Monads and Adjunctions for Global Exceptions

Paul Blain Levy [1]

School of Computer Science
University of Birmingham
Birmingham B15 2TT, U.K.

Abstract

In this paper, we look at two categorical accounts of computational effects (strong monad as a model of the monadic metalanguage, adjunction as a model of call-by-push-value with stacks), and we adapt them to incorporate global exceptions. In each case, we extend the calculus with a construct, due to Benton and Kennedy, that fuses exception handling with sequencing. This immediately gives us an equational theory, simply by adapting the equations for sequencing. We study the categorical semantics of the two equational theories.

In the case of the monadic metalanguage, we see that a monad supporting exceptions is a coalgebra for a certain comonad. We further show, using Beck's theorem, that, on a category with equalizers, the monad constructor for exceptions gives all such monads.

In the case of call-by-push-value (CBPV) with stacks, we generalize the notion of CBPV adjunction so that a stack awaiting a value can deal both with a value being returned, and with an exception being raised. We see how to obtain a model of exceptions from a CBPV adjunction, and vice versa by restricting to those stacks that are homomorphic with respect to exception raising.

Key words: exception handling, monad, adjunction, comonad, coalgebra, call-by-push-value, Beck's theorem

1 Introduction

1.1 Monads For Exceptions

In a seminal paper [19], Moggi brought together a range of imperative behaviours that have come to be called *computational effects*, including divergence, nondeterminism, storage and exceptions. The present paper is a study of the last example. For the sake of precision, let us distinguish some variations on the effect of exceptions.

[1] Email: `pbl@cs.bham.ac.uk`

- Some languages, such as Java, distinguish between *errors* and *exceptions*. The former cannot be caught: when an error is raised, execution immediately terminates. By contrast, exceptions can be caught (aka handled).
- Some languages, such as ML, allow the dynamic generation of new exception names, and this is modelled using games in [8]. But in the present paper, we look at global exceptions only.

In the study of effects, various calculi have been studied, including computational λ-calculus [18], monadic metalanguage [19], fine-grain call-by-value [12], call-by-push-value (CBPV) [11], CBPV with stacks [14]. And various categorical structures have been studied as models of these calculi, including strong monads [19], Freyd categories [16], κ-categories [16], CBPV adjunctions [14].

None of the above calculi contain constructs for specific effects; such constructs (and associated equations) must be added on to them. The same is true for the categorical structures. Take strong monads, for example. The analysis of various effects in [21], makes it possible to axiomatize the additional structure[2] that a strong monad T on $\mathcal{C}$ should be provided with.

- To model binary erratic nondeterminism, T should be equipped with a morphism $1 \xrightarrow{\mathsf{or}} T(1+1)$ satisfying equations for commutativity, associativity and idempotence.
- To model a ground storage cell, where S is the countable set of elements that can be stored, T should be equipped with a morphisms $1 \xrightarrow{\mathsf{lookup}} T\sum_{s\in S} 1$ and $\sum_{s\in S} 1 \xrightarrow{\mathsf{update}} T1$ satisfying some equations given in [21]; and similarly for several cells.
- To model printing, where $\mathcal{A}$ is the countable set of characters that can be printed, T should be equipped with a morphism $\sum_{c\in\mathcal{A}} 1 \xrightarrow{\mathsf{print}} T1$.
- To model raising of errors, where E is the countable set of errors, T should be equipped with a morphism $\sum_{e\in E} 1 \xrightarrow{\mathsf{error}} T0$.
- To model I/O, where O is the countable set of messages requesting input, and message $o \in O$ requests an input from the countable set I_o, the monad T should be equipped with a morphism $1 \xrightarrow{\mathsf{input}(o)} T\sum_{i\in I_o} 1$ for each $o \in O$. (This generalizes the previous two examples. If I_o is singleton, then o is a print-character. If I_o is empty, the o is an error message.)

All these effects are instances of a general theory developed in [21,22]. This theory involves the notion of an *algebraic operation*, i.e. an operation θ on terms such that

$$\theta\{M_i\}_{i\in I}\ \mathtt{to\ x.}\ N = \theta\{M_i\ \mathtt{to\ x.}\ N\}_{i\in I}$$

where I is the (countable) arity of θ and `to` means sequencing.

[2] We assume that the category $\mathcal{C}$ has countable coproducts of a suitable kind. Below (Def. 2.1) we state this assumption precisely.

By contrast, as explained in [22], exception handling is not an algebraic operation. So how can we add exceptions to the monadic metalanguage? What equations should be imposed, and what is the resulting categorical structure? These questions will be answered in the first part of the paper (Sect. 2–5). We shall see that the categorical structure is actually a *coalgebra* for a certain comonad.

A large class of monads for exceptions is given by a *monad transformer* that takes a monad T to $T(-+E)$ [4]. Surprisingly, we shall prove that *every* monad for exceptions (on a category with equalizers) is of this form.

1.2 Adjunctions For Exceptions

CBPV is a fine-grain calculus that includes both call-by-value and call-by-name as fragments [11,13]. It was shown in [14], that when we extend CBPV with a judgement for stacks (aka evaluation contexts), its categorical semantics is given by an adjunction. As an example, define an *E-set* to be a set X equipped with a function $E \longrightarrow X$. Then the adjunction between the category of sets and that of E-sets gives a model of errors. A stack denotes a morphism of the second category, in this example, an E-set homomorphism. Intuitively, an evaluation context, applied to a term that raises an error, gives a term that raises an error.

As noted in [14], this theory cannot account for exception handling. Stacks that involve handlers may treat an error in a non-homomorphic way. The problem here is more severe than in Sect. 1.1, because exception handling actually invalidates some of the equational laws of CBPV with stacks. (A similar phenomenon is noted in [9]: exception handling invalidates some of the standard laws for continuations.) So what is the appropriate equational theory, and what is the resulting categorical structure? These questions will be answered in the second part of the paper (Sect. 6–7).

1.3 Combining Handling With Sequencing

Both of our questions rely on finding a reasonable set of equations for raising and handling of exceptions. Using conventional syntax for handling, this would seem to be difficult. But in [2] a novel syntax was introduced for *handle-sequencing*:

$$M\ \{\mathtt{to}\ \mathtt{x}.\ N, \mathtt{catch}\ \mathtt{x}.\ N'\}$$

This means: first evaluate M. If it returns a value, bind `x` to that value and evaluate N. On the other hand, if it raises an exception, bind `x` to that exception and evaluate N'.

Many advantages of this syntax—which is equivalent, in the presence of sum types, to the traditional syntax—are discussed in [2]. But what is useful for us is that it is so similar to ordinary sequencing. So all we need to do is take the standard equations for sequencing and adapt them to this construct.

This gives an elegant theory in both of the calculi we are considering: monadic metalanguage, and CBPV with stacks.

1.4 Theories and Categorical Structures

In the course of the paper, we present various equational theories and various categorical structures. We relate these with results asserting a "correspondence" between theory and structure. This asserts an equivalence between the category $\mathcal{A}$ of theories (where a theory consists of a signature and a congruence generated by the signature) and the category $\mathcal{B}$ of structures. See e.g. [14] for a precise statement of this equivalence in the specific case of CBPV adjunctions; this statement is easily adapted to the other cases. As explained there, the morphisms in both $\mathcal{A}$ and $\mathcal{B}$ are required to preserve structure on the nose; this is a flaw pervasive in categorical semantics, whose rectification is left to future work.

2 Monadic Metalanguage

The monadic metalanguage, with finite products and sum types, is shown in Fig. 1, along with syntax for exception raising. Here, `pm` abbreviates "pattern-match", and we use `to` for sequencing. Because the rules for 1 are analogous to those for $\times$, we omit them. Throughout the paper, all constructs and equations that involve sequencing are marked ♣, because they are the ones that will need to be adapted when we add handle-sequencing into the language.

The equational theory is shown in Fig. 2. We omit the assumptions necessary to make each equation well-typed. Given a term $\Gamma \vdash M : B$ we write ${}^{\mathtt{x}}M$ for the weakened term in the context $\Gamma, \mathtt{x} : A$ where A is some suitable type. This implies that $\mathtt{x}$ is not in Γ, because the identifiers in a context must be distinct. We thereby obviate the need for the traditional $\mathtt{x} \notin \mathsf{FV}(M)$ conditions.

To interpret sum types, we adapt the following from [3,5].

Definition 2.1 Let $\mathcal{C}$ be a cartesian category, i.e. a category with distinguished terminal object and distinguished binary products.

(i) A *distributive coproduct* for a family of $\mathcal{C}$-objects $\{A_i\}_{i\in I}$ is a cocone $(V, \{\, A_i \xrightarrow{\mathsf{in}_i} V \,\}_{i\in I})$ such that, for every $\mathcal{C}$-object X, the cocone $(X \times V, \{\, X \times A_i \xrightarrow{X\times\mathsf{in}_i} X \times V \,\}_{i\in I})$ is a coproduct.

