Generic Hoare Logic for Order-Enriched Effects with Exceptions

Christoph Rauch, Sergey Goncharov, Lutz Schröder

Generic Hoare Logic for Order-Enriched Effects with Exceptions

Christoph Rauch, Sergey Goncharov, and Lutz Schröder

Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. In programming semantics, monads are used to provide a generic encapsulation of side-effects. We introduce a monad-based metalanguage that extends Moggi's *computational metalanguage* with native exceptions and iteration, interpreted over monads supporting a dcpo structure. We present a Hoare calculus with abnormal postconditions for this metalanguage and prove relative completeness using weakest liberal preconditions, extending earlier work on the exception-free case.

1 Introduction

The use of monads as a means of modelling side-effecting computations [10] is well-established in the design and semantics of programming languages. A broad range of computational effects is subsumed under this paradigm, e.g. store, non-determinism, exceptions, probabilities, and alternation.

In addition to just *modelling* side-effects, a matter of interest is how to *reason* generically about properties of side-effecting programs, one of the most important properties being functional correctness. Goncharov and Schröder [7] have developed a monad-based generic *Hoare calculus*, and proved relative completeness using weakest liberal preconditions. More precisely the calculus is parametrized by the choice of a *predicated monad*, i.e. a monad that supports iteration via a suitable dcpo enrichment of its Kleisli category (the associated category of side-effecting functions), and is moreover equipped with a suitable submonad of *innocent* computations for use in assertions. In fact, the object of truth values is generated from a predicated monad as the type of innocent computations of unit type (pre- and postconditions thus become *affirmative assertions* in the sense of Vickers [20], e.g. semi-decidable).

Exception monads $T_E X = T(X + E)$ (with E an object of exceptions and T a predicated base monad) are, in fact, predicated but include an operation that does not fit into the required scheme for basic operations, namely exception catching $\mathsf{catch} : TA \to T(A + E)$. The framework covers only operations that are *algebraic* [15], which catch is not. We therefore extend the generic Hoare calculus with explicit support for exception handling via *abnormal postconditions* [9], which govern the behaviour of a computation for the case where it terminates with an exception. The calculus thus allows, e.g., for a meaningful specification of exception raising, for which the normal postcondition is just $\bot$. Our main result is that the calculus is, again, relatively complete; we prove this by means of a weakest liberal precondition calculus featuring abnormal postconditions.

Related Work. Schröder and Mossakowski [18] present a monad-based Hoare calculus with abnormal postconditions, interpreted over a program semantics where the *base category* of the monad, rather than only the Kleisli category, is enriched over a suitable category of domains; our setting thus allows for more instances, e.g. involving monads on the category of sets. Moreover the semantics of the Hoare logic of [18] differs from ours in that the object of truth values is assumed in the base category rather than extracted from the monad as in our setting. The calculus of [18] is proved sound but a relative completeness proof is currently missing. Relative completeness proofs in *most general formula* style have been given for specific effect combinations found in concrete programming languages with exceptions, namely fragments of Java [13,16] and Eiffel [12]. Our approach to generic weakest preconditions essentially follows our previous work on the exception-free case [7]. In work subsequent to [7], Hasuo [8] provides an alternative categorical analysis of monad-based weakest preconditions, notably relying on similar assumptions to ensure well-behavedness of weakest preconditions (see Remark 11 for a detailed comparison). In this setup, no syntactic Hoare calculus, and a fortiori no relative completeness result, is provided.

2 Preliminaries: Monads and Exceptions

A *monad* $\mathbb{T}$ over a category $\mathcal{C}$ with class $|\mathcal{C}|$ of objects can be described as a *Kleisli triple* $(T, \eta, -^*)$ in which T is an endomap over $|\mathcal{C}|$ (we adopt the general convention to use blackboard characters for monads and the corresponding Roman letters for their functorial parts), η – the *unit* of the monad – is a family of morphisms $(\eta_X : X \to TX)_{X \in |\mathcal{C}|}$ and $-^*$ – the *Kleisli lifting* – is a map sending each $f : X \to TY$ to $f^* : TX \to TY$ so that the *monad laws* are satisfied:

$$\eta^* = \mathsf{id}, \qquad f^* \circ \eta = f, \qquad (f^* \circ g)^* = f^* \circ g^* \tag{1}$$

