{\mathcal{T}_1} : F_{\mathcal{T}_1}W \to \mathfrak{T}^*(F_{\mathcal{T}_2}W)$$

such that $\eta^{\mathcal{T}_2}_W = m^{\mathfrak{T}}_W \circ \eta^{\mathcal{T}_1}_W$. Following Filinski's *layered monads* [10], we call the function $m^{\mathfrak{T}}_W : |F_{\mathcal{T}_1}W| \to |F_{\mathcal{T}_2}W|$ the *layering of $F_{\mathcal{T}_1}W$ over $F_{\mathcal{T}_2}W$ along $\mathfrak{T}$*.

Finally, we introduce two common methods to combine theories [19]. Let Σ_1, Σ_2 be signatures. Their *sum* $\Sigma_1 + \Sigma_2$ is given by $|\Sigma_1 + \Sigma_2| \stackrel{\mathrm{def}}{=} |\Sigma_1| + |\Sigma_2|$ and $\mathrm{ar}_{\Sigma_1+\Sigma_2}(\iota_i f) \stackrel{\mathrm{def}}{=} \mathrm{ar}_i(f)$. Given a term t in $\mathrm{Terms}^{\Sigma_i}$, relabelling the operations in t according to the injection $\iota_i : |\Sigma_i| \to |\Sigma_1 + \Sigma_2|$ yields a term $\iota_i t$ in $\mathrm{Terms}^{\Sigma_1+\Sigma_2}$. We can apply this relabelling to binary relations over Σ_i in the obvious manner. Let $\mathcal{T}_1$, $\mathcal{T}_2$ be two theories. Their *sum* $\mathcal{T}_1 + \mathcal{T}_2$ is given by $\mathrm{Th}^{\Sigma_1+\Sigma_2}((\iota_1 \leq_1) \cup (\iota_2 \leq_2))$. Their *tensor* $\mathcal{T}_1 \otimes \mathcal{T}_2$ is given by $\mathrm{Th}^{\Sigma_1+\Sigma_2}((\iota_1 \leq_1) \cup (\iota_2 \leq_2) \cup (\mathrm{Ax}_{\Sigma_1\otimes\Sigma_2}))$, where $\mathrm{Ax}_{\Sigma_1\otimes\Sigma_2}$ contains all equations of the form:

$$f^1(\lambda a_1.f^2(\lambda a_2.\mathbf{x}_{a_1,a_2})) = f^2(\lambda a_2.f^1(\lambda a_1.\mathbf{x}_{a_1,a_2}))$$

where $f^i : A_i$ in $\iota_i\Sigma_i$, and $a_i \in A_i$.

3.3 Semantics

Given a MAIL signature Σ, a Σ-MAIL *model* consists of a quadruple $\langle\mathcal{B}[\![-]\!], \mathcal{T}_-, \mathfrak{T}_-, \mathcal{K}[\![-]\!]\rangle$ which we describe presently.

To each basic type $K \in \mathbf{Bsc}$, we assign an ω-cpo $\mathcal{B}[\![K]\!]$. This assignment allows us to interpret ground value types $G \in \mathbf{Gnd}$:

$$\mathcal{G}[\![K]\!] \stackrel{\mathrm{def}}{=} \mathcal{B}[\![K]\!]$$
$$\mathcal{G}[\![\mathbf{1}]\!] \stackrel{\mathrm{def}}{=} \mathbb{1} \quad \mathcal{G}[\![G_1 \times G_2]\!] \stackrel{\mathrm{def}}{=} \mathcal{G}[\![G_1]\!] \times \mathcal{G}[\![G_2]\!]$$
$$\mathcal{G}[\![\mathbf{0}]\!] \stackrel{\mathrm{def}}{=} \mathbb{0} \quad \mathcal{G}[\![G_1 + G_2]\!] \stackrel{\mathrm{def}}{=} \mathcal{G}[\![G_1]\!] + \mathcal{G}[\![G_2]\!]$$

We impose on $\mathcal{B}[\![-]\!]$ that, for all $\mathrm{op} : A \to P$, $\mathcal{G}[\![A]\!]$ is a finite discrete ω-cpo and $\mathcal{G}[\![P]\!]$ is discrete. For example, we may choose $\mathcal{B}[\![\mathbf{Word}]\!]$ and $\mathcal{B}[\![\mathbf{Loc}]\!]$ to be 2^{64} in a 64-bit setting, and $\mathcal{B}[\![\mathbf{Char}]\!]$ to be 2^7 for ASCII characters. As these interpretations are finite discrete domains, every ground type involving them will denote a finite discrete ω-cpo.

The signature Σ and this assignment induce an equational signature:

$$\mathcal{S}[\![|\sigma|]\!] \stackrel{\mathrm{def}}{=} \left\{\mathrm{op}_p : \mathcal{G}[\![A]\!] \;\middle|\; \mathrm{op} : A \to P \in |\sigma|, p \in \mathcal{G}[\![P]\!]\right\}$$

which, in turn, induces an equational sub-signature, for each $\varepsilon \in \mathcal{E}$:

$$|\mathcal{SUB}[\![\varepsilon]\!]| \stackrel{\mathrm{def}}{=} \left\{\mathrm{op}_p \in \mathcal{S}[\![|\sigma|]\!] \;\middle|\; \mathrm{op} \in \varepsilon\right\}$$

$$\begin{array}{rlrl}
\mathcal{V}[\![K]\!] &\stackrel{\mathrm{def}}{=} \mathcal{B}[\![K]\!] & \mathcal{CXT}[\![\Gamma]\!] &\stackrel{\mathrm{def}}{=} \prod_{x\in\mathrm{Dom}(\Gamma)} [\![\Gamma(x)]\!] \\
\mathcal{V}[\![\mathbf{1}]\!] &\stackrel{\mathrm{def}}{=} \mathbb{1} & & \\
\mathcal{V}[\![A_1 \times A_2]\!] &\stackrel{\mathrm{def}}{=} [\![A_1]\!] \times [\![A_2]\!] & & \\
\mathcal{V}[\![\mathbf{0}]\!] &\stackrel{\mathrm{def}}{=} \mathbb{0} & \mathcal{C}_\varepsilon[\![\mathbf{F}_\varepsilon A]\!] &\stackrel{\mathrm{def}}{=} F_\varepsilon[\![A]\!] \\
\mathcal{V}[\![A_1 + A_2]\!] &\stackrel{\mathrm{def}}{=} [\![A_1]\!] + [\![A_2]\!] & \mathcal{C}_\varepsilon[\![\underline{B}_1 \times \underline{B}_2]\!] &\stackrel{\mathrm{def}}{=} [\![\underline{B}_1]\!] \times [\![\underline{B}_2]\!] \\
\mathcal{V}[\![\mathbf{U}_\varepsilon \underline{B}]\!] &\stackrel{\mathrm{def}}{=} |[\![\underline{B}]\!]| & \mathcal{C}_\varepsilon[\![A \to \underline{B}]\!] &\stackrel{\mathrm{def}}{=} [\![\underline{B}]\!]^{[\![A]\!]}
\end{array}$$

Figure 4. Type Interpretation

The next component, $\mathcal{T}_-$, assigns to each $\varepsilon \in \mathcal{E}$ an inequational theory $\mathcal{T}_\varepsilon = \langle\Sigma_\varepsilon, \leq_\varepsilon\rangle$, such that $\mathcal{SUB}[\![\varepsilon]\!] \subseteq \Sigma_\varepsilon$ as a sub-signature, and such that if $\Omega \in \varepsilon$ then $\Omega \leq_\varepsilon \mathbf{x}$. Thus we get, for each $\varepsilon \in \mathcal{E}$ and ω-cpo W, the free ω-cpo $\mathcal{T}_\varepsilon$-algebra, $F_\varepsilon W$ and its corresponding unit η^ε and bijection $-^\dagger_\varepsilon$. The side condition guarantees that if $\Omega \in \varepsilon$ then all $\mathcal{T}_\varepsilon$-models are pointed ω-cpos. This assignment allows us to interpret the Σ-MAIL type system (see Figure 4): value types are interpreted as ω-cpos via $\mathcal{V}[\![-]\!]$; ε-computations as $\mathcal{T}_\varepsilon$-algebras via $\mathcal{C}_\varepsilon[\![-]\!]$; and contexts Γ as products via $\mathcal{CXT}[\![-]\!]$. For example, take $\mathcal{T}_\emptyset$ to be $\mathrm{Th}^\emptyset\emptyset$ the empty theory over the empty signature. In this case, $\mathcal{T}_\emptyset$-models are just ω-cpos, and $F_\emptyset W = W$.

The third component, $\mathfrak{T}_-$, assigns to each pair of subsets $\varepsilon_1 \subseteq \varepsilon_2$ in $\mathcal{E}$ a translation $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2} : \mathcal{T}_{\varepsilon_1} \to \mathcal{T}_{\varepsilon_2}$. We require this translation to be *operation-compatible*, $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2}(\mathrm{op}) = \mathrm{op}(\lambda a.\mathbf{x}^{\mathrm{op}}_a)$, for all $\mathrm{op} \in \varepsilon_1$, and *inclusion-compatible* (*functorial*):

$$\mathfrak{T}_{\varepsilon'\subseteq\varepsilon''}(\mathfrak{T}_{\varepsilon\subseteq\varepsilon'}(\mathrm{op})) = \mathfrak{T}_{\varepsilon\subseteq\varepsilon''}(\mathrm{op}) \qquad \mathfrak{T}_{\varepsilon\subseteq\varepsilon}(\mathrm{op}) = \mathrm{op}(\lambda a.\mathbf{x}^{\mathrm{op}}_a)$$

for every triple $\varepsilon \subseteq \varepsilon' \subseteq \varepsilon''$ in $\mathcal{E}$ and $\mathrm{op} : A \to P$ in ε. For example, if $\mathcal{T}_\emptyset = \mathrm{Th}^\emptyset\emptyset$, we can choose $\mathfrak{T}_{\emptyset\subseteq\varepsilon}$ to be the empty translation. These translations induce an inclusion-compatible layering $m^{\varepsilon_1\subseteq\varepsilon_2}$.

