Friedrich-Alexander-Universität
Erlangen-Nürnberg

T.cs

Chair for Computer Science 8
Theoretical Computer Science

Generic Hoare Logic for Order-Enriched Effects with Exceptions

Master Thesis in Computer Science

Christoph Rauch

Advisors: Prof. Dr. Lutz Schröder, Dr. Sergey Goncharov

Erlangen, August 30th, 2013

Contents

Goncharov and Schröder established a relatively complete Hoare calculus for monads equipped with a dcpo structure. Their framework supports algebraic operations $\alpha_A : (TA)^n \to TA$ in the form of generic effects, i.e. morphisms in the Kleisli category corresponding to T. We take this approach one step further by exploring how the calculus behaves when exception handling, an operation which is not algebraic, is introduced and give a similar completeness result. Exceptions are modeled using coproducts, hence our calculus supports generic coproduct types, case distinction and a loop construction performing iterated case distinction.

1 Introduction

The use of monads as a means of modeling side-effecting computations goes back to the end of the 1980s [14] and has since been studied extensively. A broad range of types of computation can be shown to have a monadic structure. This allows for practically employing the concept to model the encapsulation of side-effects, for example in functional programming languages like Haskell [25], where side-effects are otherwise conceptually impossible.

Typical examples of such side-effecting computations are global state or exceptions, that is abrupt termination with the possibility to propagate the reason of termination to the user.

In addition to just modeling side-effects, a matter of interest is how to reason about certain properties of such programs, one of the most important properties being correctness. Tony Hoare, in his seminal work [11], provided a means of proving correctness of simple imperative programs.

In the work preceding this thesis, Goncharov and Schröder [9] established a generic *Hoare calculus* for a monadic metalanguage. Generic here means that the monad can be freely chosen, provided that it fulfils certain properties which we will clarify in Chapter 3.

It is, however, important to note that, while being generally admissible in the sense that it is structurally rich enough for the framework of Goncharov and Schröder [9], the exception monad allows for an operation which does not fit into the scheme, namely *catch*, that is exception handling. Said framework captures only algebraic functions, which *catch* is not. A general theory of non-algebraic operations seems to be a challenging task. Thus, the purpose of this work is to explore a modified version of the calculus of Goncharov and Schröder with explicit support for exception handling as a single non-algebraic operation. Semantically, the introduction of exceptions acts as a *monad transformer* (cf. [14, 20]), meaning that we are dealing with a monad modeling a certain side-effect—the base monad—and stack the capability of raising and handling exceptions on top of it. This procedure yields another monad. We will show that this monad also has the desired properties and that the resulting *Hoare calculus* is also sound and relatively complete.

Remark. We assume basic knowledge in category theory and also make use of some advanced categorical techniques and notions. References for the latter are given where necessary. For the former, the reader may find a thorough introduction suitable for computer scientists in Barr and Wells [3]. Other introductory texts include Awodey [2], Herrlich and Strecker, [1] and the seminal work by MacLane [13].

2 Defining the Language

In the following chapter, we introduce the concepts needed to create a *metalanguage* supporting generic side-effects and exceptions. It is called *metalanguage*, because it abstracts from specific types of computations and interpretations and can be instantiated to actual programming languages by choosing suitable monads and categories.

2.1 Hoare Logic

The language we are going to define is supposed to be part of a verification calculus in the sense of Hoare. Before actually defining the language, we give a short description of the development of such calculi as well as the terminology involved. The capabilities of the language are crucial factors for the actual usefulness of the resulting calculus. We will therefore shortly discuss the concepts of soundness and completeness.

2.1.1 Concept

In his seminal paper [11], Hoare developed a formal system for reasoning about the correctness of computer programs. The basic idea is to annotate a program segment C which we call *command* with a precondition P and postcondition Q, forming a so-called *Hoare Triple*, which we denote by $\{P\}\ C\ \{Q\}$. The meaning of such a triple is that if the command C is executed in an environment satisfying the precondition P, then the postcondition Q holds once C has terminated. Nothing is said about cases where C does not terminate, which is why *Hoare Triples* in this sense only argue about *partial correctness*, as opposed to *total correctness*.

Remark. Assertions, i.e. pre- and postconditions, in *Hoare Triples* are formulae in predicate logic. One of our goals in the following will be to use the same language for both programs and assertions. This is practical because it allows us to define the semantics for a Hoare triple in a uniform manner. Because assertions are just programs, the whole triple can be interpreted as a certain program, and validity of a Hoare triple can be treated as an equality of the induced programs.

The inference rules for a simple *Hoare calculus* for a primitive imperative programming language supporting assignment, sequential composition, branching and looping operations is given in Figure 1.

$$\textbf{(ass)}\ \frac{}{\{P[a/x]\}\ x := a\ \{P\}} \qquad \textbf{(comp)}\ \frac{\{P\}\ S\ \{Q\} \quad \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

$$\textbf{(wk)}\ \frac{P' \Rightarrow P \quad \{P\}\ S\ \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ S\ \{Q'\}}$$

$$\textbf{(if)}\ \frac{\{P \wedge B\}\ S\ \{Q\} \quad \{P \wedge \neg B\}\ T\ \{Q\}}{\{P\}\ \mathsf{if}\ B\ \mathsf{then}\ S\ \mathsf{else}\ T\ \{Q\}} \qquad \textbf{(while)}\ \frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \mathsf{while}\ B\ \mathsf{do}\ S\ \{P \wedge \neg B\}}$$

Figure 1: The classical Hoare calculus.

2.1.2 Rule design

An important question here is how to come up with a rule set for a Hoare calculus. The most prominent desirable features of a formal system are *soundness* and *completeness*. The former states that every theorem which is derivable using the inference rules is valid with respect to the semantics of the system. The latter states that every valid statement is provable in the calculus.

Thus, the rules should be designed in such a way that they at least have the soundness property. This ensures that the calculus can be used to derive only valid programs. However, a more common application of verification calculi is—as the name already suggests—to analyze a given program for correctness. Therefore, the calculus also has to be complete. Otherwise a potentially correct program (which would yield a valid statement) might not be derivable using the calculus, which can be acceptable in other situations, but is certainly a drawback here.

We formally introduce the concepts in terms of our monadic formalisation in Chapter 4.

2.2 Monads and Categorical Semantics

The semantics we give to our metalanguage is formulated in terms of category theory. This means that types are interpreted as objects and programs as morphisms in some category (see for example [15]). However, we want our programs to be able to have side effects. In this section, we introduce the categorical notions necessary to define a metalanguage with side effects, based on the ideas of Eugenio Moggi, who used monads, a concept known for a long time in category theory [7], to describe side-effecting computations [14, 16].

2.2.1 Kleisli triples

Side effects are a common phenomenon in imperative programming. For example, a programmer might want to read and modify a global state and write procedures depending on the contents of that state. The Hoare Calculus introduced in Section 2.1 can only cope with side effecting programs with respect to a global store. Consequently, general effects have to be made explicit in the language.

We thus incorporate as the basic representation of our programming language a type system, interpreted over a category $\mathcal{C}$ whose objects are types. We discern value types A from types TA of computations over values of type A. The type constructor T is interpreted as part of a *Kleisli triple*, defined as follows.

Definition 1 (Kleisli triple)**.** Let $\mathcal{C}$ be a category. Let $\mathbb{T}$ be a triple $(T, \eta, -^*)$ with

- $T : \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{C})$ being an endomorphism;
- η being a family of morphisms $\eta_A : A \to TA$ called *unit*;
- $-^*$ being an operator called *Kleisli star* where f^* has type $TA \to TB$ for any $f : A \to TB$.

Such a triple is called a *Kleisli triple* iff it fulfils the identities

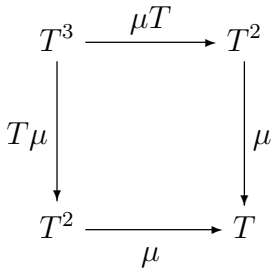
$$\eta_A^* = \mathrm{id}_{TA} \qquad f^* \circ \eta_A = f \qquad (g^* \circ f)^* = g^* \circ f^*.$$

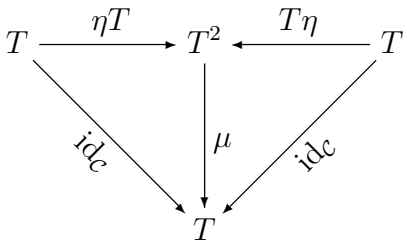
The objects of $\mathcal{C}$ and the morphisms $A \to TB$, together with a composition operation $g^* \circ f$ for $f : A \to TB, g : B \to TC$, form a category. This category, denoted $\mathcal{C}_{\mathbb{T}}$ is called the *Kleisli category* of $\mathbb{T}$.

We can consider the *Kleisli category* as a category of programs, where an object B contains the computations with result type B (and possibly other side-effects) and a program with input type A and result type B is an element of $\mathsf{Hom}_{\mathcal{C}_{\mathbb{T}}}(A, B) = \mathsf{Hom}_{\mathcal{C}}(A, TB)$ [14]. *Kleisli triples* are a common representation of monads in the context of semantics of programming languages [17].

An equivalent representation is as follows.

Definition 2 (Monad)**.** Let $\mathcal{C}$ be a category. A *monad* $\mathbb{T}$ is a triple (T, η, μ), where T is an endofunctor on $\mathcal{C}$, $\eta : \mathrm{id}_{\mathcal{C}} \to T$ and $\mu : T \circ T \to T$ are natural transformations called *unit* and *multiplication* respectively, and the following diagrams commute:





It is easy to see that a monad (T, η, μ) can be turned into a *Kleisli triple* $(T, \eta, -^*)$ by letting

$$f^* = \mu_B \circ Tf \text{ for } f : A \to TB$$

and restricting T to the object part of the functor. Then, the identities for *Kleisli triples* hold:

$$\eta_A^* = \mu_A \circ (T\eta_A) = \mathrm{id}$$

by the right-hand side of the second commutative diagram,

$$f^* \circ \eta_A = \mu_B \circ Tf \circ \eta_A = \mu_B \circ \eta_{TB} \circ f = f$$

because η is a natural transformation, that is from $\eta : F \to G$ and $f : A \to B$ follows $Gf \circ \eta_A = \eta_B \circ Ff$, and

$$\begin{aligned}(g^* \circ f)^* &= (\mu_C \circ Tg \circ f)^* \\ &= \mu_C \circ T(\mu_C \circ Tg \circ f) \\ &= \mu_C \circ T\mu_C \circ TTg \circ Tf \\ &= \mu_C \circ \mu_{TC} \circ TTg \circ Tf \\ &= \mu_C \circ Tg \circ \mu_B \circ Tf = g^* \circ f^*,\end{aligned}$$

using the fact that μ is a natural transformation and that $\mu_C \circ T\mu_C = \mu_C \circ \mu_{TC}$ (from the first commutative diagram).

Similarly, every *Kleisli triple* gives rise to a monad (proof omitted), so there is a one-to-one correspondence, allowing us to use either representation equivalently.

2.2.2 Monad examples

Let us have a look at some of the more popular monads over the category **Set** with sets as objects and functions as morphisms.

Example 1 (State)**.** Probably the most well-known example is that of a global state. Let S be a set of states. We can then describe a computation over a type A:

$$TA = S \to (A \times S)$$

Intuitively, it is a function depending on some *current* state and computing a value of type A together with a new, updated state. Intuitively, the monad *unit* for T should just leave the state unchanged:

$$\eta_A(a)(s) = (a, s), \quad a \in A, s \in S$$

The *Kleisli star* operator for this monad, which should allow us to compose computations $f : A \to TB$ and $g : B \to TC$ in the state monad, can be defined as follows:

$$f^*(t)(s) = \mathsf{let}\ (a, s') = t(s)\ \mathsf{in}\ f(a)(s'), \quad t \in TA, s \in S,$$

where the π_i, $i \in 1, 2$ are the usual product projection functions. It remains to check that the monad laws hold:

$$\begin{aligned}
\eta_A^*(t)(s) &= \mathsf{let}\ (a, s') = t(s)\ \mathsf{in}\ \eta_A(a)(s') = (a, s') = t(s) \\
(f^* \circ \eta_A)(a) &= f^*(\lambda s.(a, s)) = \lambda s'.f(a)(s') = f(a) \\
(g^* \circ f)^*(t)(s) &= \mathsf{let}\ (a, s') = t(s)\ \mathsf{in}\ (g^* \circ f)(a)(s') \\
&= \mathsf{let}\ (a, s') = t(s)\ \mathsf{in}\ g^*(f(a)(s')) = (g^* \circ f^*)(t)(s)
\end{aligned}$$

We can now define operations $\mathsf{get} = \lambda s.(s, s)$ and $\mathsf{put}\ s = \lambda u.(\star, s)$, where $\star$ is the element of the singleton set 1. The former returns the contents of the state as its result, otherwise leaving the state unchanged, the latter overwrites the state with a new state s. The types of those operations are TS and $S \to T1$, respectively.

It is, however, tedious to write programs using this purely mathematical notation. Therefore, we introduce the *monadic metalanguage* in the next section.

Example 2 (Powerset)**.** Another popular monad is the *powerset monad*, where $TA = \mathcal{P}A$, i.e. for a set A, the set of all subsets of A. The unit and *Kleisli star* are given by

$$\begin{aligned}
&\eta_A(a) = \{a\} && a \in A \\
&f^*(t) = \bigcup \{f(a) \mid a \in t\} && f : A \to TB, t : TA
\end{aligned}$$

The *powerset monad* can be used to model non-determinism. Intuitively, a program $C \to TA$ turns a value from C into a set of possible results of type A. We can now proceed by processing each of these results with another program $A \to TB$ and take the union to receive another set of possible outcomes, and so on.

Example 3 (Other examples)**.** Other than these standard examples, there is a vast choice of computationally interesting structures which can be shown to be monadic. To name a few: continuations, parsers, interprocess communication, even graphical user interfaces. The use of monads in functional programming languages like Haskell offers a broad range of implemented examples, see e.g. the *Hackage* database [10].

2.2.3 Monadic metalanguage

With the machinery of monads at hand, we are able to design a small formal system, a kind of programming language, by giving inference rules for which we define interpretations in the base category. Over the course of this work, we gradually expand the rule set and impose restrictions on the base category, so that it can support the features added to our language. The basic set of rules is given in Figure 2.

First, we fix a set of types $\mathcal{W}$. For the time being, the types of our language are given by the grammar

$$A ::= W \mid 1 \qquad C ::= A \mid TA$$

where $W \in \mathcal{W}$. The types A are called *value types*. The interpretation $[\![C]\!]$ of a type C is just an object in the category $\mathcal{C}$. We impose upon $\mathcal{C}$ that it has a terminal object 1,

$$\textbf{(var)}\ \frac{}{x : C \vdash x : C} \qquad \textbf{(op)}\ \frac{x : C \vdash t : A}{x : C \vdash f(t) : B}\ (f : A \to B \in \Sigma) \qquad \textbf{(1)}\ \frac{}{x : C \vdash \star : 1}$$

$$\textbf{(ret)}\ \frac{x : C \vdash p : A}{x : C \vdash \mathsf{ret}\, p : TA} \qquad \textbf{(do)}\ \frac{y : C \vdash p : TA \quad x : A \vdash q : TB}{y : C \vdash \mathsf{do}\ x \leftarrow p; q : TB}$$

Figure 2: Inference rules for a simple computational calculus.

such that $[\![1]\!] = 1$. Inductively, $[\![TA]\!] = T[\![A]\!]$, where $T : \mathrm{Ob}\,\mathcal{C} \to \mathrm{Ob}\,\mathcal{C}$ is the underlying object map of a monad $\mathbb{T}$.

The metalanguage is parameterized by a set of function symbols $f : A \to C$, contained in a signature Σ (for instance, from last section, $\mathsf{get}, \mathsf{put} \in \Sigma$), with $[\![f]\!] : [\![A]\!] \to [\![C]\!]$. Here, A must be a *value type* [9].

Remark. Notice that operations like get are actually of type $1 \to TS$, but we denote them with type TS as a simplification.

We introduce terms $x : A \vdash p : C$, which are interpreted as morphisms $[\![A]\!] \to [\![C]\!]$. The rule **(var)** therefore acts as the identity: $[\![x : A \vdash x : A]\!] = \mathrm{id}_{[\![A]\!]}$. For a term $x : A \vdash p : C$ with interpretation f, the conclusion of the rule **(ret)** is interpreted using the unit of the monad $\mathbb{T}$:

$$[\![x : A \vdash \mathsf{ret}\, p : TC]\!] = \eta_C \circ f$$

Similarly, **(do)** is interpreted using composition in the *Kleisli category* $\mathcal{C}_{\mathbb{T}}$. Given $[\![y : A \vdash p : TB]\!] = g$, $[\![x : B \vdash q : TB']\!] = f$, we let

$$[\![y : A \vdash \mathsf{do}\ x \leftarrow p; q : TB']\!] = f^* \circ g.$$

Finally, given $[\![x : A \vdash t : B]\!] = g$ and a function $f : B \to C \in \Sigma$, the interpretation for **(op)** is

$$[\![x : A \vdash f(t) : C]\!] = f \circ g.$$

Naturally, the monad laws are expressible in the metalanguage:

$$\begin{aligned} \mathsf{do}\ y \leftarrow (\mathsf{do}\ x \leftarrow p; q); r &= \mathsf{do}\ x \leftarrow p; y \leftarrow q; r \\ \mathsf{do}\ x \leftarrow \mathsf{ret}\, a; p &= p[a/x] \\ \mathsf{do}\ x \leftarrow p; \mathsf{ret}\, x &= p. \end{aligned}$$

Using this notation, it is easier to see that the monad laws describe syntactic transformations of programs which retain the semantics of the program and as such define an equivalence relation between programs.

2.2.4 Strong monads

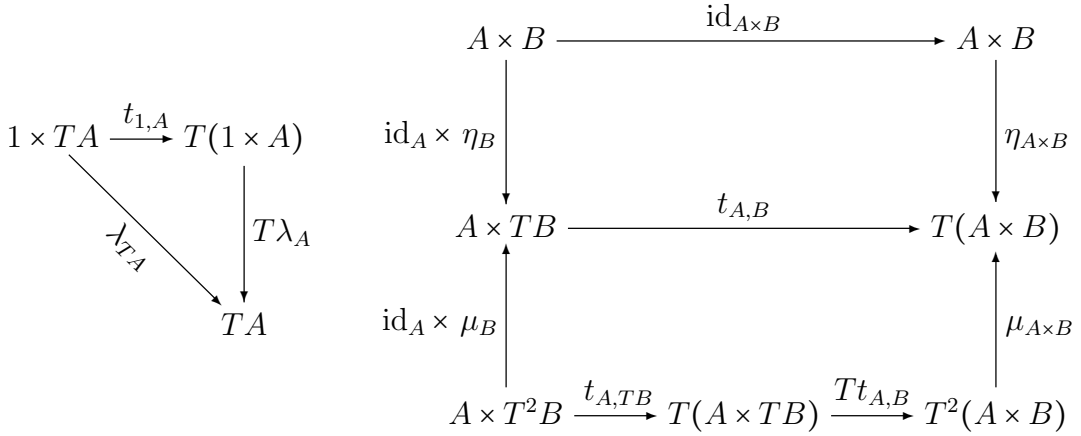
The metalanguage defined in the previous section is extremely limited, in that it only allows one single variable in the context (the left-hand side of the $\vdash$ sign) of a term. Without side-effects, to mitigate this, it would suffice to add types $(A_1 \times \cdots \times A_n)$, where the A_i, $1 \leq i \leq n$ are types, and admit terms in context $\Gamma \vdash p : A$, where $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$. We write Γ, Δ for the product of two contexts, especially $\Gamma, x : A$ to add a new variable-type pair to the context, and interpret product types in the base category as $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$, i.e. we demand that the base category have products.

