Annual Progress Report
Year II

Ohad Kammar

February 8, 2011

This document reports my progress with respect to the research proposal presented last year. A preprint submitted to LICS'11 is appended.

1 Current Work

Last year I proposed two different research topics: algebraic effect type systems and a logic for access control. The aim was to focus on the more fruitful topic. As algebraic effect type systems proved so, I focused on them. Recall that effect type systems consist of *type and effect* judgements $\Gamma \vdash M : A ! \varepsilon$, where ε is a finite set of effects that may be caused by M.

The research goals I set last year for effect type systems were: validate Wadler's conjecture regarding the existence of a coherent semantics, in which computations with effect set ε are interpreted via monads T_ε; devise general effect reconstruction algorithms, using subeffecting; devise effect reconstruction algorithms, using effect polymorphism; devise operational semantics for the type and effect system and analyse it; treat effect handlers within the type and effect system; formulate the results in terms of (general) monads; compare our work with Atkey's work on *permissions*; and analyse the proof theory of the effect type system, via Curry-Howard.

1.1 LICS'11 Submission

We have submitted a preprint to LICS'11, appended below. The preprint: validates Wadler's conjecture; shows that, over **Set**, for the theories that arise in practice, the monads T_ε are given by the conservative restriction of the Lawvere theory to the signature corresponding to ε; treats effect handlers, and generalises them to arbitrary, not necessarily algebraic, effects; and formulates the semantics abstractly, in non-algebraic terminology. When comparing the semantic model with Atkey's work on *permissions*, it seems that permissions are more general. However, unlike us, Atkey assumes the parameterised monads are predetermined, and does not offer a method to derive them.

The preprint applies our semantic results to establish modular proofs for correctness preserving transformations. Our work sheds light on the origins of

these optimisations, and yields a taxonomy of optimisations: structural, algebraic and abstract. Structural transformations stem from the underlying categorical structure of the models, such as β-laws. Algebraic transformations, in models that arise from algebraic theories, correspond to equations in the algebraic theory. Finally, abstract optimisations arise of abstract property of the underlying monad for the effect.

When the overall monad of the language is given by known combinations of simpler theories, the verification of these transformations can be localised to the conservative restrictions of the component theories. In order to do so, we defined a generalised state theory whose conservative restrictions give a unified account of the monad transformers arising out of the tensor product of theories: reader, writer (monoid monad), and state.

A key idea in our submission is that the modular composition of theories gives rise to a modular decomposition of their conservative restrictions: if $\bigcirc$ is either $+$ or $\otimes$, then restricting the combined theory $L_1 \bigcirc L_2$ to the combined effect set $\varepsilon_1 \cup \varepsilon_2$, with ε_i in the corresponding theory L_i, amounts to combining the restricted theories: $L_1^{\varepsilon_1} \bigcirc L_2^{\varepsilon_2}$. In other words, conservative restriction commutes with combination. In the preprint, we have established this result for the cases that arise in practice, over **Set**. A treatment of ω**CPO** is outlined, but deferred for future work due to complications with factorisation systems, which are the categorical counterparts to conservative restrictions.

1.2 Unpublished Results

I have also established the above-mentioned modularity result for an arbitrary sum of two **Set**-enriched finitary Lawvere theories, using techniques from term rewriting systems: unfailing Knuth-Bendix completion and the recursive path ordering. I have an outline for a proof for infinitary theories.

The corresponding tensor conservativeness result is famously false, by the Eckmann-Hilton argument. We have conjectured its validity under some conditions: If at least one of the theories have no constants, then tensoring conservative restrictions yields a conservative restriction. I have failed to mimic the proof for sum in the tensor case, and found a counter-example to this conjecture.

Another unpublished result, whose proof again uses term rewriting, is that ordinary Lawvere theories have a unique epi-mono factorisation system. Unfortunately, ω**CPO** does not share this property. This difference might explain the difficulty we encountered in choosing the correct factorisation system for ω**CPO** Lawvere theories.

1.3 Intended Submission for ICFP'11

Currently, I am working on general effect reconstruction algorithms for subeffecting and for effect polymorphism. I intend to submit the results to ICFP'11. This work could be used to instrument a type system for the currently untyped programming language *Eff*, being developed by Bauer and Pretnar [`http://math.andrej.com/2010/09/27/programming-with-effects-ii-introducing-eff/`].

2 Future Work

As the semantic account proved rich enough, we have decided to abandon research into the operational semantics and Curry-Howard proof theory of our proposed language.

While working on the LICS preprint, we realised it would be better to split it into two journal articles. The first article would give the underlying mathematics. This article would supply the mathematical foundation for our work on type and effect analysis: unique epi-mono factorisation of Lawvere theories over **Set**, justification for the factorisation system in ω**CPO**, modularity results for sum and tensor, and the decomposition of the known basic theories into their conservative restrictions.

The second article would synthesise semantics for type and effect systems using the foundation of the first article. In particular, we would prove the equivalence of our type and effect semantics with standard Call-by-Push-Value using logical relations. We would also further analyse effect dependent program transformations. Finally, we would deal thoroughly with the ω**CPO** portion, in order to accommodate recursion.

If I could manage to prove a suitable result for the tensor of arbitrary theories, under sufficiently general conditions, I could write an article applying rewriting techniques to Lawvere theories: the unique epi-mono factorisation of Lawvere theories of **Set**, conservativeness of sum, and conservativeness of tensor. The journal version of this paper would try to generalise these results to: a) inifinitary cases, and b) enriched cases. However, none of this is on the critical path towards a thesis, and unique factorisation can be put in the journal article as before, and the sum conservativeness can simply be included in the thesis, or await further work.

I was invited to present our work at the European Workshop on Computational Effects in Ljubljana. I intend to extend my visit to Slovenia in order to investigate the possible application of our work to Bauer and Pretnar's Eff programming language, mentioned above.

Should time permit, I intend to apply our system to concurrency, where effects are more interesting.

3 Proposed Schedule

The Ljubljana workshop, academic visit and ICFP deadline will take place during March. If the LICS preprint is rejected, we plan to resubmit it for MFPS (if it is rejected early enough), or POPL. I then plan to work on the two journal papers. In July, thesis writing will commence, in parallel with the journal submissions. I aim for a January thesis submission.

Modular Semantics for General Effect-Dependent Optimisations

Ohad Kammar <ohad.kammar@ed.ac.uk>
and Gordon D. Plotkin <gdp@ed.ac.uk>
Laboratory for Foundations of Computer Science
University of Edinburgh, Scotland

***Abstract*—We propose a semantic foundation for optimisations based on type and effect analysis. We present a multiple-effect CBPV-based calculus whose denotational semantics is based on an effect-indexed structure of adjunctions. When the underlying set of effects is specified by an algebraic theory, we take effects to be given by sets of operations. The required adjunction structure can then be obtained via a uniform process of conservative restriction of the theory. The calculus and its semantics is then extended by a straightforward generalisation of Pretnar and Plotkin's effect handlers to arbitrary, non-algebraic, inter-effect handlers.**

The modular composition of effects then boils down to extension conservativeness, and we show that some common ways to compose arbitrary effects by sums and tensors are indeed conservative. In particular we obtain conservative extension results for both the sum of a monad and a free monad, and also for a unification of three common instances of the tensor of theories that is obtained using monoid actions. This unification includes the usual reader, state and writer monads and monad transformers.

We use our calculus and its semantics to provide general effect-dependent optimisations. We exemplify the machinery developed by deriving a language with state, IO and exceptions, and their handlers. Many of the already known effect-dependent optimisations are then particular cases of our general ones. We have thus demonstrated the possibility of a general theory of effect optimisations based on the algebraic theory of effects.

I. Introduction

Type and effect analysis [1] in its simplest form assigns a type and an effect set to each term of a programming language. The type describes the various values the term may evaluate to. The effect set describes the various computational effects, such as memory assignments, exception raising, I/O, etc., the term may cause during its computation.

For example, consider the following term:

$$M \coloneqq \texttt{if true then x} := \texttt{1 else x} := \texttt{deref(y)}$$

Its type is the unit type, as its sole purpose is for the computational effects on the memory store. The effect set it has is $\{\texttt{update}, \texttt{lookup}\}$, as it might cause either a memory look-up or an update. Many type and effect systems assert this information as a type and effect judgement:

$$\texttt{x}:\mathbf{Loc}, \texttt{y}:\mathbf{Loc} \vdash M : \texttt{unit} ! \{\texttt{update}, \texttt{lookup}\}$$

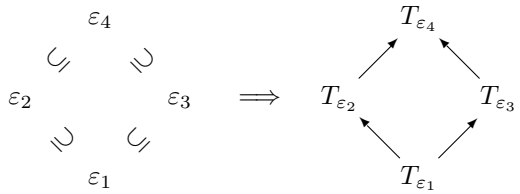
The information gathered by such an effect analysis can be used to guarantee correctness of the implementation [2], to prove authenticity properties [3] or to aid resource management [4]. One particular use of effect systems allows code transformations for optimisation. It is well-known that purely functional code can be executed out of order:

$$\texttt{let x be } M_1 \texttt{ in let y be } M_2 \texttt{ in } N = \texttt{let y be } M_2 \texttt{ in let x be } M_1 \texttt{ in } N$$

Such reordering can occur more generally when the terms M_1 and M_2 have non-interfering effects. In a sequence of papers, Benton et al. [5]–[9] prove such optimisations for increasingly complex overall sets of effects. As any change in the language requires a complete reformulation of the semantics and proof used, this approach is not modular.

One way to obtain such modularity is to study effect systems in general. However, despite their usefulness, there has been little work on a general theory for type and effect systems. Marino and Millstein [10] made the only such attempt. Unfortunately, their system is purely syntactic and has no ready connection to semantics.

In earlier work, Wadler [11] made an important connection with the monadic approach to computational effects. He established a translation of judgements of the form $\Gamma \vdash M : A ! \varepsilon$ in a region analysis calculus, to judgements of the form $\Gamma' \vdash M' : T_\varepsilon A$. These latter judgements formed part of a multi-monadic calculus in which T_ε is a monad corresponding to the effects in ε. Wadler gave an operational semantics, and conjectured the existence of a corresponding denotational semantics in which these syntactic monads have semantic counterparts. Thus the hierarchy of effect sets and inclusions should induce a hierarchy of monads and monad morphisms:



A linear hierarchy in this spirit was given in a special case by Tolmach [12]. Kieburtz [13] gave a more elaborate hierarchy, although not in the same semantic setting, and still short of full correspondence to the effect set hierarchy.

To summarise, the current body of work lacks a unifying general theory for type and effect systems with a modular semantic foundation.

Plotkin and Power's algebraic theory of effects [14], [15] proposes a semantic foundation for computational effects that focuses on the operations causing the effects. These operations form a single-sorted signature Σ, which posses a natural equational theory. Using tools from universal algebra and category theory, the usual monadic semantics for computational effects can be obtained. The shift in focus from monads to effect operations allows a modular treatment [16].

This view of effects provides the missing link needed to develop a general theory of type and effect systems. We show that when our semantics is given by an equational theory, under the right choice of the signature Σ, we can take the effect sets from type and effect analysis to be subsets $\varepsilon \subseteq \Sigma$.

Then, for each such ε, we consider the *conservative restriction* of the equational theory to the sub-signature induced by ε. That is, terms are given by the operation symbols of ε, and the equations are precisely those valid in the original theory. Our main claim is that this restriction gives the right monad T_ε for ε-computations.