The final component, $\mathcal{K}[\![-]\!]$, assigns to each built-in constant $c \in \kappa$ an element $\mathcal{K}[\![c]\!] \in \mathcal{V}[\![A_c]\!]$. It is instructive to interpret the arithmetic constants on 64-bit integers that can cause exceptions as an example.

We summarise the definition of a Σ-MAIL model:

Definition 3. *Let Σ be a* MAIL *signature. A Σ-*MAIL *model is a quadruple $\mathcal{M} = \langle\mathcal{B}[\![-]\!], \mathcal{T}_-, \mathfrak{T}_-, \mathcal{K}[\![-]\!]\rangle$, where: $\mathcal{B}[\![-]\!]$ assigns to each basic type K an ω-cpo such that all $|\sigma|$-arities are interpreted as finite discrete domains; $\mathcal{T}_-$ assigns to each effect set a theory whose signature includes $\mathcal{SUB}[\![\varepsilon]\!]$, such that if $\Omega \in \varepsilon$ then $\Omega \leq \mathbf{x}$; $\mathfrak{T}_-$ assigns to each inclusion $\varepsilon_1 \subseteq \varepsilon_2$ a translation $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2} : \mathcal{T}_{\varepsilon_1} \to \mathcal{T}_{\varepsilon_2}$ in a compatible manner; and $\mathcal{K}[\![-]\!]$ assigns to each built-in constant c an element $\mathcal{K}[\![c]\!] \in \mathcal{V}[\![A_c]\!]$.*

Given a Σ-MAIL model $\mathcal{M}$, we interpret Σ-MAIL terms as follows (see Figure 5): value terms $\Gamma \vdash_{\mathrm{v}} V : A$ are interpreted as continuous functions $\mathcal{VT}[\![V]\!] : \mathcal{CXT}[\![\Gamma]\!] \to \mathcal{V}[\![A]\!]$; and ε-computation terms $\Gamma \vdash_\varepsilon M : \underline{B}$ are interpreted as continuous functions $\mathcal{CT}_\varepsilon[\![M]\!] : \mathcal{CXT}[\![\Gamma]\!] \to |\mathcal{C}_\varepsilon[\![\underline{B}]\!]|$. The semantic functions have straightforward definitions. The only exception is coercion from ε_1-returners to ε_2-returners via the layering $m^{\varepsilon_1\subseteq\varepsilon_2}$. Also note how the type system and the side condition over $\mathcal{T}_\varepsilon$ ensures these semantic functions are well-defined.

3.4 Validity and Models

First, we define the validity of optimisations in MAIL:

Definition 4. *Let $\mathcal{M}$ be a Σ-*MAIL *model, and $\Gamma \vdash P_i : X$, $i = 1, 2$ be two well-typed Σ-*MAIL *terms. We say that the optimisation $\Gamma \vdash P_1 = P_2 : X$ is* valid *in $\mathcal{M}$ if $\mathcal{M}[\![P_1]\!] = \mathcal{M}[\![P_2]\!]$. In this case we write $\mathcal{M} \models \Gamma \vdash P_1 = P_2 : X$. When Γ and X are clear, we simply write $\mathcal{M} \models P_1 = P_2$.*

$$\mathcal{VT}[\![c]\!](\gamma) \stackrel{\text{def}}{=} [\![c]\!] \quad \mathcal{VT}[\![x]\!](\gamma) \stackrel{\text{def}}{=} \pi_x(\gamma) \quad \mathcal{VT}[\![\star]\!](\gamma) \stackrel{\text{def}}{=} \star$$

$$\mathcal{VT}[\![(V_1, V_2)]\!](\gamma) \stackrel{\text{def}}{=} \langle [\![V_1]\!](\gamma), [\![V_2]\!](\gamma)\rangle$$

$$\mathcal{VT}\Big[\!\Big[\mathbf{inj}_{\mathbf{i}}^{A_1+A_2} V\Big]\!\Big](\gamma) \stackrel{\text{def}}{=} \iota_i([\![V]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon_2}[\![\mathbf{coerce}_{\varepsilon_1\subseteq\varepsilon_2} M]\!](\gamma) \stackrel{\text{def}}{=} m^{\varepsilon_1\subseteq\varepsilon_2}([\![M]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon}[\![\mathbf{return}_{\varepsilon} V]\!](\gamma) \stackrel{\text{def}}{=} \eta^{\varepsilon}([\![V]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon}[\![M \ \mathbf{to}\ x : A.\ N]\!](\gamma) \stackrel{\text{def}}{=} (\lambda a.[\![N]\!](\gamma\,[x \mapsto a]))^{\dagger}_{\varepsilon}([\![M]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon}[\![\mathbf{i}`M]\!](\gamma) \stackrel{\text{def}}{=} \pi_i([\![M]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon}[\![V`M]\!](\gamma) \stackrel{\text{def}}{=} ([\![M]\!](\gamma))\,([\![V]\!](\gamma))$$

$$\mathcal{CT}_{\varepsilon}\Big[\!\Big[\ \mathbf{match}\ V : \mathbf{0}\ \mathbf{as}\ \{\}^{\underline{B}}\Big]\!\Big] \stackrel{\text{def}}{=} ?_{[\![\underline{B}]\!]}$$

$$\mathcal{CT}_{\varepsilon}[\![\ \mathbf{match}\ V\ \mathbf{as}\ \{\mathbf{inj_1}\, x_1 : A_1.\, M_1, \mathbf{inj_2}\, x_2 : A_2.\, M_2\}]\!](\gamma)$$

$$\stackrel{\text{def}}{=} \begin{cases} [\![M_1]\!](\gamma\,[x_1 \mapsto a_1]) & [\![V]\!](\gamma) = \iota_1 a_1 \\ [\![M_2]\!](\gamma\,[x_2 \mapsto a_2]) & [\![V]\!](\gamma) = \iota_2 a_2 \end{cases}$$

$$\mathcal{CT}_{\varepsilon}[\![\mu x : \mathbf{U}_{\varepsilon}\underline{B}.\, M]\!](\gamma) \stackrel{\text{def}}{=} \mu\lambda f \in |[\![\underline{B}]\!]|.[\![M]\!](\gamma\,[x \mapsto f])$$

$$\mathcal{CT}_{\varepsilon}\Big[\!\Big[\mathrm{op}_{V}^{\underline{B}} M\Big]\!\Big](\gamma) \stackrel{\text{def}}{=} [\![\mathrm{op}_{[\![V]\!](\gamma)}]\!]_{[\![\underline{B}]\!]}([\![M]\!](\gamma))$$

Figure 5. Term Interpretation

However, we are actually interested in the validity of optimisations with the effect annotations erased, as that will correspond to the validity of optimisations in the source language. So we need a suitable "algebraic" version of CBPV. We obtain such a language, Σ-Alg-CBPV, by specialising MAIL to signatures with $\mathcal{E} \stackrel{\text{def}}{=} \{|\sigma|\}$. Models of Σ-Alg-CBPV consist of interpretations for basic types, a *single* inequational theory $\mathcal{T}_{|\sigma|}$, and interpretations for the built-in constants. Thus we obtain a syntax and semantics for CBPV with recursion and algebraic effects.

Definition 5. *A* simple MAIL signature *is a* MAIL *signature Σ such that $|\sigma| \in \mathcal{E}$, and for each built-in constant c, the only effect set appearing in A_c is $|\sigma|$.*

Let Σ be a simple signature. Then, by replacing $\mathcal{E}$ with $\{|\sigma|\}$ and ε's with $|\sigma|$ one obtains an Alg-CBPV signature $\Sigma^\natural$. There is an obvious erasure operation $(-)^\natural$ that yields, for each syntactic Σ-MAIL entity, such as type X or term P, a corresponding $\Sigma^\natural$-Alg-CBPV entity, such as $X^\natural$ or $P^\natural$ respectively. This erasure operation preserves well-kindedness and well-typedness.

Models for Algebraic CBPV with common combinations of effects, and means to combine and construct them, are readily available in the literature [19, 20, 34]. Given a simple signature Σ and one such $\Sigma^\natural$-Alg-CBPV model, we now give three corresponding Σ-MAIL models that can be used for reasoning about $\Sigma^\natural$-Alg-CBPV programs.

Lemma 6. *Let Σ be a simple* MAIL *signature. For each $\Sigma^\natural$-Alg-*CBPV *model $\mathcal{M}$ there exists a Σ-*MAIL *model $\mathcal{M}^\flat$ such that $\mathcal{T}_\varepsilon = \mathcal{T}$, and $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2}$ is the identity translation.*

Further, for all well-typed terms $\Gamma \vdash P : X$ we have $\mathcal{M}^\flat[\![X]\!] = \mathcal{M}\big[\!\big[X^\natural\big]\!\big]$ and $\mathcal{M}^\flat[\![P]\!] = \mathcal{M}\big[\!\big[P^\natural\big]\!\big]$.