The interpretation of such a term would then be a map $([\![A_1]\!] \times \cdots \times [\![A_n]\!]) \to [\![A]\!]$. As Moggi [16] points out, however, product types do not suffice to interpret the do-construction with variable contexts:

$$\textbf{(do)}\ \frac{\Gamma \vdash p : TA \quad \Gamma, x : A \vdash q : TB}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : TB}$$

This rule can no longer be interpreted in terms of *Kleisli composition* only as suggested in the previous section. The terms in the premise have as interpretations two morphisms $f : [\![\Gamma]\!] \to T[\![A]\!]$ and $g : [\![\Gamma]\!] \times [\![A]\!] \to T[\![B]\!]$, which are not composible via *Kleisli composition*, because g^* has type $T([\![\Gamma]\!] \times [\![A]\!]) \to T[\![B]\!]$. So first we have to extend the morphism f to $\langle \mathrm{id}_C, f \rangle$ to propagate the context to g^*. Still, there is a missing link, a morphism $[\![\Gamma]\!] \times T[\![A]\!] \to T([\![\Gamma]\!] \times [\![A]\!])$. We have to require that such a morphism exists. This is captured in the following definition.

Definition 3 (Strong monad)**.** A *strong monad* [14] is given by a monad $\mathbb{T} = (T, \eta, \mu)$ together with a natural transformation $t_{A,B} : A \times TB \to T(A \times B)$, called *strength*, such that the following diagrams commute:



where $\lambda = \pi_2$ and a are the obvious natural isomorphisms

$$\lambda_A : 1 \times A \to A$$
$$a_{A,B,C} : (A \times B) \times C \to A \times (B \times C).$$

$$\textbf{(var)}\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \textbf{(op)}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash f(t) : B}\ (f : A \to B \in \Sigma) \qquad \textbf{(1)}\ \frac{}{\Gamma \vdash \star : 1}$$

$$\textbf{(pair)}\ \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} \qquad (\pi_1)\ \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_1 t : A} \qquad (\pi_2)\ \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2 t : B}$$

$$\textbf{(ret)}\ \frac{\Gamma \vdash p : A}{\Gamma \vdash \mathsf{ret}\, p : TA} \qquad \textbf{(do)}\ \frac{\Gamma \vdash p : TA \quad \Gamma, x : A \vdash q : TB}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : TB}$$

Figure 3: Inference rules for a computational calculus.

$$\begin{array}{ccccc}
(A \times B) \times TC & & \xrightarrow{\qquad t_{A\times B,C} \qquad} & & T((A \times B) \times C) \\
\Big\downarrow{\scriptstyle a_{A,B,TC}} & & & & \Big\downarrow{\scriptstyle Ta_{A,B,C}} \\
A \times (B \times TC) & \xrightarrow{\mathrm{id}_A \times t_{B,C}} & A \times T(B \times C) & \xrightarrow{t_{A,B\times C}} & T(A \times (B \times C))
\end{array}$$

The upper right diagram states that *strength* must be compatible with monad unit and multiplication.

Thus, given a strong monad $\mathbb{T}$, we can interpret the do-construction as follows:

$$\frac{[\![\Gamma \vdash p : TA]\!] = f \quad [\![\Gamma, x : A \vdash q : TB]\!] = g}{[\![\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : TB]\!] = g^* \circ t_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, f \rangle}$$

and we arrive at the rule set given in Figure 3. The additional rules for product types are interpreted as expected:

$$\frac{[\![\Gamma \vdash t : A]\!] = f \quad [\![\Gamma \vdash u : B]\!] = g}{[\![\Gamma \vdash \langle t, u \rangle : A \times B]\!] = \langle f, g \rangle} \qquad \frac{[\![\Gamma \vdash t : A \times B]\!] = f}{[\![\Gamma \vdash \pi_1 t : A]\!] = \pi_1 \circ f} \qquad \frac{[\![\Gamma \vdash t : A \times B]\!] = f}{[\![\Gamma \vdash \pi_2 t : B]\!] = \pi_2 \circ f}$$

and variables $\Gamma \vdash x : A$ are now projection maps from $[\![\Gamma]\!]$ to $[\![A]\!]$.

Example 4 (Strong monads in **Set**)**.** All monads in the category of sets are strong.

Proof. Let p_a for $a \in A$ be the map $b \mapsto (a, b)$. Then, strength in **Set** for any monad $\mathbb{T}$ may be defined as

$$t_{A,B} \ : \ (a, t) \mapsto (Tp_a)(t) \ : \ A \times TB \to T(A \times B).$$

We need to check if the conditions imposed by the diagrams above hold. Obviously, λ_A sends $(\star, a)$ to a, thus $\lambda_A \circ p_\star = \mathrm{id}_A$ and for the first diagram, we get

$$(T\lambda_A \circ t_{1,A})(\star, t) = (T\lambda_A \circ Tp_\star)(t) = (T(\lambda_A \circ p_\star))(t) = t = \lambda_{TA}(\star, t).$$

Because $a_{A,B,C}$ sends $((a,b),c)$ to $(a,(b,c))$, we have $p_a \circ p_b = a_{A,B,C} \circ p_{(a,b)}$. Using this identity, the equalities encoded in the second diagram are

$$\begin{aligned}
&(Ta_{A,B,C} \circ t_{A\times B,C})((a,b),t)\\
=&T(a_{A,B,C} \circ p_{(a,b)})(t)\\
=&(T(p_a \circ p_b))(t)\\
=&(Tp_a \circ Tp_b)(t)\\
=&t_{A,B\times C}(a,(Tp_b)(t))\\
=&(t_{A,B\times C} \circ (\mathrm{id}_A \times t_{B,C}))(a,(b,t))\\
=&(t_{A,B\times C} \circ (\mathrm{id}_A \times t_{B,C}) \circ a_{A,B,TC})((a,b),t)
\end{aligned}$$

as required. Compatibility with unit and multiplication result in the identities

$$(t_{A,B} \circ (\mathrm{id}_A \times \eta_B))(a,b) = (Tp_a)(\eta_B(b)) = \eta_{A\times B}(p_a(b)) = \eta_{A\times B}(a,b),$$

which holds by naturality of η, and

$$\begin{aligned}
(\mu_{A\times B} \circ Tt_{A,B} \circ t_{A,TB})(a,t) &= (\mu_{A\times B} \circ Tt_{A,B} \circ Tp_a)(t)\\
&= (\mu_{A\times B} \circ T(t_{A,B} \circ p_a))(t) = (\mu_{A\times B} \circ (T^2 p_a))(t)\\
&= (Tp_a \circ \mu_B)(t) = (t_{A,B} \circ (\mathrm{id}_A \times \mu_B))(a,t),
\end{aligned}$$

valid by naturality of μ and the fact that $(t_{A,B} \circ p_a)(t) = t_{A,B}(p_a(t)) = t_{A,B}(a,t) = (Tp_a)(t)$. □

An alternative proof of the previous example uses the fact that all monads on **Set** are **Set**-enriched, a notion which coincides with strength [12, 21]. The fact that every monad on **Set** admits at least one strength raises the question whether there are categories with computationally relevant monads where strength is not inherent. It is hard to find such an example. However, it is known that there are non-strong monads on **Cat**. The following example is due to Power and relies on the fact that every strong monad on **Cat** is **Cat**-enriched.

Example 5 (Monads without a strength, Example 3.1 of [19])**.** Consider the endofunctor T on **Cat** given by

$$T(C) = C + (N \times \mathrm{Ob}\,(C))$$

where N is the discrete category on the natural numbers. The action of T on maps is evident. The functor T is finitary and is the functor part of the monad on **Cat** for which an algebra is a category C together with a function $E : \mathrm{Ob}\,(C) \to \mathrm{Ob}\,(C)$.

But T cannot be extended to 2-cells, even to invertible ones. For instance, putting $C = 1$, taking D to be **Iso**, the two object category that is equivalent to 1, and letting f and g be the two distinct functors from C to D, it follows that f is isomorphic to g but Tf is not isomorphic to Tg. So there is no possible enrichment of the ordinary functor T over either **Cat** or **Gpd**.

2.3 Exceptions

The main feature of our calculus in comparison with the work of Goncharov and Schröder [9] is the support for catching exceptions. In the following section, we explain what exceptions are, how they can be used, and, more importantly, how they can be formulated semantically in our categorical setting. This treatment is in accordance with the work of Schröder and Mossakowski [20]. The metalanguage of the previous sections is furthermore extended by appropriate types and operations.

Exceptions are a concept for interrupting the flow of program execution as a response to unintended, but predictable erroneous behaviour. Many programming languages implement support for exceptions, for instance C++, Python, Lisp, PHP or Java. Basically, a framework for exception handling concerns two operations *raise* and *catch*. The former signals that an anomaly has occurred and the normal flow of execution is changed, normally jumping to the next *catch*, where the exceptional behaviour is handled.

A typical example in Java (where the *raise* operation is called *throw*) might look like this:

```
try {
        throw new ExampleException();
} catch {
        // do something
}
```

The `try`-block marks the segment of code that may terminate abruptly with an exception.

This works well in imperative languages, where the flow of execution can be interrupted and changed easily. In a functional setting, however, program flow is created by composing functions. Here, it is vital that every function (except catch) which is chained to a *raise* operation effectively acts as the identity function:

$$\mathsf{do}\ x \leftarrow \mathsf{raise}\, e; p = \mathsf{raise}\, e$$

In this way, the exception that has been raised is passed through the program until it is handled by *catch*, making it visible to the program, so that exceptional measures might be taken.

Remark. It is important to notice that exceptional abortion of a program is a different matter than divergence.

2.3.1 Coproducts

One way to model a computation of type A which might raise an exception is as a computation with result either of type A or a specified type of exceptions which we denote by E. In other words, a candidate for an endofunction for a suitable monad would be

$$TA = A + E$$

Therefore, we need to impose the existence of coproducts on our base category $\mathcal{C}$ and add types $A + B$, interpreted using those coproducts as $[\![A + B]\!] = [\![A]\!] + [\![B]\!]$. The additional term formation rules can be found in the upper part of Figure 5.

Interpretation of the additional rules for coproducts is as follows. Given $[\![\Gamma \vdash p : A]\!] = f$ or $[\![\Gamma \vdash q : B]\!] = g$, we have $[\![\Gamma \vdash \mathsf{inl}\, p : A + B]\!] = \mathsf{inl} \circ f$ and $[\![\Gamma \vdash \mathsf{inr}\, q : A + B]\!] = \mathsf{inr} \circ g$, respectively, where $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$ are the usual left and right coproduct injection maps.

For the interpretation of **(case)**, however, we have to restrict the base category even further. The interpretations of the terms in the premise give us the following functions:

$$\begin{aligned} [\![\Gamma \vdash p : A + B]\!] &= h & h &: [\![\Gamma]\!] \to [\![A]\!] + [\![B]\!] \\ [\![\Gamma, a : A \vdash q : C]\!] &= f & f &: [\![\Gamma]\!] \times [\![A]\!] \to [\![C]\!] \\ [\![\Gamma, b : B \vdash r : C]\!] &= g & g &: [\![\Gamma]\!] \times [\![B]\!] \to [\![C]\!] \end{aligned}$$

Similar to the **(do)** rule, we may first propagate the context, using the pairing morphism $\langle \mathrm{id}_{[\![\Gamma]\!]}, h \rangle : [\![\Gamma]\!] \to [\![\Gamma]\!] \times ([\![A]\!] + [\![B]\!])$. From f and g, we receive a unique copairing morphism $[f, g] : [\![\Gamma]\!] \times [\![A]\!] + [\![\Gamma]\!] \times [\![B]\!] \to [\![C]\!]$. To compose $[f, g]$ and $\langle \mathrm{id}_{[\![\Gamma]\!]}, h \rangle$, we need to assume that in our base category products distribute over coproducts, i.e. the canonical distributivity morphism

$$[\mathrm{id} \times \mathsf{inl}, \mathrm{id} \times \mathsf{inr}] : (A \times B) + (A \times C) \to A \times (B + C)$$

is an isomorphism [4], whose inverse we denote by dist. This morphism must take $\langle a, \mathsf{inl}\, b \rangle$ to $\mathsf{inl}\, \langle a, b \rangle$ and $\langle a, \mathsf{inr}\, e \rangle$ to $\mathsf{inr}\, \langle a, e \rangle$. Taking advantage of this structure, we define the semantics for **(case)** as

$$[\![\Gamma \vdash \mathsf{case}\ p\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r]\!] = [f, g] \circ \mathrm{dist} \circ \langle \mathrm{id}, h \rangle.$$

Using the sum type $1 + 1 = 2$, we can derive a *conditional* term constructor **if**,

$$\textbf{(if)}\ \frac{\Gamma \vdash b : 2 \quad \Gamma \vdash s : A \quad \Gamma \vdash t : A}{\Gamma \vdash \mathsf{if}\, b\, \mathsf{then}\, s\, \mathsf{else}\, t : A} = \frac{\Gamma \vdash b : 1 + 1 \quad \Gamma \vdash s : A \quad \Gamma \vdash t : A}{\Gamma \vdash \mathsf{case}\ b\ \mathsf{of}\ \mathsf{inl} \star \mapsto s;\ \mathsf{inr} \star \mapsto t : A}.$$

The **case** construction supports the following axioms, where $x, y \notin \mathrm{Vars}(u) \cup \mathrm{Vars}(w)$ in the last axiom (see for example [8]):

$$\mathsf{case}\ \mathsf{inl}\, s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto t;\ \mathsf{inr}\, y \mapsto u = t[s/x] \tag{1}$$

$$\mathsf{case}\ \mathsf{inr}\, s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto t;\ \mathsf{inr}\, y \mapsto u = u[s/y] \tag{2}$$

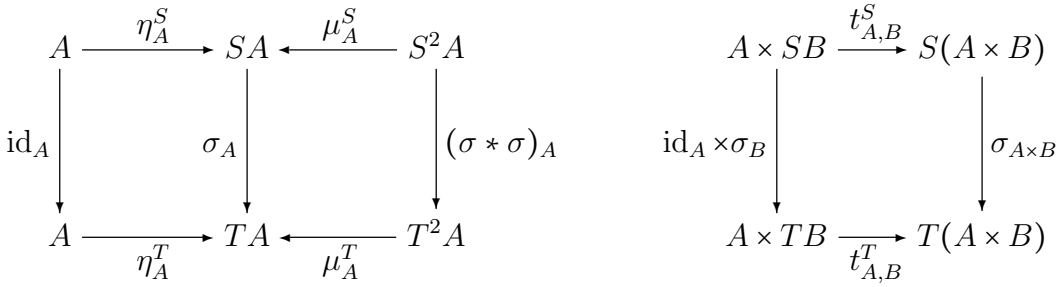
$$\mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{inl}\, x;\ \mathsf{inr}\, y \mapsto \mathsf{inr}\, y = s \tag{3}$$

$$\mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto t[u/z];\ \mathsf{inr}\, y \mapsto t[w/z] = t[(\mathsf{case}\ s\ \mathsf{of}\ \mathsf{inl}\, x \mapsto u;\ \mathsf{inr}\, y \mapsto w)/z] \tag{4}$$

2.3.2 Monad morphisms

Using $T = - + E$ is only of limited help, because what we are actually aiming for is a language which supports exceptions *in addition to* some other monad. We therefore need to construct a new monad, adding exceptions on top of an existing one. For this, we need a preliminary definition, describing morphisms between monads.

Definition 4 (Strong monad morphism)**.** Given two strong monads $\mathbb{S} = (S, \eta^S, \mu^S)$, $\mathbb{T} = (T, \eta^T, \mu^T)$ with strengths t^S, t^T, respectively, over a category $\mathcal{C}$, a natural transformation $\sigma : S \to T$ between them is called *(simplified) strong monad morphism* [14] if the following diagrams commute (where $*$ denotes the *Godement product* of natural transformations, see for example [3]):



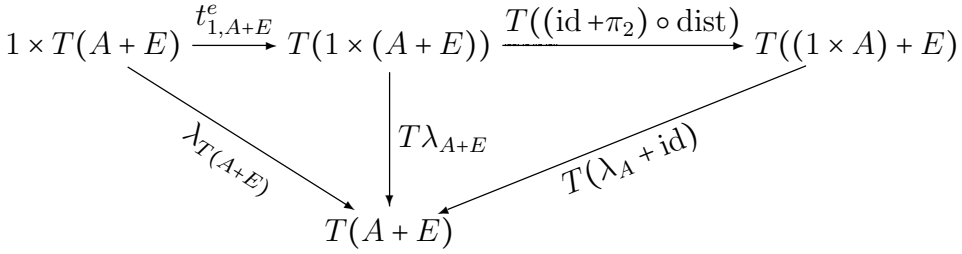
Monads supporting exceptions can be obtained by applying Moggi's exception monad transformer [14]. Schröder and Mossakowski proved in [20] that this kind of construction uniquely characterises monads modeling exceptions. We therefore present the following definition:

Definition 5 (Exception monad)**.** Let $\mathbb{T}$ be a Kleisli triple. An *exception monad* $\mathbb{T}_e$ for $\mathbb{T}$ is given by the following data:

- An object E (*exceptions*)
- $T_e = T(- + E)$ whose action on morphisms $f : A \to B$ is $T_e f = T[\mathsf{inl} \circ f, \mathsf{inr}]$
- $\eta^{T_e} = \eta^T \circ \mathsf{inl}$
- $f^* = [f, \eta^T \circ \mathsf{inr}]^*$ for $f : A \to T(B + E)$ (equivalently, $\mu^{T_e} = \mu^T \circ T[\mathrm{id}, \eta^T \circ \mathsf{inr}]$)
- $t^e_{A,B} = T((\mathrm{id} + \pi_2) \circ \mathrm{dist}) \circ t_{A,B+E}$

Proposition 1. *The morphism t^e makes $\mathbb{T}_e$ into a strong monad, i.e. for any strong monad $\mathbb{T}$, the exception monad $\mathbb{T}_e$ is a strong monad as well.*

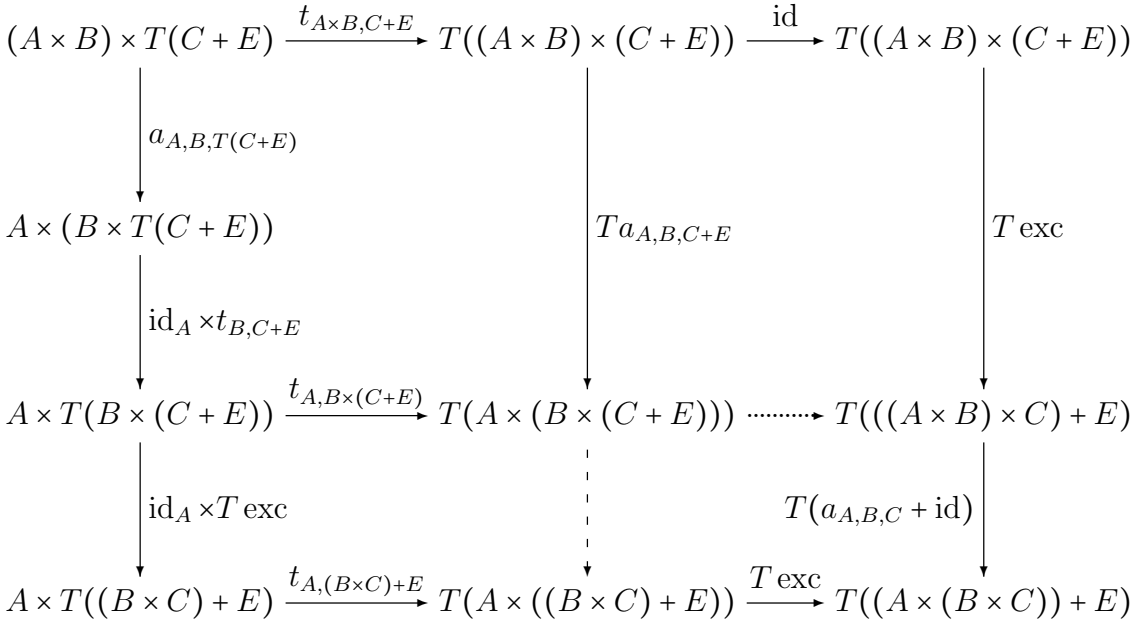
Proof. The first diagram of Definition 3 instantiates to the following:



The left half of this diagram already commutes, because t is a strength for T. Commutativity of the right half follows because, for $x : 1 \times (A + E)$,

$$((\lambda_A + \mathrm{id}) \circ (\mathrm{id} + \pi_2) \circ \mathrm{dist})(x) = ((\lambda_A + \pi_2) \circ \mathrm{dist})(x) = \lambda_{A+E}(x).$$

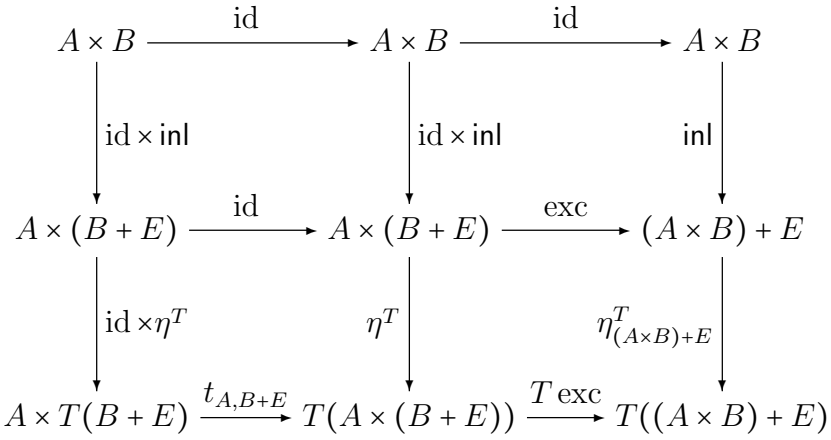
Let $\mathrm{exc} = (\mathrm{id} + \pi_2) \circ \mathrm{dist}$. From the second diagram of Definition 3, we want the outer square of the following to commute:



The dotted arrow is the unique arrow that makes the upper right square commute. The morphism exists because a is an isomorphism. It is namely $T\,\mathrm{exc} \circ T a^{-1}_{A,B,C+E}$. Then, the upper squares commute because t is strength for T. The dashed arrow in the lower square is $T(\mathrm{id}_A \times \mathrm{exc})$, so that the lower left square commutes by naturality of t. Commutativity of the lower right square then amounts to

$$((a_{A,B,C} + \pi_2) \circ \mathrm{dist} \circ a^{-1}_{A,B,C+E})\langle a, \langle b, \mathsf{inl}\, c\rangle\rangle = \mathsf{inl}\langle a, \langle b, c\rangle\rangle = (\mathrm{exc} \circ \mathrm{id} \times \mathrm{exc})\langle a, \langle b, \mathsf{inl}\, c\rangle\rangle$$
$$((a_{A,B,C} + \pi_2) \circ \mathrm{dist} \circ a^{-1}_{A,B,C+E})\langle a, \langle b, \mathsf{inr}\, e\rangle\rangle = \mathsf{inr}\, e = (\mathrm{exc} \circ \mathrm{id} \times \mathrm{exc})\langle a, \langle b, \mathsf{inr}\, e\rangle\rangle.$$

For compatibility with strength, consider the following diagram:



The upper left square commutes trivially. The lower left square is commutative by compatibility of t with η^T. Commutativity of the upper right square follows by Proposition 2.3.2. Finally, the lower right square commutes by naturality of η^T.

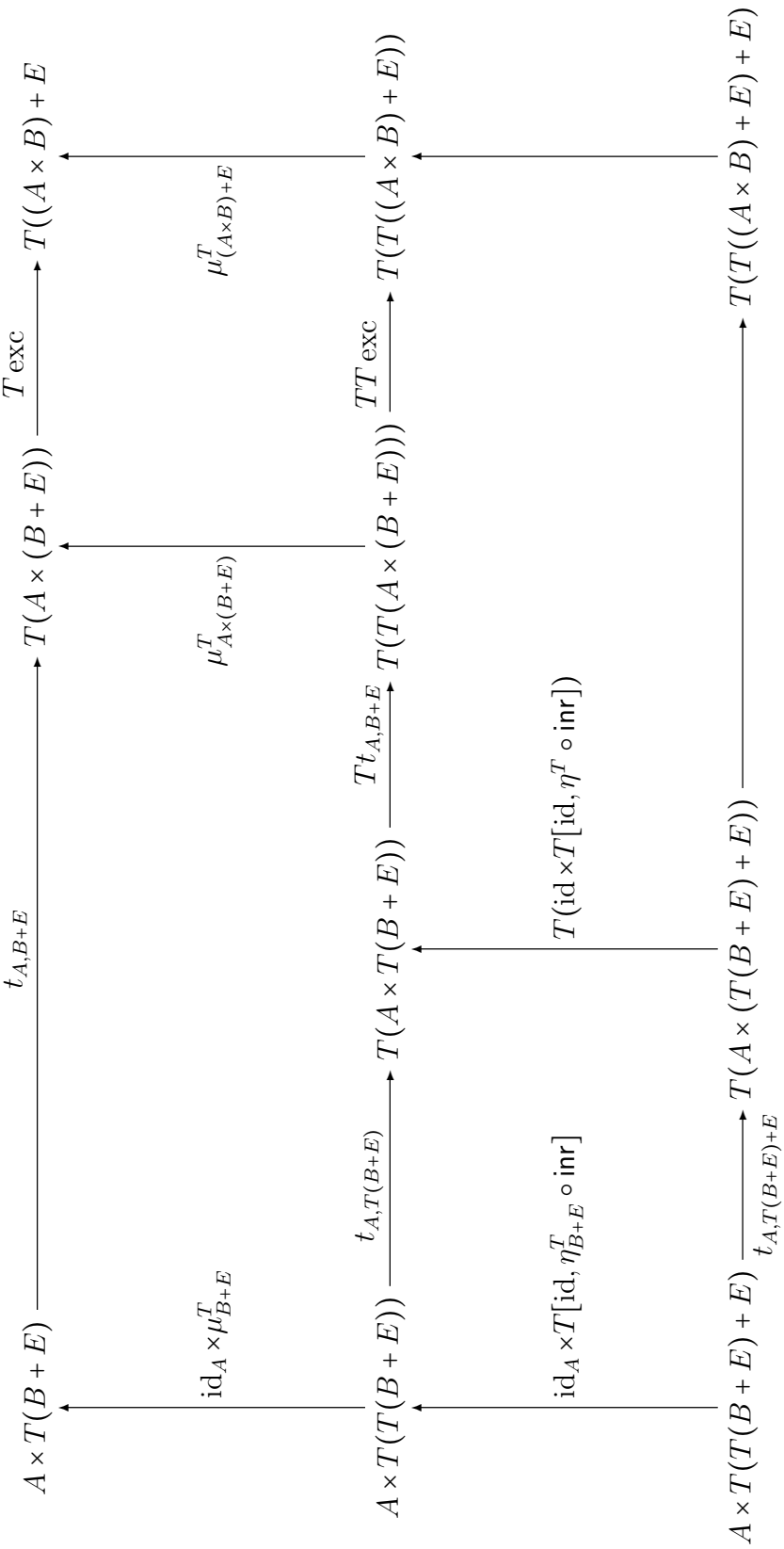


Figure 4: Compatibility of t^e with monad multiplication.

We are left to show compatibility with multiplication. The relevant diagram can be seen in Figure 4. The top left square commutes again because t is strength for T. The upper right square commutes by naturality of μ^T, the lower left one by naturality of t. The lower right square, with intermediate arrows and objects and with the outer layer of T removed, looks as follows:

$$\begin{array}{ccccc}
A \times T(B+E) & \xrightarrow{t_{A,B+E}} & T(A \times (B+E)) & \xrightarrow{T\,\mathrm{exc}} & T((A \times B)+E) \\
\uparrow{\scriptstyle \mathrm{id} \times T[\mathrm{id}, \eta^T \circ \mathsf{inr}]} & & & & \uparrow{\scriptstyle [\mathrm{id}, \eta^T \circ \mathsf{inr}]} \\
A \times (T(B+E)+E) & & & & \\
\downarrow{\scriptstyle \mathrm{exc}} & & & & \\
(A \times T(B+E)) + E & \xrightarrow{(t_{A,B+E} + \mathrm{id})} & T(A \times (B+E)) + E & \xrightarrow{(T\,\mathrm{exc} + \mathrm{id})} & T((A \times B)+E) + E
\end{array}$$

This obviously commutes, as the top and bottom routes perform the same actions; the only difference is that the outer coproduct is resolved at different stages of the computation. □

The choice of strength reflects the following: if we have a computation p from $T(B+E)$ and we want to combine this computation with a value from A, then if p raised an exception, we want the resulting computation to do so as well. Thus, the value from A is not needed, so that after distributing it in via dist, we can forget it in the right-hand side of the resulting coproduct to leave only the exception.

Next, we want to show that in our category $\mathcal{C}$ and for our way of constructing an exception monad, there is a natural candidate for a morphism which can act as the interpretation of *catch*.

Proposition 2. *For a strong monad T, the morphism $T\,\mathsf{inl} : T \to T(- + E)$ is a strong monad morphism.*

Proof. Using the fact that the monad units are natural transformations, we have

$$T\,\mathsf{inl} \circ \eta^T_A = \eta^T_{A+E} \circ \mathsf{inl} = \eta^{T(-+E)}_A.$$

The right-hand side of the left diagram of Definition 4 boils down to the following identity:

$$T\,\mathsf{inl} \circ \mu^T_A = \mu^T_{(A+E)} \circ TT\,\mathsf{inl} = \mu^{T(-+E)}_{(A+E)} \circ (T\,\mathsf{inl} * T\,\mathsf{inl}),$$

where the first equality holds by naturality of μ. Expanding the right-hand side by the definition of Godement product and exception monad above yields

$$\mu^T_{(A+E)} \circ TT\,\mathsf{inl} = \mu^T_{(A+E)} \circ T[\mathrm{id}, \eta^T_{(A+E)} \circ \mathsf{inr}] \circ (T\,\mathsf{inl})_{T(A+E)} \circ TT\,\mathsf{inl}\,.$$

This equality holds because $T[\mathrm{id}, \eta^T_{(A+E)} \circ \mathsf{inr}] \circ (T\,\mathsf{inl})_{T(A+E)}$ is easily seen to be the identity function.

Finally, compatibility with strength is represented by the identity

$$\begin{aligned}(T\,\mathsf{inl})_{A\times B} \circ t^T_{A,B} &= t^{T(-+E)}_{A,B} \circ (\mathrm{id}_A \times (T\,\mathsf{inl})_B) \\ &= T((\mathrm{id} + \pi_2) \circ \mathrm{dist}) \circ t^T_{A,B+E} \circ (\mathrm{id}_A \times (T\,\mathsf{inl})_B) \\ &= T((\mathrm{id} + \pi_2) \circ \mathrm{dist}) \circ T\,(\mathrm{id}_A \times \mathsf{inl}_B) \circ t^T_{A,B}.\end{aligned}$$

Again, we expand by definition on the right-hand side and use naturality of t. We have

$$(\mathrm{id} + \pi_2) \circ \mathrm{dist} \circ (\mathrm{id}_A \times \mathsf{inl}_B) = \mathsf{inl}\,.$$

This is because $(\mathrm{id}_A \times \mathsf{inl}_B)$ is exactly the left part of the canonical distributivity morphism whose inverse is dist. □

The exception monad construction gives rise to a strong monad morphism $catch : T_e \to T_e(-+E)$, which is the equalizer of $catch_{(-+E)}$ and $T_e\,\mathsf{inl} : T_e(-+E) \to T_e(-+E+E)$. For an exception monad $T^e = T(-+E)$ over a strong monad T, we have

$$\mathsf{catch} = T\,\mathsf{inl}\,. \tag{5}$$

With this choice of representation, we can formulate the requirements for the *catch* operation equationally, giving an equivalent characterization of exception monads (cf. [20]):

$$\mathsf{catch}(\mathsf{ret}\,x) = \mathsf{ret}(\mathsf{inl}\,x) \tag{6}$$

$$\mathsf{catch}(\mathsf{do}\;x \leftarrow p; q) = \mathsf{do}\;y \leftarrow \mathsf{catch}\,p; (\mathsf{case}\;y\;\mathsf{of}\;\mathsf{inl}\,x \mapsto \mathsf{catch}\,q; \mathsf{inr}\,e \mapsto \mathsf{ret}(\mathsf{inr}\,e)) \tag{7}$$

$$\mathsf{catch}(\mathsf{raise}\,e) = \mathsf{ret}(\mathsf{inr}\,e) \tag{8}$$

$$\mathsf{catch}(\mathsf{catch}\,p) = \mathsf{do}\;y \leftarrow \mathsf{catch}\,p; \mathsf{ret}(\mathsf{inl}\,y) \tag{9}$$

$$p = \mathsf{do}\;y \leftarrow \mathsf{catch}\,p; (\mathsf{case}\;y\;\mathsf{of}\;\mathsf{inl}\,x \mapsto \mathsf{ret}\,x; \mathsf{inr}\,e \mapsto \mathsf{raise}\,e) \tag{10}$$

Equation 6 and Equation 7, which formally state that catch is a monad morphism [20], intuitively realise the requirement that ret does not raise exceptions, so that the result of the computation can be accessed, and that catching an exception in a compound expression $\mathsf{do}\;x \leftarrow p; q$ amounts to handling possible exceptions of p and then returning a right injection if there were any, continuing with computing q otherwise, but wrapping the computation into catch again.

Obviously, Equation 8 states that exceptions are actually caught [20], making them available to the rest of the program as a right injection. The justification for this equation follows in the next section.

Equation 9 ensures that catch equalises itself and $T\,\mathsf{inl}$ [20]. In terms of programming, this means that catch does not raise exceptions, so catching the result of catch is just a left injection. Notice that this adds another "layer" of coproducts, so that previously unhandled exceptions are not lost.

The last equation, Equation 10, says that executing a program p is the same as catching possible exceptions in p, returning the result unchanged if it terminated normally and raising the same exception again if it resulted in an exception, just as expected [20].

$$\textbf{(inl)}\ \frac{\Gamma \vdash p : A}{\Gamma \vdash \mathsf{inl}\, p : A + B} \qquad \textbf{(inr)}\ \frac{\Gamma \vdash p : B}{\Gamma \vdash \mathsf{inr}\, p : A + B}$$

$$\textbf{(case)}\ \frac{\Gamma \vdash p : A + B \quad \Gamma, a : A \vdash q : C \quad \Gamma, b : B \vdash r : C}{\Gamma \vdash \mathsf{case}\, p\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q;\, \mathsf{inr}\, b \mapsto r}$$

$$\textbf{(raise)}\ \frac{\Gamma \vdash e : E}{\Gamma \vdash \mathsf{raise}\, e : TA} \qquad \textbf{(catch)}\ \frac{\Gamma \vdash p : TA}{\Gamma \vdash \mathsf{catch}\, p : T(A + E)}$$

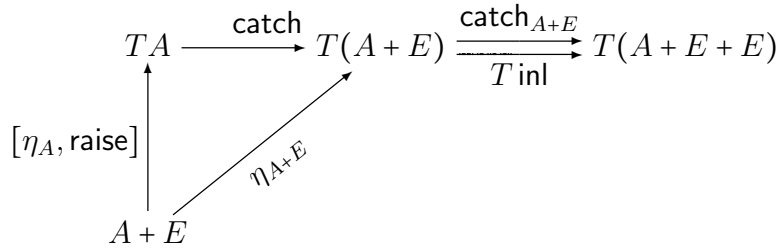
Figure 5: Term formation rules for coproducts, raising and catching exceptions.

2.3.3 Semantics

To implement the features from the last section in our metalanguage, we define the rules in the lower part of Figure 5. The intepretation of the **(catch)** rule is straightforward:

$$[\![\Gamma \vdash \mathsf{catch}\, p : T(A + E)]\!] = \mathsf{catch} \circ [\![p]\!].$$

To define semantics for **(raise)**, however, we cannot use the same approach, as our definition of exception monad does not require a separate *raise* morphism. As Schröder and Mossakowski pointed out [20], *raise* is derivable from their definition of the exception handling mechanism. Because catch is defined as the equalizer of catch and $T\,\mathsf{inl}$ and η must equalise catch_{-+E} and $T\,\mathsf{inl}$ as a consequence of compatibility of monad morphisms with the monad unit, we obtain a factorisation of η through catch as seen in the following diagram:



This is basically a diagrammatic version of Equation 8. The factorisation

$$\mathsf{catch} \circ [\eta_A, \mathsf{raise}] = \eta_{A+E}$$

must exist, because catch is a monad morphism and thus $\mathsf{catch} \circ \eta = T\,\mathsf{inl} \circ \eta = \eta \circ \mathsf{inl}$ (by naturality of η), which means that η equalises catch and $T\,\mathsf{inl}$. Hence, $[\eta_A, \mathsf{raise}]$ is taken to be the unique morphism from $A + E$ to TA. Then we get a simple interpretation for **(raise)** as well:

$$[\![\Gamma \vdash \mathsf{raise}\, e : TA]\!] = \mathsf{raise} \circ [\![e]\!].$$

2.4 Summary

We have introduced many new restrictions and features to our initial metalanguage in the previous sections. Currently, the language can be described as follows:

- A strong exception monad $\mathbb{T}$
- Types $A ::= W \mid 1 \mid A \times A \mid A + A \qquad C ::= A \mid TA$
- A signature Σ containing functions $f : A \to C$, where A is a *value type*
- The term formation rules from Figure 3 and Figure 5

The base category so far needs to have equalizers to support the *catch* and *raise* operations. The category has to be distributive in order to interpret the *case* construction and to derive a strength for the exception monad from the original strength.

3 Predicates and Truth Values

Recall the definition of a classical *Hoare triple* $\{P\}\ C\ \{Q\}$. It says that if a *precondition* P holds, then the *postcondition* Q holds after executing C. The question is, however, what language to choose for those assertions. For the simple command language in the classical calculus, Hoare chose to represent the assertions using predicate logic [11, 5]. Consider now the case of commands built using our monadic metalanguage. We need to come up with a suitable notion of predicate which is able to express assertions for monadic programs. Indeed, following Goncharov and Schröder [9], it is possible to derive a truth value object from the monad itself, given that it fulfils certain properties which we have to impose.

3.1 Truth Value Objects

The two-valued Boolean algebra with elements usually called *true* and *false* is the most common understanding for a system of *truth values*. One often forgets, however, that this, on the one hand, is only a special case of a Boolean algebra, and on the other hand, that there are many different logics, which are be represented by a multitude of mathematical objects, where Boolean algebras are those admitting *classical* propositional logic.

The concept of order is deeply intertwined with logic. In fact, a Boolean algebra is nothing but a partial order with some additional conditions. Varying these conditions yields different logics, each with its own level of expressivity. In the following, we will give definitions for some of these structures, beginning with a relatively weak one.

Definition 6 (Distributive lattice)**.** A *distributive lattice* is a *lattice*[1] L in which the distributivity laws for join and meet are satisfied, i.e.

$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$$
$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$$

for all $x, y, z \in L$.

The logic corresponding to this type of lattice is very weak. We have a bottom and top element and thus notions of truth and falsity and—as the name suggests—the usual distributivity laws, yet we only have meet and join (i.e. logical *and* and *or*) as logical connectives, but no implication!

[1] We take a *lattice* L to be the algebraic structure of a non-empty set L together with operations *meet* and *join* and the constants $\top$ and $\bot$.

Definition 7 (Heyting algebra)**.** A *Heyting algebra* is a lattice L together with an *implication* operation

$$\Rightarrow\colon L \times L \to L$$

satisfying the condition

$$x \sqcap a \sqsubseteq b \Leftrightarrow x \sqsubseteq (a \Rightarrow b).$$

Heyting algebras are the objects satisfying the axioms of *intuitionistic logic*, which is a full logic with meets, joins, implication (allowing the definition of negation as implication to bottom), and one of the de Morgan Laws, namely $\neg x \sqcap \neg y = \neg(x \sqcup y)$. Other than in classical logic, statements do not have to be either true or false. In other words, Heyting algebras in general do not admit the axiom of excluded middle, $a \sqcup \neg a = \top$. This means that if we want to attribute a truth value to a statement, we need to be able to give a proof for it. Another law from classical logic which does *not* hold here is double negation elimination, $\neg\neg a = a$. To be able to formulate these axioms and arrive at classical logic, it is only necessary for every element in the lattice to have a complement:

Definition 8 (Boolean algebra)**.** A *Heyting algebra* L in which every element admits a complement, i.e. $\forall x \,.\, \exists y \,.\, (x \sqcap y = \bot \text{ and } x \sqcup y = \top)$, is called a *Boolean algebra*.