In order to back this claim we first need a choice of language which deals with multiple effects, the Multi-Adjunctive Intermediate Language. The models for our language are a hierarchy of adjunctions between values and ε-computations, with monad morphisms that relate between them. We focus on a particular model, namely the conservative restriction model. We devise a set of tools to derive the explicit structure of this model. These tools allow the modular treatment of effects. Using our semantics, we then unify and generalise most of Benton et al.'s work on effect dependent optimisations. Finally, we exemplify our result for a concrete language with state with multiple regions of varying types, I/O and exceptions, both rollback and regular.

This paper makes the following contributions: it proposes a multi-adjunctive intermediate language for specifying and reasoning about several different effects simultaneously and explicitly; it generalises Pretnar and Plotkin's effect handlers [17] to the inter-effect, non-algebraic setting; it presents a semantic description of types and effects; it provides tools for modular decomposition of theories into conservative restrictions; it gives a unified account for the reader, writer and global state monads and monad transformers; it suggests a unifying framework and a taxonomy for effect dependent optimisations; it supplies modular proofs for these optimisations; finally, it analyses a language with rollback exceptions;

Section 2 presents our intermediate language, and its semantics. Section 3 discusses algebraic theories, extends the language with algebraic operations and gives it an algebraic semantics. It then gives a conservative restriction analysis of the sum and tensor operations on algebraic theories. Section 4 uses this material to give our general optimisations. Section 5 extends the language with user-defined handlers whose semantics is partial. Section 6 applies our results to a language with I/O, normal and rollback exceptions and memory regions. Finally, Section 7 sketches the use of ωCPO theory to handle recursion.

II. A Multi-Adjunctive Intermediate Language

The setting for our language is as an intermediate language: a compiler has analysed the source-code for a program and produced a term in our intermediate language, over which it will apply some optimising transformations. Hence the syntax is generated automatically after some static analysis, and the intrinsic types are available for generation. Therefore having Church-style intrinsic typing is a reasonable choice.

We assume a category $\mathcal{E}$ of abstract effects has been devised based on the high level language. The canonical effect category we have in mind is where the objects are effect sets ε and the morphisms are the inclusions between them $\varepsilon_1 \subseteq \varepsilon_2$. The syntax for our language, parametrised over $\mathcal{E}$, appears in Figure 1. We dub it the *$\mathcal{E}$ Multi-Adjunctive Intermediate Language*, or $\mathcal{E}$-MAIL for short.

We base our language on Levy's *call-by-push-value (CBPV)* [18]. The main reason is that we would like to include both call-by-value and call-by-name type systems. As Levy's work shows, both these paradigms have translations into CBPV that respect all known semantic structures, hence working in CBPV is the natural choice for us. To make the transition from CBPV to $\mathcal{E}$-MAIL, we have drawn ideas from Filinski's transition from CBV to $\mathrm{M}^3\mathrm{L}$ [19].

We assume disjoint countable sets of type variables X and variables x. Our language is parametrised by a choice of base types P, constants c and predefined handler constants H.

We have two main kinds of types: *values* A and *computations* $\underline{B}$. The computations are parametrised by the objects ε of the effect category $\mathcal{E}$. Although we have defined kind environments Δ as syntactic lists, we shall treat them more abstractly as finite mappings of distinct type variables to kinds. In particular, we denote by $\mathrm{Dom}\,(\Delta)$ the finite set of type variables to which Δ assigns kinds.

Given two kind environments Δ_1, Δ_2, a *type-variable substitution* $\Theta{:}\Delta_1 \to \Delta_2$ assigns to each type variable $X \in \mathrm{Dom}\,(\Delta_2)$ a type $\Theta(X)$ such that $\Delta_1 \vdash_{\mathrm{k}} \Theta(X){:}\Delta_2(X)$. Applying a substitution to types is straightforward.

Value types, other than type variables: base types, unit, product, sum and thunks. Computation types, other than type variables: returners, computation products and function spaces. For more details, see Levy's book [20].

The most important computation term for our work on types and effects is for coercing an ε_1-computation M to an ε_2-computation $\mathtt{coerce}_f\, M$ along an $\mathcal{E}$-morphism $f{:}\varepsilon_1 \to \varepsilon_2$. Handlers are explained below. The other terms are standard CBPV terms.

Sample rules in the kind system for $\mathcal{E}$-MAIL are given in Figure 2, and sample rules from the $\mathcal{E}$-MAIL type system are given in Figure 3. In order to type the constants c, we assume we are given a mapping $A_{\_}$ from the set of constants to value types: $\vdash_{\mathrm{k}} A_c{:}\mathtt{Val}$. For example, we might include a primitive type `int` of integers. We can also include constants $\underline{n}$ for every integer $n \in \mathbb{Z}$. In this case we shall choose $A_{\underline{n}} \coloneqq \mathtt{int}$. We could also choose constant operations for the arithmetic operations $+_\varepsilon : \mathtt{thunk}_\varepsilon\, \mathtt{int} \times \mathtt{int} \to \mathtt{int}$. If

$$
\begin{array}{llll}
\text{Kinds} & \mathcal{K} & ::= & \mathtt{Val} \mid \mathtt{Comp}_{\varepsilon} \mid \mathbf{handler} \\
\text{Value Types} & A, B, \ldots & ::= & P \mid X \mid \mathtt{unit} \mid A_1 \times A_2 \mid A_1 + A_2 \mid \mathtt{U}_{\varepsilon}\, \underline{B} \\
\text{Computation Types} & \underline{A}, \underline{B}, \ldots & ::= & X \mid \mathtt{F}_{\varepsilon}\, A \mid \underline{B}_1 \times \underline{B}_2 \mid A \mathrel{->} \underline{B} \\
\text{Handler Types} & R & ::= & (A; \varepsilon_1) \mathrel{=>} (\underline{B}; \varepsilon_2) \\
\text{Value terms} & V & ::= & c \mid x \mid \star \mid (V_1, V_2) \mid \mathtt{inj}_1^{A_1+A_2}\, V \mid \mathtt{inj}_2^{A_1+A_2}\, V \mid \mathtt{thunk}_{\varepsilon}\, M \\
\text{Computation terms} & M, N, \ldots & ::= & \mathtt{coerce}_{f:\varepsilon_1 \to \varepsilon_2}\, M \mid \mathtt{return}_{\varepsilon}\, M \mid \mathtt{bind}\, M\ \mathtt{to}\ x\ \mathtt{in}\ N \\
 & & & \mid \mathtt{pm}\ V\ \mathtt{as}\ (x, y)\ \mathtt{in}\ M \mid \mathtt{pm}\ V\ \mathtt{as}\ \{\mathtt{inj}_1\, x\ .\ M_1, \mathtt{inj}_2\, y\ .\ M_2\} \mid \mathtt{force}\ V \\
 & & & \mid \lambda\{\mathtt{1} \mapsto M_1, \mathtt{2} \mapsto M_2\} \mid \mathtt{1`}M \mid \mathtt{2`}M \mid \lambda x{:}A\ .\ M \mid V\mathtt{`}M \\
 & & & \mid \mathtt{try}\ M\ \mathtt{with}\ H\ @\ V\ \mathtt{as}\ x\ @\ y\ \mathtt{in}\ N
\end{array}
$$

Kind Environments $\Delta ::= \lrcorner \mid \Delta, X{:}\mathcal{K}$

Types $Q ::= A \mid \underline{B}$

Type Environments $\Gamma ::= \lrcorner \mid \Gamma, x{:}A$

Fig. 1. $\mathcal{E}$-MAIL Synatx

Kind System $\Delta \vdash_{\mathrm{k}} Q{:}\mathcal{K}$:

$$
\frac{}{\Delta \vdash_{\mathrm{k}} P{:}\mathtt{Val}} \qquad \frac{\Delta\,(X) = \mathcal{K}}{\Delta \vdash_{\mathrm{k}} X{:}\mathcal{K}} \qquad \frac{\Delta \vdash_{\mathrm{k}} A{:}\mathtt{Val} \quad \Delta \vdash_{\mathrm{k}} \underline{B}{:}\mathtt{Comp}_{\varepsilon}}{\Delta \vdash_{\mathrm{k}} A \mathrel{->} \underline{B}{:}\mathtt{Comp}_{\varepsilon}} \qquad \frac{\Delta \vdash_{\mathrm{k}} A{:}\mathtt{Val} \quad \Delta \vdash_{\mathrm{k}} \underline{B}{:}\mathtt{Comp}_{\varepsilon_2}}{\Delta \vdash_{\mathrm{k}} (A; \varepsilon_1) \mathrel{=>} (\underline{B}; \varepsilon_2){:}\mathbf{handler}}
$$

Fig. 2. $\mathcal{E}$-MAIL Kinds

our effect category have a notion of a pure effect $\emptyset$, we may choose these operations to be pure.

Another unusual part of our type system is the rule for $\mathtt{coerce}_f\, M$. One would expect that the premise will contain any ε_1-computation $\underline{B}$, and not just the returners $\mathtt{F}_{\varepsilon_1}\, A$. However, our semantics cannot interpret such coercions for non-returner types. This semantic consideration influenced our type system design decision to restrict the premise for this rule.

Finally, let us discuss handlers. Plotkin and Pretnar [17], [21] introduced handlers for algebraic effects, in the setting of a single kind of effect. We generalise them to multiple effect kinds. Intuitively, handlers for arbitrary effects change the meaning of the effects raised in a term, while, in the process, they might cause other effects to happen.

In order to explain the handler type $(A; \varepsilon_1) \mathrel{=>} (\underline{B}; \varepsilon_2)$ and intuitive meaning behind handlers, consider their usage in a try block: $\mathtt{try}\ M\ \mathtt{with}\ H\ @\ V\ \mathtt{as}\ x\ @\ y\ \mathtt{in}\ N$. This construct is a generalisation of the one introduced for exception handling by Benton and Kennedy [22]. It executes the term M, which might cause effects in ε_1. First assume it causes no effects. Then the returned value is bound to x, and the parameter V is bound to y and N, an ε_2-computation, is executed. If any effects do occur in M, they are handled by the handler H, which may refer to, and change, the parameter of type A, with the only guarantee that the resulting computation will still be of type $\underline{B}$. The classical example is that of an exception handler, which accepts as a parameter an alternative ε_2-computation of type $\underline{B}$ as a thunked value $A := U_{\varepsilon_2}\, \underline{B}$. In this case, N makes no use of the parameter. If an exception occurs in M, then the entire computation, both the remainder of M and all of N, is aborted and the parameter computation is executed instead.

We parametrise our language further by assuming a countable set of generalised handlers H with well-kinded handler type $\vdash_{\mathrm{h}}^{\Delta_H} H{:}(A_H; \varepsilon_1^H) \mathrel{=>} (\underline{B}_H; \varepsilon_2^H)$.

A. Semantics

Let $\mathcal{B}$ be the following category. The objects of $\mathcal{B}$ are the adjunctions $F \dashv G$ with $F{:}\mathbf{Set} \to \mathcal{C}$, where $\mathcal{C}$ has binary products and all $\mathbf{Set}$ cotensors (powers). Recall that GF, like all monads over $\mathbf{Set}$, is strong. The morphisms in $\mathcal{B}$ are the strong monad morphisms between these monads.