This is equivalent to the standard definition of a monad in terms of a functor T with unit and multiplication; in particular, $Tf = (\eta f)^*$ for morphisms f. One may think of TX as being a type of side-effecting computations delivering results in X; examples include non-termination ($TX = X + 1$), non-determinism ($TX = \mathcal{P}X$), statefulness (e.g. the *partial state monad* is given by $TX = S \rightharpoonup (S \times X)$ for an object S of states, where $\rightharpoonup$ takes partial function spaces), and exceptions ($TX = X + E$ for an object E of exceptions).

Furthermore, a *strong* monad is a monad $\mathbb{T}$ together with a family of morphisms $\tau : X \times TY \to T(X \times Y)$ natural in both X and Y satisfying a set of coherence axioms (cf. e.g. [11]). As originally noted by Moggi, strong monads support a *computational metalanguage* [11], incorporated into Haskell as the *do-notation* [21]: it features an operator ret, denoting the unit η, and the do-binder, which from $f : X \to TY$ and $g : X \times Y \to TZ$ builds $\mathsf{do}\, y \leftarrow f(x); g(x, y)$, in which y is syntactically bound and whose meaning is $g^\star \circ \tau \circ \langle \mathsf{id}, f \rangle : X \to TZ$. For a term p in the computational metalanguage, $FV(p)$ denotes the set of free variables of p. The monad laws (1) can be rewritten in this style, and then form

part of a complete axiomatization:

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

A monad $\mathbb{T}$ defines the *Kleisli category* $\mathcal{C}_T$ of $\mathbb{T}$, whose objects are those of $\mathcal{C}$ and $\mathsf{Hom}_{\mathcal{C}_T}(A, B) = \mathsf{Hom}_{\mathcal{C}}(A, TB)$ with composition defined using the Kleisli lifting of $\mathbb{T}$; that is, morphisms $X \to Y$ in $\mathcal{C}_T$ may be thought of side-effecting functions $X \to Y$, with side-effects specified by T.

We fix from now on an object E of exceptions and a base monad $\mathbb{T}$ on a category $\mathcal{C}$ which is *distributive*, i.e. the canonical distributivity morphism $X \times Y + X \times Z \to X \times (Y + Z)$ is an isomorphism, with inverse called dist, and which has *Kleisli exponentials* [19,15], i.e. exponentials [2] of the form TZ^Y. The latter condition means that there is a natural isomorphism $\lambda : \mathsf{Hom}_{\mathcal{C}}(X \times Y, TZ) \cong \mathsf{Hom}_{\mathcal{C}}(X, TZ^Y)$. We furthermore denote pairing and copairing morphisms by angle and square brackets, respectively.

The well-known *exception monad transformer* [4] adds exceptions as an effect to a given monad: the functor

$$T_E = T(\text{-} + E)$$

is turned into a strong monad $\mathbb{T}_E$, called an *exception monad*, by putting

- $\eta^{T_E} = \eta^T \circ \mathsf{inl}$;
- $f^* = [f, \eta^T \circ \mathsf{inr}]^*$ for $f : A \to T(B + E)$;
- $\tau^{T_E}_{A,B} = T((\mathsf{id} + \pi_2) \circ \mathsf{dist}) \circ \tau_{A,B+E}$.

That is, a computation in $\mathbb{T}_E$ performs a side effect specified by $\mathbb{T}$ and then either terminates normally or with an exception. The definition of binding in $\mathbb{T}_E$ means that in the latter case, subsequent statements have no further effects (e.g. in case $\mathbb{T}$ features statefulness, the state is frozen) other than propagating the exception. On $\mathbb{T}_E$, we have operations for raising and catching exceptions (i.e. exposing exceptions raised by a program in the output type),

$$\begin{aligned} \mathsf{raise}_X &= \eta\, \mathsf{inr} : E \to T(X + E) = T_E X \\ \mathsf{catch}_X &= T\, \mathsf{inl} : T_E X = T(X + E) \to T((X + E) + E) = T_E(X + E). \end{aligned}$$

3 Order-Enrichment and Innocence

As indicated in the introduction, we require a certain amount of infrastructure in the base monad $\mathbb{T}$ to support, on the one hand, loops, and on the other hand, an internalized assertion language. To this end, we require a complete partial order on programs, i.e. Kleisli morphisms, of a given type $A \to TB$. We will then use certain well-behaved programs as logical assertions. Formal definitions are as follows [7].