Complements, if they exist, are unique, because if y, y' are complements for the same element x, then

$$y' = y' \sqcap (x \sqcup y) = (y' \sqcap y) \sqcup (y' \sqcap x) = y' \sqcap y$$

and vice versa. Furthermore, let $x \sqcap y = \bot$ and $x \sqcup y = \top$. For any other element z with $x \sqcap z = \bot$, we have

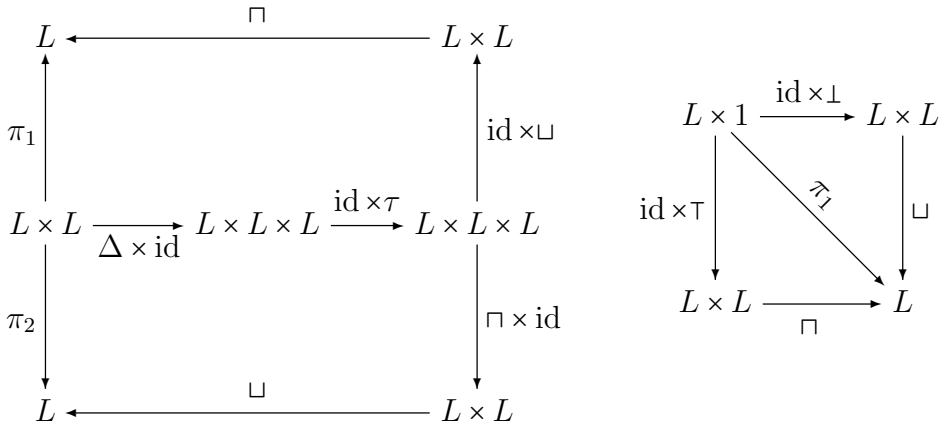
$$z = z \sqcap (x \sqcup y) = (z \sqcap x) \sqcup (z \sqcap y) = z \sqcap y$$

and thus $z \sqsubseteq y$, from which follows by the Heyting algebra condition that $y = (x \Rightarrow \bot) = \neg x$. Because this is true for every element, the excluded middle axiom holds. For double negation elimination, we already have $x \sqsubseteq \neg\neg x$ since $x \sqcap \neg x = \bot$, and we derive

$$\neg\neg x = \neg\neg x \sqcap (x \sqcup \neg x) = (\neg\neg x \sqcap x) \sqcup (\neg\neg x \sqcap \neg x) = \neg\neg x \sqcap x \sqsubseteq x.$$

Obviously, the lattices described are only sets with additional structure. Thus, if we were working in the category **Set**, we would have all those logics at our disposal. We do, however, not want to restrict ourselves to the case of **Set** and therefore have to make sure that we can internalise the mathematical objects in question in our category, that is come up with a definition which, when used in **Set**, is equivalent to the usual definition given above.

Definition 9 (Lattice object)**.** A *lattice object* in a category $\mathcal{C}$ is an object L of $\mathcal{C}$ and commutative, associative, idempotent operations $\sqcap, \sqcup : L \times L \to L$, as well as arrows $\bot, \top : 1 \to L$, such that the following diagrams commute:



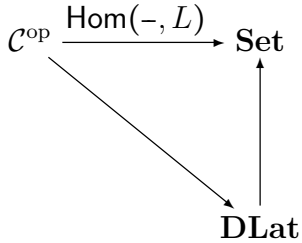
As expected, the lattice object is said to be distributive if the respective diagrams given by the axioms in Definition 6 commute.

Definition 10 (Heyting algebra object)**.** A *Heyting algebra object* in a category $\mathcal{C}$ is a *lattice object* H with an additional morphism $\Rightarrow\colon H \times H \to H$ which satisfies the diagrams given by the following identities:

$$\begin{aligned} x \Rightarrow x &= \top \\ x \sqcap (x \Rightarrow y) &= x \sqcap y \\ y \sqcap (x \Rightarrow y) &= y \\ x \Rightarrow (y \sqcap z) &= (x \Rightarrow y) \sqcap (x \Rightarrow z) \end{aligned}$$

These are easily recognized as the axioms of intuitionistic logic concerning implication.

Goncharov and Schröder define their notion of lattice object indirectly. Namely, they say that L is a distributive lattice object if the hom-functor $\mathsf{Hom}(-, L)$ factors through distributive lattices [9]:



Here, **DLat** is the category of distributive lattices and lattice homomorphisms. The diagram says that every hom-set $\mathsf{Hom}(A, L)$ is a distributive lattice and the maps $\mathsf{Hom}(f, L) : \mathsf{Hom}(A, L) \to \mathsf{Hom}(B, L)$ for $f : B \to A$ are lattice homomorphisms.

3.2 Order-Enrichment

In the light of the previous section, it becomes clear that we need to enrich our monad over a partial order structure to come closer to the goal of having a truth value object.

Informally, this means that we employ an order on programs, i.e. *Kleisli arrows*, of type $A \to TB$. This will later allow us to use certain well-behaved programs as logical assertions. Intuitively, programs are ordered by *information content* in the vein of domain theory, with a bottom element representing the fact that no information has been obtained from a computation and $p \sqsubseteq q$ specifying that p carries part of the information of q.

Definition 11 (Bounded-complete dcpo)**.** A partial order $(S, \sqsubseteq)$ over a non-empty set S is called *directed-complete* (dcpo) if all directed subsets $D \subseteq S$, i.e. subsets with joins $x \sqcup y \in D$ for all $x, y \in D$, have a supremum $\bigsqcup D$. A dcpo is furthermore called *bounded-complete* if every subset of S which has an upper bound has a join.

Remark. This definition implies the existence of a bottom element $\bot$, because the empty set obviously has an upper bound in S.

Definition 12 (Scott-continuous maps [24])**.** A function f between two posets S and T is called *Scott-continuous* if it preserves all directed joins, i.e. if D is a directed subset of S, then $f(\bigsqcup D) = \bigsqcup_{d \in D} f(d)$.

Definition 13 (Order-enriched monad [9])**.** A strong monad $\mathbb{T}$ over a category $\mathcal{C}$ is called *order-enriched* if

- every hom-set $\mathsf{Hom}_{\mathcal{C}}(A, TB)$ carries the structure of a bounded-complete dcpo,
- for any $h \in \mathsf{Hom}_{\mathcal{C}}(A', A)$ and any $u \in \mathsf{Hom}_{\mathcal{C}}(B, TB')$, the maps

  $$f \mapsto f \circ h, \qquad f \mapsto u^* \circ f, \qquad f \mapsto t \circ \langle \mathrm{id}, f \rangle$$

  preserve all existing joins (including the empty join $\bot$),
- Kleisli star is Scott-continuous.

Recall the interpretation of the **do** construction:

$$[\![\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : TB]\!] = q^* \circ t_{[\![\Gamma]\!],[\![A]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, p \rangle$$

for $[\![\Gamma \vdash p : TA]\!] = p$, $[\![\Gamma, x : A \vdash q : TB]\!] = q$. The **do** construction thus preserves all existing joins in the left argument p. Scott-continuity of the Kleisli star means that it preserves directed joins in the right argument q if the monad $\mathbb{T}$ is order-enriched [9].

Example 6 (State and order-enrichment)**.** Let us return to the state monad example. To make it work in the updated setting, we have to make a slight adjustment. The (total) state monad from Example 1, where $A \mapsto S \to (A \times S)$, does not carry a dcpo structure in a natural way. However, if we consider the *partial* state monad $TA = S \rightharpoonup (A \times S) \cong S \to (A \times S) + 1$, we can employ the so-called *extension order*.

Definition 14 (Extension order [23])**.** The partial functions between two sets A and B give rise to a partial order in the following way: For $f, g : A \rightharpoonup B$, $f \sqsubseteq g$ (we say g *extends* f) if, for all $a \in A$, the existence of $f(a)$ implies the existence of $g(a)$, and then $f(a) = g(a)$.

The bottom element $\bot$ of an object TB is simply the partial function that is undefined for all arguments, so that $\bot \sqsubseteq t$ for every $t \in TB$. Let us denote the *support* of a partial function f, i.e. the set of elements of its domain for which it is defined, by $\operatorname{supp} f$. Then two elements $s, t \in TB$ have a join $f \sqcup g$ if

$$\forall x \in \operatorname{supp} f \cap \operatorname{supp} g \,.\, f(x) = g(x).$$

Moreover, $\operatorname{supp}(f \sqcup g) = \operatorname{supp} f \cup \operatorname{supp} g$ where

$$(f \sqcup g)(x) = \begin{cases} f(x) & x \in \operatorname{supp} f \\ g(x) & \text{otherwise} \end{cases}.$$

From this definition it follows easily that every directed subset of TB has a join. Suppose now that $f \sqsubseteq h$ and $g \sqsubseteq h$ for some h. This means that

$$\operatorname{supp} h \supseteq (\operatorname{supp} f \cup \operatorname{supp} g),$$

and because

$$(\forall x \in \operatorname{supp} f \,.\, f(x) = h(x)) \wedge (\forall x \in \operatorname{supp} g \,.\, g(x) = h(x)) \\ \Rightarrow \forall x \in \operatorname{supp} f \cap \operatorname{supp} g \,.\, f(x) = g(x),$$

f and g have a join, namely the restriction of h to $(\operatorname{supp} f \cup \operatorname{supp} g)$. TA is thus bounded-complete.

Note that we have now shown that the objects TA carry the required structure. It remains to show that this can be extended to the hom-sets $\mathsf{Hom}(A, TB)$.

The partial order on $\mathsf{Hom}(A, TB)$ is defined pointwise. For two functions $f, g \in \mathsf{Hom}(A, TB)$,

$$f \sqsubseteq g \Leftrightarrow \forall x \in A \,.\, f(x) \sqsubseteq g(x).$$

Joins are also computed pointwise, that is $f \sqcup g = \lambda a. f(a) \sqcup g(a)$ if the joins on the right-hand side exist for all $a \in A$, and the bottom element is $\bot_{A,TB} = \lambda a \,.\, \bot_{TB}$.

Indeed, if D is a directed subset of $\mathsf{Hom}(A, TB)$, then for every $a \in A$, the set $\{d(a)\}_{d \in D}$ is a directed subset of TB, which has a join, so $\bigsqcup D = \bigsqcup \{d(a)\}_{d \in D}$ exists.

Precomposition, that is the map $f \mapsto f \circ h$ (with types as given in the definition above) obviously preserves all existing joins, since for any set S whose join exists

$$\bigsqcup S \circ h = \left(\lambda a \,.\, \bigsqcup_{f \in S} f(a)\right) \circ h = \lambda a' \,.\, \bigsqcup_{f \in S} f(h(a')) = \bigsqcup_{f \in S} (f \circ h).$$

For Kleisli composition, we have

$$\begin{aligned} (u^* \circ \bigsqcup S)(a) &= (u^* \circ (\lambda a \,.\, \bigsqcup_{f \in S} f(a)))(a) \\ &= u^*(\bigsqcup_{f \in S} f(a)) \overset{(*)}{=} \bigsqcup_{f \in S} u^*(f(a)) \\ &= (\lambda a \,.\, \bigsqcup_{f \in S} (u^* \circ f)(a))(a) = (\bigsqcup_{f \in S} (u^* \circ f))(a). \end{aligned}$$

We can justify the step $(*)$ with the definition of joins on TB. Postcomposition with u^* in every case of the stepwise defined function yields a stepwise defined function on TB' which qualifies for being a join.

By a similar argument (recall that $p_a : b \mapsto (a, b)$ for $a \in A$),

$$(t \circ \langle \mathrm{id}, \bigsqcup_{f \in S} f \rangle)(a) = (Tp_a)(\bigsqcup_{f \in S} f(a)) = \bigsqcup_{f \in S} (Tp_a)(f(a)) = (\bigsqcup_{f \in S} (t \circ \langle \mathrm{id}, f \rangle))(a).$$

Scott continuity of the Kleisli star operator means that if D is a directed subset of $\mathsf{Hom}(A, TB)$, then $(\bigsqcup D)^* = \bigsqcup_{d \in D} d^*$.

$$\begin{aligned}
(\bigsqcup D)^* &= (\lambda a \,.\, \bigsqcup_{d \in D} d(a))^* \\
&= \lambda r \,.\, \lambda s \,.\, \left(\lambda a \,.\, \bigsqcup_{d \in D} d(a)\right)(\pi_1 r(s))\,(\pi_2 r(s)) \\
&= \lambda r \,.\, \lambda s \,.\, \left(\bigsqcup_{d \in D} d(\pi_1 r(s))\right)(\pi_2 r(s)) \\
&\overset{(*)}{=} \lambda r \,.\, \bigsqcup_{d \in D} \lambda s \,.\, d(\pi_1 r(s))(\pi_2 r(s)) \\
&= \lambda r \,.\, \bigsqcup_{d \in D} d^*(r) = \bigsqcup_{d \in D} d^*.
\end{aligned}$$

Again, the identity marked $(*)$ holds, because the join exists by assumption, so if $D = \{d_1, \ldots, d_k\}$,

$$\left(\bigsqcup_{d \in D} d(\pi_1 r(s))\right)(x) = \begin{cases} d_1(\pi_1 r(s))(x) & x \in \operatorname{supp} d_1 \\ \vdots & \\ d_k(\pi_1 r(s))(x) & x \in \operatorname{supp} d_k. \end{cases}$$

Thus, we can push the lambda abstraction and evaluation into the different cases (which then are of type TB again) and obviously end up with another join.

3.3 Innocence

The order-enrichment discussed in the previous section alone still fails to provide a useful truth value object on which we could base a language of assertions. The object $T1$, which would be a candidate, does not come with the necessary structure, because $\mathsf{Hom}(A, T1)$ does not necessarily have a top element.

Goncharov and Schröder therefore introduced the notion of *innocent* monads. The programs in such a monad are particularly well-behaved with respect to side-effects (in other words, they are *innocently side-effecting*).

Definition 15 (Commutation [9])**.** Two programs p, q are said to *commute* if the equation

$$\mathsf{do}\ x \leftarrow p; y \leftarrow q; \mathsf{ret}\ \langle x, y \rangle = \mathsf{do}\ y \leftarrow p; x \leftarrow q; \mathsf{ret}\ \langle x, y \rangle \tag{11}$$

holds, where $x, y \notin \mathrm{Vars}(p) \cup \mathrm{Vars}(q)$. If every pair of programs for a given monad commutes, the monad is *commutative*.

Definition 16 (Innocent monad)**.** An order-enriched monad is called *innocent*, if it is *commutative* and all programs in it are *copyable*, that is

$$\mathsf{do}\ x \leftarrow p; y \leftarrow p; \mathsf{ret}\,\langle x, y\rangle = \mathsf{do}\ x \leftarrow p; \mathsf{ret}\,\langle x, x\rangle, \tag{12}$$

and *weakly discardable*, i.e.

$$\mathsf{do}\ x \leftarrow p; \mathsf{ret}\ \star \sqsubseteq \mathsf{ret}\ \star. \tag{13}$$

Programs from such a monad are called *innocent programs*.

This implies that a program in an innocent monad may not throw exceptions, because $\mathsf{do}\ x \leftarrow \mathsf{raise}\,e; \mathsf{ret}\ \star = \mathsf{raise}\,e$ cannot hold.

In such an innocent monad, binary joins are just sequential composition of two programs:

Lemma 1. *Let $\mathbb{P}$ be an innocent monad. Then, $p \sqcap q$ exists and is equal to $\mathsf{do}\ p; q = \mathsf{do}\ q; p$.*

Proof. [9] We first prove that $p \sqcap q \sqsubseteq \mathsf{do}\ p; q$:

$$\begin{aligned}
p \sqcap q &= \mathsf{do}\ p \sqcap q; p \sqcap q && \text{(copyability)}\\
&\sqsubseteq \mathsf{do}\ p; p \sqcap q && \text{(monotonicity)}\\
&= \mathsf{do}\ p \sqcap q; p && \text{(commutativity)}\\
&\sqsubseteq \mathsf{do}\ q; p. && \text{(monotonicity)}
\end{aligned}$$

Monotonicity of do is entailed by Scott-continuity of the Kleisli star: if $x \sqsubseteq y$, then $f(y) = f(\bigsqcup\{x, y\}) = \bigsqcup\{f(x), f(y)\}$, hence $f(x) \sqsubseteq f(y)$.

As p is weakly discardable, we have $p \sqsubseteq \mathsf{ret}\ \star$, which means that

$$\begin{aligned}
\mathsf{do}\ p; q &\sqsubseteq \mathsf{do}\,(\mathsf{ret}\ \star); q = q\\
\mathsf{do}\ q; p &\sqsubseteq \mathsf{do}\,(\mathsf{ret}\ \star); p = p.
\end{aligned}$$

Thus, $\mathsf{do}\ p; q = \mathsf{do}\ q; p$ is a lower bound of p and q, hence

$$\mathsf{do}\ q; p \sqsubseteq p \sqcap q.$$

□

More often than not, computationally relevant monads do not fulfil the requirements above. It is, however, often possible to identify the set of programs in a monad which do fulfil them with programs in a different, smaller monad, which turns out to be innocent. The following definitions capture this idea:

Definition 17 (Submonad [9])**.** A *submonad* of a monad $\mathbb{T}$ is a pair $(\mathbb{P}, \iota)$, where $\mathbb{P}$ is a monad and $\iota : P \hookrightarrow T$ is a componentwise monomorphic monad morphism (*inclusion morphism*).

Definition 18 (Order-enriched submonad)**.** A *submonad* of a monad $\mathbb{T}$ is order-enriched if its inclusion morphism is Scott-continuous.

Definition 19 (Predicated monad)**.** The pair $(\mathbb{T}, \mathbb{P})$ of an order-enriched monad $\mathbb{T}$ and an innocent order-enriched submonad $\mathbb{P}$ of $\mathbb{T}$ is called a *predicated monad*.

Example 7 (State and innocence)**.** To support copyability, a program p in the partial state monad $\mathbb{T}$ needs to be deterministic. Weak discardability of p implies that p may not write the state: the type $T1 = S \rightharpoonup (1 \times S)$ is isomorphic to $S \rightharpoonup S$, so the interpretation of $\mathsf{ret} \star : T1$, that is $\eta \circ !$, is basically the identity function on S.

Obviously, only few programs in the partial state monad fulfil those conditions. However, the partial reader monad $PA = S \rightharpoonup A$ consists of exactly the programs satisfying them. This monad can be embedded into $\mathbb{T}$ as a submonad $(\mathbb{P}, \iota)$ with $\iota_A = \lambda r \,.\, \lambda s \,.\, (r(s), s)$ for $r : PA$ which is obviously componentwise monic. It can easily be checked that ι is a monad morphism.

The most important property of innocent monads, however, is that they support a logic on our base category which we will use as the underlying logic of the assertions of our Hoare calculus in Chapter 4. This result by Goncharov and Schröder is captured in the following central theorem:

Theorem 1 (Theorem 14 of [9])**.** *Let $\mathbb{P}$ be an innocent monad. Then $P1$ is a distributive lattice object in $\mathcal{C}$ under the monad ordering, i.e. its hom-functor $\mathsf{Hom}(-, P1)$ factors through distributive lattices. In fact, $\mathsf{Hom}(-, P1)$ factors through frames, i.e. $P1$ has external joins, and finite meets in $P1$ distribute over these.*

From now on, we will also denote $P1$ by Ω, reminiscent of the subobject classifier in a topos.

As seen in the beginning of this chapter, the logic of a distributive lattice is very weak. However, as noted by Goncharov and Schröder [9], for any innocent monad $\mathbb{P}$ on **Set**, the hom-sets $\mathsf{Hom}, (A, P1)$ are actually complete Heyting algebras. There might be similar results for other concrete categories.

3.4 Order-Enrichment of Exception Monads

The framework was so far constructed in a way that one can choose an arbitrary monad and instantiate the metalanguage using the exception monad arising from it. As we have now restricted the choice of monads, we need to ensure that our construction mechanism for exception monads also preserves order-enrichment.