(ii) A *distributive* (resp. *countably distributive*) category is a cartesian category with a distributive coproduct for every finite (resp. countable) family of objects.

(iii) Let T be a strong monad on $\mathcal{C}$.
- T has *Kleisli exponentials* when it is equipped, for every pair of $\mathcal{C}$-objects A, B, with a representing object for the functor $\mathcal{C}(- \times A, TB) : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$.

$$\textbf{Types}\quad A ::= \quad \textstyle\sum_{i\in I} A_i \mid 1 \mid A \times A \mid A \rightharpoonup A \quad (I \text{ finite})$$

$$\frac{}{\Gamma \vdash \mathtt{x} : A}\,(\mathtt{x}:A)\in\Gamma$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{return}\ M : TA} \qquad \frac{\Gamma \vdash M : TA \quad \Gamma, \mathtt{x} : A \vdash N : TB}{\Gamma \vdash M\ \mathtt{to}\ \mathtt{x}.\ N : TB}\,\clubsuit$$

$$\frac{\Gamma \vdash M : A_{\hat\imath}}{\Gamma \vdash \langle \hat\imath, M\rangle : \sum_{i\in I} A_i}\,\hat\imath \in I \quad \frac{\Gamma \vdash M : \sum_{i\in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash N_i : B\ (\forall i \in I)}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle . N_i\}_{i\in I}}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash M' : A'}{\Gamma \vdash \langle M, M'\rangle : A \times A'} \qquad \frac{\Gamma \vdash M : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash N : B}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle .\ N : B}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash M : TB}{\Gamma \vdash \lambda\mathtt{x}.M : A \rightharpoonup B} \qquad \frac{\Gamma \vdash M : A \rightharpoonup B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : TB}$$

Fig. 1. Syntax Of Monadic Metalanguage And Exception Raising

Laws of Sequencing

$\clubsuit$ $(\mathtt{return}\ M)\ \mathtt{to}\ \mathtt{x}.\ N = N[M/\mathtt{x}]$

$\clubsuit$ $M = M\ \mathtt{to}\ \mathtt{x}.\ \mathtt{return}\ \mathtt{x}$

$\clubsuit$ $(M\ \mathtt{to}\ \mathtt{x}.\ N)\ \mathtt{to}\ \mathtt{y}.\ P = M\ \mathtt{to}\ \mathtt{x}.\ (N\ \mathtt{to}\ \mathtt{y}.\ {}^{\mathtt{x}}P)$

β-laws

$\mathtt{pm}\ \langle \hat\imath, M\rangle\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle . N_i\}_{i\in I} = N_{\hat\imath}[M/\mathtt{x}]$

$\mathtt{pm}\ \langle M, M'\rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle . N = N[M/\mathtt{x}, M'/\mathtt{y}]$

$(\lambda\mathtt{x}.M)N = M[N/\mathtt{x}]$

η-laws

$N[M/\mathtt{z}] = \mathtt{pm}\ M\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle .\ {}^{\mathtt{x}}N[\langle i, \mathtt{x}\rangle/\mathtt{z}]\}_{i\in I}$

$N[M/\mathtt{z}] = \mathtt{pm}\ M\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle .\ {}^{\mathtt{xy}}N[\langle \mathtt{x}, \mathtt{y}\rangle/\mathtt{z}]$

$M = \lambda\mathtt{x}.({}^{\mathtt{x}}M\mathtt{x})$

Fig. 2. Equations For Monadic Metalanguage

- T has *countable products of Kleisli exponentials* when it is equipped, for every countable family of pairs of $\mathcal{C}$-objects $\{(A_i, B_i)\}_{i\in I}$, with a representing object for the functor $\prod_{i\in I} \mathcal{C}(- \times A_i, TB_i) : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathbf{Set}$. (This clearly implies that T has Kleisli exponentials.)

□

Proposition 2.2 There is a theory/model correspondence (see Sect. 1.4) between

- a theory of the monadic metalanguage
- a distributive category, together with a strong monad with Kleisli exponentials.

□

Remark 2.3 An infinitary variant of the metalanguage can be formed by including both countable sum types, and a fusion of function type and countable product types in the style of [15], to the calculus. There is then a correspondence between

- a theory of the infinitary monadic metalanguage
- a countably distributive category, together with a strong monad with countable products of Kleisli exponentials.

□

3 Exceptions

3.1 Raising Exceptions

Types with Exceptions $A ::= \sum_{i \in I} A_i \mid 1 \mid A \times A \mid A \rightharpoonup A \mid \mathtt{exn} \quad (I \text{ finite})$

Exception Raising $$\frac{\Gamma \vdash M : \mathtt{exn}}{\Gamma \vdash \mathtt{raise}_B M : TB}$$

Raising Is Algebraic

$\clubsuit \quad (\mathtt{raise}\ M)\ \mathtt{to}\ \mathtt{x}.\ N = \mathtt{raise}\ M$

Fig. 3. Syntax and Equations For Exception Raising In Monadic Metalanguage

We first treat the *raising* of exceptions, which requires an additional type exn, as shown in Fig. 3.1. Exception raising is an *algebraic effect*, in the sense of [20], and consequently its semantics is very simple—unlike that of exception *handling*. Indeed, the semantics of raise is determined by that of the computation $\mathtt{x} : \mathtt{exn} \vdash \mathtt{raise}\ \mathtt{x} : T0$, because

$$\mathtt{raise}_B V = (\mathtt{raise}_0 \mathtt{x})[V/\mathtt{x}]\ \mathtt{to}\ \mathtt{y}.\ \mathtt{raise}_B V$$

This is an instance of a general result [22]: an algebraic operation corresponds to a generic element. We define a semantic structure accordingly:

Definition 3.1 Let $\mathcal{C}$ be a distributive category, and let E be an object of $\mathcal{C}$. A *strong monad supporting E-raising* is a strong monad T on $\mathcal{C}$ together with a $\mathcal{C}$-morphism from E to $T0$. $\square$

The "monad constructor for exceptions" [4] provides the main class of examples.

Definition 3.2 Let $\mathcal{C}$ be a distributive category, with an object E. Let T be a strong monad on $\mathcal{C}$. Then we define T_E to be the strong monad $T(- + E)$. $\square$

Note that if T has Kleisli exponentials (resp. countable products of Kleisli exponentials) then so does T_E.

Proposition 3.3 There is a theory-model correspondence (Sect. 1.4) between

- a theory of the monadic metalanguage with exception raising
- a distributive category $\mathcal{C}$, with distinguished object E and a strong monad on $\mathcal{C}$, with Kleisli exponentials, supporting E-raising.

$\square$

3.2 Exception Handling—The Syntax

The changes required to obtain the monadic metalanguage with exceptions are shown in Fig. 4. We *define*

$$M \;\texttt{to x.}\; N \;\text{ to be }\; M \left\{ \begin{array}{ll} \texttt{to x.} & N \\ \texttt{catch x.} & \texttt{raise x} \end{array} \right\}$$

$$M \;\texttt{catch x.}\; N \;\text{ to be }\; M \left\{ \begin{array}{ll} \texttt{to x.} & \texttt{return x} \\ \texttt{catch x.} & N \end{array} \right\}$$

We can then prove all the equations marked ♣ (that are no longer needed as axioms), and the following:

$$\begin{array}{rcll}
(\texttt{raise } V) \texttt{ catch x. } M & = & M[V/\texttt{x}] & (1) \\
(\texttt{return } V) \texttt{ catch x. } M & = & \texttt{return } V & (2) \\
M & = & M \texttt{ catch x. raise x} & (3) \\
(M \texttt{ catch x. } N) \texttt{ catch y. } & P = & & \\
M \texttt{ catch x} & (N \texttt{ catch y. } {}^{\texttt{x}}P) & & (4) \\
((M \texttt{ catch w. return } V) \texttt{ to x. } N) \texttt{ catch y. } & P = & & \\
(M \texttt{ catch w. return } V) \texttt{ to x. } & (N \texttt{ catch y. } {}^{\texttt{x}}P) & & (5) \\
((M \texttt{ to w. raise } V) \texttt{ catch x. } N) \texttt{ to y. } & P = & & \\
(M \texttt{ to w. raise } V) \texttt{ catch x . } & (N \texttt{ to y. } {}^{\texttt{x}}P) & & (6) \\
M\{\texttt{to x. } N, \texttt{catch x. } N'\} & = & & \\
((M \texttt{ to w. return inl w}) \texttt{ catch y. } & \texttt{return inr y}) \texttt{ to z.} & & \\
\texttt{pm z} & \texttt{as} & \{\texttt{inl x. } {}^{\texttt{z}}N, \texttt{inr x. } {}^{\texttt{z}}N'\} & (7)
\end{array}$$