The proof is straightforward, where the notion of a *simple* signature guarantees our ability to choose interpretations for Σ-MAIL's constants. (It would be useful to extend our results to non-simple signatures.) We call $\mathcal{M}^\flat$ the *benchmark model*. Lemma 6 immediately yields a connection between Σ-MAIL and $\Sigma^\natural$-Alg-CBPV validity:

Theorem 7. *Let $\mathcal{M}$ be a $\Sigma^\natural$-Alg-*CBPV *model. For any two well-typed Σ-*MAIL *programs $\vdash P_i : \mathbf{F}_\varepsilon G$, $i = 1, 2$, we have:*

$$\mathcal{M} \models P_1^\natural = P_2^\natural \iff \mathcal{M}^\flat \models P_1 = P_2$$

This theorem then allows us to compare other Σ-MAIL models to a given $\Sigma^\natural$-Alg-CBPV model via logical relations arguments [11, 30, 41] comparing Σ-MAIL models with the benchmark model.

The benchmark model completely ignores the effect annotations. We next introduce a model that takes them into account:

Definition 8. *Let $\mathcal{T} = \langle \Sigma, \leq\rangle$ be an inequational theory. For any $\varepsilon \subseteq \Sigma$, the* conservative restriction of $\mathcal{T}$ to ε, $\mathcal{T}|_\varepsilon$, *is the inequational theory $\langle \varepsilon, \leq \cap\, \mathrm{Terms}^\varepsilon \times \mathrm{Terms}^\varepsilon\rangle$.*

Theorem 9. *Let $\mathcal{M}$ be an $\Sigma^\natural$-Alg-*CBPV *model. There exists a Σ-*MAIL *model $\mathcal{M}^\sharp$ such that $\mathcal{T}_\varepsilon$ is $\mathcal{T}|_{[\![\varepsilon]\!]}$, and $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2}$ is the trivial translation.*

*Further, for any two well-typed Σ-*MAIL *programs $\vdash P_i : \mathbf{F}_\varepsilon G$, $i = 1, 2$, we have:*

$$\mathcal{M} \models P_1^\natural = P_2^\natural \iff \mathcal{M}^\sharp \models P_1 = P_2$$

We call $\mathcal{M}^\sharp$ the *conservative restriction model*.

Validating optimisations in $\mathcal{M}^\sharp$ depends on finding $\mathcal{T}_\varepsilon$ explicitly. This is non-trivial, but is easier with simpler theories. One might hope to express restrictions of a combination of theories in terms of the restrictions of their components. This hope leads to the following (incomplete) conjecture:

Conjecture 10. *Let $\mathcal{T}_1$,$\mathcal{T}_2$ be inequational theories, and let $\varepsilon_1 + \varepsilon_2 \subseteq \Sigma_1 + \Sigma_2$ be any subset. Under some conditions, the restriction $(\mathcal{T}_1 + \mathcal{T}_2)|_{\varepsilon_1+\varepsilon_2}$ is identical to the sum of restrictions $\mathcal{T}_1|_{\varepsilon_1} + \mathcal{T}_2|_{\varepsilon_2}$. Similarly, under some conditions, $(\mathcal{T}_1 \otimes \mathcal{T}_2)|_{\varepsilon_1+\varepsilon_2}$ is identical to $\mathcal{T}_1|_{\varepsilon_1} \otimes \mathcal{T}_2|_{\varepsilon_2}$.*

To see that the conjecture is not trivial, consider the theory for monoids. The signature is $\{\cdot : 2, e : \mathbb{0}\}$ and the axioms are:

$$(\mathbf{x}\cdot\mathbf{y})\cdot\mathbf{z} = \mathbf{x}\cdot(\mathbf{y}\cdot\mathbf{z}) \quad \mathbf{x}\cdot e = \mathbf{x} = e\cdot\mathbf{x}$$

The non-intuitive, yet elementary, Eckmann-Hilton argument [9] shows that the two theories $(\text{Monoids} \otimes \text{Monoids})|_{\{\cdot,e\}+\emptyset}$ and $\text{Monoids}|_{\{\cdot,e\}} \otimes \text{Monoids}|_{\emptyset}$ are different: multiplication is commutative in the former, but not the latter.

As an intermediate solution, we give a model based on the presentation of a theory as a collection of axioms.

Definition 11. *Let Σ be a signature and Ax a set of axioms over it. For any $\varepsilon \subseteq \Sigma$, the* axiomatic restriction of Ax to ε, $\mathrm{Th}|_\varepsilon\langle\Sigma, \mathrm{Ax}\rangle$, *is the inequational theory $\mathrm{Th}^\varepsilon(\mathrm{Ax} \cap (\mathrm{Terms}^\varepsilon \times \mathrm{Terms}^\varepsilon))$.*

We then have an analogue of Theorem 9:

Theorem 12. *Let Ax be a set of axioms over Σ, and $\mathcal{M}$ a $\Sigma^\natural$-Alg-*CBPV *model such that $\mathcal{T} = \mathrm{Th}^\Sigma \mathrm{Ax}$. There exists a Σ-*MAIL *model $\mathcal{M}_\sharp$ such that $\mathcal{T}_\varepsilon$ is $\mathrm{Th}|_{[\![\varepsilon]\!]}\langle[\![\Sigma]\!], \mathrm{Ax}\rangle$, and $\mathfrak{T}_{\varepsilon_1\subseteq\varepsilon_2}$ is the trivial translation.*

Further, for any two well-typed programs $\vdash P_i : X$, $i = 1, 2$, we have:

$$\mathcal{M} \models {P_1}^\natural = {P_2}^\natural \impliedby \mathcal{M}_\sharp \models P_1 = P_2$$

We call $\mathcal{M}_\sharp$ the *axiomatic restriction model*. Note that by construction $\mathcal{M}_\sharp$ has an explicit axiomatisation of $\mathcal{T}_\varepsilon$. Moreover, if $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$ and $\mathcal{T}_i = \mathrm{Th}^{[\![\Sigma_i]\!]}\mathrm{Ax}_i$, then $\mathcal{T} = \mathrm{Th}(\mathrm{Ax}_1 + \mathrm{Ax}_2)$ and the axiomatic restriction model resulting for $\mathrm{Ax}_1 + \mathrm{Ax}_2$ satisfies for all $\varepsilon_i \subseteq \Sigma_i$, $i = 1, 2$:

$$\begin{aligned}\mathcal{T}_{\varepsilon_1+\varepsilon_2} &= \mathrm{Th}|_{[\![\varepsilon_1+\varepsilon_2]\!]}\langle[\![\Sigma_1+\Sigma_2]\!], (\mathrm{Ax}_1+\mathrm{Ax}_2)\rangle \\ &= \mathrm{Th}^{[\![\Sigma_1]\!]}\mathrm{Ax}_1 + \mathrm{Th}^{[\![\Sigma_2]\!]}\mathrm{Ax}_2 = \mathcal{T}_{\varepsilon_1} + \mathcal{T}_{\varepsilon_2}\end{aligned}$$

β laws: $\mathbf{match}\ (V, V')\ \mathbf{as}\ (x : A, y : A')\,.\,M = M\,[V/x, V'/y]$

$\mathbf{match}\ \mathbf{inj}_{\mathbf{i}}^{A_1+A_2} V\ \mathbf{as}\ \{\mathbf{inj_1}\, x_1 : A_1\,.\,M_1, \mathbf{inj_2}\, x_2 : A_2\,.\,M_2\} = M_i\,[V/x_i]$

$\mathbf{force}\ (\ \mathbf{thunk}\ M) = M \qquad (\mathbf{return}_\varepsilon V)\ \mathbf{to}\ x : A.\, M = M\,[V/x]$

$\mathbf{i}\text{`}\lambda\,\{\mathbf{1} \mapsto M_1, \mathbf{2} \mapsto M_2\} = M_i \qquad V\text{`}\lambda x : A.\, M = M\,[V/x]$

η laws: $V = \star$

$M\,[V/z] = \ \mathbf{match}\ V\ \mathbf{as}\ (x : A, y : A')\,.\,M\,[(x, y)/z] \quad x, y$ fresh in M

$M = \ \mathbf{match}\ V : \mathbf{0}\ \mathbf{as}\ \{\}^{\underline{B}}$

$M\,[V/z] = \ \mathbf{match}\ V\ \mathbf{as}\ \begin{cases} \mathbf{inj_1}\, x : A\ .\, M\,[\mathbf{inj}_1 x/z], \\ \mathbf{inj_2}\, y : A'.\, M\,[\mathbf{inj}_2 y/z] \end{cases} \quad x, y$ fresh in M

$V = \ \mathbf{thunk}\ (\ \mathbf{force}\ V) \qquad M = M\ \mathbf{to}\ x : A.\, \mathbf{return}_\varepsilon x$

$M = \lambda\,\{\mathbf{1} \mapsto \mathbf{1}\text{`}M, \mathbf{2} \mapsto \mathbf{2}\text{`}M\} \qquad N = \lambda x : A.\, x\text{`}N \qquad x$ fresh in N

Sequencing:

$M\ \mathbf{to}\ x : A.\ (N\ \mathbf{to}\ y : A'.\, N') = (M\ \mathbf{to}\ x : A.\, N)\ \mathbf{to}\ y : A'.\, N' \quad x$ fresh in N'

$M\ \mathbf{to}\ x : A.\, \lambda\,\{\mathbf{1} \mapsto N_1, \mathbf{2} \mapsto N_2\} = \lambda \begin{cases} \mathbf{1} \mapsto M\ \mathbf{to}\ x : A.\, N_1, \\ \mathbf{2} \mapsto M\ \mathbf{to}\ x : A.\, N_2 \end{cases}$

$M\ \mathbf{to}\ x : A.\, \lambda y : A'.\, N = \lambda y : A'.\, (M\ \mathbf{to}\ x : A.\, N) \quad y$ fresh in M

Effects:

$\mathrm{op}_V^{\underline{B}} \lambda x : A.\ (M\ \mathbf{to}\ y : A'.\, N) = \left(\mathrm{op}_V^{\mathbf{F}_\varepsilon A'} \lambda x : A.\, M\right) \mathbf{to}\ y : A'.\, N \quad x$ fresh in N