Lemma 2. *Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad. Then, the exception monad $\mathbb{T}_e$ is also order-enriched. Furthermore, $\mathbb{P}$ is an innocent submonad of $\mathbb{T}_e$, so that $(\mathbb{T}_e, \mathbb{P})$ is also a predicated monad.*

Proof. The hom-sets $\mathsf{Hom}(A, T_e B)$ are just $\mathsf{Hom}(A, T(B+E))$ by definition and thus carry the required structure by order-enrichment of $\mathbb{T}$.

The maps $f \mapsto f \circ h$ with $h : A' \to A$ and $f \mapsto u^* \circ f$ with $u : B \to T(B'+E)$ preserve directed joins by order-enrichment of T. The map $f \mapsto t^e \circ \langle \mathrm{id}, f \rangle = T([\mathsf{inl}, \mathsf{inr} \circ \pi_2] \circ \mathrm{dist}) \circ t \circ \langle \mathrm{id}, f \rangle$ preserves directed joins because $Tf = (\eta \circ f)^*$, so we have an instance of the map $f \mapsto u^* \circ f$.

Obviously, copairing with a fixed morphism $\eta \circ \mathsf{inr}$ is an order-preserving operation, such that Scott-continuity of the Kleisli star holds.

$(\mathbb{P}, \iota)$ is an innocent submonad of $\mathbb{T}$. It can be mapped into $\mathbb{T}_e$ by letting the inclusion morphism be $\iota^e_A = T\,\mathsf{inl} \circ \iota_A$. This morphism is Scott-continuous: $T\,\mathsf{inl} = (\eta \circ \mathsf{inl})^*$ is Scott-continuous by order-enrichment of T, ι is Scott-continuous by the definition of order-enriched submonad, and the composition of two continuous maps is continuous. □

Because catch operates on $\mathbb{T}_e$, innocent programs need to be transformed into programs of $\mathbb{T}_e$ before catch can act on them. From the definition of inclusion map from P to $T(- + E) = T_e$, it is clear that innocent programs p do not raise exceptions (also by weak discardability) and thus we can derive

$$\mathsf{catch}\, p = \mathsf{do}\ x \leftarrow p; \mathsf{ret}\,\mathsf{inl}\, x, \quad p \text{ innocent.} \tag{14}$$

3.5 Fixed Points and Loops

Order-enrichment of the underlying monad is not only the basis for the Hoare calculus. Programming languages that do not provide the possibility of iterated execution of a program segment are barely useful in practice. The order imposed on the monad structure, however, allows us to instantiate the following theorem.

Theorem 2 (Fixed point theorem [26])**.** *If D is a cpo and $f : D \to D$ is continuous, then $fix(f) = \bigsqcup_{n \in \omega} f^n(\bot)$, where ω is the first infinite ordinal, is a fixed point of f and the least prefixed point of f:*

1. $f(fix(f)) = fix(f)$
2. *if $f(d) \sqsubseteq d$ then $fix(f) \sqsubseteq d$.*

We will see in the next section how least fixed points relate to loops in programming languages.

3.5.1 Iterated case

Goncharov and Schröder [9] equip their metalanguage with a standard while loop, which is basically an iterated **if**. Since we have coproducts, we introduce a more general rule called **itercase**. The looping construction $\mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r$ initializes the variable x with the result of p and then proceeds with executing q

repeatedly as long as c is a left injection. When c eventually is a right injection, the loop terminates and r is executed once.

Interpretation of **itercase** follows [9]. Let $? : A + B \to PA$ be the operator defined by $c? = \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{ret}\, y;\ \mathsf{inr}\, z \mapsto \bot$ and let $\overline{c} = \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{inr}\, y;\ \mathsf{inr}\, z \mapsto \mathsf{inl}\, z$ for $c : A + B$ be the unique isomorphism from $A + B$ to $B + A$.

Lemma 3. *Assume terms* $\Gamma, y : A \vdash p : TC$, $\Gamma, z : B \vdash q : TC$, *and* $\Gamma \vdash c : A + B$. *Then the join* $(\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q)$ *exists and equals* $\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q$.

Proof.

$$\begin{aligned}
\mathsf{do}\ y \leftarrow c?; p &= \mathsf{do}\ y \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{ret}\, y;\ \mathsf{inr}\, z \mapsto \bot); p \\
&= \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\ y \leftarrow \mathsf{ret}\, y; p);\ \mathsf{inr}\, z \mapsto (\mathsf{do}\ y \leftarrow \bot; p) \\
&= \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto \bot \\
&\sqsubseteq \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q.
\end{aligned}$$

The inequality follows from the fact that $c?$ is weakly discardable and

$$\begin{aligned}
\mathsf{do}\ c?; (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q) &= \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\ c?; p);\ \mathsf{inr}\, z \mapsto (\mathsf{do}\ c?; q) \\
&= \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto \bot.
\end{aligned}$$

Analogously, we have $\mathsf{do}\ z \leftarrow \overline{c}?; q \sqsubseteq \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q$. Since $\mathbb{T}$ is order-enriched, the join must exist, because $(\mathsf{do}\ y \leftarrow c?; p)$ and $(\mathsf{do}\ z \leftarrow \overline{c}?; q)$ have a common upper bound.

Since

$$(\mathsf{do}\ y \leftarrow (\mathsf{inl}\, a)?; p) \sqcup (\mathsf{do}\ z \leftarrow \overline{\mathsf{inl}\, a}?; q) = p \sqcup \bot = p$$

and vice versa, the following equalities hold:

$$\begin{aligned}
\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q &= \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q); \\
&\qquad \mathsf{inr}\, z \mapsto (\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q) \\
&= (\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q)
\end{aligned}$$

□

As a result, **case** is Scott-continuous in both program arguments: for a directed subset $D \subseteq \mathsf{Hom}_{\mathcal{C}}(A, TB)$, we have

$$\begin{aligned}
\bigsqcup_{d \in D} \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto d;\ \mathsf{inr}\, z \mapsto q &= \bigsqcup_{d \in D} ((\mathsf{do}\ y \leftarrow c?; d) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q)) \\
&= (\bigsqcup_{d \in D} \mathsf{do}\ y \leftarrow c?; d) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q) \\
&= (\mathsf{do}\ y \leftarrow c?; \bigsqcup D) \sqcup (\mathsf{do}\ z \leftarrow \overline{c}?; q),
\end{aligned}$$

where the last equality follows from Scott continuity of the Kleisli star. The computation for the second argument is analogous.

Thus, in analogy to Goncharov and Schröder [9], we can define an interpretation for the special case $\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r$ as the least fixed point of the map

$$f \mapsto \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\ x \leftarrow q; f); \mathsf{inr}\, z \mapsto r. \tag{15}$$

This fixed point exists by Scott-continuity of **case** and Theorem 2.

The full construction is then interpreted as

$$\begin{aligned}&[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r : TA]\!] = \\ &\qquad [\![\Gamma \vdash \mathsf{do}\ x \leftarrow p; (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r) : TA]\!].\end{aligned} \tag{16}$$

3.5.2 While loop

An *iterated case* loop is unusual in everyday programming languages. However, a **while** loop can easily be derived using **itercase**. Let $b : 2$, then

$$[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{while}\, b\, \mathsf{do}\, q]\!] := [\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\, b\ \mathsf{of}\ \mathsf{inl}\, \star \mapsto q; \mathsf{inr}\, \star \mapsto \mathsf{ret}\, x]\!].$$

In fact, we can also express **itercase** in terms of the newly defined **while**, showing that **while** is equally expressive as **itercase**, by observing that

$$\begin{aligned}&[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r]\!] = \\ &\qquad [\![\Gamma \vdash \mathsf{do}\ x \leftarrow (\mathsf{init}\, x \leftarrow p\ \mathsf{while}\, (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{inl}\, \star; \mathsf{inr}\, z \mapsto \mathsf{inr}\, \star)\, \mathsf{do}\, q'); r']\!]\end{aligned} \tag{17}$$

where $q' = \mathsf{do}\ y \leftarrow c?; q$ and $r' = \mathsf{do}\ z \leftarrow \bar{c}?; r$.

Proof. Let $f \mapsto \mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto (\mathsf{do}\, x \leftarrow q; f); \mathsf{inr}\, z \mapsto r$ be the functional defining **itercase**. We can semantically replace it by

$$\begin{aligned}&f \mapsto \mathsf{do}\ x \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto (\mathsf{do}\, x \leftarrow q; f); \mathsf{inr}\, z \mapsto \mathsf{ret}\, x); \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{ret}\, x; \mathsf{inr}\, z \mapsto r.\end{aligned} \tag{18}$$

The recursion stops only if c is a right injection, so in the second **case**, which we can represent by a join $(\mathsf{do}\, y \leftarrow c?; \mathsf{ret}\, x) \sqcup (\mathsf{do}\, z \leftarrow \bar{c}?; r)$, the first part evaluates to bottom. Hence, we have

$$\begin{aligned}&[\![\Gamma \vdash \mathsf{do}\ x \leftarrow (\mathsf{init}\, x \leftarrow p\ \mathsf{while}\, (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{inl}\, \star; \mathsf{inr}\, z \mapsto \mathsf{inr}\, \star)\, \mathsf{do}\, q'); z \leftarrow \bar{c}?; r]\!] \\ =&[\![\Gamma \vdash \mathsf{do}\ x \leftarrow (\mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\, (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \mathsf{inl}\, \star; \mathsf{inr}\, z \mapsto \mathsf{inr}\, \star)\ \mathsf{of} \\ &\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{inl}\, \star \mapsto q'; \mathsf{inr}\, \star \mapsto \mathsf{ret}\, x); z \leftarrow \bar{c}?; r]\!] \\ =&[\![\Gamma \vdash \mathsf{do}\ x \leftarrow (\mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto \mathsf{ret}\, x); z \leftarrow \bar{c}?; r]\!] \\ =&[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r]\!].\end{aligned}$$

□

3.6 Final Metalanguage

3.6.1 Predicate types

As an addition to the standard first-order types, Goncharov and Schröder allow a certain kind of function types of the form $A \to \Omega$, called *predicate types*, where A is a value type. The reason why no arbitrary function types are allowed is that we only assume the base category $\mathcal{C}$ to be distributive, that is it need not have exponential objects. Contexts are now tuples containing variables for value types or predicate types.

The inclusion of predicate types thus forces Goncharov and Schröder to change once again the interpretation of metalanguage terms. The relevant definitions are found in [9]. We reproduce them here for completeness.

- Contexts $\Gamma = (x_1 : A_1, \dots, x_n : A_n, X_1 : B_1 \to \Omega, \dots, X_k : B_k \to \Omega)$ are interpreted as pairs (C, H), where
$$C = [\![A_1]\!] \times \cdots \times [\![A_n]\!]$$
$$H = \mathsf{Hom}(C \times [\![B_1]\!], [\![\Omega]\!]) \times \cdots \times \mathsf{Hom}(C \times [\![B_k]\!], [\![\Omega]\!]).$$
- First-order terms $\Gamma \vdash t : A$ are interpreted as maps $[\![t]\!] : H \to \mathsf{Hom}(C, [\![A]\!])$.
- Terms of predicate type $\Gamma \vdash \phi : A \to \Omega$ are interpreted as maps $[\![\phi]\!] : H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$.

For the sake of this work, we take a different approach and once again restrict the base category. To interpret arbitrary function types, one would have to demand that the category $\mathcal{C}$ be cartesian closed. As we only involve types $A \to P1$, where P is the functor part of a monad, we only need exponential objects in the Kleisli category:

Definition 20 (Kleisli exponentials, cf. [22, 18])**.** Given a monad $\mathbb{P}$ on a category $\mathcal{C}$, associated with the adjunct pair $F_P \dashv U_P$, where $F_P A = A$ and, for $g : A \to B$, $F_P g = Pg \circ \eta_A$ (see e.g. [3]), $\mathcal{C}$ is said to have *Kleisli exponentials* if, for all objects X of $\mathcal{C}$, the functor $F_P \circ (- \times X) : \mathcal{C} \to \mathcal{C}_P$ has a right adjoint $(-)_P^X : \mathcal{C}_P \to \mathcal{C}$.

In other words, there is a natural isomorphism $\mathsf{Hom}_{\mathcal{C}}(X \times Y, PZ) \cong \mathsf{Hom}_{\mathcal{C}}(X, Z_P^Y)$, so we can interpret a predicate type $A \to P1$ as the image of $[\![1]\!]$ under the functor $(-)_P^{[\![A]\!]}$ and define evaluation and currying maps:

$$\epsilon : [\![1]\!]_P^{[\![A]\!]} \times [\![A]\!] \to [\![\Omega]\!]$$
$$\lambda f : [\![X]\!] \to [\![1]\!]_P^{[\![A]\!]} \text{ for } f : [\![X]\!] \times [\![A]\!] \to [\![\Omega]\!].$$

The categories used in the examples (most notably **Set**) in this thesis are cartesian closed and thus automatically have Kleisli exponentials, because cartesian closed categories have arbitrary exponentials. The functor U_P from the definition above sends an object A to PA and maps $f : A \to PB$ to $\mu_B \circ Pf$. The right adjoint for $(-)_P^X$ in a cartesian closed category can then be defined as $(-)^X \circ U_P$ [22].

$$\frac{}{\Gamma \vdash \top : \Omega} \qquad \frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \wedge \psi : \Omega} \qquad \frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \vee \psi : \Omega}$$

$$\frac{}{\Gamma \vdash \bot : \Omega} \qquad \frac{\Gamma, X : A \to \Omega \vdash \phi : A \to \Omega}{\Gamma \vdash \mu X.\, \phi : A \to \Omega} \qquad \frac{\Gamma, X : A \to \Omega \vdash \phi : A \to \Omega}{\Gamma \vdash \nu X.\, \phi : A \to \Omega}$$

$$\frac{X : A \to \Omega \text{ in } \Gamma}{\Gamma \vdash X : A \to \Omega} \qquad \frac{\Gamma, x : A \vdash t : \Omega}{\Gamma \vdash \lambda x.\, t : A \to \Omega} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash s : A \to \Omega}{\Gamma \vdash s(t) : \Omega}$$

Figure 6: Assertion language.

$$\frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash \psi : \Omega}{\Gamma \vdash \phi \Rightarrow \psi : \Omega} \qquad \frac{\Gamma, x : A \vdash \phi : \Omega}{\Gamma \vdash \exists x.\, \phi : \Omega} \qquad \frac{\Gamma, x : A \vdash \phi : \Omega}{\Gamma \vdash \forall x.\, \phi : \Omega}$$

Figure 7: Additional rules in **Set**.

3.6.2 Type system

With the concepts of Chapter 3, we can now give the type system to the full extent, interpreted over a distributive category $\mathcal{C}$ with Kleisli exponentials and a predicated exception monad $(\mathbb{T}, \mathbb{P})$ on $\mathcal{C}$. The grammar for types is then

$$A ::= W \mid 1 \mid A \times A \mid A + A \qquad C ::= A \mid \Omega \mid TA \mid PA \mid A \to \Omega$$

with W ranging over a set of basic types $\mathcal{W}$ (objects of $\mathcal{C}$) and interpretations

$$[\![W]\!] = W \in \mathcal{W} \qquad [\![1]\!] = 1 \qquad [\![A \times A]\!] = [\![A]\!] \times [\![A]\!] \qquad [\![A + A]\!] = [\![A]\!] + [\![A]\!]$$
$$[\![\Omega]\!] = P1 \qquad [\![TA]\!] = T[\![A]\!] \qquad [\![PA]\!] = P[\![A]\!] \qquad [\![A \to \Omega]\!] = [\![1]\!]_P^{[\![A]\!]}.$$

3.6.3 Assertion language

In addition to the metalanguage of programs, we can now use the fact that $P1 = \Omega$ carries a logic to develop rules for an assertion language. These rules are identical to those in [9], because, due to Lemma 2, we are dealing with the same innocent submonads. The rules are given in Figure 6 and Figure 7. The latter are additional rules if the base category is **Set**.

Interpretation of assertion terms is standard. Logical conjunction and disjunction are interpreted as meets and joins, respectively. For lambda abstraction and evaluation

of predicates, we have

$$\frac{[\![\Gamma, x : A \vdash t : \Omega]\!] = f}{[\![\Gamma \vdash \lambda x\,.\,t : A \to \Omega]\!] = \lambda f.} \qquad \frac{[\![\Gamma \vdash t : A]\!] = f \quad [\![\Gamma \vdash s : A \to \Omega]\!] = g}{[\![\Gamma \vdash s(t) : \Omega]\!] = \epsilon \circ \langle g, f \rangle}$$

Fixpoints are interpreted using the fact that the hom-sets $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$ are complete lattices. We restrict the rule such that fixed points may only be taken of ϕ which monotonously depend on their input. The interpretation of such a ϕ is a morphism $g : [\![\Gamma]\!] \times [\![1]\!]_P^{[\![A]\!]} \to [\![1]\!]_P^{[\![A]\!]}$. Using this morphism, we define an endomorphism F over $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$, of which we can later take the fixed point. An input for F is a morphism $h : [\![\Gamma]\!] \times [\![A]\!] \to [\![\Omega]\!]$, or in curried form $\lambda h : [\![\Gamma]\!] \to [\![1]\!]_P^{[\![A]\!]}$. We define $F(h)$ to be the morphism indicated in the following diagram:

$$[\![\Gamma]\!] \times [\![A]\!] \xrightarrow{\langle \mathrm{id}, \lambda h \circ \pi_1 \rangle} [\![\Gamma]\!] \times [\![A]\!] \times [\![1]\!]_P^{[\![A]\!]} \xrightarrow{\langle g \circ \langle \pi_1, \pi_3 \rangle, \pi_2 \rangle} [\![1]\!]_P^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\epsilon} [\![\Omega]\!]$$

We take $[\![\Gamma \vdash \eta X\,.\,\phi]\!]$ to be $\lambda \eta F$ for $\eta \in \{\mu, \nu\}$, i.e. we take the respective fixed point of F and curry. By the restrictions on ϕ, F is monotone [9] and thus the existence of the greatest and least fixed points of F are ensured by the Knaster-Tarski theorem for complete lattices [26, Theorems 5.15 and 5.16].

Implication and quantification are interpreted using the additional structure on $P1$ if the base category is **Set**. Then, $\mathsf{Hom}(A, P1)$ is a complete Heyting algebra and therefore supports implication and infinite meets and joins.

Definition 21 (Valid assertion [9])**.** A judgment $\Gamma \vdash \phi \sqsubseteq \psi$ is *valid* with respect to a predicated monad $(\mathbb{T}, \mathbb{P})$ if $[\![\phi]\!] \sqsubseteq [\![\psi]\!]$.

3.6.4 Computational metalanguage

The full term formation language for our metalanguage is displayed in Figure 8. We have all the rules from Chapter 2, that is rules for products, coproducts, sequential composition in $\mathbb{T}$, application of functions in the signature Σ, and most notably exception raising and handling. Additionally, we incorporate rules for innocent computations and a rule to *cast* an innocent computation to one in $\mathbb{T}$ (interpreted using the inclusion morphism of the submonad—note that then it is sufficient to give a unit rule (**ret**) only for $\mathbb{P}$) and finally the looping construction introduced in the previous section.

$$\frac{x : A \text{ in } \Gamma}{\Gamma \vdash x : A} \qquad \frac{f : A \to C \in \Sigma \quad \Gamma \vdash t : A}{\Gamma \vdash f(t) : C} \qquad \frac{}{\Gamma \vdash \star : 1}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_1 t : A} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \pi_2 t : B} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{inl}\, t : A + B} \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \mathsf{inr}\, t : A + B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \times B} \qquad \frac{\Gamma \vdash c : A + B \quad \Gamma, a : A \vdash t : C \quad \Gamma, b : B \vdash u : C}{\Gamma \vdash \mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto t; \mathsf{inr}\, b \mapsto u : C}$$

$$\frac{\Gamma \vdash p : PA \quad \Gamma, x : A \vdash q : PB}{\Gamma \vdash \mathsf{do}\, x \leftarrow p; q : PB} \qquad \frac{\Gamma \vdash p : TA \quad \Gamma, x : A \vdash q : TB}{\Gamma \vdash \mathsf{do}\, x \leftarrow p; q : TB} \qquad \frac{\Gamma \vdash p : PA}{\Gamma \vdash p : TA}$$

$$\frac{\Gamma \vdash p : A}{\Gamma \vdash \mathsf{ret}\, p : PA}\ (A \text{ value type}) \qquad \frac{\Gamma \vdash e : E}{\Gamma \vdash \mathsf{raise}\, e : TA} \qquad \frac{\Gamma \vdash p : TA}{\Gamma \vdash \mathsf{catch}\, p : T(A + E)}$$

$$\frac{\Gamma, x : A \vdash c : B + C \quad \Gamma \vdash p : TA \quad \Gamma, y : B \vdash q : TA \quad \Gamma, z : C \vdash r : TA}{\Gamma \vdash \mathrm{init}\, x \leftarrow p\ \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q; \mathsf{inr}\, z \mapsto r : TA}$$

Figure 8: Term formation rules for the simple imperative metalanguage with exceptions.