Definition 1 (Order-enriched monad). The monad $\mathbb{T}$ is *order-enriched* if

– every hom-set $\mathsf{Hom}_{\mathcal{C}}(A, TB)$ carries a bounded-complete and directed-complete partial ordering $\sqsubseteq$ (i.e. joins exist for finite bounded subsets and for directed subsets, and consequently for all bounded subsets),
– for any $h \in \mathsf{Hom}_{\mathcal{C}}(A', A)$ and any $u \in \mathsf{Hom}_{\mathcal{C}}(B, TB')$, the maps

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

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

The ordering $\sqsubseteq$ is to be thought of as the usual information ordering; in particular, non-termination is below termination. Note that we do not include continuity of copairing in the above definition; in fact this follows from preservation of joins (Lemma 6).

We identify a class of *innocent* computations suitable for use within assertions of the Hoare logic. Intuitively, such a computation may read but should not modify the state, as required already in earlier work on monad-based program logics [17]. Instead of assuming a type of truth values in the base category, we will identify truth with termination, so that truth values are just innocent computations of unit type. Formally, the conditions for innocence are as follows [7].

Definition 2 (Innocence). A monad is *commutative* if it satisfies

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

for all p, q, and fresh x, y, in the computational metalanguage (Section 2). An order-enriched monad is *innocent* if it is commutative and satisfies *copyability*

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

(for all p and fresh x, y) and *weak discardability*

$$\mathsf{do}\ x \leftarrow p; \mathsf{ret}\star \sqsubseteq \mathsf{ret}\star. \tag{2}$$

Notice that the notion of weak discardability uses the ordering of the monad. It allows for non-termination; e.g. if $p = \bot$ is the everywhere diverging program, then the left hand side of (2) is also $\bot$ so that the inequality is satisfied. (Contrastingly, previous approaches [17] used *discardability* $\mathsf{do}\ x \leftarrow p; \mathsf{ret}\star = \mathsf{ret}\star$, which implies termination.) On the other hand, weakly discardable computations cannot raise exceptions, as these fail to be below $\mathsf{ret}\star$. Intuitively, the only side-effects an innocent computation may exhibit are possible non-termination and read operations on implicit states encapsulated in the base monad.

A key property of innocent monads $\mathbb{P}$ is that they support a logic on the base category, which we will later use as the underlying logic of the assertions of our Hoare calculus. Specifically, $P1$ *is a distributive lattice object in* $\mathcal{C}$, i.e. $\mathsf{Hom}(\text{–}, P1)$ factors through distributive lattices, in fact even through *frames* [7],

which are complete lattices in which finite meets distribute over all joins (with morphisms preserving finite meets and all joins). Consequently, $P1$ supports a form of *geometric logic* [20]. Notably, finite meets in $P1$ are given by sequential composition. In case $\mathcal{C} = \mathbf{Set}$, $P1$ is in fact a Heyting algebra, and thus supports full intuitionistic logic.

Computationally relevant monads, e.g. stateful ones, often fail to be innocent but will typically contain a useful innocent submonad. Formally:

Definition 3 (Predicated monad). A *predicated monad* $(\mathbb{T}, \mathbb{P})$ consists of an order-enriched monad $\mathbb{T}$ and an innocent order-enriched submonad $\mathbb{P}$ of $\mathbb{T}$ whose inclusion into $\mathbb{T}$ is Scott-continuous.

Example 4. In the *nondeterministic state monad* $TX = S \to \mathcal{P}(S \times X)$, a program is weakly discardable iff it (possibly reads but) does not change the state. Such a program is copyable iff it is deterministic; all copyable and weakly discardable programs in $\mathbb{T}$ commute. We thus obtain an innocent submonad $\mathbb{P}$ of $\mathbb{T}$ given by $PX = S \rightharpoonup X$, the *partial reader monad*. Taking $\mathbb{T}$ to be either the partial state monad (Section 2) or the nondeterministic state monad and $\mathbb{P}$ the partial reader monad, we thus thus obtain a predicated monad $(\mathbb{T}, \mathbb{P})$.