The following constructs and equations replace those marked ♣ in Fig. 1–3.1.

$$\frac{\Gamma \vdash M : TA \quad \Gamma, \mathtt{x} : A \vdash N : TB \quad \Gamma, \mathtt{x} : \mathtt{exn} \vdash N' : TB}{\Gamma \vdash M\ \{\mathtt{to}\ \mathtt{x}.\ N, \mathtt{catch}\ \mathtt{x}.\ N'\} : TB}$$

$$(\mathtt{return}\ M) \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{x}. & N \\ \mathtt{catch}\ \mathtt{x}. & N' \end{array} \right\} = N[M/\mathtt{x}]$$

$$(\mathtt{raise}\ M) \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{x}. & N \\ \mathtt{catch}\ \mathtt{x}. & N' \end{array} \right\} = N'[M/\mathtt{x}]$$

$$M = M \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{x}. & \mathtt{return}\ \mathtt{x} \\ \mathtt{catch}\ \mathtt{x}. & \mathtt{raise}\ \mathtt{x} \end{array} \right\}$$

$$(M \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{x}. & N \\ \mathtt{catch}\ \mathtt{x}. & N' \end{array} \right\}) \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{y}. & P \\ \mathtt{catch}\ \mathtt{y}. & P' \end{array} \right\} =$$

$$M \left\{ \begin{array}{ll} \mathtt{to}\ \mathtt{x}. & N\ \{\mathtt{to}\ \mathtt{y}.\ {}^{\mathtt{x}}P, \mathtt{catch}\ \mathtt{y}.\ {}^{\mathtt{x}}P'\} \\ \mathtt{catch}\ \mathtt{X}. & N'\ \{\mathtt{to}\ \mathtt{y}.\ {}^{\mathtt{x}}P, \mathtt{catch}\ \mathtt{y}.\ {}^{\mathtt{x}}P'\} \end{array} \right\}$$

Fig. 4. Handle-Sequencing In The Monadic Metalanguage

Equations (1)–(6) give properties of plain handling. (7) shows that handle-sequencing is no more expressive (in the presence of sum types) than ordinary handling and sequencing; it is merely a syntactic convenience.

To give categorical semantics for handle-sequencing will require more sophisticated machinery, which we now develop. But for the moment, note that T_E always gives a model.

4 Coalgebras On Algebras

In this section, we review and develop some abstract theory of coalgebras.

Proposition 4.1 Let $\mathcal{A} \underset{G}{\overset{F}{\rightleftarrows}} \mathcal{B}$ (with $\perp$) be an adjunction with unit η and counit ϵ. Write L for the induced comonad $(FG, \epsilon, F\eta G)$.

(i) There is a unique comparison K from the resolution $(\mathcal{A}, F, G, \eta)$ of L to the co-Eilenberg-Moore resolution (into the category of L-coalgebras). It maps an $\mathcal{A}$-object X to $(FX, F\eta X)$, and a morphism $X \xrightarrow{f} X'$ to Ff.

(ii) Suppose $\mathcal{A}$ has all equalizers, and F preserves them. Then K has a right adjoint Q (not necessarily a comparison) and the counit of $K \dashv Q$ is an isomorphism.

□

Proof. This is proved in [1]. The right adjoint of K maps an L-coalgebra (Y,ϕ) to the equalizer in $\mathcal{A}$ of

$$\begin{tikzcd} GY \arrow[r, shift left, "G\phi"] \arrow[r, shift right, "\eta GY"'] & GFGY \end{tikzcd}$$

□

Now suppose that $(\mathcal{T},\eta,\mu)$ is a monad on a category $\mathcal{M}$. We form the following two adjunctions:

$$\begin{tikzcd} \mathcal{M} \arrow[rd, shift left] \arrow[dd, dashed, "K(\mathcal{T})"'] & & \text{Eilenberg-Moore} \\ & \mathcal{M}^{\mathcal{T}} \arrow[lu, shift left] \arrow[lu, phantom, "\perp"] \arrow[ld, shift left] & \\ (\mathcal{M}^{\mathcal{T}})^{\mathcal{L}(\mathcal{T})} \arrow[ru, shift left] \arrow[ru, phantom, "\perp"] & & \text{co-Eilenberg-Moore} \end{tikzcd}$$

This proceeds in three steps.

The first step is to form the Eilenberg-Moore resolution $(\mathcal{M}^T, G^T, F^T, \epsilon^T)$ of $\mathcal{T}$. This induces a comonad on $\mathcal{M}^{\mathcal{T}}$, which we call $\mathcal{L}(\mathcal{T})$. Explicitly:

- it maps an object (X,θ) to $(\mathcal{T}X,\mu X)$
- it maps a morphism $(X,\theta)\xrightarrow{f}(Y,\theta')$ to $(\mathcal{T}X,\mu X)\xrightarrow{\mathcal{T}f}(\mathcal{T}Y,\mu Y)$
- the counit at (X,θ) is $(\mathcal{T}X,\mu X)\xrightarrow{\theta}(X,\theta)$
- the comultiplication at (X,θ) is $(\mathcal{T}X,\mu X)\xrightarrow{\mathcal{T}\eta X}(\mathcal{T}^2X,\mu\mathcal{T}X)$.

The second step is to form the co-Eilenberg-Moore resolution of this comonad. Explicitly:

- a coalgebra for $\mathcal{L}(\mathcal{T})$ is (X,θ,ϕ), where (X,θ) is a T-algebra, and $(X,\theta)\xrightarrow{\phi}(\mathcal{T}X,\mu X)$ is a T-algebra homomorphism such that

$$\begin{tikzcd} X \arrow[r, "\phi"] \arrow[rd, "\mathsf{id}"'] & \mathcal{T}X \arrow[d, "\theta"] \\ & X \end{tikzcd} \qquad \begin{tikzcd} X \arrow[r, "\phi"] \arrow[d, "\phi"'] & \mathcal{T}X \arrow[d, "\mathcal{T}\phi"] \\ \mathcal{T}X \arrow[r, "\mathcal{T}\eta X"'] & \mathcal{T}^2X \end{tikzcd} \quad \text{commute}$$

- a coalgebra homomorphism from (X,θ,ϕ) to (Y,θ',ϕ') is a T-algebra homomorphism $(X,\theta) \xrightarrow{f} (Y,\theta')$ such that

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, "\phi"'] & Y \arrow[d, "\phi"] \\ \mathcal{L}X \arrow[r, "\mathcal{L}f"'] & \mathcal{L}Y \end{tikzcd} \quad \text{commutes}$$

- the forgetful functor (the left adjoint) maps an object (X,θ,ϕ) to (X,θ) and a morphism $(X,\theta,\phi) \xrightarrow{f} (Y,\theta',\phi')$ to f
- the free functor (the right adjoint) maps an object (X,θ) to $(\mathcal{T}X,\mu X,\mathcal{T}\eta X)$ and a morphism $(X,\theta) \xrightarrow{f} (Y,\theta')$ to $\mathcal{T}f$
- the unit at (X,θ,ϕ) is ϕ.

The third step is to look at the unique comparison from our first resolution of $\mathcal{L}(\mathcal{T})$ to the co-Eilenberg-Moore resolution, which is terminal. It is a functor from $\mathcal{M}$ to the coalgebra category, mapping

- an object X to $(\mathcal{T}X,\mu X,\mathcal{T}\eta X)$
- a morphism $X \xrightarrow{f} Y$ to $\mathcal{T}f$.

We call this comparison $K(\mathcal{T})$.

Proposition 4.2 Let $(\mathcal{T},\eta,\mu)$ be a monad on a category $\mathcal{M}$. Suppose $\mathcal{M}$ has all equalizers, and $\mathcal{T}$ preserves them. Then $K(\mathcal{T})$ has a right adjoint Q, and the counit of $K \dashv Q$ is an isomorphism. □

Proof. Since $\mathcal{T}$ preserves equalizers, the free algebra functor F^T must do so too. We then apply Prop. 4.1(ii). □

5 Monad Models For Exceptions

5.1 General Monads

In this section, let $\mathcal{C}$ be a distributive category, and E an object of it. We define $\mathcal{M}_{\mathcal{C}}$ to be the category of strong monads on $\mathcal{C}$. We first recall the following result, mentioned in [7].