In this paper we shall only deal with set-theoretic models of $\mathcal{E}$-MAIL. Let $[\![-]\!] : \mathcal{E} \to \mathcal{B}$ be a functor. Then for each $\mathcal{E}$-object ε there is an adjunction $F_{\varepsilon} \dashv U_{\varepsilon}$ that gives rise to a strong monad T_{ε} with strength $\mathrm{str}_{\varepsilon}$. Denote by $\mathcal{C}_{\varepsilon}$ the codomain of $F_{\varepsilon} : \mathcal{V} \to \mathcal{C}_{\varepsilon}$. Let the unit of the adjunction be η_{ε} and the counit θ_{ε}. For each $\mathcal{E}$-morphism $f : \varepsilon_1 \to \varepsilon_2$ there is a strong monad morphism $m_f : U_{\varepsilon_1} F_{\varepsilon_1} \to U_{\varepsilon_2} F_{\varepsilon_2}$.

With this notation in mind, we define the models of $\mathcal{E}$-MAIL:

Definition 1. *Let $\mathcal{E}$ be a category. A* model *for $\mathcal{E}$-MAIL $[\![-]\!]$ consists of: A functor $[\![-]\!] : \mathcal{E} \to \mathcal{B}$; For each base type P a set $[\![P]\!]$; This data is enough to interpret $\mathcal{E}$-MAIL kinds and types, as in Figure 4. For each constant c, an element $[\![c]\!] \in [\![A_c]\!]_{\emptyset}$; For each predefined handler constant H a family $\left[\!\left[\varepsilon_1^H\right]\!\right]$ of objects in $\left[\!\left[(A_H; \varepsilon_1^H) \mathrel{=>} (\underline{B}_H; \varepsilon_2^H)\right]\!\right]$.*

The semantics of $\mathcal{E}$-MAIL terms, except the try block, are given in Figure 4. Note that $\mathbf{Set}$ has all products, distributive coproducts, and is cartesian closed. To give semantics to the try block, consider a well-typed try block, as in Figure 3.

Due to the adjunction $F_{\varepsilon_1} \dashv U_{\varepsilon_1}$, for all $\delta \in [\![\Delta_1]\!]$ there exists a unique h_{δ} making the following diagram in $\mathbf{Set}$:

$$
\begin{array}{ccc}
[\![\Gamma]\!]_{\delta} \times [\![A_M]\!]_{\delta} & \xrightarrow{[\![N]\!]_{\delta}} & U_{\varepsilon_2}\, [\![\underline{B}]\!]_{[\![\Theta]\!]_{\delta}}^{[\![A]\!]_{[\![\Theta]\!]_{\delta}}} \\
\Big\downarrow{\scriptstyle \eta_{\varepsilon_1}} & & \Big\Vert \\
T_{\varepsilon_1}([\![\Gamma]\!]_{\delta} \times [\![A_M]\!]_{\delta}) & \overset{h_{\delta}}{\dashrightarrow} & U_{\varepsilon_1}\, [\![H]\!]_{[\![\Theta]\!]_{\delta}}
\end{array}
$$

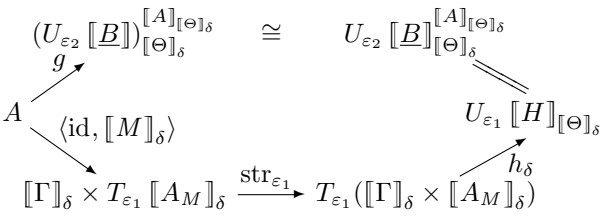
Judgements $\Gamma \vdash_{\mathrm{v}}^{\Delta} V : A$, $\Gamma \vdash_{\varepsilon}^{\Delta} M : \underline{B}$, and $\vdash_{\mathrm{h}}^{\Delta} H : (A; \varepsilon_1) => (\underline{B}; \varepsilon_2)$, assuming $\Delta \vdash_{\mathrm{k}} A : \mathtt{Val}$, $\Delta \vdash_{\mathrm{k}} \underline{B} : \mathtt{Comp}_{\varepsilon}$, and $\Delta \vdash_{\mathrm{k}} (A; \varepsilon_1) => (\underline{B}; \varepsilon_2) : \mathbf{handler}$:

$$\frac{}{\Gamma \vdash_{\mathrm{v}}^{\Delta} c : A_c} \qquad \frac{\Gamma \vdash_{\varepsilon}^{\Delta} M : \underline{B}}{\Gamma \vdash_{\mathrm{v}}^{\Delta} \mathtt{thunk}_{\varepsilon}\, M : U_{\varepsilon}\, \underline{B}} \qquad \frac{\Gamma \vdash_{\varepsilon_1}^{\Delta} M : F_{\varepsilon_1}\, A}{\Gamma \vdash_{\varepsilon_2}^{\Delta} \mathtt{coerce}_{f : \varepsilon_1 \to \varepsilon_2}\, M : F_{\varepsilon_2}\, A} \qquad \frac{\Gamma \vdash_{\varepsilon}^{\Delta} N : \mathtt{F}_{\varepsilon}\, A \quad \Gamma, x : A \vdash_{\varepsilon}^{\Delta} M : \underline{B}}{\Gamma \vdash_{\varepsilon}^{\Delta} \mathtt{bind}\, N\, \mathtt{to}\, x\, \mathtt{in}\, M : \underline{B}}$$

$$\frac{\Gamma \vdash_{\varepsilon_1}^{\Delta_1} M : F_{\varepsilon_1}\, A_M \quad \vdash_{\mathrm{h}}^{\Delta_2} H : (A; \varepsilon_1) => (\underline{B}; \varepsilon_2) \quad \Gamma \vdash_{\mathrm{v}}^{\Delta_1} V : A\Theta \quad \Gamma, x : A_M \vdash_{\varepsilon}^{\Delta_1} N : (A -> \underline{B})\Theta}{\Gamma \vdash_{\varepsilon_2}^{\Delta_1} \mathtt{try}\, M\, \mathtt{with}\, H\, @\, V\, \mathtt{as}\, x\, @\, y\, \mathtt{in}\, N : \underline{B}\Theta}\ \Theta : \Delta_1 \to \Delta_2$$

Fig. 3. $\mathcal{E}$-MAIL Types

Using h, define the following morphism g:



Now the semantics of the try block are given by:

$$[\![\Gamma]\!]_\delta \xrightarrow{\langle g, [\![V]\!]_\delta \rangle} (U_{\varepsilon_2}\, [\![\underline{B}]\!])^{[\![A]\!]_{[\![\Theta]\!]_\delta}}_{[\![\Theta]\!]_\delta} \times [\![A]\!]_{[\![\Theta]\!]_\delta} \xrightarrow{\mathrm{eval}} U_{\varepsilon_2}\, [\![\underline{B}]\!]_{[\![\Theta]\!]_\delta}$$

Our choice of interpreting $[\![f : \varepsilon_1 \to \varepsilon_2]\!]$ as a strong monad morphism arose from our usage of $\mathcal{E}$-MAIL. There are other possible choices for such maps, such as a functor $U_f f$ satisfying $U_{\varepsilon_1} U_f = U_{\varepsilon_2}$ or an adjunction.

III. Algebraic Theories

In the description of $\mathcal{E}$-MAIL we did not mention any computational effects. As with standard metalanguages, the effects need to be layered on top of the calculus. However, if we know that all these effects are *algebraic* [15], we can incorporate them into our language and semantics, obtaining *Algebraic* $\mathcal{E}$-MAIL.

Given such an equational theory $\langle \Sigma, E \rangle$, we can choose $\mathcal{E}$ as a subcategory of Σ-subsets and inclusions between them. Note that if we have a countable signature, then its powerset is uncountable, making $\mathcal{E}$ and our calculus uncountable *necessarily*. Therefore we decouple $\mathcal{E}$ from the powerset by taking a subcategory $\mathcal{E}$.

We shall investigate the structure of this algebraic variant of $\mathcal{E}$-MAIL, and its *conservative restriction* model.

A. Lawvere Theories

Let κ be one of $\aleph_0$, $\aleph_1$, the first countable or uncountable cardinal. κ can be considered as a category with objects being all cardinals $\kappa > \lambda$ and the functions between them.

Definition 2. *A* κ-Lawvere theory *is a strict up to* κ*-product preserving, identity-on-objects functor* $J : \kappa^{\mathrm{op}} \to \mathcal{L}$.

A morphism of κ-Lawvere theories $F : J_1 \to J_2$ *is a product preserving, identity-on-objects functor* $F : \mathcal{L}_1 \to \mathcal{L}_2$ *such that* $F \circ J_1 = J_2$. *We denote by* κ-$\mathbf{Law}$ *the category of all* κ*-Lawvere theories and their morphisms.*

We shall usually identify the Lawvere theory with the underlying category $\mathcal{L}$.

Lawvere theories offer a categorical view to universal algebra and equational logic. The arrows $\mathcal{L}(n, 1)$ correspond to equivalence classes of provably equal terms over n variables. A more detailed account is given by Borceux [23] in the unenriched case and by Power [24] in the enriched case. We shall only quote the definitions and relevant results.

When we move to infinite Lawvere theories, the correspondence with universal algebra continues. In this case, our terms can be infinitely wide, but well-founded. Given a function symbol f with arity $n < \kappa$, we adopt the notation $f(\lambda r.t_r)$ for the term whose r-th component is t_r for all $0 \leq r < n$.

Definition 3. *Let* $J : \kappa^{\mathrm{op}} \to \mathcal{L}$ *be a* κ*-Lawvere theory, and* $\mathcal{C}$ *be a category with up to all* κ *products. A* $\mathcal{C}$-model *of* $\mathcal{L}$ *is functor* $M : \mathcal{L} \to \mathcal{C}$ *that preserves up to all* κ *products. A* morphism of $\mathcal{C}$-models of $\mathcal{L}$ *is a natural transformation between models. We denote by* $\mathsf{Mod}(\mathcal{L}, \mathcal{C})$ *the category of* $\mathcal{C}$*-models of* $\mathcal{L}$ *and their morphisms.*

The forgetful functor $U : \mathsf{Mod}(\mathcal{L}, \mathcal{C}) \to \mathcal{C}$, given by $M \mapsto M1$, maps a model to its carrier set. Its left adjoint, when it exists, is called the *free model functor*. The free set-theoretic model functor always exists. If we view $\mathcal{L}$ as an equational theory, then the free model functor assigns to each set X the evident model whose carrier set is the set of equivalence classes of terms over variables in X.

Thus, every κ-Lawvere theory gives rise to a monad over $\mathbf{Set}$, given by the free model adjunction, $T_{\mathcal{L}} \coloneqq UF$. The unit of this monad is the inclusion of the set of variables, and the multiplication map is given by substitution. This monad always has rank κ, that is, it preserves κ-filtered colimits.

Conversely, any κ-ranked monad T over $\mathbf{Set}$ is the free model monad for some κ-Lawvere theory. This Lawvere theory can be obtained as follows: consider the Kleisli category for T, $\mathrm{Kl}\, T$. Restrict to the sub-category $\mathrm{Kl}_\kappa\, T$ given by the representatives n of the sets with cardinality $n < \kappa$. Then the opposite category $\mathrm{Kl}_\kappa^{\mathrm{op}} T$ is a κ-Lawvere theory, and its free model monad is isomorphic to T.

These two processes form an equivalence of categories between κ-$\mathbf{Law}$ and the category of κ-ranked monads over $\mathbf{Set}$. Thus, a ranked monad T is essentially a Lawvere theory. Its terms over n variables are the arrows $1 \to Tn$ of the Kleisli category.