$\mathrm{op}_V^{\underline{B}_1 \times \underline{B}_2} \lambda x : A.\, \lambda \begin{cases} \mathbf{1} \mapsto M_1, \\ \mathbf{2} \mapsto M_2 \end{cases} = \lambda \begin{cases} \mathbf{1} \mapsto \mathrm{op}_V^{\underline{B}_1} \lambda x : A.\, M_1, \\ \mathbf{2} \mapsto \mathrm{op}_V^{\underline{B}_2} \lambda x : A.\, M_2 \end{cases}$

$\mathrm{op}_V^{A' \to \underline{B}} \lambda x : A.\, \lambda y : A'.\, M = \lambda y : A'.\, \mathrm{op}_V^{\underline{B}} \lambda x : A.\, M \quad x \neq y$

Recursion: $\mu x : \mathbf{U}_\varepsilon \underline{B}.\, M = M\,[\ \mathbf{thunk}\ \mu x : \mathbf{U}_\varepsilon \underline{B}.\, M/x]$

Coercion: $\mathbf{coerce}_{\varepsilon_2 \subseteq \varepsilon_3}(\mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M) = \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_3} M$

$\mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2}(\mathbf{return}_{\varepsilon_1} M) = \mathbf{return}_{\varepsilon_2} M$

$\mathbf{coerce}_{\varepsilon \subseteq \varepsilon'}(M\ \mathbf{to}\ x : A.\, N) = \left(\mathbf{coerce}_{\varepsilon \subseteq \varepsilon'} M\right) \mathbf{to}\ x : A.\, \mathbf{coerce}_{\varepsilon \subseteq \varepsilon'} N$

$\mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2}\left(\mathrm{op}_V^{\mathbf{F}_{\varepsilon_1} A'} \lambda x : A.\, M\right) = \mathrm{op}_V^{\mathbf{F}_{\varepsilon_2} A'} \lambda x : A.\, \mathbf{coerce}_{\varepsilon_1 \subseteq \varepsilon_2} M$

Figure 6. Structural Optimisations

Similarly, if $\mathcal{T} = \mathcal{T}_1 \otimes \mathcal{T}_2$, then $\mathcal{T}_{\varepsilon_1+\varepsilon_2} = \mathcal{T}_{\varepsilon_1} \otimes \mathcal{T}_{\varepsilon_2}$ in the axiomatic restriction model resulting for $\mathrm{Ax}_1 + \mathrm{Ax}_2 + (\mathrm{Ax}_{\Sigma_1 \otimes \Sigma_2})$. Thus the analogue of Conjecture 10 holds by construction for $\mathcal{M}_\sharp$.

Also note that for our purposes, this theorem implies that validation of optimisations in the modular approximation model is *sound*. However, it may not be *complete* — some sound optimisations may not be valid in this model.

4. Optimisations

We turn to validating effect-dependent optimisations, formulated as MAIL equations. This is done semantically, using Theorems 9 or 12 (either one can be used in all cases considered below). We divide optimisations into *structural*, *algebraic*, and *abstract* groups. We validate versions of almost all the optimisations in Benton et al. [4–8], and also some others, not previously considered in the semantics literature. The only optimisations considered by Benton et al. [4–8] that we do not treat deal with exception handlers (see Section 7).

Structural optimisations reflect the general structure of our models. Let Σ be a MAIL signature. Then a Σ-MAIL structural optimisation is one that is valid in all Σ-MAIL models. Figure 6 shows example schemes for such optimisations. They are all variants of the standard β, η, sequencing and similar laws found in the CBPV literature [26, 36], together with ones concerning effect coercion.

Algebraic optimisations originate locally in the inequational theories $\mathcal{T}_\varepsilon$ associated to a given model of Σ-MAIL. Each equation yields such an optimisation. To derive these optimisations from the equations, one follows [36]. We just give an example here. The global state theory includes the axiom

$$\mathtt{update}_{v_0}\,(\mathtt{lookup}\,(\lambda v.\mathbf{x}_v)) = \mathtt{update}_{v_0}\,(\mathbf{x}_{v_0})$$

The corresponding algebraic optimisation is

$$\mathtt{update}_V^{\underline{B}}\,(\mathtt{lookup}\,M) = \mathtt{update}_V^{\underline{B}}\,(V\text{`}M)$$

Abstract optimisations follow from overall, global properties of the theories. Each appears in two forms, which we call *utilitarian* and *pristine*. The utilitarian form readily applies to program optimisation; the pristine form is shorter and easier to validate, but perhaps less useful. For example, the utilitarian form of Discard is

$$\frac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \qquad \Gamma \vdash_{\varepsilon'} N : \underline{B}}{\mathcal{M} \models \mathbf{coerce}_{\varepsilon \subseteq \varepsilon'} M\ \mathbf{to}\ x : A.\, N = N}\ (\varepsilon \subseteq \varepsilon')$$

and its pristine form is

$$\frac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{\mathcal{M} \models M\ \mathbf{to}\ x : A.\, \mathbf{return}_\varepsilon \star = \mathbf{return}_\varepsilon \star}$$

The two forms can always be shown equivalent using structural optimisations. Instances of both appear in the work of Benton et al.

This optimisation is valid if, for example, $\mathcal{T}_\varepsilon$ is the environment theory $\mathcal{T}_{\mathrm{Env}(\mathbb{V})}$, whose signature is $\{\mathtt{lookup} : \mathbb{V}\}$ and axioms are:

$$\mathbf{x} = \mathtt{lookup}(\lambda v.x)$$

$$\mathtt{lookup}\,(\lambda v_1.\mathtt{lookup}\,(\lambda v_2.\mathbf{x}_{v_1,v_2})) = \mathtt{lookup}\,(\lambda v.\mathbf{x}_{v,v})$$

Note that the optimisation obtained by erasure, $M\ \mathbf{to}\ x : A.\, N = N$ is in general not valid.

Let $\vdash P : \mathbf{F}_\varepsilon G$ be a well-typed Σ-MAIL program. Let P' be a well-typed Σ-MAIL program obtained by applying the Discard optimisation to P. If Discard is valid in the conservative (axiomatic) restriction model, then Theorem 9 (respectively, Theorem 12) guarantees that $P^\natural = P'^\natural$ is valid. This equation is valid, even if the erased Discard used is invalid. Thus Theorems 9 and 12 formalise the justification for effect dependent optimisation.

Discard also supplies a non-example. Taking M to be a thunk in the pristine form of the optimisation yields on the right hand side the term $\lambda \mathtt{m} : \mathbf{U}_\varepsilon \mathbf{F}_\varepsilon A.\ \mathbf{force}\ \mathtt{m}\ \mathbf{to}\ x : A.\, \mathbf{return}_\varepsilon \star$, and on the left the term $\lambda \mathtt{m} : \mathbf{U}_\varepsilon \mathbf{F}_\varepsilon A.\, \mathbf{return}_\varepsilon \star$. When Discard is valid, these two terms are equal. When the erased Discard is invalid, the erased terms are not equal. Note that these are closed term of the *non-ground* type $\mathbf{U}_\varepsilon \mathbf{F}_\varepsilon A \to \mathbf{F}_\varepsilon \mathbf{1}$. This non-example demonstrates why it is important to restrict to *ground* returners in Theorems 9 and 12.

Validating abstract optimisations denotationally reveals an intimate connection[2] to Führmann's work on the structure of call-by-value [13, 14]: each has a characterisation in semantic terms. For example, the Discard optimisation holds iff $\mathcal{T}_\varepsilon$ is an *affine* theory [13, 21, 24], that is, iff $\eta^\varepsilon_{\mathbb{1}} : \mathbb{1} \to |F_\varepsilon \mathbb{1}|$ is an isomorphism.

There are also *algebraic* characterisations of these semantic properties. For example, a theory $\mathcal{T}$ is affine iff for every term t, $t(\mathbf{x}, \ldots, \mathbf{x}) = \mathbf{x}$, that is, iff a *global absorption law* holds. (Here, and below, we are displaying *all* the variables of the terms at hand.) Such algebraic properties can be investigated using the equational presentation of the theory. As an example, the theory for environments $\mathcal{T}_{\mathrm{Env}(\mathbb{V})}$ and the various semilattice theories are affine.

A summary of the results appears in Figure 7; in any row, all the conditions are equivalent in any model of MAIL, taking the condition in the first column as universally quantified over all ε' such that $\varepsilon' \supseteq \varepsilon$. Benton et al. [4–8] (Führmann [13, 14]) considered call-by-value analogues of the optimisations labelled by **B** (respectively **F**) in Figure 7, albeit for particular combinations of effects (respectively for a fixed monad); Führmann also showed the

[2] Alex Simpson, private communication.