Proposition 5.1 Let T be a strong monad on $\mathcal{C}$. Then T_E is a coproduct of T and $-+E$. The injection $T \longrightarrow T_E$ is given at A by $TA \xrightarrow{T\mathsf{inl}_{A,E}} T(A+E)$. The injection $-+E \longrightarrow T_E$ is given at A by $A+E \xrightarrow{\eta(A+E)} T(A+E)$. □

Now, in general, if $\mathcal{E}$ is an object of a category $\mathcal{M}$ such that every $\mathcal{M}$-object U has a coproduct with $\mathcal{E}$, then $U \mapsto U+\mathcal{E}$ gives a monad on $\mathcal{M}$. So in particuar, we obtain a monad $\mathcal{T}_{\mathcal{C},E}$ that maps T to T_E. Its unit at T maps

X to $TX \xrightarrow{T\mathsf{inl}} T(X+E)$. The multiplication at T maps X to

$$T((X+E)+E) \xrightarrow{T[\mathsf{id},\mathsf{inr}]} T(X+E)$$

Furthermore, let us write $\mathcal{M}^{kl}_{\mathcal{C}}$ (resp. $\mathcal{M}^{\omega kl}_{\mathcal{C}}$) for the full subcategory of $\mathcal{M}_{\mathcal{C}}$ consisting of strong monads with Kleisli exponentials (resp. countable products of Kleisli exponentials). Then $\mathcal{T}_{\mathcal{C},E}$ restricts to a monad on $\mathcal{M}^{kl}_{\mathcal{C}}$ (resp. on $\mathcal{M}^{\omega kl}_{\mathcal{C}}$), though the strong monad $- + E$ might lack Kleisli exponentials. We call this restricted monad $\mathcal{T}^{kl}_{\mathcal{C},E}$ (resp. $\mathcal{T}^{\omega kl}_{\mathcal{C},E}$).

We can now formulate our main definition.

Definition 5.2
- A *strong monad supporting E-exceptions* on $\mathcal{C}$ is a coalgebra for $\mathcal{L}(\mathcal{T}_{\mathcal{C},E})$.
- A *strong monad with Kleisli exponentials* (resp. *with countable products of Kleisli exponentials*) *supporting E-exceptions* is a coalgebra for $\mathcal{L}(\mathcal{T}^{kl}_{\mathcal{C},E})$ (resp. $\mathcal{L}(\mathcal{T}^{\omega kl}_{\mathcal{C},E})$).

□

Let us unpack this definition.

Firstly, an algebra for the $\mathcal{T}_{\mathcal{C},E}$ monad is precisely a strong monad T with a strong monad morphism from $-+E$ to T, and such a strong monad morphism corresponds to a $\mathcal{C}$-morphism $E \xrightarrow{\mathsf{raise}} T0$ (by the general theory of algebraic operations [22]). Thus, an algebra is a strong monad on $\mathcal{C}$ supporting E-raising. The algebra structure θ is given at X by

$$T(X+E) \xrightarrow{T[\eta X,(\mathsf{raise};T[])]} T^2X \xrightarrow{\mu X} TX$$

A coalgebra for the induced comonad consists of a strong monad T supporting E-raising, together with, for each $\mathcal{C}$-object X, a morphism $TX \xrightarrow{eX} T(X+E)$ that is natural in X, is strong monad homomorphic

$$\begin{tikzcd} & X \arrow[dl, "\eta X"'] \arrow[dr, "\eta\,\mathsf{inl}_{X,E}"] & \\ TX \arrow[rr, "eX"'] & & T(X+E) \end{tikzcd}$$

$$\begin{tikzcd} T^2X \arrow[r, "e^2X"] \arrow[dd, "\mu X"'] & T(T(X+E)+E) \arrow[d, "T({[\mathsf{id},\eta\,\mathsf{inr}_{X,E}]})"] \\ & T^2(X+E) \arrow[d, "\mu(X+E)"] \\ TX \arrow[r, "eX"'] & T(X+E) \end{tikzcd}$$

$$\begin{tikzcd}
TX \times Y \arrow[r, "(eX)\times Y"] \arrow[dd, "t_{X,Y}"'] & T(X+E)\times Y \arrow[d, "t_{X+E,Y}"] \\
 & T((X+E)\times Y) \arrow[d] \\
T(X\times Y) \arrow[r, "(eX\times Y)"'] & T(X\times Y+E)
\end{tikzcd}$$

is a $\mathcal{T}_{\mathcal{C},E}$-algebra homomorphism

$$\begin{tikzcd}
T(X+E) \arrow[r, "eX+E"] \arrow[d, "\theta X"'] & T((X+E)+E) \arrow[d, "T[\mathsf{id},\mathsf{inr}_{X+E,E}]"] \\
TX \arrow[r, "ex"'] & T(X+E)
\end{tikzcd} \tag{8}$$

and is coalgebraic

$$\begin{tikzcd}
TX \arrow[r, "eX"] \arrow[dr, "\mathsf{id}"'] & T(X+E) \arrow[d, "\theta X"] \\
 & TX
\end{tikzcd}$$

$$\begin{tikzcd}
TX \arrow[r, "eX"] \arrow[d, "eX"'] & T(X+E) \arrow[d, "e(X+E)"] \\
T(X+E) \arrow[r, "T\mathsf{inl}_{X+E,E}"'] & T((X+E)+E)
\end{tikzcd}$$

Proposition 5.3 The condition (8) is equivalent, in the presence of all the other equations, to the condition

$$\begin{tikzcd}
E \arrow[r, "\mathsf{raise}"] \arrow[dr, "\eta\,\mathsf{inr}_{0,E}"'] & T0 \arrow[d, "e0"] \\
 & T(0+E)
\end{tikzcd}$$

□

5.2 Monad Semantics Of Exceptions

The above structure is precisely what we require to interpret handling. For given terms $\Gamma \vdash M : TA$ and $\Gamma, \mathtt{x} : A \vdash N : TB$ and $\Gamma, \mathtt{x} : \mathtt{exn} \vdash N' : TB$, the term $M\ \{\mathtt{to\ x}.\ N, \mathtt{catch\ x}.\ N'\}$ denotes the composite

$$\begin{tikzcd}
\llbracket\Gamma\rrbracket \arrow[r, "\langle\mathsf{id},\llbracket M\rrbracket\rangle"] & \llbracket\Gamma\rrbracket \times T\llbracket A\rrbracket \arrow[r, "\llbracket\Gamma\rrbracket\times e\llbracket A\rrbracket"] & \llbracket\Gamma\rrbracket \times T(\llbracket A\rrbracket + E) \arrow[r, "t"] & T(\llbracket\Gamma\rrbracket \times (\llbracket A\rrbracket + E)e) \arrow[d, "T[\llbracket N\rrbracket,\llbracket N\rrbracket]"] \\
 & & & T\llbracket B\rrbracket
\end{tikzcd}$$

It is then easy to see that all the equational laws of Fig. 2 are validated.

Conversely, we can construct such a coalgebra out of the syntax of handling. We define eA to be the congruence class of the term

$$\mathtt{x} : TA \vdash \mathtt{x}\ \{\mathtt{to\ y.\ return\ inl\ y}, \mathtt{catch\ y.\ return\ inr\ y}\} : T(A+E)$$

and all the required commutativity diagrams follow from the laws.

These two directions enable us to prove:

Proposition 5.4 There is a theory/model correspondence (Sect. 1.4) between

- a theory of the monadic metalanguage with exceptions
- a distributive category $\mathcal{C}$, with a distinguished object E, and a strong monad, with Kleisli exponentials, supporting E-exceptions.

$\square$

5.3 The Comparison Functor

Let us now unpack the comparison functor $K(\mathcal{T}_{\mathcal{C},E})$ defined in Sect. 4. It maps a strong monad T on $\mathcal{C}$ to the monad T_E, so it is precisely the exceptions monad transformer.

Proposition 5.5 Let E be an object of a distributive category $\mathcal{C}$. Suppose $\mathcal{C}$ has equalizers. Then the functors $K(\mathcal{T}_{\mathcal{C},E})$ and $K(\mathcal{T}^{kl}_{\mathcal{C},E}$ and $K(\mathcal{T}^{\omega kl}_{\mathcal{C},E})$ each have a right adjoint, and, in each case, the counit of the adjunction is an isomorphism. $\square$

Proof. We have to check that the conditions of Prop. 4.2 are satisfied. Given a diagram of strong monads

$$\begin{tikzcd} T \arrow[r, shift left, "\alpha"] \arrow[r, shift right, "\beta"'] & T' \end{tikzcd}$$

the equalizer $S \xrightarrow{\gamma} T$ is computed pointwise. Kleisli exponentials are just equalizers of Kleisli exponentials for T and T', and similarly for countable products of Kleisli exponentials. Preservation by $\mathcal{T}_{\mathcal{C},E}$ is trivial. $\square$

Corollary 5.6 Let T be a strong monad on $\mathcal{C}$ supporting E-exceptions. If $\mathcal{C}$ has equalizers, then $T \cong T'_E$ for some strong monad T' on $\mathcal{C}$, possessing Kleisli exponentials (resp. countable products of Kleisli exponentials) if T possesses them. $\square$

We note that T' might not be unique up to isomorphism. For example, let $\mathcal{C}$ be **Set**, let E be 1, let T' be the monad $(- \to 0) \to 0$, and let T'' be the unit monad (mapping everything to 1). Then T''_E and T'_E are isomorphic (they are the unit monad), but $T''0 \ncong T'0$.