Kinds: $[\![\mathcal{K}]\!]$ is a category: $[\![\mathtt{Val}]\!] \coloneqq \mathcal{V} \coloneqq \mathbf{Set} \quad [\![\mathtt{Comp}_\varepsilon]\!] \coloneqq \mathcal{C}_\varepsilon$.
Kind environments: $[\![\Delta]\!]$ is a class of mappings from $\mathrm{Dom}\,(\Delta)$ to objects such that if $\Delta(X) = \mathcal{K}$ then $[\![\Delta]\!]\,(X) \in \mathrm{Ob}\,([\![\mathcal{K}]\!])$:

$$[\![\llcorner]\!] \coloneqq \{\emptyset\} \quad [\![\Delta, X{:}\mathcal{K}]\!] \coloneqq \left\{ \delta\,[X \mapsto W] \,\middle|\, \delta \in [\![\Delta]\!]\,, W \in [\![\mathcal{K}]\!] \right\}$$

Type-variable substitutions: $[\![\Theta{:}\Delta_1 \to \Delta_2]\!]$ is the function $[\![\Theta]\!] : [\![\Delta_1]\!] \to [\![\Delta_2]\!]$ given by $[\![\Theta]\!]_\delta\,(Y \in \mathrm{Dom}\,(\Delta_2)) \coloneqq [\![\Theta(Y)]\!]_\delta$
Kind judgements: $[\![\Delta \vdash_{\mathrm{k}} Q{:}\mathcal{K}]\!]$ is a mappings $[\![Q]\!] : [\![\Delta]\!] \to \mathrm{Ob}\,([\![\mathcal{K}]\!])$:

$$[\![\Delta \vdash_{\mathrm{k}} X{:}\mathcal{K}]\!]_\delta \coloneqq \delta(X) \quad [\![\Delta \vdash_{\mathrm{k}} A \mathrel{->} \underline{B}]\!]_\delta \coloneqq [\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}$$

Type environments: $[\![\Gamma]\!]$ is a mapping from $[\![\Delta]\!]$ to $\mathcal{V}$-objects: $[\![\llcorner]\!]_\delta \coloneqq \mathbb{1} \quad [\![\Delta, x{:}A]\!]_\delta \coloneqq [\![\Delta]\!]_\delta \times [\![A]\!]_\delta$. Handlers: $\left[\!\left[\vdash^{\Delta}_{\mathrm{h}} H{:}(A;\varepsilon_1) \mathrel{=>} (\underline{B};\varepsilon_2)\right]\!\right]$ is a family of $\mathcal{C}_{\varepsilon_1}$-objects parametrised by $[\![\Delta]\!]$ such that: $U_{\varepsilon_1}\,[\![H]\!]_\delta = U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}$.
Value type judgements: $\left[\!\left[\Gamma \vdash^{\Delta}_{\mathrm{v}} V{:}A\right]\!\right]$ is a map from $[\![\Delta]\!]$ to $\mathcal{V}$-morphisms such that $[\![V]\!]_\delta : [\![\Gamma]\!]_\delta \to [\![A]\!]_\delta$.
Computation type judgements: $\left[\!\left[\Gamma \vdash^{\Delta}_{\varepsilon} M{:}\underline{B}\right]\!\right]$ is a map from $[\![\Delta]\!]$ to $\mathcal{V}$-morphisms such that $[\![M]\!]_\delta : [\![\Gamma]\!]_\delta \to U_\varepsilon\,[\![\underline{B}]\!]_\delta$:

$$\left[\!\left[\Gamma \vdash^{\Delta}_{\mathrm{v}} \mathtt{thunk}_{\,\varepsilon}\, M : U_\varepsilon\, \underline{B}\right]\!\right]_\delta : [\![\Gamma]\!]_\delta \xrightarrow{[\![M]\!]_\delta} U_\varepsilon\,[\![\underline{B}]\!]_\delta$$

$$\left[\!\left[\Gamma \vdash^{\Delta}_{\mathrm{v}} \mathtt{coerce}_{f:\varepsilon_1 \to \varepsilon_2}\, M : F_{\varepsilon_2}\, A\right]\!\right]_\delta : [\![\Gamma]\!]_\delta \xrightarrow{[\![M]\!]_\delta} T_{\varepsilon_1} A \xrightarrow{m_f} T_{\varepsilon_2} A$$

$$\left[\!\left[\Gamma \vdash^{\Delta}_{\varepsilon} \mathtt{bind}\, N\ \mathtt{to}\ x\ \mathtt{in}\ M{:}\underline{B}\right]\!\right]_\delta : [\![\Gamma]\!]_\delta \xrightarrow{\langle \mathrm{id}, [\![N]\!]_\delta \rangle} [\![\Gamma]\!]_\delta \times T_\varepsilon\,[\![A]\!]_\delta \xrightarrow{\mathrm{str}\varepsilon} T_\varepsilon([\![\Gamma]\!]_\delta \times [\![A]\!]_\delta)$$

Fig. 4. $\mathcal{E}$-MAIL Semantics

The category κ-**Law** is cocomplete, and in particular has binary coproducts $\mathcal{L}_1 + \mathcal{L}_2$. In terms of equational theories over a signature, also known as presentations, the coproduct of two Lawvere theories amounts to taking the disjoint union of their function symbols, and including all the equations from both theories and nothing else. The initial object in this category is the empty theory κ^{op}, also known as the theory of sets.

The category κ-**Law** admits a symmetric monoidal structure, the *tensor*, $\langle \kappa\text{-}\mathbf{Law}, \otimes, \kappa^{\mathrm{op}} \rangle$. In terms of presentations, the tensor $\mathcal{L}_1 \otimes \mathcal{L}_2$ amounts to taking the disjoint union of the function symbols from both theories, as well as requiring that operations from different theories commute with each other. More explicitly, the two function symbols $f{:}n$ and $g{:}m$ commute when $f(\lambda r.g(\lambda s.\mathtt{x}_{r,s})) = g(\lambda s.f(\lambda r.\mathtt{x}_{r,s}))$.

More abstractly, for any arrow $t{:}m_1 \to m_2$ in $\mathcal{L}$ we denote by $t \otimes n{:}m_1 \times n \to m_2 \times n$ the arrow obtained by taking n copies of t: $\pi_{i,j} \circ t \otimes n = (\pi_i \circ t)(\lambda j.\mathtt{x}_{i,j})$. Then two arrows $t_1{:}n_1 \to m_1$ and $t_2{:}n_2 \to m_2$ commute when $(m_1 \otimes t_2) \circ (t_1 \otimes n_2) = (t_1 \otimes m_2) \circ (n_1 \otimes t_2)$.

An essential property of the tensor is that for any two κ-Lawvere theories $\mathcal{L}_1$, $\mathcal{L}_2$, we have an equivalence of categories between $\mathsf{Mod}(\mathcal{L}_1 \otimes \mathcal{L}_2, \mathcal{C})$ and $\mathsf{Mod}(\mathcal{L}_1, \mathsf{Mod}(\mathcal{L}_2, \mathcal{C}))$

Plotkin and Power [14] realised that most of the monads used for computational effects, with the notable exception of the continuations monad have a rank, and therefore are equivalent to some Lawvere theory. They also realised [25] that the operations causing the effects come in two equivalent flavours.

For concreteness, let $f{:}n \to m$ be any arrow in a Lawvere theory. The first notion, that of a *generic effect*, is to regard f as an arrow $m \to Tn$ in the Kleisli subcategory $\mathrm{Kl}_\kappa\, T$. The second notion, that of an *algebraic operation*, is a transformation $(Tx)^m \to (Tx)^n$, that is a natural transformation in the Kleisli category.

Finally, Hyland et al. [16] realised that the two notions for combining Lawvere theories, sum and tensor, correspond to the notion of monad transformers [26]. Their work allowed to directly lift effect operations in either flavour to the combined theory, a feat that was only recently discovered for monad transformers by Jaskelioff and Moggi [27].

B. Algebraic $\mathcal{E}$-MAIL

Let κ be an infinite cardinal. A *κ-signature* is a pair $\langle \Sigma, arity \rangle$ where Σ is a set and $arity{:}\Sigma \to \kappa \times \kappa$ is a function called the *arity*. We shall use infix notation for arities, $f{:}n \to m$, as customary. We shall also interchange cardinals n with their representatives n and sets with that cardinality.

Given a κ-signature Σ, we denote by $\aleph_\Sigma^{\mathrm{op}}$ the free κ-Lawvere theory for that signature: it is the smallest κ-Lawvere theory such that each $f{:}A \to B$ in Σ is a different morphism of $\aleph_\Sigma^{\mathrm{op}}$. A presentation of a κ-Lawvere theory is then nothing but a full functor $\aleph_\Sigma^{\mathrm{op}} \to \mathcal{L}$ that is a morphism of Lawvere theories, for some signature Σ. $\aleph_\Sigma^{\mathrm{op}}$ extends to a functor from signatures and inclusions to faithful Lawvere morphisms.

Fix a presentation $\aleph_\Sigma^{\mathrm{op}} \to \mathcal{L}$, and choose a sub-category $\mathcal{E}$ of the partial order of subsets of Σ for the remainder of this section. Algebraic $\mathcal{E}$-MAIL contains one new construct for computational effects:

Definition 4. *The* Algebraic $\mathcal{E}$-MAIL *is the* $\mathcal{E}$-MAIL *extended with its syntax extended, for all* $\mathrm{op}{:}A_1 \to A_2$ *in* Σ*:*

$$P ::= \ldots \mid \mathtt{A}_{A_1} \mid \mathtt{A}_{A_2}$$
$$M, N ::= \ldots \mid \mathrm{op}_V\, M$$

The type system is extended with the following rule:

$$\frac{\Gamma \vdash^{\Delta}_{\mathrm{v}} V{:}\mathtt{A}_{A_2} \quad \Gamma \vdash^{\Delta}_{\varepsilon} M{:}\mathtt{A}_{A_1} \to \underline{B}}{\Gamma \vdash^{\Delta}_{\varepsilon} \mathrm{op}_V\, M{:}\underline{B}}\ \mathrm{op}{:}A_1 \to A_2 \in \varepsilon$$

The construct $\mathrm{op}_V(M)$ is the algebraic operation for $\mathrm{op}{:}A_1 \to A_2$. The generic effect for op is definable as:

$$\mathtt{gen}^{\varepsilon}_{\mathrm{op}}(V) \coloneqq \mathrm{op}_V(\lambda x{:}\mathtt{A}_{A_1}\ .\ \mathtt{return}_\varepsilon\, x)$$

We turn to the semantics of Algebraic $\mathcal{E}$-MAIL. The category κ-**Law** has a natural factorisation system: each morphism $F{:}\mathcal{L}_1 \to \mathcal{L}_2$ can be factored as $F = M_F E_F$, where E is a full functor and M is a faithful functor. We use this factorisation system to define the categorical counterpart of the notion of a *conservative extension*:

Definition 5. *Let $P{:}\aleph_\Sigma^{\mathrm{op}} \to \mathcal{L}$ be a presentation of a κ-Lawvere theory. For all $\varepsilon \subseteq \Sigma$, consider the unique factorisation ME satisfying*

$$\begin{array}{ccc} \aleph_\varepsilon^{\mathrm{op}} & \xrightarrow{\aleph^{\mathrm{op}}_{\varepsilon\subseteq\Sigma}} & \aleph_\Sigma^{\mathrm{op}} \\ {\scriptstyle E}\downarrow & = & \downarrow{\scriptstyle P} \\ \mathcal{L}_\varepsilon & \xrightarrow{M} & \mathcal{L} \end{array}$$

The Lawvere theory $\mathcal{L}_\varepsilon$ is the conservative restriction *of $\mathcal{L}$ to ε. We shall usually refer to the restriction by the faithful morphism M only.*

In other words, the theory $\mathcal{L}_\varepsilon$ equates two ε-terms precisely when $\mathcal{L}$ equates them.