name	utilitarian form	pristine form	abstract side condition	algebraic equivalent	example basic theories
Discard	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \quad \Gamma \vdash_{\varepsilon'} N : \underline{B}}{(\mathbf{coerce} M) \mathbf{\ to\ } x : A.\, N = N}$	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{M \mathbf{\ to\ } x : A.\, \mathbf{return}_\varepsilon \star = \mathbf{return}_\varepsilon \star}$	$\mathcal{T}_\varepsilon$ affine: **F** $\eta^\varepsilon_{\mathbb{1}} : \mathbb{1} \to \lvert F_\varepsilon \mathbb{1} \rvert$ has a continuous inverse	For all ε-terms t: $t(\mathbf{x}, \ldots, \mathbf{x}) = \mathbf{x}$	read-only state, convex, upper and lower semilattices
Copy	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \quad \Gamma, x : A, y : A \vdash_{\varepsilon'} N : \underline{B}}{\begin{array}{l}\mathbf{coerce} M \mathbf{\ to\ } x : A.\\ \mathbf{coerce} M \mathbf{\ to\ } y : A.\, N = \\ \mathbf{coerce} M \mathbf{\ to\ } x : A.\, N\,[x/y]\end{array}}$	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{\begin{array}{l}M \mathbf{\ to\ } x : A.\, M \mathbf{\ to\ } y : A.\, \mathbf{return}_\varepsilon (x, y) \\ = \\ M \mathbf{\ to\ } x : A.\, \mathbf{return}_\varepsilon (x, x)\end{array}}$	$\mathcal{T}_\varepsilon$ relevant: **F** $\psi_\varepsilon \circ \delta = L^\varepsilon \delta$	For all ε-terms t: $t(t(\mathbf{x}_{11}, \ldots, \mathbf{x}_{1n}), \ldots,$ $t(\mathbf{x}_{n1}, \ldots, \mathbf{x}_{nn}))$ $= t(\mathbf{x}_{11}, \ldots, \mathbf{x}_{nn})$	exceptions, lifting, read-only state, write-only state
Weak Copy	$\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \quad \Gamma, x : A \vdash_{\varepsilon'} N : \underline{B}}{\begin{array}{l}\mathbf{coerce} M \mathbf{\ to\ } x : A.\\ \mathbf{coerce} M \mathbf{\ to\ } y : A.\, N = \\ \mathbf{coerce} M \mathbf{\ to\ } x : A.\, N\end{array}}$	$\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{M \mathbf{\ to\ } x : A.\, M = M}$	$\mu^\varepsilon \circ \mathrm{str}^\varepsilon \circ \delta = \mathrm{id}$	For all ε-terms t: $t(t(\mathbf{x}_1, \ldots, \mathbf{x}_n), \ldots,$ $t(\mathbf{x}_1, \ldots, \mathbf{x}_n))$ $= t(\mathbf{x}_1, \ldots, \mathbf{x}_n)$	any affine or relevant theory: lifting, exceptions, read-only and write-only state, all three semilattice theories
Swap	**B F** $\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2 \quad \Gamma, x_1 : A_1, x_2 : A_2 \vdash_{\varepsilon'} N}{\begin{array}{l}\mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\, N = \\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\\ \mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\, N\end{array}}$	**B F** $\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2}{\begin{array}{l}\mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\, \mathbf{return}_\varepsilon (x_1, x_2) = \\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\\ \mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\, \mathbf{return}_\varepsilon (x_1, x_2)\end{array}}$	$\mathfrak{T}_{\varepsilon_1 \subseteq \varepsilon}, \mathfrak{T}_{\varepsilon_2 \subseteq \varepsilon}$ commute: **F** $\psi_\varepsilon \circ (m^{\varepsilon_1 \subseteq \varepsilon} \times m^{\varepsilon_2 \subseteq \varepsilon}) = \tilde{\psi}_\varepsilon \circ (m^{\varepsilon_1 \subseteq \varepsilon} \times m^{\varepsilon_2 \subseteq \varepsilon})$	$\mathfrak{T}_{\varepsilon_1 \subseteq \varepsilon}$ translations commute with $\mathfrak{T}_{\varepsilon_2 \subseteq \varepsilon}$ translations (see tensor equations)	$\mathcal{T}_1 \to \mathcal{T}_1 \otimes \mathcal{T}_2 \leftarrow \mathcal{T}_2$, e.g., distinct global memory cells
Weak Swap	$\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2 \quad \Gamma, x_1 : A_1 \vdash_{\varepsilon'} N}{\text{(same as Swap)}}$	$\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2}{\begin{array}{l}\mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\, \mathbf{return}_\varepsilon x_1 = \\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\\ \mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\, \mathbf{return}_\varepsilon x_1\end{array}}$	$\psi_\varepsilon \circ (m^{\varepsilon_1} \times m^{\varepsilon_2}) \circ (\mathrm{id} \times \eta^{\varepsilon_2}_{\mathbb{1}}) = \tilde{\psi}_\varepsilon \circ (m^{\varepsilon_1} \times m^{\varepsilon_2}) \circ (\mathrm{id} \times \eta^{\varepsilon_2}_{\mathbb{1}})$	For all ε-terms $t = \mathfrak{T}_1(t')$, $s = \mathfrak{T}_2(s')$: $t(s(\mathbf{x}_1, \ldots, \mathbf{x}_1), \ldots,$ $s(\mathbf{x}_n, \ldots, \mathbf{x}_n)) =$ $s(t(\mathbf{x}_1, \ldots, \mathbf{x}_n), \ldots,$ $t(\mathbf{x}_1, \ldots, \mathbf{x}_n))$	when $\mathcal{T}_{\varepsilon_2}$ is affine, e.g.: read-only state and convex, upper and lower semilattices.
Isolated Swap	$\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2 \quad \Gamma \vdash_{\varepsilon'} N}{\text{(same as Swap)}}$	$\dfrac{\Gamma \vdash_{\varepsilon_1} M_1 : \mathbf{F}_{\varepsilon_1} A_1 \quad \Gamma \vdash_{\varepsilon_2} M_2 : \mathbf{F}_{\varepsilon_2} A_2}{\begin{array}{l}\mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\, \mathbf{return}_\varepsilon \star = \\ \mathbf{coerce} M_2 \mathbf{\ to\ } x_2 : A_2.\\ \mathbf{coerce} M_1 \mathbf{\ to\ } x_1 : A_1.\, \mathbf{return}_\varepsilon \star\end{array}}$	$\psi_\varepsilon \circ (m^{\varepsilon_1} \times m^{\varepsilon_2}) \circ (\eta^{\varepsilon_1}_{\mathbb{1}} \times \eta^{\varepsilon_2}_{\mathbb{1}}) = \tilde{\psi}_\varepsilon \circ (m^{\varepsilon_1} \times m^{\varepsilon_2}) \circ (\eta^{\varepsilon_1}_{\mathbb{1}} \times \eta^{\varepsilon_2}_{\mathbb{1}})$	For all ε-terms $t = \mathfrak{T}_1(t')$, $s = \mathfrak{T}_2(s')$: $t(s(\mathbf{x}, \ldots, \mathbf{x}), \ldots,$ $s(\mathbf{x}, \ldots, \mathbf{x})) =$ $s(t(\mathbf{x}, \ldots, \mathbf{x}), \ldots,$ $t(\mathbf{x}, \ldots, \mathbf{x}))$	when $\mathcal{T}\varepsilon_1$ is affine: read-only state and convex, upper and lower semilattices.
Unique	**B** $\dfrac{\Gamma \vdash_\varepsilon M_i : \mathbf{F}_\varepsilon \mathbf{0}, i = 1, 2}{M_1 = M_2}$	(same as utilitarian form)	$F_\varepsilon \mathbb{0} = \mathbb{0}, \mathbb{1}$	$\mathcal{T}_\varepsilon$ equates all ε-constants	all three state theories, all three semilattice theories, a single unparameterised exception, lifting
Pure Hoist	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \quad \Gamma, x : A \vdash_{\varepsilon'} N : \underline{B}}{\begin{array}{l}\mathbf{return}_\varepsilon \mathbf{\ thunk\ } (\mathbf{coerce} M \mathbf{\ to\ } x : A.\, N) \\ = M \mathbf{\ to\ } x : A.\, \mathbf{return}_\varepsilon \mathbf{\ thunk\ } N\end{array}}$	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{\begin{array}{l}\mathbf{return}_\varepsilon \mathbf{\ thunk\ } M = \\ M \mathbf{\ to\ } x : A.\, \mathbf{return}_\varepsilon \mathbf{\ thunk\ } \mathbf{return}_\varepsilon x\end{array}}$	**F** $L^\varepsilon \eta^\varepsilon_W = \eta^\varepsilon_{\lvert F_\varepsilon W \rvert}$	all ε-terms are equal to variables in $\mathcal{T}_\varepsilon$	the empty theory, inconsistent theories
Hoist	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A \quad \Gamma, x : A \vdash_{\varepsilon'} N : \underline{B}}{\begin{array}{l}M \mathbf{\ to\ } x : A.\\ \mathbf{return}_\varepsilon \mathbf{\ thunk\ } (\mathbf{coerce} M \mathbf{\ to\ } x : A.\, N) \\ = M \mathbf{\ to\ } x : A.\, \mathbf{return}_\varepsilon \mathbf{\ thunk\ } N\end{array}}$	**B F** $\dfrac{\Gamma \vdash_\varepsilon M : \mathbf{F}_\varepsilon A}{\begin{array}{l}M \mathbf{\ to\ } x : A.\\ \quad \mathbf{thunk\ return}_\varepsilon (x, \mathbf{\ thunk\ } M) = \\ M \mathbf{\ to\ } x : A.\\ \quad \mathbf{thunk\ return}_\varepsilon (x, \mathbf{\ thunk\ return}_\varepsilon x)\end{array}}$	**F** $L^\varepsilon \langle \eta^\varepsilon, \mathrm{id} \rangle = \mathrm{str}^\varepsilon \circ \delta$	all ε-terms are either a variable or independent of their variables via $\mathcal{T}_\varepsilon$	all theories containing only constants: lifting and exceptions

Figure 7. Abstract Optimisations

equivalence of the two forms and the abstract side condition in his setting.