4 Hoare Calculus for Order-Enriched Effects with Exceptions

4.1 The Calculus

We now have all we need to define Hoare logic for programs in a predicated monad. We first state a version without exceptions as given by Goncharov and Schröder.

Definition 22 (Hoare triple [9])**.** Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad. *Hoare triples* are formed by the rule

$$\frac{\Gamma \vdash \phi : \Omega \quad \Gamma \vdash p : A \quad \Gamma, x : A \vdash \psi : \Omega}{\Gamma \vdash \{\phi\}\ x \leftarrow p\ \{\psi\}}$$

and interpreted as

$$\Gamma \vdash \{\phi\}\ x \leftarrow p\ \{\psi\} \Leftrightarrow \mathsf{do}\ \phi; x \leftarrow p; \psi; \mathsf{ret}\, x = \mathsf{do}\ \phi; p.$$

If this equation holds given an interpretation of basic programs, we write $\mathbb{T}, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi\}$.

Example 8 (Hoare Triples in the State monad)**.** Traditionally, Hoare triples were defined for stateful computations in terms of imperative programming (see also Section 2.1). Thus, one can formulate the condition for a Hoare triple $\{\phi\}\ x \leftarrow p\ \{\psi\}$ to hold classically as follows:

$$\forall s\,.\,(\phi(s) \wedge p(s)\downarrow) \Rightarrow \psi(\pi_2 p(s)) \tag{19}$$

In other words, the Hoare triple holds if for every state in which p terminates and ϕ holds, ψ holds in the state resulting from executing p.

Now let $(\mathbb{T}, \mathbb{P})$ be the predicated monad where $\mathbb{T}$ is the partial state monad and $\mathbb{P}$ the partial reader monad, as given in Example 7, and consider the equality defining a Hoare triple as given above:

$$\mathsf{do}\ \phi; x \leftarrow p; \psi; \mathsf{ret}\, x = \mathsf{do}\ \phi; p$$

Here, ϕ and ψ are innocent programs of type Ω, so by the definition of inclusion morphism from $\mathbb{P}$ to $\mathbb{T}$, they do not change the state. Therefore, the contents of the state on either side of the equation can only be changed by p. This modified state is passed on to ψ. Since $\Omega = P1 = S \rightharpoonup 1 \cong S \to 2$, the results of ϕ and ψ can only be $\top$ or $\bot$. Thus, Equation 19 is easily seen to hold in this scenario: if ϕ returns $\bot$,

the result follows because sequential composition preserves the empty join in the left argument. The same argument applies if ϕ returns $\top$ and p fails. Finally, if both ϕ and p do not fail, the equation can only hold if ψ returns $\top$. Otherwise, we would have $\mathsf{do}\ \phi; x \leftarrow p; \bot = \mathsf{do}\ \phi; p$, which obviously cannot hold.

To cope with exceptional behaviour, it seems natural to split the postcondition of a *Hoare triple* in two parts: a *normal* postcondition, which has to be satisfied if the program terminates normally, returning a value, and an *abnormal* postcondition, dealing with exceptional termination. As seen in the definition of *Hoare triple* above, the postcondition ψ may use the result of the program p. This is not possible directly if p throws an exception, because then, x is not assigned any value and the exception can not be observed. Equation 10 of the equational characterization of catch, however, suggests the following definition:

Definition 23 (Hoare quadruple)**.** Let $\phi, \psi : \Omega$ be assertions and let $\varepsilon : E \to \Omega$ be a predicate on exceptions. A *Hoare quadruple* with *abnormal postcondition* ε is defined by the equivalence

$$\begin{array}{c} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \\ \Updownarrow \\ \{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi;\ \mathsf{inr}\, e \mapsto \varepsilon(e)\}. \end{array}$$

Definition 24. Given a predicated monad $(\mathbb{T}, \mathbb{P})$, inequality assertions $\phi \sqsubseteq \psi$ are *valid* in a context Γ iff $[\![\Gamma \vdash \phi]\!] \sqsubseteq [\![\Gamma \vdash \psi]\!]$. Let furthermore Δ be a set of axioms for basic programs. If a Hoare quadruple $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is derivable from the valid inequality assertions and the axioms in Δ, we write

$$\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

4.2 Rule Design

The set of rules for our calculus can be found in Figure 9. Most of the rules can directly be derived from the rules in [9]. This applies to the rules **(basic)**, **(ret)**, **(do)** and **(wk)**. Here, only the abnormal postconditions were added. Notice that the inequalities in **(wk)**, as in Goncharov and Schröder [9], are expected to be discharged outside the calculus. Furthermore, the inequality assertion $\varepsilon \sqsubseteq \varepsilon'$ is interpreted pointwise, that is $\varepsilon(e) \sqsubseteq \varepsilon'(e)$ for all exceptions e.

4.2.1 Rules for exceptions

The design of the rule **(catch)** for exception handling is taken from [20]. It is based on the fact that, having a Hoare quadruple $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$, an exception possibly thrown by p is not visible in the regular postcondition, but adding another *catch* makes it accessible by packing it up in a coproduct. As a result, the abnormal postcondition

cannot be reached, because *catch* itself does not throw exceptions, so we can assume $\bot$ here (which is the function from E to Ω which sends every exception to $\bot$).

Exceptions can be raised in every stage of a computation and thus can be introduced without any premises. Therefore, it would be valid to take $\bot$ as a precondition, but the rule is stronger if we take as a precondition the value of the abnormal postcondition ε at the exception that is actually raised. As *raise* always terminates with an exception, the normal postcondition can be taken to be $\bot$.

4.2.2 Test rules

Our calculus includes a rule $(\mathbf{test}_l)$, which assigns the value of a coproduct c to a variable x if it is a left injection and evaluates to $\bot$ in case c is a right injection. Of course, we can derive a similar rule

$$(\mathbf{test}_r)\ \frac{}{\{c? \vee (\mathsf{do}\ x \leftarrow \bar{c}?; \phi)\}\ x \leftarrow \bar{c}?\ \{\phi \mid \varepsilon\},}$$

which we do not include in the calculus because it is derivable from $(\mathbf{test}_l)$, but we use it in the following for simplicity.

The precondition of the rule is designed in a way that it reflects the weakest precondition of a program $x \leftarrow c?$. However, in most cases, the rule is used just to assign a value to a variable in a more complex program. Because $\mathsf{do}\ a \leftarrow c?; \phi \sqsubseteq c? \vee (\mathsf{do}\ a \leftarrow \bar{c}?; \phi)$, by weakening and sequential composition, we can derive a rule

$$\frac{\{\phi\}\ x \leftarrow p\ \{\psi\}}{\{\mathsf{do}\ a \leftarrow c?; \phi\}\ x \leftarrow (\mathsf{do}\ a \leftarrow c?; p)\ \{\psi \mid \varepsilon\}.}$$

In the following, if we say we *apply* one of the test rules *to* a program, we actually apply the preceding rule.

4.2.3 Case rule

In the premises of the case rule, we have two programs with possibly different preconditions, but the same postcondition. It is easy to see that we could as well choose two distinct postconditions ψ_1 and ψ_2, but because $\psi_1 \sqsubseteq \psi_1 \vee \psi_2$, we see by weakening that the rule would not be more general.

In the conclusion of the rule, we assign values to the variables a and b. This means that the quadruples in the precondition depend on a and b, respectively. This explains the precondition of the conclusion. The preconditions may also depend on a and b, so we need to make sure this dependency is resolved as well.

Technically, the rule works as follows: we apply $(\mathbf{test}_l)$ and $(\mathbf{test}_r)$, respectively, to the premises to get

$$\{\mathsf{do}\ a \leftarrow c?; \phi\}\ x \leftarrow (\mathsf{do}\ a \leftarrow c?; q)\ \{\psi \mid \varepsilon\}$$
$$\{\mathsf{do}\ b \leftarrow \bar{c}?; \xi\}\ x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r)\ \{\psi \mid \varepsilon\}.$$

By Lemma 3, the case construction in the conclusion of the rule is exactly the join of the two programs in these quadruples. Therefore, we take the precondition of the conclusion to be the join of the preconditions thereof as well.

4.2.4 Iterated case rule

The full itercase construction, as defined in Equation 16, is actually a sequential composition of two programs. In the Hoare calculus, for simplicity, we only use the special case where $p = \mathsf{ret}\, x$ and abbreviate as follows:

$$\mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r := \mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r.$$

In contrast to the rule for **while** in [9], we cannot assert that after the loop, the value of the variable c in the test contains a right injection. This is because the program r, which is executed after the loop terminates, might change c again in such a way that it contains a left injection again, yet the loop, of course, does not continue to run.

The idea behind the complex postcondition for the program q in the premise is the following: we want to iterate over the program q, which depends on the variable a. To be able to iterate, we need to have an invariant for the loop. This invariant is realised in the assertion ψ. In every iteration of the loop, q is executed with a value for a corresponding to the contents of c. Now, if q changes the contents of c, we need to ensure that either the invariant still holds for the next iteration, that is $\mathsf{do}\, a \leftarrow c?; \psi$ holds, or the precondition of r holds, with b replaced by the value contained in c. The conclusion of the rule is then designed similar to **(case)**.

4.3 Soundness

Theorem 3. *The Hoare calculus for order-enriched effects with exceptions is sound, that is for premises* Ξ*,*

$$\Xi \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi\}\ \textit{implies}\ \mathbb{T}, \mathbb{P}, \Xi \vDash \{\phi\}\ x \leftarrow p\ \{\psi\}.$$

Proof. To prove soundness of the calculus, we need to ensure that in every given rule, the identities encoded in the premises entail the identities of the conclusion. We first introduce notation to make the proofs more readable:

$$\phi\,|_x^y\,\varepsilon := \mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \phi;\ \mathsf{inr}\, e \mapsto \varepsilon(e).$$

For the soundness proofs involving joins, we need another intermediate lemma:

Lemma 4. *The exception handling morphism* catch *preserves* case *constructions, i.e.*

$$\mathsf{catch}(\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r) = \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, b \mapsto \mathsf{catch}\, r.$$

$$\textbf{(basic)}\ \frac{\{\phi\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}}{\{\phi[t/x]\}\ x \leftarrow f(t)\ \{\psi \mid \varepsilon\}}\ (f : A \to TB \in \Sigma)$$

$$\textbf{(test}_l\textbf{)}\ \frac{}{\{(\mathsf{do}\ x \leftarrow c?; \phi) \vee \bar{c}?\}\ x \leftarrow c?\ \{\phi \mid \varepsilon\}}$$

$$\textbf{(ret)}\ \frac{}{\{\phi[t/x]\}\ x \leftarrow \mathsf{ret}(t)\ \{\phi \mid \varepsilon\}} \qquad \textbf{(do)}\ \frac{\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \quad \{\psi\}\ y \leftarrow q\ \{\chi \mid \varepsilon\}}{\{\phi\}\ y \leftarrow \mathsf{do}\ x \leftarrow p; q\ \{\chi \mid \varepsilon\}}$$

$$\textbf{(case)}\ \frac{\{\phi\}\ x \leftarrow q\ \{\psi \mid \varepsilon\} \quad \{\xi\}\ x \leftarrow r\ \{\psi \mid \varepsilon\}}{\{(\mathsf{do}\ a \leftarrow c?; \phi) \vee (\mathsf{do}\ b \leftarrow \bar{c}?; \xi)\}\ x \leftarrow \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r\ \{\psi \mid \varepsilon\}}$$

$$\textbf{(wk)}\ \frac{\phi' \sqsubseteq \phi \quad \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \quad \psi \sqsubseteq \psi' \quad \varepsilon \sqsubseteq \varepsilon'}{\{\phi'\}\ x \leftarrow p\ \{\psi' \mid \varepsilon'\}} \qquad \textbf{(raise)}\ \frac{}{\{\varepsilon(e)\}\ x \leftarrow \mathsf{raise}\, e\ \{\bot \mid \varepsilon\}}$$

$$\textbf{(catch)}\ \frac{\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}}{\{\phi\}\ y \leftarrow (\mathsf{catch}\, p)\ \{\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi; \mathsf{inr}\, e \mapsto \varepsilon(e) \mid \bot\}}$$

$$\textbf{(itercase)}\ \frac{\{\psi\}\ x \leftarrow q\ \{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi \mid \varepsilon\} \qquad \{\xi\}\ x \leftarrow r\ \{\chi \mid \varepsilon\}}{\{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi\}\ x \leftarrow (\mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r)\ \{\chi \mid \varepsilon\}}$$

Figure 9: Hoare calculus for order-enriched effects with exceptions.

Proof. We have by Equation 5 that $\mathsf{catch} = T\,\mathsf{inl} = (\eta \circ \mathsf{inl})^*$, so that by order-enrichment, the map $f \mapsto \mathsf{catch} \circ f$ preserves all joins. Therefore,

$$\begin{aligned}
&\mathsf{catch}(\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r)\\
=\ &\mathsf{catch}\,((\mathsf{do}\ a \leftarrow c?; q) \sqcup (\mathsf{do}\ b \leftarrow \bar{c}?; r))\\
=\ &\mathsf{catch}(\mathsf{do}\ a \leftarrow c?; q) \sqcup \mathsf{catch}(\mathsf{do}\ b \leftarrow \bar{c}?; r).
\end{aligned}$$

The left branch of this join can be transformed as follows:

$$\begin{aligned}
&\mathsf{catch}(\mathsf{do}\ a \leftarrow c?; q)\\
=\ &\mathsf{do}\ y \leftarrow \mathsf{catch}\, c?; (\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q; \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e)\\
=\ &\mathsf{do}\ y \leftarrow (\mathsf{do}\ a \leftarrow c?; \mathsf{ret}\,\mathsf{inl}\, a); (\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q; \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e)\\
=\ &\mathsf{do}\ a \leftarrow c?; y \leftarrow \mathsf{ret}\,\mathsf{inl}\, a; (\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q; \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e)\\
=\ &\mathsf{do}\ a \leftarrow c?; \mathsf{catch}\, q.
\end{aligned}$$

We used, in this order, Equation 7, Equation 14, associativity and Equation 1. The right branch can be transformed equivalently, so the claim holds. □

*Soundness of **(ret)***

Expanding the definitions of $\mathsf{catch}(\mathsf{ret}\, t)$ and applying copyability of assertions, we get

$$\begin{aligned}
&\mathsf{do}\ \phi[t/x]; y \leftarrow \mathsf{catch}(\mathsf{ret}\, t); \phi\ |_x^y\ \varepsilon; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi[t/x]; y \leftarrow \mathsf{ret}(\mathsf{inl}\, t); \phi\ |_x^y\ \varepsilon; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi[t/x]; (\mathsf{case}\, (\mathsf{inl}\, t)\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \phi;\ \mathsf{inr}\, e \mapsto \varepsilon); \mathsf{ret}(\mathsf{inl}\, t) \\
=\ &\mathsf{do}\ \phi[t/x]; \phi[t/x]; \mathsf{ret}(\mathsf{inl}\, t) \\
=\ &\mathsf{do}\ \phi[t/x]; \mathsf{catch}(\mathsf{ret}\, t).
\end{aligned}$$

*Soundness of **(do)***

This involves a straightforward, but rather lengthy transformation:

$$\begin{aligned}
&\mathsf{do}\ \phi; z \leftarrow \mathsf{catch}(\mathsf{do}\ x \leftarrow p; q); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ \phi; z \leftarrow (\mathsf{do}\ w \leftarrow \mathsf{catch}\, p; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e)); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; z \leftarrow (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ w \leftarrow (\mathsf{do}\ \phi; \mathsf{catch}\, p); z \leftarrow (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ w \leftarrow (\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; \psi\ |_x^w\ \varepsilon; \mathsf{ret}\, w); \\
&\qquad z \leftarrow (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; \psi\ |_x^w\ \varepsilon; \\
&\qquad z \leftarrow (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi;\ \mathsf{inr}\, e \mapsto \varepsilon(e)); \\
&\qquad z \leftarrow (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e); \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto (\mathsf{do}\ \psi; z \leftarrow \mathsf{catch}\, q; \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z); \\
&\qquad\qquad\qquad\qquad \mathsf{inr}\, e \mapsto (\mathsf{do}\ \varepsilon(e); z \leftarrow \mathsf{ret}\,\mathsf{inr}\, e; \chi\ |_y^z\ \varepsilon; \mathsf{ret}\, z)) \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{do}\ \psi; \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{do}\ \varepsilon(e); \varepsilon(e); \mathsf{ret}\,\mathsf{inr}\, e) \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; \psi\ |_x^w\ \varepsilon; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e) \\
=\ &\mathsf{do}\ w \leftarrow (\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; \psi\ |_x^w\ \varepsilon; \mathsf{ret}\, w); (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e) \\
=\ &\mathsf{do}\ w \leftarrow (\mathsf{do}\ \phi; \mathsf{catch}\, p); (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e) \\
=\ &\mathsf{do}\ \phi; w \leftarrow \mathsf{catch}\, p; (\mathsf{case}\, w\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, e \mapsto \mathsf{ret}\,\mathsf{inr}\, e) \\
=\ &\mathsf{do}\ \phi; \mathsf{catch}(\mathsf{do}\ x \leftarrow p; q)
\end{aligned}$$

*Soundness of **(wk)***

The inequalities in the premises can be written in terms of meets by Lemma 1, for example $\phi' \sqsubseteq \phi \Leftrightarrow \phi' \sqcap \phi = (\mathsf{do}\ \phi'; \phi) = \phi'$. Moreover,

$$\mathsf{do}\ \psi\ |_x^y\ \varepsilon; \psi'\ |_x^y\ \varepsilon' = \mathsf{case}\, y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto (\mathsf{do}\ \psi; \psi');\ \mathsf{inr}\, e \mapsto (\mathsf{do}\ \varepsilon(e); \varepsilon'(e)) = \psi\ |_x^y\ \varepsilon.$$

Then we have

$$\begin{aligned}
&\mathsf{do}\ \phi'; y \leftarrow \mathsf{catch}\, p; \psi' \mid_x^y \varepsilon'; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; \phi; y \leftarrow \mathsf{catch}\, p; \psi' \mid_x^y \varepsilon'; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; y \leftarrow (\mathsf{do}\ \phi; \mathsf{catch}\, p); \psi' \mid_x^y \varepsilon'; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; y \leftarrow (\mathsf{do}\ \phi; y \leftarrow \mathsf{catch}\, p; \psi \mid_x^y \varepsilon; \mathsf{ret}\, y); \psi' \mid_x^y \varepsilon'; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; \phi; y \leftarrow \mathsf{catch}\, p; \psi \mid_x^y \varepsilon; \psi' \mid_x^y \varepsilon'; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; \phi; y \leftarrow \mathsf{catch}\, p; \psi \mid_x^y \varepsilon; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \phi'; \phi; \mathsf{catch}\, p = \mathsf{do}\ \phi'; \mathsf{catch}\, p.
\end{aligned}$$