We have now characterized all monads on **Set** that model exceptions and validate the laws of Fig. 2. We next look at some non-examples. Here are two monads on **Set**, supporting E-raising, that do *not* support E-exceptions in general:

(i) [6] the monad mapping X to $S \to ((S \times X) + E)$, where S is some set

(ii) the monad mapping X to $(S \times ((S \times X) \to R)) \to R$, where R is some set and S is $E \to R$.

The second example has been provided independently by Andrzej Filinski and Hayo Thielecke [personal communication] as a model for the `catch` and `escape` facility provided in NJ-SML.

In each case there is a candidate interpretation for handle-sequencing. Suppose $\Gamma \vdash M : TA$ and $\Gamma, \mathtt{x} : A \vdash N : TB$ and $\Gamma, \mathtt{x} : \mathtt{exn} \vdash N' : TB$. Then, we define $[\![M\ \{\mathtt{to}\ \mathtt{x}.\ N, \mathtt{catch}\ \mathtt{x}.\ N'\}]\!]$ to map $\rho \in [\![\Gamma]\!]$ to

(i) the function mapping $s \in S$ to
- $([\![N]\!](\rho, \mathtt{x} \mapsto b))s'$ if $([\![M]\!]\rho)s = \mathsf{inl}\ (s', b)$
- $([\![N']\!](\rho, \mathtt{x} \mapsto e))s$ if $([\![M]\!]\rho)s = \mathsf{inr}\ e$.

(ii) the function mapping $s \in S$ and $k \in (S \times [\![B]\!]) \to R$ to

$$([\![M]\!]\rho)((\lambda e.(([\![N']\!]\rho)(s,k))), (\lambda(s',b).(([\![N]\!]\rho)(s',k))))$$

Corollary 5.6 suggests that these interpretations do not (in general) validate the equations of Fig. 4. This can be checked directly: (i) breaks equation (5), and (ii) breaks (3). Filinski has also shown [personal communication] that (3) is broken, as an observational equivalence, by `catch` and `escape`.

Two alternative conclusions may be drawn:

- these monads are inappropriate for modelling exceptions, and the constructs they model (such as `catch` and `escape`) are unnatural
- the laws in Fig. 4 are too demanding, for exceptions in general.

6 Review of Call-By-Push-Value With Stacks

A model of the monadic metalanguage with exceptions is still a model of the monadic metalanguage, albeit with extra structure. By contrast, in the case of CBPV with stacks, to which we now turn, the addition of exceptions necessitates a genuinely different structure. We first review CBPV. Our account is for infinitary CBPV; replace "countable" by "finite" throughout for the finitary version.

CBPV has two disjoint classes of terms: values and computations. It likewise has two disjoint classes of types: a value has a value type, while a computation has a computation type. For clarity, we underline computation types. The types are given by

value types	$A ::= U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A$
computation types	$\underline{B} ::= FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$

where I can be any countable set (finite, in finitary CBPV). The meaning of F and U is as follows. A computation of type FA *produces* a value of type A. A value of type $U\underline{B}$ is a *thunk* of a computation of type $\underline{B}$, i.e. the computation is frozen into a value so that it can be passed around. When later required, it can be *forced* i.e. executed.

As an example model, suppose we have a monad T on a cartesian closed category $\mathcal{C}$ with countable coproducts and products. Then each value type denotes a $\mathcal{C}$-object, and each computation type a T-algebra, in the evident way. U and F follow the Eilenberg-Moore adjunction, whilst $\prod_{i \in I}$ and $\to$ denote product algebra and exponential algebra.

Like in call-by-value, an identifier in CBPV can be bound only to a value, so it must have value type. We accordingly define a *context* Γ to be a sequence

$$\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1}$$

of distinct identifiers with associated value types. We write $\Gamma \vdash^{\mathsf{v}} V : A$ to mean that V is a value of type A, and we write $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ to mean that M is a computation of type $\underline{B}$.

In the monad semantics, a value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to $[\![A]\!]$, and a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ denotes a $\mathcal{C}$-morphism from $[\![\Gamma]\!]$ to the carrier of $[\![\underline{B}]\!]$.

The terms of CBPV are given in Fig. 5. The symbol ‘ represents application in reverse order.

A third judgement is $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$. This comes from the CK-machine of[14], and means that K is a *stack* or evaluation context of type $\underline{C}$, with a $\underline{B}$-typed hole. We do not treat the CK-machine in this paper, but give the typing rules for stacks.

For the monad semantics, given strong monad T on $\mathcal{C}$, we define a *homomorphism* from T-algebra (Y, θ) to T-algebra (Z, ϕ) over $\mathcal{C}$-object X to be a $\mathcal{C}$-morphism $X \times Y \xrightarrow{f} Z$ satisfying

$$\begin{tikzcd}
X \times TY \arrow[r, "t(X,Y)"] \arrow[d, "X\times\theta"'] & T(X \times Y) \arrow[r, "Tf"] & TZ \arrow[d, "\phi"] \\
X \times Y \arrow[rr, "f"'] & & Z
\end{tikzcd}$$

A stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ then denotes a homomorphism from $[\![\underline{B}]\!]$ to $[\![\underline{C}]\!]$ over $[\![\Gamma]\!]$.

The *complex values* are an extension of pure CBPV that are needed to achieve theory/model correspondence, though they complicate operational semantics. It is shown in [10] that this extension is conservative on computations. We can similarly add complex stacks, as explained in [14]. The syntax of complex values and complex stacks is shown in Fig. 6.

Given a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$, we obtain a

Types

value types	$A ::= \;\; U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A$
computation types	$\underline{B} ::= \;\; FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$

Values and Computations

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}\clubsuit$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \langle \hat{\imath}, V \rangle : \sum_{i\in I} A_i}\ \hat{\imath} \in I \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B}\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i\in I} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V' \rangle : A \times A'} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle . M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \lambda\{i.M_i\}_{i\in I} : \prod_{i\in I} \underline{B}_i} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \prod_{i\in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \hat{\imath}`M : \underline{B}_{\hat{\imath}}}\ \hat{\imath} \in I$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V`M : \underline{B}}$$

Stacks

$$\frac{}{\Gamma | \underline{C} \vdash^{\mathsf{k}} \mathtt{nil}\ : \underline{C}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | FA \vdash^{\mathsf{k}}\ \mathtt{to}\ \mathtt{x}.\ M :: K : \underline{C}}\clubsuit$$

$$\frac{\Gamma | \underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | \prod_{i\in I} \underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

Fig. 5. Terms of CBPV with stacks

computation $\Gamma \vdash^{\mathsf{c}} M \bullet K : \underline{C}$ by *dismantling* K on $\underline{C}$, defined by induction on K in the obvious way.

Given a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ and $\Gamma|\underline{C} \vdash^{\mathsf{k}} L : \underline{D}$, we can *concatenate* K and L to give $\Gamma|\underline{B} \vdash^{\mathsf{k}} K +\!\!+ L : \underline{D}$, defined by induction on K in the obvious way.

Complex Values

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ W : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{v}} W_i : B \ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.W_i\}_{i \in I} : B}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{v}} W : B}{\Gamma \vdash^{\mathsf{v}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.W : B}$$

Complex Stacks

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x}.A|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\underline{B} \vdash^{\mathsf{k}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ K : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \quad \Gamma, \mathtt{x} : A_i|\underline{B} \vdash^{\mathsf{k}} K_i : \underline{C} \ \ (\forall i \in I)}{\Gamma|\underline{B} \vdash^{\mathsf{k}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.K_i\}_{i \in I} : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad \Gamma, \mathtt{x} : A, \mathtt{y} : A'|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|\underline{B} \vdash^{\mathsf{k}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.K : \underline{C}}$$

$$\frac{\Gamma|\underline{C} \vdash^{\mathsf{k}} K : \underline{B} \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma|\underline{C} \vdash^{\mathsf{k}} K\ \mathtt{where}\ \mathtt{nil}\ \mathtt{is}\ L : \underline{D}}$$

$$\frac{\Gamma|\underline{C} \vdash^{\mathsf{k}} K_i : \underline{B}_i \ \ (\forall i \in I) \quad \Gamma|\prod_{i \in I} \underline{B}_i \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma|\underline{C} \vdash^{\mathsf{k}} \{K_i\ \mathtt{where}\ i :: \mathtt{nil}\}_{i \in I}\ \mathtt{is}\ L : \underline{D}}$$

$$\frac{\Gamma, \mathtt{x} : A|\underline{C} \vdash^{\mathsf{k}} K : \underline{B} \quad \Gamma|A \to \underline{B} \vdash^{\mathsf{k}} L : \underline{D}}{\Gamma|\underline{C} \vdash^{\mathsf{k}} K\ \mathtt{where}\ \mathtt{x} :: \mathtt{nil}\ \mathtt{is}\ L : \underline{D}}$$