The abstract conditions in Figure 7 use the following standard categorical notions. The diagonal function $\delta_W : W \to W \times W$ is given by $\delta(w) \stackrel{\text{def}}{=} \langle w, w\rangle$. Let $\mathcal{T}$ be an inequational theory. Given a continuous function $f : V \to W$, we can *lift* it to a homomorphism $(\eta \circ f)^\dagger : FV \to FW$. We denote the underlying map by $Lf : |FV| \to |FW|$. The *multiplication* $\mu : |F|FW|| \to |FW|$ is the underlying map of the homomorphism $\mathrm{id}^\dagger : F|FW| \to FW$. The *(left) strength*, $\mathrm{str} : V \times |FW| \to |F(V \times W)|$, is given by $\lambda\langle v, w\rangle.(\mathrm{curry}\,\eta)^\dagger(w)(v)$. Similarly we define the *right strength*, $\mathrm{str}' : |FV| \times W \to |F(V \times W)|$. The two *double strength* functions $\psi, \tilde{\psi} : |FV| \times |FW| \to |F(V \times W)|$ are defined by $\psi = \theta \circ L\,\mathrm{str}' \circ \mathrm{str}$ and by $\tilde{\psi} = \theta \circ L\,\mathrm{str} \circ \mathrm{str}'$.

Note how Copy corresponds to a global *idempotency* law, and how Swap corresponds to *commutativity*. Slight variations on these two laws yield the Weak Copy[3], and Weak and Isolated Swap optimisations, which are new in the formal methods optimisation literature. Also note the algebraic condition for Pure Hoist. It means that $\mathcal{T}_\varepsilon$ is either inconsistent, or the operations project one of their arguments without effect.

Note too the two hoisting optimisations and compare them to the structural optimisations dealing with effect operations (Figure 6). The simplicity of the latter over the former suggests that the complications arise from the process of *thunking* rather than abstracting over variables. This clean separation between thunks and abstraction supports our use of CBPV.

Most (but, unfortunately, not all) of the optimisations can be validated *operation-wise*:

Theorem 13. *For each of the optimisations Discard, Pure Hoist, and Hoist, the algebraic condition holds for a theory $\mathcal{T}$ iff for all operations $f : A$ it holds for the term $t = f(\lambda a.\mathbf{x}_a)$.*

Analogously, for each of the various Swap optimisations, the algebraic condition holds for $\mathcal{T}_1 \xrightarrow{\mathfrak{T}_1} \mathcal{T} \xleftarrow{\mathfrak{T}_2} \mathcal{T}_2$ iff for each $f^i :_{\Sigma_i} A_i$, $i = 1, 2$, it holds for the terms $t = f^1(\lambda a_1.\mathbf{x}_{a_1})$ and $s = f^2(\lambda a_2.\mathbf{x}_{a_2})$.

The theorem is proved by a straightforward induction over ε-terms, demonstrating the benefits of the algebraic characterisation. The theorem was used to obtain most of the examples column in Figure 7. We can also use this it to deduce optimisation validity modularly:

Corollary 14. *For each algebraic condition for Discard, Pure Hoist, and Hoist, if $\mathcal{T}^1$,$\mathcal{T}^2$ satisfy it, then so does $\mathcal{T}^1 + \mathcal{T}^2$.*

Analogously, for each algebraic condition for the various Swap optimisations, if $\mathcal{T}_1^i \xrightarrow{\mathfrak{T}_1^i} \mathcal{T} \xleftarrow{\mathfrak{T}_2^j} \mathcal{T}_2^j$, for all $i, j = 1, 2$, satisfy it, then so does $\mathcal{T}_1^1 + \mathcal{T}_1^2 \to \mathcal{T} \leftarrow \mathcal{T}_2^1 + \mathcal{T}_2^2$.

Optimisation validity is inherited by super-theories:

Proposition 15. *If $\mathcal{T}$ satisfies any of the algebraic conditions in Figure 7, and if $\mathcal{T}'$ is any theory with $\Sigma = \Sigma'$ and $(\leq) \subseteq (\leq')$, then $\mathcal{T}'$ satisfies the same condition.*

Analogously, if $\mathcal{T}_1 \xrightarrow{\mathfrak{T}_1} \mathcal{T}_3 \xleftarrow{\mathfrak{T}_2} \mathcal{T}_2$ satisfy the algebraic condition for any of the various Swap optimisations, and if, further, $\mathcal{T}_1' \xrightarrow{\mathfrak{T}_1'} \mathcal{T}_3' \xleftarrow{\mathfrak{T}_2'} \mathcal{T}_2'$ are such that $\Sigma_i = \Sigma_i'$, $\mathcal{T}_i \subseteq \mathcal{T}_i'$ and for each $f \in \Sigma_i$, $\mathfrak{T}_i(f) = \mathfrak{T}_i'(f)$, then $\mathcal{T}_1' \xrightarrow{\mathfrak{T}_1'} \mathcal{T}_3' \xleftarrow{\mathfrak{T}_2'} \mathcal{T}_2'$ satisfy the same algebraic condition.

For example, the semilattice theories are *commutative* — the identity translations commute. Thus given two locations ℓ, ℓ' in memory with associated operations $\mathtt{lookup}^\ell$ and $\mathtt{update}_v^{\ell'}$, the trivial translations of $\mathcal{T}_{\mathrm{Env}(\mathbb{V})} \otimes \mathcal{T}_{\mathrm{ND}}$, $\mathcal{T}_{\mathrm{ND}} \otimes \mathcal{T}_{\mathrm{OW}(\mathbb{V})}$ into the theory with two distinct memory cells and non-determinism, $\mathcal{T}_{\mathrm{GS}(\mathbb{V})} \otimes \mathcal{T}_{\mathrm{ND}} \otimes \mathcal{T}_{\mathrm{GS}(\mathbb{V})}$, commute.

Note that each algebraic condition for Discard and Copy implies that of Weak Copy. Similarly, the algebraic condition for Swap implies that of Weak Swap, which implies that of Isolated Swap.

The notable exceptions to Theorem 13 are the Copy, Weak Copy and Unique optimisations. For example, in the global state theory $\mathcal{T}_{\mathrm{GS}(2)}$ both `lookup` and `update` are idempotent, but the term $\mathtt{lookup}(\lambda a.\mathtt{update}_0\mathbf{x}_a)$ is not idempotent. However, in practice, we can use equational reasoning to establish them. For example, the write-only state theory is relevant, because any term can be rewritten to a single operation $\mathtt{update}_v\mathbf{x}$. As these *are* idempotent, all terms satisfy the idempotency law.

We can use similar equational arguments to establish these idempotency laws for theory combinations:

Theorem 16. *Let $\mathcal{T}$, $\mathcal{T}'$ be two relevant algebraic theories.*

- *If all Σ operations have arity $\mathbb{0}$, then $\mathcal{T} + \mathcal{T}'$ is relevant.*
- *If all Σ operations have arity $\mathbb{1}$, then $\mathcal{T} \otimes \mathcal{T}'$ is relevant.*
- *If $\mathcal{T}$ is also affine, then $\mathcal{T} \otimes \mathcal{T}'$ is relevant.*

The same is true replacing "relevant" by the Weak Copy characterisation.

In practice, the relevant theories with which one usually tensors are the lifting theory or with the read-only and write-only state theories, and they all satisfy the conditions of this theorem.

Similarly, we can deduce the condition for the unique optimisation in the following case:

Proposition 17. *Let $\mathcal{T}$ be an inequational theory. If every nullary operation in Σ commutes with every Σ-operation, then $\mathcal{T}$ satisfies the algebraic condition for the* Unique *optimisation.*

5. Example Language

To demonstrate our results, we consider a non-trivial language. Assume the memory has been partitioned into a finite set of disjoint regions [27] $\mathbf{Reg}$. The set $\mathbf{Reg}$ is partitioned into three subsets: read-only regions $\mathbf{Reg_{RO}}$; write-only regions $\mathbf{Reg_{WO}}$; and read-write regions $\mathbf{Reg_{RW}}$. We denote by $\mathbf{Reg_R}$ the read-able regions $\mathbf{Reg_{RO}} \cup \mathbf{Reg_{RW}}$, and by $\mathbf{Reg_W}$ the write-able regions $\mathbf{Reg_{WO}} \cup \mathbf{Reg_{RW}}$.

The MAIL signature in question Σ is given as follows. The basic types $\mathbf{Bsc}$ are $\mathbf{Char}$, $\mathbf{Word}$, $\mathbf{Str}$ and $\mathbf{Loc}$. The effect operations $|\sigma|$ and their arities are: $\Omega : \mathbf{0}$, as required; $\mathtt{input} : \mathbf{Char}$ for terminal input; $\mathtt{output} : \mathbf{1} \to \mathbf{Char}$ for terminal output; $\mathtt{raise} : \mathbf{0}$ for causing an exception; $\mathtt{throw} : \mathbf{0} \to \mathbf{Str}$ for causing an exception with an error message; $\mathtt{rollback} : \mathbf{0}$ for causing a rollback exception; $\mathtt{abort} : \mathbf{0} \to \mathbf{Str}$ for causing a rollback exception with an error message; for all write-able regions $\rho \in \mathbf{Reg_R}$, $\mathtt{lookup}^\rho : \mathbf{Word} \to \mathbf{Loc}$, note that each region consists of $\mathbf{Loc}$-many locations; for all read-able regions ρ in $\mathbf{Reg_W}$, $\mathtt{update}^\rho : \mathbf{1} \to \mathbf{Loc} \times \mathbf{Word}$; and finally, $\vee : \mathbf{2}$ for non-deterministic choice. The effect sets $\mathcal{E}$ are given by all (finite) effect subsets $\mathcal{P}(|\sigma|)$. Finally, the built-in constants κ we choose are: $'\mathtt{c}'$: $\mathbf{Char}$ for each ASCII character c; $\mathtt{n}$ for each 64-bit number n; $''\mathtt{s}''$ for each character string s; $\mathtt{l}$ for each 64-bit memory address ℓ. The resulting Σ is indeed simple.