Soundness of ***(basic)***

From **(ret)**, we have $\{\phi[t/z]\}\ z \leftarrow \mathsf{ret}\, t\ \{\phi \mid \varepsilon\}$. Together with the premise $\{\phi\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}$ we can apply **(do)** and get $\{\phi[t/z]\}\ x \leftarrow (\mathsf{do}\ z \leftarrow \mathsf{ret}\, t; f(z))\ \{\psi \mid \varepsilon\}$. Soundness then follows from

$$\mathsf{do}\ z \leftarrow \mathsf{ret}\, t; f(z) = f(z)[t/z] = f(t).$$

Soundness of ***(raise)***

Straightforward by Equation 8 and copyability.

$$\begin{aligned}
&\mathsf{do}\ \varepsilon(e); y \leftarrow \mathsf{catch}(\mathsf{raise}\, e); \bot \mid_x^y \varepsilon; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \varepsilon(e); y \leftarrow \mathsf{ret}\, \mathsf{inr}\, e; \bot \mid_x^y \varepsilon; \mathsf{ret}\, y \\
=\ &\mathsf{do}\ \varepsilon(e); \varepsilon(e); \mathsf{ret}\, \mathsf{inr}\, e \\
=\ &\mathsf{do}\ \varepsilon(e); \mathsf{catch}(\mathsf{raise}\, e).
\end{aligned}$$

Soundness of ***(catch)***

Using Equation 9, we have

$$\begin{aligned}
&\mathsf{do}\ \phi; r \leftarrow \mathsf{catch}(\mathsf{catch}\, p); (\psi \mid_x^y \varepsilon) \mid_y^r \bot; \mathsf{ret}\, r \\
=\ &\mathsf{do}\ \phi; r \leftarrow (\mathsf{do}\ s \leftarrow \mathsf{catch}\, p; \mathsf{ret}\, \mathsf{inl}\, s); (\psi \mid_x^y \varepsilon) \mid_y^r \bot; \mathsf{ret}\, r \\
=\ &\mathsf{do}\ \phi; s \leftarrow \mathsf{catch}\, p; r \leftarrow \mathsf{ret}\, \mathsf{inl}\, s; (\psi \mid_x^y \varepsilon) \mid_y^r \bot; \mathsf{ret}\, r \\
=\ &\mathsf{do}\ \phi; s \leftarrow \mathsf{catch}\, p; \psi \mid_x^s \varepsilon; \mathsf{ret}\, \mathsf{inl}\, s \\
=\ &\mathsf{do}\ s \leftarrow (\mathsf{do}\ \phi; s \leftarrow \mathsf{catch}\, p; \psi \mid_x^s \varepsilon; \mathsf{ret}\, s); \mathsf{ret}\, \mathsf{inl}\, s \\
=\ &\mathsf{do}\ s \leftarrow (\mathsf{do}\ \phi; \mathsf{catch}\, p); \mathsf{ret}\, \mathsf{inl}\, s \\
=\ &\mathsf{do}\ \phi; s \leftarrow \mathsf{catch}\, p; \mathsf{ret}\, \mathsf{inl}\, s \\
=\ &\mathsf{do}\ \phi; \mathsf{catch}(\mathsf{catch}\, p).
\end{aligned}$$

Soundness of ***(test$_l$)***

Recall that $c? : PA$ is innocent, so we can use copyability, weak discardability and commutativity to see that

$$\begin{aligned}
&\mathsf{do}\,(\mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?); y \leftarrow \mathsf{catch}\, c?; \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y \\
=&\,\mathsf{do}\,(\mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?); x \leftarrow c?; \psi; \mathsf{ret}\,\mathsf{inl}\, x \\
=&(\mathsf{do}\ x \leftarrow c?\psi; x \leftarrow c?; \psi; \mathsf{ret}\,\mathsf{inl}\, x) \sqcup (\mathsf{do}\ \bar{c}?; x \leftarrow c?; \psi; \mathsf{ret}\,\mathsf{inl}\, x) \\
=&(\mathsf{do}\ x \leftarrow c?\psi; x \leftarrow c?; \psi; \mathsf{ret}\,\mathsf{inl}\, x) \sqcup \bot \\
=&(\mathsf{do}\ x \leftarrow c?; \psi; x \leftarrow c?; \mathsf{ret}\,\mathsf{inl}\, x) \sqcup \bot \\
=&\,\mathsf{do}\,(\mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?); x \leftarrow c?; \mathsf{ret}\,\mathsf{inl}\, x \\
=&\,\mathsf{do}\,(\mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?); \mathsf{catch}\, c?.
\end{aligned}$$

Soundness of $(\mathbf{test}_r)$ follows analogously.

Soundness of ***(case)***

We use Lemma 4 and the fact that sequential composition of $a \leftarrow c?$ and $b \leftarrow \bar{c}?$ yields $\bot$. We may thus discharge some of the programs in the resulting joins and have

$$\begin{aligned}
&\mathsf{do}\,((\mathsf{do}\ a \leftarrow c?; \phi) \vee (\mathsf{do}\ b \leftarrow \bar{c}?; \xi));\\
&\qquad y \leftarrow \mathsf{catch}(\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r); \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y \\
=&(\mathsf{do}\ a \leftarrow c?; \phi; y \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, b \mapsto \mathsf{catch}\, r); \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y) \\
\sqcup&(\mathsf{do}\ b \leftarrow \bar{c}?; \xi; y \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathsf{catch}\, q;\ \mathsf{inr}\, b \mapsto \mathsf{catch}\, r); \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y) \\
=&(\mathsf{do}\ a \leftarrow c?; \phi; y \leftarrow \mathsf{catch}\, q; \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y) \\
\sqcup&(\mathsf{do}\ b \leftarrow \bar{c}?; \xi; y \leftarrow \mathsf{catch}\, r; \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y) \\
=&(\mathsf{do}\ a \leftarrow c?; \phi; \mathsf{catch}\, q) \sqcup (\mathsf{do}\ b \leftarrow \bar{c}?; \xi; \mathsf{catch}\, r) \\
=&\,\mathsf{do}\,((\mathsf{do}\ a \leftarrow c?; \phi) \vee (\mathsf{do}\ b \leftarrow \bar{c}?; \xi)); \mathsf{catch}(\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r).
\end{aligned}$$

Soundness of ***(itercase)***

The proof is by fixpoint induction. We prove that the conclusion of the rule holds for the finite approximants of the fixed point of the functional in Equation 15. That is, we show that, assuming the premises of the rule,

$$\{\psi\}\ x \leftarrow \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto (\mathsf{do}\ x \leftarrow q; p);\ \mathsf{inr}\, b \mapsto r\ \{\chi \mid \varepsilon\}$$

holds, or equivalently

$$\{\psi\}\ x \leftarrow (\mathsf{do}\ x \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto (\mathsf{do}\ x \leftarrow q; p);\ \mathsf{inr}\, b \mapsto \mathsf{ret}\, x); b \leftarrow \bar{c}?; r)\ \{\chi \mid \varepsilon\},$$

by Equation 18.

Let F be the functional

$$p \mapsto \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto (\mathsf{do}\ x \leftarrow q; p);\ \mathsf{inr}\, b \mapsto \mathsf{ret}\, x.$$

We first show that from $\{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi\}\ x \leftarrow p\ \{\mathsf{do}\ b \leftarrow \bar{c}?; \xi \mid \varepsilon\}$ it follows that $\{\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi\}\ x \leftarrow F(p)\ \{\mathsf{do}\ b \leftarrow \bar{c}?; \xi \mid \varepsilon\}$. Let $\phi = \mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; \xi$ and therefore $\bar{c}? \wedge \phi = \mathsf{do}\ b \leftarrow \bar{c}?; \xi$.

Base case. We prove that the claim holds for the bottom element. The quadruple

$$\{\phi\}\ x \leftarrow \bot\ \{\bar{c}? \wedge \phi \mid \varepsilon\}$$

translates into

$$\mathsf{do}\ \phi; y \leftarrow \mathsf{catch}\ \bot; \bar{c}? \wedge \phi\ |^{y}_{x}\ \varepsilon; \mathsf{ret}\ y = \mathsf{do}\ \phi; \mathsf{catch}\ \bot.$$

Because $\mathsf{catch} = T\,\mathsf{inl} = (\eta \circ \mathsf{inl})^*$, catch preserves bottom by order-enrichment. The quadruple is therefore valid by preservation of joins in the left argument by do.

Inductive step. We have the following Hoare quadruples:

$$\{\psi\}\ x \leftarrow q\ \{\phi \mid \varepsilon\} \tag{q}$$
$$\{\phi\}\ x \leftarrow p\ \{\bar{c}? \wedge \phi \mid \varepsilon\} \tag{p}$$

The quadruple (q) is from the premises of **(itercase)** and (p) is assumed to hold by the induction hypothesis. Applying **(do)** to (q) and (p) yields

$$\{\psi\}\ x \leftarrow (\mathsf{do}\ x \leftarrow q; p)\ \{\bar{c}? \wedge \phi \mid \varepsilon\}.$$

Using **(ret)**, we get

$$\{\bar{c}? \wedge \phi\}\ x \leftarrow \mathsf{ret}\, x\ \{\bar{c}? \wedge \phi \mid \varepsilon\}.$$

We apply **(case)** to the previous two quadruples to obtain the derivation

$$\frac{\{\psi\}\ x \leftarrow (\mathsf{do}\ x \leftarrow q; p)\ \{\bar{c}? \wedge \phi \mid \varepsilon\} \qquad \{\bar{c}? \wedge \phi\}\ x \leftarrow \mathsf{ret}\, x\ \{\bar{c}? \wedge \phi \mid \varepsilon\}}{\{\phi\}\ x \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto (\mathsf{do}\ x \leftarrow q; p);\ \mathsf{inr}\, b \mapsto \mathsf{ret}\, x)\ \{\bar{c}? \wedge \phi \mid \varepsilon\},}$$

since $\mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow c?; (\bar{c}? \wedge \phi) = \mathsf{do}\ a \leftarrow c?; \psi \vee \mathsf{do}\ b \leftarrow \bar{c}?; b \leftarrow \bar{c}?; \xi = \phi$ by copyability. Validity of the conclusion of **(itercase)** as stated above then follows by applying $(\mathbf{test}_r)$ to

$$\{\xi\}\ x \leftarrow r\ \{\chi \mid \varepsilon\}$$

from the premises of the rule and sequential composition.

Soundness of **(itercase)** follows from Scott-continuity of and therefore the existence of the least fixed point of F. □

4.4 Relative Completeness

For a Hoare calculus to be complete means that it is possible to derive every valid partial correctness assertion using the rules of the calculus. It is a well-known fact that Hoare logic is not complete. Therefore, Cook coined the notion of *relative completeness*. Applied to our calculus, this amounts to saying that $\mathbb{T}, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ implies $\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ for a predicated monad $(\mathbb{T}, \mathbb{P})$ and axioms Δ for basic programs.

The main idea for Cook's original proof of relative completeness of ordinary Hoare logic is a calculus of *weakest liberal preconditions* or shorter *weakest preconditions* [6],

that is (adapted to our setting), for every program p, postcondition ψ and abnormal postcondition ε, a precondition $\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ such that

$$\{\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \tag{20}$$

is a valid Hoare quadruple and $\phi \sqsubseteq \mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ whenever $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$. If we can prove that all quadruples of the form in Equation 20 are derivable in the calculus then relative completeness follows by soundness and application of **(wk)**.

The fact that every precondition that fulfils a Hoare quadruple with a given postcondition $\psi \mid \varepsilon$ has to be smaller than the weakest precondition for the respective command and postcondition suggests the following definition of weakest preconditions (cf. [9]):

$$\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon) = \bigsqcup\{\phi \mid \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}\}$$

Because every hom-set $\mathsf{Hom}(X, \Omega)$ is a complete lattice, this join exists, so the definition is sound.

Lemma 5. *For all programs p and postconditions ψ,*

$$\{\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

Proof. Recall that $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ means $\mathsf{do}\ \phi; y \leftarrow \mathsf{catch}\, p; \psi\ |_x^y\ \varepsilon; \mathsf{ret}\, y = \mathsf{do}\ \phi; \mathsf{catch}\, p$. Let now S be the set of all preconditions fulfilling this equation. As **do** preserves joins in the left argument, the equation must also hold for the join of S, which, by definition, is $\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$. □

We only gave a formal definition for weakest preconditions, but it is in no way self-understood that the so-defined joins are expressible in terms of our metalanguage. It is thus necessary to define the weakest preconditions inductively based on the available terms and then prove that these two definitions coincide.

The definition for a syntactic version wp_s of the weakest precondition calculus is found in Figure 10. The syntactic weakest preconditions are heavily based on those given by Goncharov and Schröder in [9].

As Goncharov and Schröder pointed out, however, the definition of the syntactic weakest precondition of $\mathsf{do}\ x \leftarrow p; q$ is problematic. We need to impose a restriction on the monad in order to be able to decompose it:

Definition 25 (Sequential Compatibility [9])**.** We call a predicated monad $(\mathbb{T}, \mathbb{P})$ *sequentially compatible* if for all programs p, q, assertions ψ and abnormal postconditions ε, we have

$$\mathrm{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}(y \leftarrow p, \mathrm{wp}(x \leftarrow q, \psi \mid \varepsilon)).$$

Also, the definition of wp_s for *catch* is not self-explanatory. Because *catch* itself never fails with an exception, the abnormal postcondition in a quadruple containing

$$\mathrm{wp}_s(x \leftarrow f(t), \psi \mid \varepsilon) = \mathrm{wp}(x \leftarrow f(z), \psi \mid \varepsilon)[t/z] \tag{21}$$
$$\mathrm{wp}_s(x \leftarrow c?, \psi \mid \varepsilon) = \mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}? \tag{22}$$
$$\mathrm{wp}_s(x \leftarrow \bar{c}?, \psi \mid \varepsilon) = c? \vee \mathsf{do}\ x \leftarrow \bar{c}?; \psi \tag{23}$$
$$\mathrm{wp}_s(x \leftarrow \mathsf{ret}\, t, \psi \mid \varepsilon) = \psi[t/x] \tag{24}$$
$$\mathrm{wp}_s(y \leftarrow (\mathsf{do}\ x \leftarrow p; q), \psi \mid \varepsilon) = \mathrm{wp}_s(x \leftarrow p, \mathrm{wp}_s(y \leftarrow q, \psi \mid \varepsilon) \mid \varepsilon) \tag{25}$$
$$\mathrm{wp}_s(x \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \tag{26}$$
$$\quad \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathrm{wp}_s(x \leftarrow q, \psi \mid \varepsilon); \mathsf{inr}\, b \mapsto \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon)$$
$$\mathrm{wp}_s(x \leftarrow (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \tag{27}$$
$$\quad (\nu X\,.\,\lambda x\,.\ \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathrm{wp}_s(x \leftarrow q, X(x) \mid \varepsilon); \mathsf{inr}\, b \mapsto \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon))\,(x)$$
$$\mathrm{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon) = \epsilon(e) \tag{28}$$
$$\mathrm{wp}_s(y \leftarrow \mathsf{catch}\, p, \psi \mid \varepsilon) = \mathrm{wp}_s(x \leftarrow p, \psi[\mathsf{inl}\, x/y] \mid \lambda e.\psi[\mathsf{inr}\, e/y]) \tag{29}$$

Figure 10: Syntactic definition of weakest preconditions.

$y \leftarrow \mathsf{catch}\, p$ is irrelevant. The whole information about the post-state has to be captured in the normal postcondition, so we have:

$$\begin{aligned}
&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi \mid \varepsilon\}\\
\Leftrightarrow&\{\phi\}\ z \leftarrow \mathsf{catch}(\mathsf{catch}\, p)\ \{\mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \psi; \mathsf{inr}\, e \mapsto \varepsilon\}\\
\Leftrightarrow&\{\phi\}\ z \leftarrow (\mathsf{do}\ y \leftarrow \mathsf{catch}\, p; \mathsf{ret}\, \mathsf{inl}\, y)\ \{\mathsf{case}\ z\ \mathsf{of}\ \mathsf{inl}\, y \mapsto \psi; \mathsf{inr}\, e \mapsto \varepsilon\}\\
\Leftrightarrow&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\}
\end{aligned}$$

We can now eliminate *catch* by decomposing ψ into a case construction and applying the definition of Hoare quadruple.

$$\begin{aligned}
&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\}\\
\Leftrightarrow&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi[\mathsf{inl}\, x/y]; \mathsf{inr}\, e \mapsto \psi[\mathsf{inr}\, e/y]\}\\
\Leftrightarrow&\{\phi\}\ x \leftarrow p\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y]\}
\end{aligned}$$

This equivalence lets us construct a weakest precondition inductively via the premise of the **(catch)** rule.

For basic programs, we also have to assume that weakest preconditions $\mathrm{wp}(x \leftarrow f(z), \psi)$ for every ψ are expressible in the assertion language, giving an axiom

$$(\mathbf{wp}_f)\ \frac{}{\{\mathrm{wp}(x \leftarrow f(z), \psi)\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}}.$$

Lemma 6. *Given a sequentially compatible predicated monad, we have for all programs p and postconditions ψ*

$$\Delta_\Sigma \vdash_{\mathbb{P}} \{\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\},$$

where Δ_Σ are axioms for the basic programs in the signature Σ.

Proof. The proof is via induction over p.

Base cases. The base cases correspond to those programs which are derivable in one step using the calculus, that is whose introduction rules do not have premises or, in the case of **(basic)**, whose premises are axioms of the calculus.

$\{\psi[t/x]\}\ x \leftarrow \mathsf{ret}\, t\ \{\psi \mid \varepsilon\} = \{\mathrm{wp}_s(x \leftarrow \mathsf{ret}\, t, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is trivially derivable in the calculus using the rule **(ret)**. The same is true for $\{\mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?\}\ x \leftarrow c?\ \{\psi \mid \varepsilon\} = \{\mathrm{wp}_s(x \leftarrow c?, \psi \mid \varepsilon)\}\ x \leftarrow c?\ \{\psi \mid \varepsilon\}$ and respectively $\bar{c}?$. Using $\mathbf{(WP}_f\mathbf{)}$ and **(basic)**, we are able to derive $\{\mathrm{wp}(x \leftarrow f(t), \psi \mid \varepsilon)\}\ x \leftarrow f(t)\ \{\psi \mid \varepsilon\}$. Application of **(wk)** to a term produced by **(raise)** yields $\{\mathrm{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon)\}\ x \leftarrow \mathsf{raise}\, e\ \{\psi \mid \varepsilon\}$. Thus, the claim holds for the smallest possible programs.

Inductive step. We assume that for programs q syntactically smaller than p, the Hoare quadruples $\{\mathrm{wp}(x \leftarrow q, \psi \mid \varepsilon)\}\ x \leftarrow q\ \{\psi \mid \varepsilon\}$ can be derived.

Case $p = \mathsf{catch}\, q$.

For $y \leftarrow p$, let ξ denote $\mathrm{wp}_s(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y])$. Then we have

$$\{\xi\}\ x \leftarrow q\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y]\}$$

by induction hypothesis. Applying **(catch)** results in

$$\begin{aligned}
&\{\xi\}\ y \leftarrow \mathsf{catch}\, q\ \{\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi[\mathsf{inl}\, x/y];\ \mathsf{inr}\, e \mapsto (\lambda e\,.\,\psi[\mathsf{inr}\, e/y])(e) \mid \bot\}\\
\Leftrightarrow&\{\xi\}\ y \leftarrow \mathsf{catch}\, q\ \{\psi[(\mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{inl}\, x;\ \mathsf{inr}\, e \mapsto \mathsf{inr}\, e)/y] \mid \bot\}\\
\Leftrightarrow&\{\mathrm{wp}_s(y \leftarrow p, \psi \mid \varepsilon)\}\ y \leftarrow p\ \{\psi \mid \bot\}
\end{aligned}$$

An application of **(wk)** yields the desired result.