Fig. 6. Complex Values and Stacks

The equational theory for CBPV is shown in Fig. 7 (with the same conventions as Fig. 2), and the additional law for exception raising in Fig. 8. Here

β-laws

$$\begin{aligned}
\texttt{let } V \texttt{ be x. } Q &= Q[V/\texttt{x}]\\
K \texttt{ where nil is } L &= K +\!\!+ L\\
\texttt{pm } \langle \hat{\imath}, V\rangle \texttt{ as } \{\langle i, \texttt{x}\rangle.Q_i\}_{i\in I} &= Q_{\hat{\imath}}[V/\texttt{x}]\\
\texttt{pm } \langle V, V'\rangle \texttt{ as } \langle \texttt{x}, \texttt{y}\rangle.Q &= Q[V/\texttt{x}, V'/\texttt{y}]\\
\texttt{force thunk } M &= M\\
\clubsuit \quad (\texttt{return } V) \texttt{ to x. } M &= M[V/\texttt{x}]\\
\hat{\imath}`\lambda\{i.M_i\}_{i\in I} &= M_{\hat{\imath}}\\
\{K_i \texttt{ where } i :: \texttt{nil}\}_{i\in I} \texttt{ is } \hat{\imath} :: L &= K_{\hat{\imath}} +\!\!+ L\\
V`\lambda\texttt{x}.M &= M[V/\texttt{x}]\\
K \texttt{ where x} :: \texttt{nil is } V :: L &= K[V/\texttt{x}] +\!\!+ L
\end{aligned}$$

η-laws

$$\begin{aligned}
Q[V/\texttt{z}] &= \texttt{pm } V \texttt{ as } \{\langle i, \texttt{x}\rangle.\, {}^{\texttt{x}}Q[\langle i, \texttt{x}\rangle/\texttt{z}]\}_{i\in I}\\
Q[V/\texttt{z}] &= \texttt{pm } V \texttt{ as } \langle \texttt{x}, \texttt{y}\rangle.\, {}^{\texttt{xy}}Q[\langle \texttt{x}, \texttt{y}\rangle/\texttt{z}]\\
V &= \texttt{thunk force } V\\
\clubsuit \quad K +\!\!+ L &= \texttt{ to x. } ((\texttt{return x}) \bullet {}^{\texttt{x}}K) :: L\\
M &= \lambda\{\ldots, i.i`M, \ldots\}\\
K +\!\!+ L &= \{(K +\!\!+ i :: \texttt{nil}) \texttt{ where } i :: \texttt{nil}\}_{i\in I} \texttt{ is } L\\
M &= \lambda\texttt{x}.(\texttt{x}`\,{}^{\texttt{x}}M)\\
K +\!\!+ L &= ({}^{\texttt{x}}K +\!\!+ \texttt{x} :: \texttt{nil}) \texttt{ where x} :: \texttt{nil is } L
\end{aligned}$$

Fig. 7. Equational laws for CBPV + stacks

Types with Exceptions

value types	$A ::= \; U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \texttt{exn}$
computation types	$\underline{B} ::= \; FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B}$

Exception Raising

$$\frac{\Gamma \vdash^{\mathsf{v}} V : \texttt{exn}}{\Gamma \vdash^{\mathsf{c}} \texttt{raise}_{\underline{B}} M : \underline{B}}$$

Raising Is Algebraic

$$(\texttt{raise } V) \texttt{ to x. } M = \texttt{raise } V$$

Fig. 8. Syntax and Equations For Exception Raising In CBPV

are some consequences:

$$\begin{aligned}
M \bullet K &= M \texttt{ to x. } ((\texttt{return x}) \bullet {}^{\texttt{x}}K) \\
M &= M \texttt{ to x. return x} \\
(M \texttt{ to x. } N) \bullet K &= M \texttt{ to x. } (N \bullet K) \\
(M \texttt{ to x. } N) \texttt{ to y. } P &= M \texttt{ to x. } (N \texttt{ to y. } P) \\
\lambda\{i.(M \texttt{ to x. } N_i)\}_{i \in I} &= M \texttt{ to x. } \lambda\{i.N_i\}_{i \in I} \\
\lambda\texttt{y}.(M \texttt{ to x. } N) &= M \texttt{ to x. } \lambda\texttt{y}.N \\
(\texttt{raise } V) \bullet K &= \texttt{raise } V \\
\lambda\{i.\texttt{raise } V\}_{i \in I} &= \texttt{raise } V \\
\lambda\texttt{x. raise } V &= \texttt{raise } V
\end{aligned}$$

Definition 6.1 Let $\mathcal{D}$ be a category. A *right $\mathcal{D}$-module* is a functor $\mathcal{O} : \mathcal{D} \longrightarrow$ **Set**. Explicitly,

- it provides for each A a set $\mathcal{O}A$ of "morphisms", an element of which is written $\xrightarrow{g} A$,
- we can compose $\xrightarrow{g} A$ with $A \xrightarrow{h} B$ to obtain $\xrightarrow{g;h} A$
- this composition satisfies associativity and right-identity laws.

□

Definition 6.2 Let $\mathcal{C}$ be a category.

A *locally $\mathcal{C}$-indexed category* is a strictly $\mathcal{C}$-indexed category $\mathcal{D}$ in which all the fibres have the same objects ($\mathsf{ob}\ \mathcal{D}$) and all the reindexing functors are identity on objects; equivalently, a $[\mathcal{C}^{\mathsf{op}}, \mathbf{Set}]$ enriched category.

Let $\mathcal{D}$ be a locally $\mathcal{C}$-indexed category. A *right $\mathcal{D}$-module* consists of

- for each $X \in \mathsf{ob}\ \mathcal{C}$ and $\underline{Y} \in \mathsf{ob}\ \mathcal{D}$, a small set $\mathcal{O}_X\underline{Y}$, an element of which we call an $\mathcal{O}$-morphism over X to $\underline{Y}$ and write $\xrightarrow[X]{g} \underline{Y}$
- for each $X' \xrightarrow{k} X$ and $\xrightarrow[X]{g} \underline{Y}$ a reindexed $\mathcal{O}$ morphism $\xrightarrow[X']{k^*g} \underline{Y}$
- for each $\xrightarrow[X]{g} \underline{Y}$ and $\underline{Y} \xrightarrow[X]{h} \underline{Y}'$ a composite $\mathcal{O}$ morphism $\xrightarrow[X]{g;h} \underline{Y}'$

satisfying right-identity, associativity and reindexing laws. □

Various equivalent versions of these definitions are given in [14].

Definition 6.3 A *CBPV judgement model* consists of

- a cartesian category $\mathcal{C}$
- a locally $\mathcal{C}$-indexed category $\mathcal{D}$
- a right $\mathcal{D}$-module $\mathcal{O}$.

□

Given such a structure, we interpret

- a value type or context by a $\mathcal{C}$-object
- a computation type by a $\mathcal{D}$-object

- a value $\Gamma \vdash^{\mathsf{v}} V : A$ by a $\mathcal{C}$-morphism $[\![\Gamma]\!] \xrightarrow{[\![V]\!]} [\![A]\!]$
- a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K\underline{C}$ by a $\mathcal{D}$-morphism $[\![\underline{B}]\!] \xrightarrow[{[\![\Gamma]\!]}]{[\![K]\!]} [\![\underline{C}]\!]$
- a computation $\Gamma \vdash^{\mathsf{c}} M : \underline{C}$ by an $\mathcal{O}$-morphism $\xrightarrow[{[\![\underline{B}]\!]}]{[\![M]\!]} [\![\Gamma]\!]$

This structure interprets the connectives $\times$ and 1. To interpret the remaining connectives, we require the following.