The Alg-CBPV signature $\Sigma^\natural$ is $\left\langle \langle \mathbf{Bsc}, |\sigma|, \Omega, \kappa\rangle, \mathrm{ar}, A_- \right\rangle$. The chosen $\Sigma^\natural$-Alg-CBPV model $\mathcal{M}$ interprets the basic types as follows: $[\![\mathbf{Char}]\!]$ is 2^8, the usual ASCII encoding; $[\![\mathbf{Word}]\!]$ is 2^{64}, which we also denote by $\mathbb{V}$; $[\![\mathbf{Str}]\!]$ is $[\![\mathbf{Char}]\!]^*$, the set of finite $[\![\mathbf{Char}]\!]$ sequences; and $[\![\mathbf{Loc}]\!]$ is 2^{64}. The theory $\mathcal{T}$ is given

[3] Paul B. Levy, private communication.

by [19]:

$$\mathrm{Th}^{\{\texttt{raise},\texttt{throw}\}}\emptyset + (\mathcal{T}_{\mathrm{Env}(\mathbb{V})}^{\bigotimes_{\rho\in\mathbf{Reg_{RO}},\ell}} \otimes \mathcal{T}_{\mathrm{OW}(\mathbb{V})}^{\bigotimes_{\rho\in\mathbf{Reg_{WO}},\ell}} \otimes \mathcal{T}_{\mathrm{GS}(\mathbb{V})}^{\bigotimes_{\rho\in\mathbf{Reg_{RW}},\ell}} \otimes (\mathrm{Th}^{\{\texttt{rollback},\texttt{abort}\}}\emptyset + \mathrm{Th}^{\{\texttt{input}\}}\emptyset + \mathrm{Th}^{\{\texttt{output}\}}\emptyset + (\mathcal{T}_{\mathrm{ND}} \otimes \mathcal{T}_{\Omega})))$$

where the `lookup` operations in the signature in the $\langle\rho,\ell\rangle$-th component of the folded tensor $\mathcal{T}_{\mathrm{Env}(\mathbb{V})}^{\bigotimes_{\rho\in\mathbf{Reg_{RO}}}}$ are tagged with ρ and ℓ, i.e. $\texttt{lookup}_\ell^\rho$, and similarly for the other folded tensors. Finally, the constants are given the obvious interpretations.

By Theorem 12, we have an axiomatic restriction model $\mathcal{M}_\sharp$ for the natural modular axiomatisation of $\mathcal{T}$. Validating optimisations in this model yields valid optimisations for $\Sigma^\natural$-Alg-CBPV.

For each optimisation o in {Discard, Pure Hoist, Hoist}, we define a set $\zeta^o \subseteq |\sigma|$ of the operations that satisfy its algebraic condition:

$$\zeta^{\text{Discard}} \stackrel{\text{def}}{=} \{\vee, \texttt{lookup}^\rho \mid \rho \in \mathbf{Reg_R}\} \qquad \zeta^{\text{Pure Hoist}} \stackrel{\text{def}}{=} \emptyset$$

$$\zeta^{\text{Hoist}} \stackrel{\text{def}}{=} \{\texttt{raise}, \texttt{throw}, \texttt{abort}, \texttt{rollback}, \Omega\}$$

By Theorem 13 we obtain the following condition:

Proposition 18. *For each optimisation o in {Discard, Pure Hoist, Hoist}, if $\varepsilon \subseteq \zeta^o$ then o is valid in the theory $\mathcal{T}_\varepsilon$ of $\mathcal{M}_\sharp$.*

Analogously, for each o in {Swap, Weak Swap (WSwap), isolated swap (ISwap)} and for each $\mathrm{op} \in |\sigma|$ define the set $\zeta^o(\mathrm{op})$ of effect operations that o-commute with op. For Weak Swap, op replaces t in the algebraic condition. Note that because Swap implies Weak Swap, which implies Isolated Swap, we have $\zeta^{\text{Swap}}(\mathrm{op}) \subseteq \zeta^{\text{WSwap}}(\mathrm{op}) \subseteq \zeta^{\text{ISwap}}(\mathrm{op})$. These sets are given in Figure 8.

From Theorem 13 we can deduce the validity of swapping:

Proposition 19. *Let o be one of the various Swap optimisations. Let ε_1, ε_2 be two effect sets. If $\varepsilon_2 \subseteq \bigcap_{\mathrm{op}\in\varepsilon_1} \zeta^o(\mathrm{op})$ then the optimisation o is valid for $\mathcal{T}_{\varepsilon_1} \to \mathcal{T}_{\varepsilon_1\cup\varepsilon_2} \leftarrow \mathcal{T}_{\varepsilon_2}$.*

We should note that despite our brute force method of examining 200 pairs of effect operations, we are still exponentially better off than trying to exhaust the space of 2^{20} possible pairs $\varepsilon_1, \varepsilon_2$. It is worthwhile to wonder whether mechanised assistance is possible. It would also be good to be able to decide exactly which of the 2^{20} possible optimisations is valid in the benchmark model.

Theorem 20. *Let $\varepsilon \subseteq |\sigma|$. If $\texttt{input}, \texttt{update}, \vee \notin \varepsilon$ and for all ρ in $\mathbf{Reg_{RW}}$, $\{\texttt{lookup}^\rho, \texttt{update}^\rho\} \not\subseteq \varepsilon$, then $\mathcal{T}_\varepsilon$ validates the Copy optimisation.*

Note that the premise of the theorem and the structure of the axiomatic restriction model guarantees that $\mathcal{T}_\varepsilon$ is of the form:

$$\mathrm{Th}^{\{\texttt{raise},\texttt{throw}\}}\emptyset + (\mathcal{T}_{\mathrm{Env}(\mathbb{V})}^{\otimes} \otimes \mathcal{T}_{\mathrm{OW}(\mathbb{V})}^{\otimes} \otimes (\mathrm{Th}^{\{\texttt{rollback},\texttt{abort}\}}\emptyset + \mathcal{T}_\Omega))$$

where each of the theories may be omitted. Note that all of them are relevant. Repeated application of Theorem 16 shows the combination's relevance.

The optimisation Weak Copy is treated similarly:

Theorem 21. *Let $\varepsilon \subseteq |\sigma|$. If $\texttt{input}, \texttt{output} \notin \varepsilon$ and for all ρ in $\mathbf{Reg_{RW}}$, $\{\texttt{lookup}^\rho, \texttt{update}^\rho\} \not\subseteq \varepsilon$, then $\mathcal{T}_\varepsilon$ validates the Weak Copy optimisation.*

To conclude our example, we treat the Unique optimisation using Proposition 17:

Theorem 22. *Let $\varepsilon \subseteq |\sigma|$. If $\varepsilon \cap \{\Omega, \texttt{raise}, \texttt{rollback}\} = \{\mathrm{op}\}$ and $\varepsilon \subseteq \zeta^{\text{Swap}}(\mathrm{op})$, or if the intersection is empty, then $\mathcal{T}_\varepsilon$ validates the Unique optimisation.*

6. Related Work

Filinski used $\mathrm{M}^3\mathrm{L}$ [10, 11] to investigate *effect reification* [12]. Its earlier call-by-value versions had a denotational semantics and inspired our CBPV MAIL. Marino and Millstein [28] developed a technique to derive type and effect frameworks based on the notions of *redexes* and *contexts* [48]. These frameworks are parameterised by two functions `check` and `adjust`. They applied their technique to a fixed ML-like, call-by-value language with recursion, exceptions and higher-order dynamic allocation, capturing many existing effect systems, mostly more general than the Gifford-style. They gave a mechanically checked soundness proof of their effect system, with respect to a suitable operational semantics.

Atkey applied *permission-parameterised monads* to the semantics of type and effect systems [2, 3] more general than Gifford-style ones. His parameterised monads can be generated by a notion of equational theory in which the signature assigns input and output permissions to operations, thereby connecting operations and effects (qua permissions). He also established relations to standard monadic semantics analogous to Theorems 9 and 12.

7. Conclusions and Further Work

We have given a theory of Gifford-style effect systems, generalising and unifying existing work. The algebraic approach provides a valuable general point of view, resulting in: the connection between effect operations and effect sets; the conservative and the axiomatic restriction models; the relation between effect-annotated semantics and standard algebraic semantics; optimisation classification and discovery of new optimisations; criteria for the validity of abstract optimisations; and methods to derive the validity of optimisations for combinations of effects modularly. Rather than having to proceed from case to case by analogy, we hope that the generality of our approach will provide a first step on the way to obtaining a scientifically-based engineering methodology.

The use of CBPV enabled a systematic account that highlights the interplay between programming constructs and effects. Categorical language greatly helped the organisation of this work. It was also crucial in seeing the connection with Führmann's work, and unifying it with Benton et al.'s. Finally, the example language demonstrated the ease of application of the theory to a language equipped with an algebraic semantics. Note too that model constructions are not ad hoc: they come for *free* given only the algebraic theory of the effects at hand.

Further work abounds. It would be good to resolve Conjecture 10, and to establish means to decide the validity of optimisations in the benchmark model. It is also important to include *effect handlers* [37, 40] such as exception handlers and rollback in the language. Pretnar and Plotkin describe a β rule for handlers, and a rule for the interaction of algebraic operations and handlers. These rules have many consequences in their logic, and will surely be valid in our setting. There may also be useful abstract optimisations involving handlers, possibly unifying existing accounts [4].

Effect reconstruction is of immediate importance. It should be possible to derive general algorithms for type and effect annotation. Our semantics can then be used to give semantics to such programs. Levy's translations of call-by-value and call-by-name into CBPV could then be used to deduce general effect systems for these paradigms, generalising and unifying existing work.