Definition 6. *A* conservative restriction model *for an Algebraic $\mathcal{E}$-MAIL is an $\mathcal{E}$-MAIL model in which $[\![\varepsilon]\!]$ is the free model adjunction $F_\varepsilon \dashv U_\varepsilon : \mathsf{Mod}(\mathcal{L}_\varepsilon, \mathbf{Set})$, and $[\![\varepsilon_1 \subseteq \varepsilon_2]\!]$ is the strong monad morphism that is equivalent to the conservative restriction morphism $\mathcal{L}_\varepsilon \hookrightarrow \mathcal{L}_\varepsilon$.*

Given an Algebraic $\mathcal{E}$-MAIL model $[\![-]\!]$, we interpret the constructs in the extension as follows. The additional base types $[\![\mathtt{A}_{A_i}]\!]_\emptyset$ are interpreted as A_i. For each $\mathrm{op}{:}A_1 \to A_2 \in \varepsilon$, because $\mathrm{op}{:}A_1 \to A_2$ in $\mathcal{L}_\varepsilon$, we have an arrow $A_2 \to T_\varepsilon A_1$ in the Kleisli category $\mathrm{Kl}\, T_\varepsilon$. Therefore we can interpret $\mathrm{op}_V(M)$ as:

$$\begin{aligned} [\![\Gamma]\!]_\delta &\xrightarrow{\langle \mathrm{id}, [\![V]\!]_\delta\rangle} [\![\Gamma]\!]_\delta \times A_2 \xrightarrow{\mathrm{id}\times\mathrm{op}} \\ &[\![\Gamma]\!]_\delta \times T_\varepsilon A_1 \xrightarrow{\mathrm{str}\varepsilon} T_\varepsilon([\![\Gamma]\!]_\delta \times A_1) \xrightarrow{T_\varepsilon([\![M]\!]_\delta\times\mathrm{id})} \\ &T_\varepsilon\left(U_\varepsilon\left([\![\underline{B}]\!]_\delta^{A_1}\right)\times A_1\right) \cong T_\varepsilon\left((U_\varepsilon\,[\![\underline{B}]\!]_\delta)^{A_1}\times A_1\right) \\ &\xrightarrow{T_\varepsilon \mathrm{eval}} U_\varepsilon\, F_\varepsilon\, U_\varepsilon\, [\![B]\!]_\delta \xrightarrow{U_\varepsilon\,\theta\varepsilon} U_\varepsilon\, [\![\underline{B}]\!]_\delta \end{aligned}$$

C. Combining Conservative Restrictions

We have shown how the conservative restriction hierarchy can serve as semantics for type and effect systems. One of our goals is to devise a *modular* foundations for effect systems.

Given two presentations $\aleph^{\mathrm{op}}_{\Sigma_i} \xrightarrow{P_i} \mathcal{L}_i$, $i = 1, 2$, consider the combined theory $\mathcal{L}_1 \bigcirc \mathcal{L}_2$, for either combination $\bigcirc \in \{+, \otimes\}$. From the construction of $\mathcal{L}_1 \bigcirc \mathcal{L}_2$ we know that there is a presentation $\aleph^{\mathrm{op}}_{\Sigma_1+\Sigma_2} \to \mathcal{L}_1 \bigcirc \mathcal{L}_2$. All of the subsets of $\Sigma_1 + \Sigma_2$ are of the form $\varepsilon_1 \cup \varepsilon_2$ with $\varepsilon_i \subseteq \Sigma_i$. We shall show that all the conservative restrictions of $\mathcal{L}_1 \bigcirc \mathcal{L}_2$ in cases that arise in practise are of the form $\mathcal{L}_1^{\varepsilon_1} \bigcirc \mathcal{L}_2^{\varepsilon_2}$. Therefore analysing the conservative restriction hierarchy of the combined theory amounts to a separate analysis of each theory.

1) Sum: We say that a κ-Lawvere theory is *completely free* when it is of the form $\aleph_\Sigma^{\mathrm{op}}$ for some κ-signature Σ. That is, when it has no equations.

Combining effects by summing arises in practise when we sum with a completely free theory [16]. Therefore we shall only concentrate on the case $\mathcal{L}{+}\aleph_\Sigma^{\mathrm{op}}$. Hyland et al. [16] showed that the if $\mathcal{L}$ is any other κ-theory with monad $T_\mathcal{L}$, then the monad for $\mathcal{L} + \aleph_\Sigma^{\mathrm{op}}$ is

$$T_{\mathcal{L}+\aleph_\Sigma^{\mathrm{op}}} X \coloneqq \mu Z.T_\mathcal{L}\left(X + \sum_{\mathrm{op}:I_{\mathrm{op}}\to O_{\mathrm{op}}\in\Sigma} O_{\mathrm{op}} \times Z^{I_{\mathrm{op}}}\right)$$

If $\varepsilon \subseteq \Sigma$ then $\aleph_\varepsilon^{\mathrm{op}} = \aleph_\varepsilon^{\mathrm{op}} \hookrightarrow \aleph_\Sigma^{\mathrm{op}}$ is a factorisation into a full-faithful pair. Therefore the conservative restriction of a completely free theory to ε is the completely free theory $\aleph_\varepsilon^{\mathrm{op}}$.

Theorem 7. *Let $\aleph^{\mathrm{op}}_{\Sigma_1} \xrightarrow{P_1} \mathcal{L}$ be a κ-presentation and Σ_2 a κ-signature.*

For each $\varepsilon_1 \cup \varepsilon_2 \subseteq \Sigma_1 + \Sigma_2$ with $\varepsilon_i \subseteq \Sigma_i$, we have

$$(\mathcal{L} + \aleph^{\mathrm{op}}_{\Sigma_2})_{\varepsilon_1\cup\varepsilon_2} = \mathcal{L}_{\varepsilon_1} + \aleph^{\mathrm{op}}_{\varepsilon_2}$$

2) Tensor: Hyland et al. [16] have shown that in practise we tensor three kinds of theories with other theories: the theory for the *global state monad* $S \times (-)^S$, the theory for the *reader monad* $(-)^S$, and the theory for the *writer monad* $M \times (-)$ where M is a monoid. We unify the analysis of these three cases by defining a *theory for generalised global state*. The conservative restrictions of these three cases are then special cases.

The computational model behind the theory for generalised state is of a state in a set S, on which actions can be performed by elements of a monoid M.

Definition 8 (Generalised State Theory)**.** *Let $\langle S, M, \cdot\rangle$ be a monoid action with $|S|, |M| < \kappa$. The κ-Lawvere theory for* generalised global state *$\mathcal{L}_{GGS(\cdot)}$ for this action is given by the following operations and equations. The operations are*

$$\mathtt{lookup}{:}S \to 1 \qquad \mathtt{act}{:}1 \to M$$

Given $m \in M$, we write $\mathtt{act}_m\mathtt{x}$ for the m-th projection of $\mathtt{act}\,\mathtt{x}$. The equations are

$$\begin{aligned} \mathtt{lookup}(\lambda s.\mathtt{x}) &= \mathtt{x} \\ \mathtt{lookup}(\lambda s.\mathtt{lookup}(\lambda t.\mathtt{x}_{s,t})) &= \mathtt{lookup}(\lambda r.\mathtt{x}_{r,r}) \\ \mathtt{act}_{m_1}\,\mathtt{act}_{m_2}\,\mathtt{x} &= \mathtt{act}_{m_2 m_1}\,\mathtt{x} \\ \mathtt{act}_1\,\mathtt{x} &= \mathtt{x} \\ \mathtt{act}_m\,\mathtt{lookup}(\lambda s.\mathtt{x}_s) &= \mathtt{lookup}(\lambda r.\mathtt{act}_m\,\mathtt{x}_{m\cdot r}) \end{aligned}$$

and for all S-indexed sequences of pairs $\langle m_s, n_s\rangle_{s\in S}$ for which $m_s \cdot s = n_s \cdot s$, the equation:

$$\mathtt{lookup}(\lambda s.\mathtt{act}_{m_s}\,\mathtt{x}_s) = \mathtt{lookup}(\lambda s.\mathtt{act}_{n_s}\,\mathtt{x}_s)$$

Given a set S, consider the overwrite monoid $M \coloneqq \mathbb{1} + S$, given by $s_1 s_2 = s_1$. Let $(\cdot)$ be the monoid action induced by this monoid over S. Then the generalised state theory for this

monoid action coincides with the *global state theory* $\mathcal{L}_{GS_S}$ for a single location with values in S.

A study of the conservative restriction of particular instances of the generalised state theory finds the reader, writer and state theories. Thus, all instances of tensoring are covered by tensoring with generalised state, or one of its conservative restrictions: The theory for global state is an instance of generalised state; its conservative restriction to look-ups only, yields the reader theory for the same set of states; and the writer theory is a conservative restriction of an instance of the generalised state theory to actions only. Moreover, the writer theory for the overwrite monoid for a given set of states, $\mathcal{L}_{OW_S}$, is the conservative restriction of the theory for global state to writes only. This relationship provides empirical evidence that conservative restrictions gives us the right hierarchy for type and effect systems.

Theorem 9. *Let* $\langle S, M, \cdot \rangle$ *be a monoid action with* $|S|, |M| < \kappa$, $\aleph_{\Sigma_1}^{\mathrm{op}} \xrightarrow{\mathcal{L}}$ *a* κ*-presentation. Let* $\Sigma_2 \coloneqq \{\mathtt{lookup}, \mathtt{act}m | m \in M\}$.

For all $\varepsilon_1 \cup \varepsilon_2 \subseteq \Sigma_1 + \Sigma_2$,

$$(\mathcal{L} \otimes \mathcal{L}_{GGS(\cdot)})_{\varepsilon_1 \cup \varepsilon_2} = \mathcal{L}_{\varepsilon_1} \otimes \mathcal{L}^{\varepsilon_2}_{GGS(\cdot)}$$

IV. Optimisation Taxonomy

Now that we have a semantic foundation for type and effect analysis, we can examine code transformation optimisations in this light. We list three sources for such optimisations. Comparing to Benton et al. [5]–[9], we have covered all the optimisations that make sense in our language, in one form or another. The advantage of our method is that our proofs are now modular.

In order to talk about optimisations, we introduce *validty*:

Definition 10. *Consider some* $\mathcal{E}$-MAIL *language. Let* $\Gamma \vdash^{\Delta} M_i : Q$ *be any two terms of the same type.*

For any $\mathcal{E}$-MAIL*-model* $\mathcal{M}$ *we say that the equation* $\Gamma \vdash^{\Delta} M_1 = M_2 : Q$ *is* valid*, or* holds*, in* $\mathcal{M}$*, and write*

$$\mathcal{M} \models \Gamma \vdash^{\Delta} M_1 = M_2 : Q$$

if $[\![M_1]\!] = [\![M_2]\!]$ *in* $\mathcal{M}$.

We say that an equation is valid *if it holds in all* $\mathcal{E}$-MAIL*-models. In this case we write:*

$$\Gamma \vdash^{\Delta} M_1 = M_2 : Q$$

When Γ and Q can be inferred from the context, we shall omit them and simply write $\mathcal{M} \models M_1 = M_2$ or $M_1 = M_2$.