Definition 6.4 In a CBPV judgement model $(\mathcal{C}, \mathcal{D}, \mathcal{O})$,

$\boldsymbol{U\underline{B}}$ a *right adjunctive* for a $\mathcal{D}$-object $\underline{B}$ is a $\mathcal{C}$-object V and an $\mathcal{O}$-morphism $\xrightarrow[V]{\mathsf{force}} \underline{B}$, such that the functions

$$\begin{array}{rll} \mathcal{C}(X, V) & \longrightarrow \mathcal{O}_X\underline{B} & \text{for all } X \\ f & \longmapsto f^*\mathsf{force} & \end{array}$$

are isomorphisms

$\boldsymbol{FA}$ a *left adjunctive* for a $\mathcal{C}$-object A is a $\mathcal{D}$-object $\underline{V}$ and an $\mathcal{O}$-morphism $\xrightarrow[A]{\mathsf{return}} \underline{V}$, such that the functions

$$\begin{array}{rll} \mathcal{D}_X(\underline{V}, \underline{Y}) & \longrightarrow \mathcal{O}_{X\times A}\underline{Y} & \text{for all } X, \underline{Y} \\ h & \longmapsto (\pi'^*_{X,A}\mathsf{return}); (\pi^*_{X,A}h) & \end{array}$$

are isomorphisms

$\sum_{\boldsymbol{i\in I}}\boldsymbol{A_i}$ a *distributive coproduct* for a family $\{A_i\}_{i\in I}$ of $\mathcal{C}$-objects is a $\mathcal{C}$-object V and, for each $i \in I$, a $\mathcal{C}$-morphism $A_i \xrightarrow{\mathsf{in}_i} V$, such that the functions

$$\begin{array}{rll} \mathcal{C}(X \times V, Y) & \longrightarrow \prod_{i\in I}\mathcal{C}(X \times A_i, Y) & \text{for all } X, Y \\ \mathcal{O}_{X\times V}\underline{Y} & \longrightarrow \prod_{i\in I}\mathcal{O}_{X\times A_i}\underline{Y} & \text{for all } X, \underline{Y} \\ \mathcal{D}_{X\times V}(\underline{Y}, \underline{Z}) & \longrightarrow \prod_{i\in I}\mathcal{D}_{X\times A_i}(\underline{Y}, \underline{Z}) & \text{for all } X, \underline{Y}, \underline{Z} \\ f & \longmapsto \lambda i.((X \times \mathsf{in}_i)^*f) & \end{array}$$

are isomorphisms

$\prod_{\boldsymbol{i\in I}}\underline{\boldsymbol{B}}_{\boldsymbol{i}}$ a *product* for a family $\{\underline{B}_i\}_{i\in I}$ of $\mathcal{D}$-objects is a $\mathcal{D}$-object $\underline{V}$ and, for each $i \in I$, a $\mathcal{D}$-morphism $\underline{V} \xrightarrow[1]{\pi_i} \underline{B}_i$, such that the functions

$$\begin{array}{rll} \mathcal{O}_X\underline{V} & \longrightarrow \prod_{i\in I}\mathcal{O}_X\underline{B}_i & \text{for all } X \\ \mathcal{D}_X(\underline{Y}, \underline{V}) & \longrightarrow \prod_{i\in I}\mathcal{D}_X(\underline{Y}, \underline{B}_i) & \text{for all } X, \underline{Y} \\ h & \longmapsto \lambda i.(h; (\,)^*\pi_i) & \end{array}$$

are isomorphisms

$\boldsymbol{A \to \underline{B}}$ an *exponential* from a $\mathcal{C}$-object A to a $\mathcal{D}$-object $\underline{B}$ is a $\mathcal{D}$-object $\underline{V}$ and a $\mathcal{D}$-morphism $\underline{V} \xrightarrow[A]{\mathsf{ev}} \underline{B}$, such that the functions

$$\begin{array}{rll} \mathcal{O}_X\underline{V} & \longrightarrow \mathcal{O}_{X\times A}\underline{B} & \text{for all } X \\ \mathcal{D}_X(\underline{Y}, \underline{V}) & \longrightarrow \mathcal{D}_{X\times A}(\underline{Y}, \underline{B}) & \text{for all } X, \underline{Y} \\ h & \longmapsto (\pi^*_{X,A}h); (\pi'^*_{X,A}\mathsf{ev}) & \end{array}$$

are isomorphisms.

□

Definition 6.5 A *CBPV adjunction* is a CBPV adjunction model with all left adjunctives, all right adjunctives, all countable products, all countable distributive coproducts and all exponentials. □

For example, suppose T is a strong monad on a countably distributive category $\mathcal{C}$, and all countable products of, and exponentials to, carriers of T-algebras exist in $\mathcal{C}$. Then the Eilenberg-Moore resolution of T is a CBPV adjunction. Many other examples are given in [14,13]

Proposition 6.6 [14] There is a theory/model correspondence (Sect. 1.4) between

- a theory of CBPV with stacks (including complex values and complex stacks)
- a CBPV adjunction.

□

To interpet exception raising, we again use the "generic element", as in Sect. 3.1.

Definition 6.7 Let E be an object of a cartesian category $\mathcal{C}$. A *CBPV adjunction supporting E-raising* is a CBPV adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ together with an $\mathcal{O}$-morphism $\xrightarrow[E]{\texttt{raise}} F0$. □

Proposition 6.8 There is a theory/model correspondence between

- a theory of CBPV with stacks and exception raising
- a cartesian category $\mathcal{C}$ with object E and CBPV adjunction supporting E-raising.

□

7 CBPV With Handle-Sequencing

Fig. 9 shows how to add handle-sequencing to CBPV with stacks and exception raising. Given a strong monad T on $\mathcal{C}$, we interpret values and computations as before, using the monad T_E. In particular, FA denotes the free T_E-algebra on A, and $U\underline{B}$ denotes the carrier of $[\![\underline{B}]\!]$. But a stack does *not* denote a T_E algebra homomorphism, because $\texttt{raise}\ V \bullet K$ might not raise V. It has to denote a T-algebra homomorphism. More precisely, if $\underline{Y} = (X, \theta)$ is a T_E-algebra, write $p\underline{Y}$ for the T-algebra $(X, (T\mathsf{inl}_{X,E}; \theta))$. Then a stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ denotes a T-algebra homomorphism from $p[\![\underline{B}]\!]$ to $p[\![\underline{C}]\!]$ over $[\![\Gamma]\!]$.

We see in this class of examples that F and U do not describe an adjunction between values and stacks. We can see this in the equational theory too.

Define M `to x.` N as in Sect. 2, and similarly define `to x.` $N :: K$ to be

$$\left\{\begin{array}{ll} \texttt{to x.} & N \\ \texttt{catch x.} & \texttt{raise x} \end{array}\right\} :: K$$

Then one of the equations that appears in Fig. 7 and is valid in any CBPV adjunction, viz.

$$K +\!\!+ L = \texttt{to x.}\ ((\texttt{return x}) \bullet {}^{\mathsf{x}}K) :: L$$

ceases to be valid. To see this, suppose that V is a closed value of type `exn`, take L to be `nil` , and let K be `to x. return` $\langle\rangle$. When we dismantle the two sides onto `raise` V, then the LHS returns the value $\langle\rangle$, whereas the RHS raises exception V.

The following constructs and equations replace those marked ♣ in Fig. 5–7.

$$\frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B} \quad \Gamma, \mathtt{x} : \mathtt{exn} \vdash^{\mathsf{c}} N' : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \{\texttt{to x.}\ N, \texttt{catch x.}\ N'\} : \underline{B}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma, \mathtt{x} : \mathtt{exn} \vdash^{\mathsf{c}} M' : \underline{B} \quad \Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma|FA \vdash^{\mathsf{k}}\ \{\texttt{to x.}\ M, \texttt{catch x.}\ M'\} :: K : \underline{C}}$$

$$(\texttt{return}\ V)\left\{\begin{array}{ll} \texttt{to x.} & N \\ \texttt{catch x.} & N' \end{array}\right\} = N[V/\mathtt{x}]$$

$$(\texttt{raise}\ V)\left\{\begin{array}{ll} \texttt{to x.} & N \\ \texttt{catch x.} & N' \end{array}\right\} = N'[V/\mathtt{x}]$$

$$K +\!\!+ L = \left\{\begin{array}{ll} \texttt{to x.} & ((\texttt{return x}) \bullet {}^{\mathsf{x}}K) \\ \texttt{catch x.} & ((\texttt{raise x}) \bullet {}^{\mathsf{x}}K) \end{array}\right\} :: L$$

Fig. 9. Exception Handling In CBPV With Stacks

Thus, in the presence of exceptions, the type FA is not a left adjunctive for A. To model F, we need to generalize the notion of left adjunctive.

Definition 7.1 (i) Let $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ be a CBPV judgement model, and let E be an object of $\mathcal{C}$. A *left E-adjunctive* for a $\mathcal{C}$-object A consists of a $\mathcal{D}$-object $\underline{V}$, an $\mathcal{O}$-morphism $\xrightarrow[A]{\mathsf{return}} \underline{V}$ and an $\mathcal{O}$-morphism $\xrightarrow[E]{\mathsf{raise}} \underline{V}$ such that the functions

$$\begin{aligned}\mathcal{D}_X(\underline{V},\underline{Y}) &\longrightarrow \mathcal{O}_{X\times A}\underline{Y} \times \mathcal{O}_{X\times E}\underline{Y} \quad \text{for all } X, \underline{Y} \\ h &\longmapsto (((\pi'^*_{X,A}\mathsf{return}); (\pi^*_{X,A}h)), \\ &\qquad\qquad ((\pi'^*_{X,A}\mathsf{raise}); (\pi^*_{X,A}h)))\end{aligned}$$

are isomorphisms.