Notions of locality, particularly local state, are very important. It may be possible to make use of work on the algebraic treatment of locality, e.g., [29, 34, 42], to obtain a more general optimisation theory. This should enable the work of Benton et al. on dynamic allocation [8] to be incorporated. (Incorporating higher-order store [6] would require solving recursive domain equations [1, 25].)

op	ζ^{Swap}	$\zeta^{\mathrm{WSwap}} \setminus \zeta^{\mathrm{Swap}}$	$\zeta^{\mathrm{ISwap}} \setminus \zeta^{\mathrm{WSwap}}$	$\lvert\sigma\rvert \setminus \zeta^{\mathrm{ISwap}}$
Ω	Ω, $\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$, $\vee$	$\emptyset$	$\emptyset$	$\mathtt{input}$, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$
$\mathtt{input}$	$\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$	$\vee$	$\mathtt{input}$	Ω, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$
$\mathtt{output}$	$\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$	$\vee$	$\mathtt{output}$	Ω, $\mathtt{input}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$
$\mathtt{raise}$	$\mathtt{lookup}^{\rho}$, $\mathtt{raise}$, $\vee$	$\emptyset$	$\emptyset$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$, $\mathtt{update}^{\rho}$
$\mathtt{throw}$	$\mathtt{lookup}^{\rho}$	$\vee$	$\emptyset$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$, $\mathtt{update}^{\rho}$
$\mathtt{rollback}$	$\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$, $\mathtt{rollback}$, $\vee$	$\emptyset$	$\emptyset$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{abort}$
$\mathtt{abort}$	$\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$	$\vee$	$\emptyset$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$
$\mathtt{lookup}^{\rho 0}$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{abort}$, $\mathtt{rollback}$, $\mathtt{abort}$, $\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho \neq \rho 0}$, $\vee$	$\emptyset$	$\mathtt{update}^{\rho 0}$	$\emptyset$
$\mathtt{update}^{\rho 0}$	Ω, $\mathtt{input}$, $\mathtt{output}$, $\mathtt{rollback}$, $\mathtt{abort}$, $\mathtt{lookup}^{\rho \neq \rho 0}$, $\mathtt{update}^{\rho \neq \rho 0}$, $\vee$	$\mathtt{lookup}^{\rho 0}$	$\emptyset$	$\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{update}^{\rho 0}$
$\vee$	Ω, $\mathtt{raise}$, $\mathtt{throw}$, $\mathtt{rollback}$, $\mathtt{abort}$, $\mathtt{lookup}^{\rho}$, $\mathtt{update}^{\rho}$, $\vee$	$\emptyset$	$\mathtt{input}$, $\mathtt{output}$	$\emptyset$

Figure 8. Swap Sets

In a different direction, the distributive combination of theories [17] (used for combining ordinary and probabilistic computation) should be investigated. Also, our theory should be extended to include non-algebraic effects, such as continuations [20].

Our arities are required to be *ground*, *finite* and *discrete*. The finiteness and discreteness conditions can be relaxed by using, respectively, *infinitary (in)equational theories* [39]; and *enriched Lawvere theories* [38]. Our treatment of the finite discrete case exploited inequational logic; in the infinitary, enriched case, corresponding proof theories are needed [32]. Generalising arities to non-ground types may involve recursive domain equations, as in the hypothesised treatment of higher-order store.

The logic we used for our optimisations is a simple equational logic — we have only considered equations between terms. It seems straightforward to devise a richer effect-dependent counterpart to Plotkin and Pretnar's logic [36, 40].

Additional work is needed to distill the general account into methodologies. In particular, it is desirable to reproduce the exponential improvement in operation-wise validation for the other optimisations. Also, machine assistance and model-checking tools could alleviate the repetitive burden of operation-wise validation.

We conjecture that Benton et al.'s logical relations method can be related to the conservative restriction model. Our approach is denotational; it would be interesting to devise an operational account [22, 33]. A precise relationship between our theory and Atkey's is in order, particularly with his parameterised monads. Finally it would be good to go beyond Gifford-style, e.g., allowing effect traces, and to account for parallelism, as in Gifford's work [27].

Acknowledgments

We thank Robert Atkey, Andrej Bauer, Nick Benton, Jeff Egger, Ben Kavanagh, Neelakantan Krishnaswami, Paul Levy, Sam Lindley, Matija Pretnar, Alex Simpson, Ross Tate, Phil Wadler, and the anonymous reviewers for helpful discussions and comments. This research was supported by a Scottish Informatics and Computer Science Alliance studentship and a Royal Society Wolfson Award.

References

[1] S. Abramsky and A. Jung. Domain theory. *Handbook of Logic in Computer Science*, 3:1–168, 1994.

[2] R. Atkey. Algebras for Parameterised Monads. *Proc. 3rd CALCO*, 3–17, 2009.

[3] R. Atkey. Parameterised notions of computation. *Journal of Functional Programming*, 19:335–376, 2009.

[4] N. Benton and P. Buchlovsky. Semantics of an effect analysis for exceptions. *Proc. 3rd TLDI*, 15–26, 2007.

[5] N. Benton and A. Kennedy. Monads, effects and transformations. *Electronic Notes in Theoretical Computer Science*, 26:3–20, 1999.

[6] N. Benton et al. Relational semantics for effect-based program transformations: higher-order store. *Proc. 11th PPDP*, 301–312, 2009.

[7] N. Benton et al. Reading, writing and relations. *Proc. 4th APLAS*, 114–130, 2006.

[8] N. Benton et al. Relational semantics for effect-based program transformations with dynamic allocation. *Proc. 9th PPDP*, 87–96, 2007.

[9] B. Eckmann and P. J. Hilton. Group-like structures in general categories I: multiplications and comultiplications *Mathematische Annalen*, 145(3):227–255, 1962.

[10] A. Filinski. Representing layered monads. *Proc. 26th POPL*, 175–188, 1999.

[11] A. Filinski. On the relations between monadic semantics. *Theoretical Computer Science*, 375(1-3):41–75, 2007.

[12] A. Filinski. Monads in action. *Proc. 37th POPL*, 483–494, 2010.

[13] C. Führmann. The structure of call-by-value. PhD thesis, University of Edinburgh, 2000.

[14] C. Führmann. Varieties of effects. *Proc. 5th FoSSaCS*, 144–158, 2002.

[15] A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12:435–483, 2004.

[16] M. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. *Proc. 8th MFCS*, 108–120, 1979.

[17] M. Hyland and J. Power. Discrete Lawvere theories and computational effects. *Theoretical Computer Science*, 366(1-2):144–162, 2006.

[18] M. Hyland and J. Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, 2007.

[19] M. Hyland, G. D. Plotkin, and J. Power. Combining effects: sum and tensor. *Theoretical Computer Science*, 357(1-3):70–99, 2006.

[20] M. Hyland et al. Combining algebraic effects with continuations. *Theoretical Computer Science*, 375(1-3):20–40, 2007.

[21] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69(1):73–106, 1994.

[22] P. Johann et al. A generic operational metatheory for algebraic effects. *Proc. 25th LICS*, 209–218, 2010.

[23] R. B. Kieburtz. Taming effects with monadic typing. *Proc. 3rd ICFP*, 51–62, 1998.

[24] A. Kock. Bilinearity and cartesian closed monads. *Mathematica Scandinavia*, 29:161–174, 1971.

[25] P. B. Levy. Possible world semantics for general storage in call-by-value. *Proc. 16th CSL*, 232–246, 2002.

[26] P. B. Levy. Call-by-Push-Value: a functional/imperative synthesis. *Call-by-Push-Value*, Springer, 2004.

[27] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. *Proc. 15th POPL*, 47–57, 1988.

[28] D. Marino and T. Millstein. A generic type-and-effect system. *Proc. 4th TLDI*, 39–50, 2009.

[29] P.-A. Melliès. Segal condition meets computational effects. *Proc. 25th LICS*, 150–159, 2010.

[30] J. C. Mitchell. Type systems for programming languages. *Handbook of Theoretical Computer Science*, B:365–458, 1990.

[31] G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5:452–487, 1976.

[32] G. D. Plotkin. Some varieties of equational logic. *Lecture Notes in Computer Science*, 4060:150–156, 2006.

[33] G. D. Plotkin and J. Power. Adequacy for algebraic effects. *Proc. 4th FoSSaCS*, 1–24, 2001.

[34] G. D. Plotkin and J. Power. Notions of computation determine monads. *Lecture Notes in Computer Science*, 2303:342–356, 2002.

[35] G. D. Plotkin and J. Power. Computational effects and operations: an overview. *Electronic Notes in Theoretical Computer Science*, 73:149–163, 2004.

[36] G. D. Plotkin and M. Pretnar. A logic for algebraic effects. *Proc. 23rd LICS*, 118–129, 2008.

[37] G. D. Plotkin and M. Pretnar. Handlers of algebraic effects. *Proc. 19th ESOP*, 80–94, 2009.

[38] J. Power. Enriched Lawvere theories. *Theory and Applications of Categories*, 6:83–93, 2000.

[39] J. Power. Countable lawvere theories and computational effects. *Electronic Notes in Theoretical Computer Science*, 161:59–71, 2006.

[40] M. Pretnar. The logic and handling of algebraic effects. PhD thesis, University of Edinburgh, 2009.

[41] J. Reynolds. On the relation between direct and continuation semantics. *Proc. 2nd ICALP*, 141–156, 1974.

[42] S. Staton. Two cotensors in one: presentations of algebraic theories for local state and fresh names. *Electronic Notes in Theoretical Computer Science*, 249:471–490, 2009.

[43] T. Coquand et al. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, 1991.

[44] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

[45] A. P. Tolmach. Optimizing ML using a hierarchy of monadic types. *Proc. 2nd TIC*, 97–115, 1998.

[46] P. Wadler. The Marriage of Effects and Monads. *Proc. 3rd ICFP*, 63–74, 1998.

[47] P. Wadler and P. Thiemann. The marriage of effects and monads. *ACM Trans. Comp. Log.*, 4(1):1–32, 2003.

[48] A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.