A. Structural Optimisations

Structural optimisations arise from the structure of our multi-adjunction language. They are true in all models, regardless of the interpretation we choose. Examples appear in Figure 5. Variants of these optimisations appear in Benton et al.'s work as "effect independent optimisations".

B. Algebraic Optimisations

Consider Algebraic $\mathcal{E}$-MAIL and one of its conservative restriction models $[\![-]\!]$. Each computation category $\mathcal{C}_\varepsilon$ in this model arises out of a κ-Lawvere theory. Given a term $t : n \to 1$ in this theory, if our $\mathcal{E}$-MAIL language is rich enough, we can express this term as a program. More precisely, there exists a term $\vdash^{\Delta}_{\varepsilon} M_t$ such that $[\![M_t]\!]_\emptyset = t$. Therefore, for every equation $t_1 = t_2$ in this theory there is a corresponding optimisation $M_{t_1} = M_{t_2}$ that holds in this model.

Thus, the *algebraic optimisations* are those equations that arise out of the equational nature of our semantics. Given a stronger logic to reason about program phrases, a fuller portion of this semantic information can be utilised to reason about programs. In Benton and Kennedy's work [5], the algebraic optimisations appeared also as effect independent optimisations.

C. Abstract Optimisations

The last class of optimisations we consider takes advantage of particular properties of the computation category $\mathcal{C}_\varepsilon$. For each of these optimisations we present an abstract characterisation in terms of the underlying monad T_ε. It turns out that when these monads are algebraic, each of these characterisations has a neat algebraic counterpart over the terms of the Lawvere theory. We utilise this connection to easily deduce preservation results for the combination of theories. We dub these optimisations *algebraic*.

Definition 11. *Let* T *be a strong monad over a category with finite products. Let* $\mathrm{flip} : A \times B \xrightarrow{\langle \pi_2, \pi_1 \rangle} B \times A$*. The* left strength *is the following natural transformation* $\mathrm{stl} : (TA) \times B \to T(A \times B)$*, given by:* $(TA) \times B \xrightarrow{\mathrm{flip}} B \times TA \xrightarrow{\mathrm{str}} T(B \times A) \xrightarrow{T\mathrm{flip}} T(A \times B)$

For the notions of *affine monad* and *relevant monad*, see Jacobs and related literature [28]–[30].

Definition 12. *Let* $\mathcal{C}$ *be a category with finite products.* Commuting morphisms *are two strong monad morphisms* $T_1 \xrightarrow{m_1} T \xleftarrow{m_2} T_2$ *on* $\mathcal{C}$ *such that*

$$\begin{array}{ccc} T_1A \times T_2B & \xrightarrow{m_1 \times m_2} & TA \times TB \\ \Big\downarrow{\scriptstyle m_1 \times m_2} & & \Big\downarrow{\scriptstyle \tilde{\psi}} \\ TA \times TB & \xrightarrow{\psi} & T(A \times B) \end{array}$$

where $\psi \coloneqq \mu \circ T\mathrm{str} \circ \mathrm{stl}$ *and* $\tilde{\psi} \coloneqq \mu \circ T\mathrm{str} \circ \mathrm{stl}$.

The importance of these characterisations stems from the optimisations they allow, which are depicted in Figure 6.

The affine and relevant monads were defined in [28], [29]. Their relevance to program equivalence was noted by Führmann [30], although in a single monadic setting.

The last optimisation in Figure 6 is known as "lambda hoist". However, we do not know any satisfying abstract generalisation.

β-laws such as $V\text{`}\lambda x{:}A\,.\,M = M\,[V/x]$.
η-laws, such as: $M\,[V/z] = \mathtt{pm}\ V\ \mathtt{as}\ \left\{\mathtt{inj}_1\,x_1\,.\,M\left[\mathtt{inj}_1^A\,x_1/z\right], \mathtt{inj}_2\,x_2\,.\,M\left[\mathtt{inj}_2^A\,x_2/z\right]\right\}$ with x_1, x_2 fresh in M
Sequencing: $\mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ \lambda y{:}B\,.\,M = \lambda y{:}B\,.\,\mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ N$ (with y fresh in M)

$\mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ \lambda\,\{\mathtt{1} \mapsto N_1, \mathtt{2} \mapsto N_2\} = \lambda\,\{\mathtt{1} \mapsto \mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ N_1, \mathtt{2} \mapsto \mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ N_2\}$

$\mathtt{bind}\,(\mathtt{bind}\,M_1\ \mathtt{to}\ x_1\ \mathtt{in}\ M_2)\ \mathtt{to}\ x_2\ \mathtt{in}\ N = \mathtt{bind}\,M_1\ \mathtt{to}\ x_1\ \mathtt{in}\ (\mathtt{bind}\,M_2\ \mathtt{to}\ x_2\ \mathtt{in}\ N)$ (with x_1 fresh in N)

Coercion: $\mathtt{coerce}_{f:\varepsilon_1\to\varepsilon_2}\,\mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ N = \mathtt{bind}\,\mathtt{coerce}_{f:\varepsilon_1\to\varepsilon_2}\,M\ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{coerce}_{f:\varepsilon_1\to\varepsilon_2}\,N$

$\mathtt{coerce}_{g:\varepsilon_2\to\varepsilon_3}\,\mathtt{coerce}_{f:\varepsilon_1\to\varepsilon_2}\,M = \mathtt{coerce}_{g\circ f:\varepsilon_1\to\varepsilon_3}\,M \qquad \mathtt{coerce}_{f:\varepsilon_1\to\varepsilon_2}\,\mathtt{return}_{\varepsilon_1}\,M = \mathtt{return}_{\varepsilon_2}\,M$

Fig. 5. Structural Optimisations

Let $\mathcal{M}$ be an $\mathcal{E}$-MAIL-model, and $\mathcal{E}$-morphisms $\varepsilon \xrightarrow{f} \varepsilon'$, $\varepsilon_1 \xrightarrow{g_1} \varepsilon \xleftarrow{g_2} \varepsilon_2$:

$$\frac{\Gamma \vdash^{\Delta}_{\varepsilon} M : F_\varepsilon\,A \qquad \Gamma \vdash^{\Delta}_{\varepsilon'} N : \underline{B} \qquad T_\varepsilon \text{ is affine}}{\mathcal{M} \models \Gamma \vdash^{\Delta}_{\varepsilon'} \mathtt{bind}\,\mathtt{coerce}_f\,M\ \mathtt{to}\ x\ \mathtt{in}\ N = N : \underline{B}}$$

$$\frac{\Gamma \vdash^{\Delta}_{\varepsilon_1} M_1 : F_{\varepsilon_1}\,A_1 \qquad \Gamma \vdash^{\Delta}_{\varepsilon_2} M_2 : F_{\varepsilon_2}\,A_2 \qquad \Gamma, x_1{:}A_1, x_2{:}A_2 \vdash^{\Delta}_{\varepsilon'} N : \underline{B} \qquad T_{\varepsilon_1} \xrightarrow{m_{g_1}} T_\varepsilon \xleftarrow{m_{g_2}} T_{\varepsilon_2} \text{ commute}}{\begin{array}{l}\mathcal{M} \models \Gamma \vdash^{\Delta}_{\varepsilon'} \mathtt{bind}\,\mathtt{coerce}_{f\circ g_1}\,M_1\ \mathtt{to}\ x_1\ \mathtt{in}\ \mathtt{bind}\,\mathtt{coerce}_{f\circ g_2}\,M_2\ \mathtt{to}\ x_2\ \mathtt{in}\ N = \\ \qquad\qquad \mathtt{bind}\,\mathtt{coerce}_{f\circ g_2}\,M_2\ \mathtt{to}\ x_2\ \mathtt{in}\ \mathtt{bind}\,\mathtt{coerce}_{f\circ g_1}\,M_1\ \mathtt{to}\ x_1\ \mathtt{in}\ N : \underline{B}\end{array}}$$

$$\frac{\Gamma \vdash^{\Delta}_{\varepsilon} M : F_\varepsilon\,A \qquad \Gamma \vdash^{\Delta}_{\varepsilon'} N : \underline{B} \qquad T_\varepsilon \text{ is relevant}}{\mathcal{M} \models \Gamma \vdash^{\Delta}_{\varepsilon'} \mathtt{bind}\,\mathtt{coerce}_f\,M\ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{bind}\,\mathtt{coerce}_f\,M\ \mathtt{to}\ y\ \mathtt{in}\ N = \mathtt{bind}\,\mathtt{coerce}_f\,M\ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{bind}\,\mathtt{return}_{\varepsilon'}\,x\ \mathtt{to}\ y\ \mathtt{in}\ N : \underline{B}}$$

$$\frac{\Gamma \vdash^{\Delta}_{\varepsilon_1} M : \mathtt{F}_{\varepsilon_1}\,A \quad \Gamma, x{:}A \vdash^{\Delta}_{\varepsilon_2} N : \underline{B} \qquad T_{\varepsilon_1} \text{ is the identity monad}}{\mathcal{M} \models \Gamma \vdash^{\Delta}_{\varepsilon_1} \mathtt{return}_{\varepsilon_1}\,\mathtt{thunk}_{\varepsilon_2}\,\mathtt{bind}\,\mathtt{coerce}_{\varepsilon_1\subseteq\varepsilon_2}\,M\ \mathtt{to}\ x\ \mathtt{in}\ N = \mathtt{bind}\,M\ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{return}_{\varepsilon_1}\,\mathtt{thunk}_{\varepsilon_2}\,N : \mathtt{F}_{\varepsilon_1}\,\mathtt{U}_{\varepsilon_2}\,\underline{B}}$$

Fig. 6. Abstract Optimisations

Viewing the arrows in a Lawvere theory as generalised terms sheds light on these monadic properties. Affinity corresponds to generalised absorption: $f(\lambda r.\mathtt{x}) = x$ for any term f. Relevance corresponds to generalised idempotency: $f(\lambda r.f(\lambda s.\mathtt{x}_{r,s})) = f(\lambda t.\mathtt{x}_{t,t})$ for any term f. Commuting arrows correspond to the requirement that terms from the two different domain theories, when embedded in the codomain theory, commute.

Theorem 13. *Modular composition of monadic properties:*

1) *If L_1 and L_2 are theories with affine monads, then both $L_1 + L_2$ and $L_1 \otimes L_2$ have affine monads.*
2) *Let L_1 and L_2 be theories with relevant monads*
   a) *If, for some $i \in \{1, 2\}$, L_i has a presentation consisting solely of unary arrows $f : \mathbb{1} \to A$, then $L_1 \otimes L_2$ has a relevant monad.*
   b) *If one of the theories has an affine monad, then $L_1 \otimes L_2$ has a relevant monad.*
   c) *If one of the theories has a presentation consisting solely of constant operations $f : \mathbb{0} \to A$ then both $L_1 + L_2$ and $L_1 \otimes L_2$ have relevant monads.*
3) *The monad maps induced by the canonical morphisms $L_1 \to L_1 \otimes L_2 \leftarrow L_2$ commute.*

The algebraic characterisations also make it particularly easy to verify these characteristics in concrete theories. For example, the pure, read-only, and non-deterministic choice without the constant are all affine. The pure, read-only, write-only, exceptions and non-deterministic choice without constants are all relevant.