(ii) Let E be an object of a cartesian category $\mathcal{C}$. A *CBPV E-adjunction* is a CBPV judgement model $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ with all left E-adjunctives, all right adjunctives, all countable products, all countable distributive coproducts and all exponentials. We write FA for (the vertex of) the left E-adjunctive of A, and likewise for the other operations on objects.

(iii) Let E' and E be objects of a cartesian category $\mathcal{C}$. A *CBPV E'-adjunction supporting E-raising* is CBPV E'-adjunction $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ together with an $\mathcal{O}$-morphism $\xrightarrow[E]{\mathtt{raise}} F0$.

□

Note that a CBPV 0-adjunction is just a CBPV adjunction.

Proposition 7.2 There is a theory/model correspondence between

- a theory of CBPV with stacks and exceptions
- an object E of a cartesian category $\mathcal{C}$, and a CBPV E-adjunction.

□

We relate this to strong monads as follows.

Definition 7.3 Let E be an object of a cartesian category $\mathcal{C}$, and let $(\mathcal{C}, \mathcal{D}, \mathcal{O})$ be a CBPV E-adjunction. Then $\mathcal{C}$ is countably distributive, and we obtain a CBV strong monad on $\mathcal{C}$, supporting E-exceptions: on objects it is given by $A \mapsto UFA$, and the rest is evident. □

Another way of seeing this construction is as a translation from the monadic metalanguage with exceptions to CBPV with exceptions.

We can also translate CBPV with exceptions to CBPV without exceptions, so we can convert an adjunction into an E-adjunction. More generally, an E' adjunction can be converted into an $E' + E$ adjunction:

Definition 7.4 Let E' and E be objects in a cartesian category $\mathcal{C}$, and let $\mathcal{A} = (\mathcal{C}, \mathcal{O}, \mathcal{D})$ be an E'-adjunction. We construct an $E' + E$ adjunction $\mathcal{A}_E = (\mathcal{C}, \mathcal{D}', \mathcal{O}')$ as follows.

- An object of $\mathcal{D}'$ is a $\mathcal{D}$-object $\underline{B}$ together with an $\mathcal{O}$-morphism $\xrightarrow[E]{t} \underline{B}$.
- A $\mathcal{D}'$-morphism $(\underline{B}, r) \xrightarrow[A]{} (\underline{C}, s)$ is a $\mathcal{D}$-morphism $\underline{B} \xrightarrow[A]{} \underline{C}$.
- An $\mathcal{O}'$-morphism $\xrightarrow[A]{} (\underline{B}, r)$ is an $\mathcal{O}$-morphism $\xrightarrow[A]{} \underline{B}$.
- Distributive coproducts are unchanged.
- The right adjunctive of $(\underline{B}, r)$ is $U\underline{B}$.
- The product of $\{\underline{B}_i, r_i\}_{i\in I}$ is $(\prod_{i\in I}\underline{B}_i, s)$ where $s; ()^*\pi_i = r_i$,

- The exponential from A to $(\underline{B}, r)$ is $(A \to \underline{B}, s)$ where $\pi^*_{A,E}s; \pi'^*_{A,E}\mathsf{ev} = \pi'^*_{A,E}r$
- The left $E + E'$ adjunctive of A is $(F(A + E), \mathsf{inr}^*_{A,E}\mathsf{return})$.

□

Proposition 7.5 If T is the strong monad obtained from E'-adjunction $\mathcal{A}$, then the strong monad obtained from $(E' + E)$-adjunction $\mathcal{A}_E$ is T_E. □

Conversely, suppose we are given an E adjunction. We can obtain an ordinary adjunction supporting E-raising by restricting to those stacks that are "homomorphic" in exception raising. More generally, given an $E' + E$ adjunction, we can obtain an E' adjunction:

Definition 7.6 Let E' and E be objects in a cartesian category $\mathcal{C}$, and let $(\mathcal{C}, \mathcal{O}, \mathcal{D})$ be an $E' + E$ adjunction. We form an E' adjunction $(\mathcal{C}, \mathcal{O}, \mathcal{D}')$ supporting E raising, as follows. $\mathcal{D}'$ has the same objects as $\mathcal{D}$. A $\mathcal{D}'$-morphism $\underline{B} \xrightarrow[X]{} \underline{C}$ is a $\mathcal{D}$-morphism $\underline{B} \xrightarrow[X]{h} \underline{C}$ which is "homomorphic for E" i.e.

$$\begin{tikzcd}
 & {} \arrow[dl, "\pi'^*_{X,E}\mathsf{raise}"'] \arrow[dr, "\pi'^*_{X,E}\mathsf{raise}"] & \\
\underline{B} \arrow[rr, "\pi^*_{X,E}h"'] & & \underline{C}
\end{tikzcd}$$

as a diagram over $X \times E$ commutes.

The left E' adjunctive of A is FA with $\mathsf{inl}^*_{E',E}\mathsf{raise}$. All the other connectives are unchanged. This monad supports E-raising via the morphism $\mathsf{inr}^*_{E',E}\mathsf{raise}$. □

As an example, suppose we take the monad semantics of CBPV with stacks, using strong monad T on category $\mathcal{C}$. If we apply the construction in Def. 7.4, we obtain the semantics of CBPV with stacks and exception handling described above, where a computation object is a T_E-algebra but a stack is a T-algebra homomorphism. If we then apply the construction in Def. 7.6, then we get the monad semantics for errors, where a computation object is a T_E-algebra and a stack is a T_E-algebra homomorphism.

Acknowledgements I thank A. Filinski and H. Thielecke for their help.

References

[1] Beck, J. M., *Triples, algebras and cohomology*, Reprints in Theory and Applications of Categories **2** (2003), pp. 1–59.

[2] Benton, N. and A. Kennedy, *Exceptional syntax*, Journal of Functional Programming **11** (2001), pp. 395–410.

[3] Carboni, A., S. Lack and R. F. C. Walters, *Introduction to extensive and distributive categories*, J. of Pure and Applied Algebra **84** (1993), pp. 145–158.

[4] Cenciarelli, P. and E. Moggi, *A syntactic approach to modularity in denotational semantics*, in: *CTCS 1993*, 1993.

[5] Cockett, J. R. B., *Introduction to distributive categories*, Mathematical Structures in Computer Science **3** (1993), pp. 277–307.

[6] Filinski, A., *Representing layered monads*, in: *Proc., 26th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, 1999, pp. 175–188.

[7] Hyland, J. M. E., G. D. Plotkin and A. J. Power, *Combining effects: Sum and tensor* (2006), to appear in Theoretical Computer Science.

[8] Laird, J., *A fully abstract game semantics of local exceptions*, in: *Proc. of the 16th Annual IEEE Symp. on Logic in Computer Science* (2001), pp. 105–114.

[9] Laird, J., *Exceptions, continuations and macro-expressiveness*, LNCS **2305** (2002).

[10] Levy, P. B., *Call-by-push-value: Decomposing call-by-value and call-by-name*, submitted.

[11] Levy, P. B., *Call-by-push-value: a subsuming paradigm (extended abstract)*, in: J.-Y. Girard, editor, *Typed Lambda-Calculi and Applications*, LNCS **1581**, 1999.

[12] Levy, P. B., *Possible world semantics for general storage in call-by-value*, in: J. Bradfield, editor, *Proceedings, 16th CSL*, LNCS **2471** (2002).

[13] Levy, P. B., "Call-By-Push-Value. A Functional/Imperative Synthesis," Semantic Structures in Computation, Springer, 2004.

[14] Levy, P. B., *Adjunction models for call-by-push-value with stacks*, Theory and Applications of Categories **14** (2005), pp. 75–110.

[15] Levy, P. B., *Jumbo λ-calculus* (2006), to appear in Proc., 33rd ICALP, in LNCS.

[16] Levy, P. B., A. J. Power and H. Thielecke, *Modelling environments in call-by-value programming languages*, Inf. and Comp. **185** (2003), pp. 182–210.

[17] Mac Lane, S., "Categories for the Working Mathematician," Graduate Texts in Mathematics **5**, Springer, New York, 1971, ix+262 pp.

[18] Moggi, E., *Computational lambda-calculus and monads*, in: *LICS'89, Proc. 4th Ann. Symp. on Logic in Comp. Sci.*, IEEE, 1989, pp. 14–23.

[19] Moggi, E., *Notions of computation and monads*, Inf. and Comp. **93** (1991).

[20] Plotkin, G. and J. Power, *Adequacy for algebraic effects*, LNCS **2030** (2001).

[21] Plotkin, G. and J. Power, *Notions of computation determine monads*, in: *Proc., Foundations of Software Sci. and Comp. Struct., 2002*, LNCS **2303** (2002).

[22] Plotkin, G. D. and A. J. Power, *Algebraic operations and generic effects*, Applied Categorical Structures **11** (2003), pp. 69–94.