Case $p = \mathsf{do}\ x \leftarrow q; r$

We have

$$\{\mathrm{wp}_s(x \leftarrow q, \mathrm{wp}_s(y \leftarrow r, \psi \mid \varepsilon) \mid \varepsilon)\}\ x \leftarrow q\ \{\mathrm{wp}_s(y \leftarrow r, \psi \mid \varepsilon)\}$$

and

$$\{\mathrm{wp}_s(y \leftarrow r, \psi \mid \varepsilon)\}\ y \leftarrow r\ \{\psi \mid \varepsilon\}$$

by induction hypothesis. Application of **(do)** directly gives us

$$\begin{aligned}
&\{\mathrm{wp}_s(x \leftarrow q, \mathrm{wp}_s(y \leftarrow r, \psi \mid \varepsilon) \mid \varepsilon)\}\ y \leftarrow (\mathsf{do}\ x \leftarrow q; r)\ \{\psi \mid \varepsilon\}\\
\Leftrightarrow&\{\mathrm{wp}_s(y \leftarrow (\mathsf{do}\ x \leftarrow q; r), \psi \mid \varepsilon)\}\ y \leftarrow (\mathsf{do}\ x \leftarrow q; r)\ \{\psi \mid \varepsilon\}
\end{aligned}$$

Case $p = \mathsf{case}\, c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r$.

Since $x \leftarrow q$ is a program syntactically smaller than p, we have by induction hypothesis that

$$\{\mathrm{wp}_s(x \leftarrow q), \psi \mid \varepsilon)\}\ x \leftarrow q\ \{\psi \mid \varepsilon\}$$

and the analogous triple for r are derivable in the calculus. We are done immediately by applying **(case)** and by Lemma 3.

Case $p = \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r$.

Finally, for **(itercase)**, let

$$\xi = (\nu X\,.\,\lambda x\,.\,\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathrm{wp}_s(x \leftarrow q, X(x) \mid \varepsilon);\ \mathsf{inr}\, b \mapsto \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon))\,(x).$$

By induction hypothesis, $\{\mathrm{wp}_s(x \leftarrow q, \xi \mid \varepsilon)\}\ x \leftarrow q\ \{\xi \mid \varepsilon\}$ is derivable. This quadruple is equivalent to

$$\{\mathrm{wp}_s(x \leftarrow q, \xi \mid \varepsilon)\}\ x \leftarrow q\ \{\mathsf{do}\ a \leftarrow c?; \mathrm{wp}_s(x \leftarrow q, \xi \mid \varepsilon) \vee \mathsf{do}\ b \leftarrow \bar{c}?; \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\}.$$

We also have $\{\mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\}\ x \leftarrow r\ \{\psi \mid \varepsilon\}$ by induction. The required quadruple is obtained directly by applying **(itercase)** to the previous two quadruples. □

This proof already shows that $\mathrm{wp}_s(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ for all programs p, postconditions ψ and maps $\epsilon : E \to \Omega$, by definition of wp and by soundness. Therefore, we only need to prove that

$$\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}_s(x \leftarrow p, \psi \mid \varepsilon) \tag{30}$$

to see that our assertion language is actually expressive enough to formulate our defined weakest preconditions.

Lemma 7. *Given a sequentially compatible predicated monad, inequality 30 holds for all programs p, postconditions ψ and abnormal postconditions ε.*

Proof. Because by definition of weakest precondition $\phi \sqsubseteq \mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ for all valid Hoare quadruples $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$, we need to show that for all such ϕ, the inequality

$$\phi \sqsubseteq \mathrm{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$$

holds. We do this by induction over p. Let therefore $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ be a valid Hoare quadruple and assume as induction hypothesis that the claim above holds for programs syntactically smaller than p.

Case $p = \mathsf{ret}\, t$.

We have

$$\begin{aligned}
&\mathsf{do}\ \phi; y \leftarrow \mathsf{catch}(\mathsf{ret}\, t); \mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \phi;\ \mathsf{inr}\, e \mapsto \varepsilon(e); \mathsf{ret}\, y = \mathsf{do}\ \phi; \mathsf{catch}(\mathsf{ret}\, t)\\
\Leftrightarrow\ &\mathsf{do}\ \phi; y \leftarrow \mathsf{ret}\,\mathsf{inl}\, t; \mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \phi;\ \mathsf{inr}\, e \mapsto \varepsilon(e); \mathsf{ret}\, y = \mathsf{do}\ \phi; \mathsf{ret}\,\mathsf{inl}\, t\\
\Leftrightarrow\ &\mathsf{do}\ \phi; \mathsf{case}\ \mathsf{inl}\, t\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \phi;\ \mathsf{inr}\, e \mapsto \varepsilon(e); \mathsf{ret}\,\mathsf{inl}\, t = \mathsf{do}\ \phi; \mathsf{ret}\,\mathsf{inl}\, t\\
\Leftrightarrow\ &\mathsf{do}\ \phi; \phi[t/x]; \mathsf{ret}\,\mathsf{inl}\, t = \mathsf{do}\ \phi; \mathsf{ret}\,\mathsf{inl}\, t.
\end{aligned}$$

Because do is monotone and by Lemma 1, $\phi \sqcap \phi[t/x] = \phi$, which means that $\phi \sqsubseteq \phi[t/x] = \mathrm{wp}_s(x \leftarrow \mathsf{ret}\, t, \psi \mid \varepsilon)$.

Case $p = c?$.

$$\mathsf{do}\ \phi; c?; \mathsf{ret}\ \star = \mathsf{do}\ \phi; x \leftarrow c?; \psi; \mathsf{ret}\ \star$$

and by weak discardability and monotonicity of joins, we have

$$\begin{aligned}\phi &= \mathsf{do}\ \phi; c? \vee \mathsf{do}\ \phi; \bar{c}?\\ &\sqsubseteq \mathsf{do}\ \phi; c? \vee \bar{c}?\\ &\sqsubseteq \mathsf{do}\ \phi; x \leftarrow c?; \psi \vee \bar{c}?\\ &\sqsubseteq \mathsf{do}\ x \leftarrow c?; \psi \vee \bar{c}?.\end{aligned}$$

Case $p = \mathsf{do}\ x \leftarrow q; r$.

Our monad is sequentially compatible by assumption, so we have $\phi \sqsubseteq \mathrm{wp}(y \leftarrow (\mathsf{do}\ x \leftarrow q; r), \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}(x \leftarrow q, \mathrm{wp}(y \leftarrow r, \psi \mid \varepsilon) \mid \varepsilon)$, from which the claim follows by induction.

Case $p = f(z), f \in \Sigma$.

Similar to the case for soundness of **(basic)**, we replace $f(z)$ by $\mathsf{do}\ z \leftarrow \mathsf{ret}\, t; f(z)$, so that we have $\{\phi\}\ x \leftarrow (\mathsf{do}\ z \leftarrow \mathsf{ret}\, t; f(z))\ \{\psi \mid \varepsilon\}$ and therefore

$$\phi \sqsubseteq \mathrm{wp}(x \leftarrow (\mathsf{do}\ z \leftarrow \mathsf{ret}\, t; f(z)), \psi \mid \varepsilon).$$

By sequential compatibility, we get $\phi \sqsubseteq \mathrm{wp}(z \leftarrow \mathsf{ret}\, t, \mathrm{wp}(x \leftarrow f(z), \psi \mid \varepsilon) \mid \varepsilon)$ and by induction and equations 24 and 21,

$$\begin{aligned}\phi &\sqsubseteq \mathrm{wp}_s(z \leftarrow \mathsf{ret}\, t, \mathrm{wp}(x \leftarrow f(z), \psi \mid \varepsilon) \mid \varepsilon)\\ &= \mathrm{wp}(x \leftarrow f(z), \psi \mid \varepsilon)[t/x]\\ &= \mathrm{wp}_s(x \leftarrow f(t), \psi \mid \varepsilon).\end{aligned}$$

Case $p = \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q; \mathsf{inr}\, b \mapsto r$.

We have the Hoare quadruple $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$. Because $\mathsf{do}\ c?; \phi \sqsubseteq \phi$, and likewise for $\bar{c}?$, we get, after weakening:

$$\begin{aligned}\{c? \wedge \phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} &\Leftrightarrow \{c? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ a \leftarrow c?; q)\ \{\psi \mid \varepsilon\}\\ \{\bar{c}? \wedge \phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} &\Leftrightarrow \{\bar{c}? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r)\ \{\psi \mid \varepsilon\},\end{aligned}$$

which means that

$$\begin{aligned}c? \wedge \phi &\sqsubseteq c? \wedge \mathrm{wp}(x \leftarrow (\mathsf{do}\ a \leftarrow c?; q), \psi \mid \varepsilon) \sqsubseteq c? \wedge \mathrm{wp}(a \leftarrow c?, \mathrm{wp}(x \leftarrow q, \psi \mid \varepsilon) \mid \varepsilon)\\ \bar{c}? \wedge \phi &\sqsubseteq \bar{c}? \wedge \mathrm{wp}(x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r), \psi \mid \varepsilon) \sqsubseteq \bar{c}? \wedge \mathrm{wp}(b \leftarrow \bar{c}?, \mathrm{wp}(x \leftarrow r, \psi \mid \varepsilon) \mid \varepsilon)\end{aligned}$$

by sequential compatibility. We have already shown that $\mathrm{wp}(a \leftarrow c?, \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}_s(a \leftarrow c?, \psi \mid \varepsilon)$ and likewise for $\bar{c}?$ above, so this can be further transformed to

$$\begin{aligned}c? \wedge \phi &\sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathrm{wp}(x \leftarrow q, \psi \mid \varepsilon)\\ \bar{c}? \wedge \phi &\sqsubseteq \mathsf{do}\ b \leftarrow \bar{c}?; \mathrm{wp}(x \leftarrow r, \psi \mid \varepsilon)\end{aligned}$$

By induction, the weakest preconditions on the right-hand sides are smaller than the respective syntactic weakest preconditions, so that

$$\phi = (c? \wedge \phi) \vee (\bar{c}? \wedge \phi) \sqsubseteq \mathrm{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$$

as desired.

Case $p = \mathsf{itercase}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r$.

Let

$$\xi = (\nu X \,.\, \lambda x \,.\, \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathrm{wp}_s(x \leftarrow q, X(x) \mid \varepsilon);\ \mathsf{inr}\, b \mapsto \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon))\,(x).$$

Following [9], we need to prove that $\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon) =: \phi_0$ is a postfixed point of the functional defining ξ:

Definition 26. In any partial order P and function $f : P \to P$, an element p of P such that $p \sqsubseteq f(p)$ is called a *postfixed point* of f.

By the definition of weakest precondition, this amounts to showing that every ϕ satisfying $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is smaller than the functional evaluated at ϕ_0, i.e.

$$\phi_0 \sqsubseteq \mathsf{case}\ c\ \mathsf{of}\ \mathsf{inl}\, a \mapsto \mathrm{wp}_s(x \leftarrow q, \phi_0 \mid \varepsilon);\ \mathsf{inr}\, b \mapsto \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon).$$

Therefore, by case distinction, we need to show that

$$c? \wedge \phi_0 \sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathrm{wp}_s(x \leftarrow q, \phi_0 \mid \varepsilon)$$
$$\bar{c}? \wedge \phi_0 \sqsubseteq \mathsf{do}\ b \leftarrow \bar{c}?; \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon).$$

Continuation branch. Unrolling the first iteration of the loop, we see that $\{c? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ a \leftarrow c?; x \leftarrow q; p)\ \{\psi \mid \varepsilon\}$ holds and thus

$$\begin{aligned} c? \wedge \phi &\sqsubseteq \mathrm{wp}(x \leftarrow (\mathsf{do}\ a \leftarrow c?; x \leftarrow q; p), \psi \mid \varepsilon) \\ &\sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathrm{wp}(\mathsf{do}\ x \leftarrow q; p, \psi \mid \varepsilon). \end{aligned}$$

By sequential compatibility, $\mathrm{wp}(x \leftarrow (\mathsf{do}\ x \leftarrow q; p), \psi \mid \varepsilon) \sqsubseteq \mathrm{wp}(x \leftarrow q, \mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon))$ and by induction $c? \wedge \phi \sqsubseteq \mathsf{do}\ a \leftarrow c?; \mathrm{wp}_s(x \leftarrow q, \psi_0 \mid \varepsilon)$.

Termination branch. By definition of **(itercase)**, $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ decomposes sequentially into

$$\{\phi\}\ x \leftarrow p[\mathsf{ret}\, x/r]\ \{\bar{c}? \wedge \phi\} \text{ and } \{\bar{c}? \wedge \phi\}\ x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r)\ \{\psi \mid \varepsilon\}.$$

From the latter quadruple, we have $\bar{c}? \wedge \phi \sqsubseteq \mathrm{wp}(x \leftarrow (\mathsf{do}\ b \leftarrow \bar{c}?; r), \psi \mid \varepsilon)$. Again, $\bar{c}? \wedge \phi \sqsubseteq \mathsf{do}\ b \leftarrow \bar{c}?; \mathrm{wp}_s(x \leftarrow r, \psi \mid \varepsilon)$ by sequential compatibility and by induction.

Case $p = \mathsf{catch}\, q$.

As seen in the deduction of the syntactic weakest precondition for catch, the Hoare quadruple in question is equivalent to

$$\{\phi\}\ x \leftarrow q\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y]\}$$

Thus, $\phi \sqsubseteq \mathrm{wp}(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y])$ and by induction

$$\mathrm{wp}(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr})/y]) \\ \sqsubseteq \mathrm{wp}_s(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e\,.\,\psi[\mathsf{inr}\, e/y]) = \mathrm{wp}_s(x \leftarrow p, \psi \mid \varepsilon).$$

Case $p = \mathsf{raise}\, e$.

Since $\mathsf{catch}(\mathsf{raise}\, e) = \mathsf{ret}\,\mathsf{inr}\, e$,

$$\begin{aligned} &\mathsf{do}\ \phi; y \leftarrow \mathsf{catch}(\mathsf{raise}\, e); \mathsf{case}\ y\ \mathsf{of}\ \mathsf{inl}\, x \mapsto \psi;\ \mathsf{inr}\, e \mapsto \varepsilon(e); \mathsf{ret}\, y = \mathsf{do}\ \phi; \mathsf{ret}\,\mathsf{inr}\, e \\ \Leftrightarrow\ &\mathsf{do}\ \phi; \varepsilon(e); \mathsf{ret}\,\mathsf{inr}\, e = \mathsf{do}\ \phi; \mathsf{ret}\,\mathsf{inr}\, e \end{aligned}$$

So, $\phi \sqcap \varepsilon(e) = \phi$, and therefore $\phi \sqsubseteq \varepsilon(e) = \mathrm{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon)$. □

Theorem 4 (Relative completeness)**.** *Let* $(\mathbb{T}, \mathbb{P})$ *be a sequentially compatible predicated monad and let the weakest preconditions for basic programs be expressible in the assertion language. Then*

$$\mathbb{T}, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}\ \textit{implies}\ \Delta_\Sigma \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\},$$

where Δ_Σ *are axioms for the basic programs in the signature* Σ.

Proof. Using Lemma 6 and Lemma 7, we can express the weakest precondition $\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ and thus

$$\Delta_\Sigma \vdash_{\mathbb{P}} \{\mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

By definition, $\phi \sqsubseteq \mathrm{wp}(x \leftarrow p, \psi \mid \varepsilon)$ for valid Hoare triples, so we can use **(wk)** to derive the required triple. □

5 Conclusion

5.1 Achievements

The main result of this work is slightly hidden. We want to emphasize at this point that our metalanguage supports a signature Σ of operations. These operations cannot have arbitrary types, in the sense that they need to be of type $A \to B$, where A is a value type. So, we do support operations $A \to TC$ for a value type C, which, by a result of Plotkin and Power [18], correspond to algebraic operations.

The type of the catch operation, however, is $TA \to T(A + E)$ and thus catch is not algebraic. We have shown in this work that the inclusion of a single non-algebraic operation can still yield a sound and complete Hoare calculus. That is, we showed that the logic of our assertion language is strong enough to express a weakest precondition for the exception handling mechanism.

Because of the way we modeled exception monads (which by Mossakowski and Schröder [20] is the only way), we needed to introduce generic coproducts into the term formation language. We therefore also expanded the Hoare calculus with support for case distinction on coproduct types and a looping construction based on case distinction. We showed that this itercase loop can be expressed using the while loop from [9] if case distinction is present in the calculus.

We have also shown that Moggi's exception monad transformer preserves order-enrichment. This ensures that our framework supports exactly the same base monads as the framework by Goncharov and Schröder.

5.2 Further Work

Our work shows that the introduction of single non-algebraic operations can be handled adequatly, yet we are far from a generic framework. We have seen that already the case of exception handling required a detailed study of the operation itself and even an extension of the usual Hoare triples to quadruples, so this raises the question whether such a framework is indeed feasible.

Another point of interest is the choice of base category. Recall that our notion of predicate relies on order-enrichment of the monad, so that the restrictions on the base category are fairly mild, i.e. we only require a distributive category with equalizers and Kleisli exponentials for an innocent submonad of our targeted monad. Goncharov and Schröder [9] already proved that the choice of **Set** as the base category makes the logic induced by the innocent submonad a full intuitionistic logic. It would be interesting to know which logics arise if we instantiate the framework to other concrete categories.

Goncharov and Schröder [9], in addition to a weakest precondition calculus, also considered the dual case of *strongest preconditions*, which also gives them a simple characterisation of sequential compatibility (see Definition 25). An equal treatment for the calculus at hand would be desirable.

Our calculus makes a distinction of divergence and exceptional termination. However, the separation of the postcondition into normal and abnormal postcondition is reminiscent of Hoare's total correctness assertions. It may be possible to derive a calculus for total correctness from the present work.

Bibliography

[1] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and concrete categories*. John Wiley & Sons Inc., 1990.

[2] S. Awodey. *Category Theory (Oxford Logic Guides)*. Oxford University Press, USA, July 2006.

[3] M. Barr and C. Wells, editors. *Category theory for computing science, 2nd ed.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.

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

[5] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, pages 70–90, 1978.

[6] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, Aug. 1975.

[7] R. Godement. *Topologie algébrique et théorie des faisceaux*. Actualites Scientifiques et Industrielles. 1252. Publications de l'Institut de Mathematique de l'Universite de Strasbourg. XIII. Paris: Hermann & Cie. VIII, 283 p., 1958.

[8] S. Goncharov and L. Schröder. A coinductive calculus for asynchronous side-effecting processes. In O. Owe, M. Steffen, and J. A. Telle, editors, *Fundamentals of Computation Theory (FCT 2011)*, volume 6914 of *Lecture Notes in Computer Science*. Springer, 2011.

[9] S. Goncharov and L. Schröder. A relatively complete hoare logic for order-enriched effects. In O. Kupferman, editor, *Logic in Computer Science (LICS 2013)*. IEEE, 2013.

[10] HackageDB. http://hackage.haskell.org, July 2013.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12:576–580, 1969.

[12] A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.

[13] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[14] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Edinburgh Univ., Dept. of Comp. Sci., June 1989. Lecture Notes for course CS 359, Stanford Univ.

[15] E. Moggi. Computational lambda-calculus and monads. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 14–23. IEEE Computer Society Press, 1989.

[16] E. Moggi. A modular approach to denotational semantics. In *Category Theory and Computer Science, CTCS 1991*, volume 530 of *LNCS*, pages 138–139. Springer, 1991.

[17] N. S. Papaspyrou. A resumption monad transformer and its applications in the semantics of concurrency. In *Panhellenic Logic Symposium*, 2001.

[18] G. Plotkin and J. Power. Algebraic operations and generic effects. *Appl. Cat. Struct.*, 11:69–94, 2003.

[19] J. Power. Unicity of enrichment over cat or gpd. *Applied Categorical Structures*, 19(1):293–299, Feb. 2011.

[20] L. Schröder and T. Mossakowski. Generic exception handling and the Java monad. In *Algebraic Methodology and Software Technology, AMAST 2004*, volume 3116 of *LNCS*, pages 443–459. Springer, 2004.

[21] K. Segrt. *Morita Theory in Enriched Context*. PhD thesis, University of Nice Sophia-Antipolis, Feb. 2012.

[22] A. K. Simpson. Recursive types in Kleisli categories. Technical report, MFPS Tutorial, April 2007, 1992.

[23] P. Taylor. *Practical Foundations of Mathematics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[24] S. Vickers. *Topology via Logic*. Cambridge University Press, 1989.

[25] P. Wadler. How to declare an imperative. *ACM Comput. Surveys*, 29:240–263, 1997.

[26] G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.