V. User-defined Handlers

Definition 14. *The* Algebraic $\mathcal{E}$-MAIL with user-defined handlers *is Algebraic $\mathcal{E}$-MAIL with its syntax extended by*

$$H ::= \ldots \mid \mathtt{handler} \mathbin{@} x\left\langle \mathrm{op}_y(z) \coloneqq M_{\mathrm{op}} \right\rangle_{\mathrm{op}\in\varepsilon_1}$$

and with its type system extended by the rule:

$$\frac{\left\langle x{:}A, y{:}\mathtt{A}_{A_2}, z{:}\mathtt{U}_{\varepsilon_2}((A \times \mathtt{A}_{A_1}) \mathrel{->} \underline{B}) \vdash^{\Delta}_{\varepsilon_2} M_{\mathrm{op}} : \underline{B} \right\rangle_{\mathrm{op}:A_1\to A_2\in\varepsilon_1}}{\vdash^{\Delta}_{\mathrm{h}} \mathtt{handler} \mathbin{@} x\left\langle \mathrm{op}_y(z) \coloneqq M_{\mathrm{op}} \right\rangle_{\mathrm{op}\in\varepsilon_1} : (A;\varepsilon_1) \mathrel{=>} (\underline{B};\varepsilon_2)}$$

The syntax is parametrised by the effect operations in ε_1, so a user-defined handler of the type above contains $|\varepsilon_1|$ terms. Now our language can become infinite. A careful choice of the operations in the signature can maintain a finitary syntax: instead of countably infinite number of operations $\mathrm{op}_n : A_1 \to A_2$, we can choose the single operation $\mathrm{op}_n : A_1 \to \mathbb{N} \otimes A_2$.

Operationally, the terms M_{op} can be thought to re-implement the effects of ε_1 in terms of effects in ε_2. However, there is no guarantee that the user-specified implementation will respect the equations in $\mathcal{L}_{\varepsilon_1}$, and so the semantic function $[\![-]\!]$ is partial. The semantics of the other constructs in the language is defined as before, provided all the sub-terms have well-defined semantics. Recall that a handler H of type $(A;\varepsilon_1) \mathrel{=>} (B;\varepsilon_2)$ is interpreted as a $\mathcal{C}_{\varepsilon_1}$ object satisfying $U_{\varepsilon_1}[\![H]\!]_\delta = U_{\varepsilon_2}[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}$. If $\mathcal{C}_{\varepsilon_1}$ is $\mathsf{Mod}(\mathcal{L}_{\varepsilon_1}, \mathbf{Set})$, then

this requirement amounts to giving a $\mathcal{L}_{\varepsilon_1}$ algebra structure to $U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}$.

For each $\mathrm{op}{:}A_1 \to A_2 \in \varepsilon_1$, we interpret $[\![M_{\mathrm{op}}]\!]_\delta$ as an arrow:

$$[\![A]\!]_\delta \times [\![A_2]\!]_\delta \times U_{\varepsilon_2}\left([\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta \times [\![A_2]\!]_\delta}\right) \to U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta$$

or, equivalently:

$$\left(U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}\right)^{[\![A_1]\!]_\delta} \to \left(U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}\right)^{[\![A_2]\!]_\delta}$$

Thus we have equipped $U_{\varepsilon_2}\,[\![\underline{B}]\!]_\delta^{[\![A]\!]_\delta}$ with an ε_1-algebra structure. However, it may not be an $\mathcal{L}_{\varepsilon_1}$-algebra structure, i.e. the equations might not hold. At this point we use partiality, and define the semantics of the user-defined handler if and only if we obtain an $\mathcal{L}_{\varepsilon_1}$-algebra.

Partial semantics complicates our model theoretic definition of optimisations. We now require a side condition that both sides are defined. It may be possible to devise a program logic that encodes the equations in $\mathcal{L}_{\varepsilon_1}$ as an additional syntactic premise in the rule for handlers.

VI. Decomposing State, IO and Exceptions

We give a modular analysis of a complete effect hierarchy. We assume that the static analyser has sliced the memory into regions, annotating memory access operations.

Let $\mathbf{R} \coloneqq \mathbf{R}_{\mathtt{rw}} \cup \mathbf{R}_{\mathtt{ro}} \cup \mathbf{R}_{\mathtt{wo}}$ be a finite set of regions where $\mathbf{R}_{\mathtt{rw}}$, $\mathbf{R}_{\mathtt{ro}}$ and $\mathbf{R}_{\mathtt{wo}}$ denote read-write, read-only and write-only memory regions. We make use of the notation $\mathbf{R}_{\mathtt{r}} \coloneqq \mathbf{R}_{\mathtt{rw}} \cup \mathbf{R}_{\mathtt{ro}}$ and $\mathbf{R}_{\mathtt{w}} \coloneqq \mathbf{R}_{\mathtt{rw}} \cup \mathbf{R}_{\mathtt{wo}}$ for read-capable and write-capable regions. For each region $r \in \mathbf{R}$ let $\mathbf{L}_r$ be a finite set of locations. Let $\mathbf{V}$ be a countable set of values that can be stored in memory. Given a set R of regions, we denote by $\mathbf{L}_R$ the set of all their locations: $\mathbf{L}_R \coloneqq \sum_{r\in R} \mathbf{L}_r$. Let $\mathbf{E} \coloneqq \mathbf{E}_{\mathtt{n}} \cup \mathbf{E}_{\mathtt{r}}$ be a countable set of exceptions, where $\mathbf{E}_{\mathtt{n}}$ and $\mathbf{E}_{\mathtt{r}}$ denote normal and roll-back exceptions. Let $\mathbf{C} \coloneqq \mathbf{C}_{\mathtt{i}} \cup \mathbf{C}_{\mathtt{o}}$ be a countable set of IO characters, where $\mathbf{C}_{\mathtt{i}}$ and $\mathbf{C}_{\mathtt{o}}$ denote denote input and output characters.

Let Σ be the following $\aleph_1$-signature:

$$\Sigma \coloneqq \left\{ \begin{array}{l} \mathtt{lookup}^{r_{\mathtt{r}}}{:}\mathbf{V} \to \mathbf{L}_{r_{\mathtt{r}}}, \\ \mathtt{update}^{r_{\mathtt{w}}}{:}1 \to \mathbf{L}_{r_{\mathtt{w}}} \otimes \mathbf{V} \\ e{:}\mathbb{0}, \mathtt{getc}{:}\mathbf{C}_{\mathtt{i}} \to \mathbb{1}, \mathtt{putc}{:}\mathbb{1} \to \mathbf{C}_0 \end{array} \middle| \begin{array}{l} r_{\mathtt{r}} \in \mathbf{R}_{\mathtt{r}}, \\ r_{\mathtt{w}} \in \mathbf{R}_{\mathtt{w}}, \\ e \in \mathbf{E} \end{array} \right\}$$

Then the $\aleph_1$-Lawvere theory $\mathcal{L}$ we study is given by:

$$\aleph_{\{\mathtt{getc}\}}^{\mathrm{op}} + \aleph_{\{\mathtt{putc}\}}^{\mathrm{op}} + \aleph_{\mathbf{E}_{\mathtt{n}}}^{\mathrm{op}} + \left(\mathcal{L}_{GS_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{rw}}}} \otimes \mathcal{L}_{R_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{wo}}}} \otimes \mathcal{L}_{OW_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{ro}}}} \otimes \aleph_{\mathbf{E}_{\mathtt{r}}}^{\mathrm{op}}\right)$$

where $\mathcal{L}^{\otimes A}$ is the A-fold tensor of the theory $\mathcal{L}$ with itself.

The corresponding monad is:

$$\mu Z.\big(\mathbf{E}_{\mathtt{r}} + \big(Z^{\mathbf{C}_{\mathtt{i}}} + \mathbf{C}_{\mathtt{o}} \otimes Z + \mathbf{E}_{\mathtt{n}} + -\big) \otimes \mathbf{V}^{\mathbf{L}_{\mathbf{R}_{\mathtt{w}}}}\big)^{\mathbf{L}_{\mathbf{R}_{\mathtt{r}}}}$$

Let $\varepsilon \subseteq \Sigma$ be a finite subset. We use the following abbreviations:

$$\begin{array}{l} \mathbf{R}_{\mathtt{rw}}^{\varepsilon} \coloneqq \{r \in \mathbf{R} | \mathtt{lookup}^r, \mathtt{update}^r \in \varepsilon\} \\ \mathbf{R}_{\mathtt{ro}}^{\varepsilon} \coloneqq \{r \in \mathbf{R} | \mathtt{lookup}^r \in \varepsilon, \mathtt{update}^r \notin \varepsilon\}, \mathbf{R}_{\mathtt{r}}^{\varepsilon} \coloneqq \mathbf{R}_{\mathtt{rw}}^{\varepsilon} \cup \mathbf{R}_{\mathtt{ro}}^{\varepsilon} \\ \mathbf{R}_{\mathtt{wo}}^{\varepsilon} \coloneqq \{r \in \mathbf{R} | \mathtt{lookup}^r \notin \varepsilon, \mathtt{update}^r \in \varepsilon\}, \mathbf{R}_{\mathtt{w}}^{\varepsilon} \coloneqq \mathbf{R}_{\mathtt{rw}}^{\varepsilon} \cup \mathbf{R}_{\mathtt{wo}}^{\varepsilon} \\ \mathbf{E}_{\mathtt{n}}^{\varepsilon} \coloneqq \{e \in \mathbf{E}_{\mathtt{n}} | e \in \varepsilon\}, \mathbf{E}_{\mathtt{r}}^{\varepsilon} \coloneqq \{e \in \mathbf{E}_{\mathtt{r}} | e \in \varepsilon\} \\ \varepsilon_{\mathtt{I}} \coloneqq \varepsilon \cap \{\mathtt{getc}\}, \varepsilon_{\mathtt{0}} \coloneqq \varepsilon \cap \{\mathtt{putc}\} \end{array}$$

By Theorems 7 and 9, $\mathcal{L}_\varepsilon$ is:

$$\aleph_{\varepsilon_{\mathtt{I}}}^{\mathrm{op}} + \aleph_{\varepsilon_{\mathtt{0}}}^{\mathrm{op}} + \aleph_{\mathbf{E}_{\mathtt{n}}^{\varepsilon}}^{\mathrm{op}} + \left(\mathcal{L}_{GS_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{rw}}^{\varepsilon}}} \otimes \mathcal{L}_{R_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{wo}}^{\varepsilon}}} \otimes \mathcal{L}_{OW_{\mathbf{V}}}^{\otimes \mathbf{L}_{\mathbf{R}_{\mathtt{ro}}^{\varepsilon}}} \otimes \aleph_{\mathbf{E}_{\mathtt{r}}^{\varepsilon}}^{\mathrm{op}}\right)$$

The corresponding monad is then:

$$\mu Z.\Big(\mathbf{E}_{\mathtt{r}}^{\varepsilon} + \big(\varepsilon_{\mathtt{I}} \otimes Z^{\mathbf{C}_{\mathtt{i}}} + \varepsilon_{\mathtt{0}} \otimes \mathbf{C}_{\mathtt{o}} \otimes Z + \mathbf{E}_{\mathtt{n}}^{\varepsilon} + -\big) \otimes \mathbf{V}^{\mathbf{L}_{\mathbf{R}_{\mathtt{w}}^{\varepsilon}}}\Big)^{\mathbf{L}_{\mathbf{R}_{\mathtt{ro}}^{\varepsilon}}}$$

As always, all the structural optimisations are valid. As an example equational optimisation, let $\mathtt{deref}_r$ be the generic operation corresponding to $\mathtt{lookup}^r$ and $\mathtt{set}_r$ the generic operation corresponding to $\mathtt{update}^r$:

$$\begin{array}{l} \mathtt{bind}\,\mathtt{set}_r^{\varepsilon}((\ell_0, \underline{1}))\ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{bind}\,\mathtt{deref}_r^{\varepsilon}(\ell_0)\ \mathtt{to}\ y\ \mathtt{in}\ M \\ \qquad = \mathtt{set}_r^{\varepsilon}((\ell_0, \underline{1}))xA\mathtt{bind}\,\mathtt{return}_{\varepsilon}\,\underline{1}\ \mathtt{to}\ y\ \mathtt{in}\ M \end{array}$$

The following proposition analyses validity of the abstract optimisations based on Corollary 13:

Proposition 15. *Let $\varepsilon \subseteq \Sigma$ be any subset.*

1) *The monad for $\mathcal{L}_\varepsilon$ is affine when $\varepsilon \subseteq \{\mathtt{lookup}^r | r \in \mathbf{R}\}$.*
2) *The monad for $\mathcal{L}_\varepsilon$ is relevant when ε is contained in $\{\mathtt{lookup}^r, \mathtt{update}^r, e | r \in \mathbf{R}, e \in \mathbf{E}\}$ and $\mathbf{R}_{\mathtt{rw}}^{\varepsilon} = \emptyset$.*
3) *Let $\varepsilon_1 \subseteq \varepsilon \supseteq \varepsilon_2$ be finite subsets of Σ. If the following five conditions hold:*
   a) $\varepsilon_1 \cup \varepsilon_2 \subseteq \{\mathtt{lookup}^r, \mathtt{update}^r, e | r \in \mathbf{R}, e \in \mathbf{E}_{\mathtt{r}}\}$.
   b) $\mathbf{E}_{\mathtt{r}}^{\varepsilon_1} = \emptyset$ *or* $\mathbf{E}_{\mathtt{r}}^{\varepsilon_2} = \emptyset$.
   c) $(\mathbf{R}_{\mathtt{wo}}^{\varepsilon_1} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_1}) \cap (\mathbf{R}_{\mathtt{wo}}^{\varepsilon_1} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_2}) = \emptyset$.
   d) $(\mathbf{R}_{\mathtt{wo}}^{\varepsilon_1} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_1}) \cap (\mathbf{R}_{\mathtt{ro}}^{\varepsilon_2} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_2}) = \emptyset$
   e) $(\mathbf{R}_{\mathtt{wo}}^{\varepsilon_2} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_2}) \cap (\mathbf{R}_{\mathtt{ro}}^{\varepsilon_1} \cup \mathbf{R}_{\mathtt{rw}}^{\varepsilon_2}) = \emptyset$.

   then the morphisms $\mathcal{L}_{\varepsilon_1} \to \mathcal{L}_\varepsilon \leftarrow \mathcal{L}_{\varepsilon_2}$ commute.

And, as usual, the lambda hoist optimisation applies for $T_\emptyset$.

VII. Recursion

We briefly discuss the addition of recursion, passing from Set to $\omega\mathbf{CPO}$, the category of ω-cpo's and continuous functions. An ω-cpo is a partial order with lubs of increasing ω-sequences; a function is *continuous* if it is monotone and preserves these lubs. We need some $\omega\mathbf{CPO}$-enriched notions: enriched categories, functors, natural transformations, monads and their morphisms. See [16], [31] and the reference therein for material on enrichment. For enriched models of $\mathcal{E}$-MAIL, and its semantics, we proceed analogously to before, but taking objects of $\mathcal{B}$ to be enriched adjunctions $F \vdash U{:}\mathcal{C} \to \omega\mathbf{CPO}$, where $\mathcal{C}$ has enriched binary products and cotensors, and interpreting each basic type by an ω-cpo. If every $T_\varepsilon(P)$ contains $\bot$ we can interpret recursive definitions by the usual

least fixed-points. Such recursive definitions can be added to the intermediate language by the following rule:

$$\frac{\Gamma, x : \mathtt{U}_\varepsilon\, \underline{B} \vdash^{\Delta}_{\varepsilon} M : \underline{B}}{\Gamma \vdash^{\Delta}_{\varepsilon} \mu x : \mathtt{U}_\varepsilon\, \underline{B}.M : \underline{B}}$$

An *enriched* $\aleph_1$*-Lawvere theory* is a strict product-preserving, identity-on-objects functor $J : \aleph_1 \to \mathcal{L}$ where $\mathcal{L}$ is enriched; a morphism of such theories is an enriched morphism of the underlying $\aleph_1$-Lawvere theories. These concepts were introduced in [31], to which we refer for details omitted below. Such theories can be presented by terms t and inequations $t \leq u$ over a $\aleph_1$-ary signature. Examples can be found in [16], [31]; they are typically just the equations discussed above, together with the theory $\mathcal{L}_\Omega$ of a least element, presented by the constant Ω and the inequation $\Omega \leq x$. All our theories will contain $\mathcal{L}_\Omega$. Enriched models and their morphisms are defined in an evident way, free models exist, and the theory can be recovered from the resulting enriched monads by the usual Kleisli construction. The category of theories has all finite sums and a tensor product, and these correspond to sums and tensors of the corresponding enriched monads.

Turning to algebraic $\mathcal{E}$-MAIL, we assume a κ signature that includes a constant Ω and let $\aleph_\Sigma^{\mathrm{op}}$ be the free enriched $\aleph_1$-Lawvere theory for the presentation with that signature and the axiom $\Omega \leq x$. We fix a full functor $\aleph_\Sigma^{\mathrm{op}} \to \mathcal{L}$ that is a morphism of enriched $\aleph_1$-Lawvere theories, and choose a sub-category of $\mathcal{E}$ as before, but ensuring that Ω is in every ε.

A continuous function $f : P \to Q$ is *generative* iff Q is the least sub-ω-cpo of Q containing $f(P)$, and it is *order-reflecting (o.r.)* if $fx \leq fy$ implies $x \leq y$. The generative and o.r. maps form an epi-mono factorisation system, that pointwise induces a factorisation system of the subcategory of discrete enriched $\aleph_1$-Lawvere theories containing $\mathcal{L}_\Omega$ (the proof of the second point uses both discreteness and the existence of a least element, the latter ensuring countable products of the epis—cf Definition 43 of [31]). One can then define conservative restriction models and interpret Algebraic $\mathcal{E}$-MAIL, extended with recursion.

One can now consider combining conservative restrictions. As before, taking sums or tensors of conservative restrictions are conservative for the cases that arise in practise, where now we always include $\mathcal{L}_\Omega$ in the theories. We again omit details, but these are now substantial.

One can then go on to consider optimisations and user-defined handlers.

ACKNOWLEDGEMENTS

We would like to thank Bob Atkey, Nick Benton, Jeff Egger, Sam Lindley, Matija Pretnar and Alex Simpson for valuable comments and suggestions.

REFERENCES

[1] J. M. Lucassen and D. K. Gifford, "Polymorphic effect systems," ser. POPL '88, 1988, pp. 47–57.

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

[3] A. D. Gordon and A. Jeffrey, "Types and effects for asymmetric cryptographic protocols," *J. Comput. Secur.*, vol. 12, pp. 435–483, 2004.

[4] M. Tofte and J.-P. Talpin, "Region-based memory management," *Information and Computation*, vol. 132, no. 2, pp. 109 – 176, 1997.

[5] N. Benton and A. Kennedy, "Monads, effects and transformations," in *ENTCS*, 1999, pp. 1–18.

[6] N. Benton et al., "Reading, writing and relations," in *Programming Languages and Systems*, ser. LNCS, N. Kobayashi, Ed., 2006, vol. 4279, pp. 114–130.

[7] N. Benton and P. Buchlovsky, "Semantics of an effect analysis for exceptions," ser. TLDI '07. ACM, 2007, pp. 15–26.

[8] N. Benton, et al., "Relational semantics for effect-based program transformations with dynamic allocation." ACM, 2007, pp. 87–96.

[9] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann, "Relational semantics for effect-based program transformations: higher-order store," in *PPPDP'09*, 2009, pp. 301–312.

[10] D. Marino and T. Millstein, "A generic type-and-effect system," in *TLDI '09: Proc. 4th intl. work. on Types in lang. design and impl.*, 2009, pp. 39–50.

[11] P. Wadler, "The marriage of effects and monads," in *ICFP '98*. ACM, 1998, pp. 63–74.

[12] A. P. Tolmach, "Optimizing ML using a hierarchy of monadic types," in *Types in Compilation*, 1998, pp. 97–115.

[13] R. B. Kieburtz, "Taming effects with monadic typing," in *ICFP*, 1998, pp. 51–62.

[14] G. D. Plotkin and J. Power, "Notions of computation determine monads," in *LNCS'02*. Springer, 2002, pp. 342–356.

[15] ——, "Computational effects and operations: An overview," *Electr. Notes Theor. Comput. Sci.*, vol. 73, pp. 149–163, 2004.

[16] M. Hyland, G. D. Plotkin, and J. Power, "Combining effects: Sum and tensor," *Theor. Comput. Sci.*, vol. 357, no. 1-3, pp. 70–99, 2006.

[17] G. D. Plotkin and M. Pretnar, "Handlers of algebraic effects," in *ESOP*, 2009, pp. 80–94.

[18] P. B. Levy, "Call-by-push-value: A subsuming paradigm," in *TLCA*, 1999, pp. 228–242.

[19] A. Filinski, "Representing layered monads," in *POPL '99*, 1999, pp. 175–188.

[20] P. B. Levy, *Call-By-Push-Value: A Functional/Imperative Synthesis*, ser. Semantics Structures in Computation. Springer, 2004, vol. 2.

[21] M. Pretnar, "The logic and handling of algebraic effects," Ph.D. dissertation, University of Edinburgh, 2009.

[22] N. Benton and A. Kennedy, "Exceptional syntax," *J. Funct. Program.*, vol. 11, pp. 395–410, July 2001.

[23] F. Borceux, *Handbook of Categorical Algebra 2. Categories and Structures*, ser. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994, vol. 51.

[24] A. J. Power, "Enriched lawvere theories," *Theory and Applications of Categories*, vol. 6, pp. 83–93, 2000.

[25] G. D. Plotkin and J. Power, "Semantics for algebraic operations," *Electr. Notes Theor. Comput. Sci.*, vol. 45, pp. 332 – 345, 2001.

[26] E. Moggi, "A modular approach to denotational semantics," in *Category Theory and Computer Science*, 1991, pp. 138–139.

[27] M. Jaskelioff and E. Moggi, "Monad transformers as monoid transformers," *Theor. Comput. Sci.*, vol. 411, no. 51-52, pp. 4441–4466, 2010.

[28] B. Jacobs, "Semantics of weakening and contraction," *Annals of Pure and Applied Logic*, vol. 69, no. 1, pp. 73 – 106, 1994.

[29] A. Kock, "Bilinearity and cartesian closed monads," *Mathematica Scandinavia*, vol. 29, pp. 161 – 174, 1971.

[30] C. Führmann, "The structure of call-by-value," School of Informatics, University of Edinburgh, PhD Thesis, 2000.

[31] M. Hyland and J. Power, "Discrete lawvere theories and computational effects," *TCS*, vol. 366, no. 1-2, pp. 144 – 162, 2006.