Lemma 5. *Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad. Then the exception monad $\mathbb{T}_E$ is also order-enriched, and $(\mathbb{T}_E, \mathbb{P})$ is a predicated monad.*

From now on, we fix a predicated monad $(\mathbb{T}, \mathbb{P})$.

4 A Computational Metalanguage

We proceed to define the syntax of a monad-based metalanguage that extends Moggi's computational metalanguage (Section 2) with native support for iteration and innocence. We introduce a system of *value types* A and *computation types* C by the grammar

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

with W ranging over a set $\mathcal{W}$ of basic types (denoting objects of $\mathcal{C}$). While the type constructor T_E represents side-effecting computations possibly raising exceptions in E, the type constructor P represents innocent computations. The type Ω of truth values is just a synonym for $P1$. Additionally, we fix a signature Σ of typed function symbols $f : A \to C$.

The term language is defined by means of the set of typing rules for terms in context shown in Figure 1; contexts Γ consist of assignments $x : A$ of value types A to variables x. We have the usual term formation rules for products, coproducts, sequential composition of computations, application of functions from the signature Σ, and most notably for exception raising and handling. Additionally, we incorporate rules for innocent computations saying essentially that P is a submonad of T_E, i.e. closed under unit and binding, and each PA has a least element $\bot$ ($\bot$ could of course be defined as a nonterminating loop but

for our purposes it is technically more convenient to make it a first-class citizen). Our loop construct is tuned to sum types:

$$\mathsf{init}\ x\ \leftarrow\ p\ \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q;\ \mathsf{inr}\, z \mapsto r$$

initializes the loop variable x with the result of p and then proceeds to execute q repeatedly, binding the result to x, as long as $c : B + C$ remains in the left hand summand. Once c ends up in the right hand summand, the loop terminates and r is executed once, with the result bound to x. Note that the construct at hand allows passing *values* through the loop; boolean loop conditions as in standard while loops arise as special cases of our sum-type-based loop conditions, as the type 2 of Booleans is the sum type $1 + 1$.

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

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

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

$$\frac{\Gamma \vdash p : MA \quad \Gamma, x : A \vdash q : MB}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : MB}\ M \in \{T_E, P\} \qquad \frac{\Gamma \vdash p : A}{\Gamma \vdash \mathsf{ret}\, p : PA}\ (A \text{ value type})$$

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

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

Fig. 1. Term formation rules for the simple imperative metalanguage with exceptions.

In addition to the metalanguage of programs, we can now use the fact that $P1 = \Omega$ serves as an object of truth values in geometric logic to develop rules for an assertion language. These rules, given in Figure 2 and understood as extending those of Figure 1, are identical to those for the exception-free case [7], essentially because innocent computations do not raise exceptions. The logic combines a very restricted set of propositional connectives with least and greatest fixpoint constructors over dedicated predicate variables. It is designed to accommodate weakest preconditions, i.e. is strong enough to guarantee relative completeness of the Hoare calculus.

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

$$\frac{\Gamma \vdash p : PA \quad \Gamma, x : A \vdash q : \Omega}{\Gamma \vdash \mathsf{do}\ x \leftarrow p; q : \Omega} \qquad \frac{\Gamma, X : A \to \Omega \vdash \phi : A \to \Omega}{\Gamma \vdash \eta X.\, \phi : A \to \Omega}\ (\eta \in \{\mu, \nu\})$$

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

Fig. 2. Assertion language.

5 Semantics

The types C of our metalanguage are interpreted over the predicated (exception) monad $(\mathbb{T}_E, \mathbb{P})$ as $\mathcal{C}$-objects $[\![C]\!]$ in the expected way, given an assignment of $\mathcal{C}$-objects $[\![W]\!]$ to basic types W, with $+$ and $\times$ interpreted as categorical sum and product, P and T_E as the corresponding object maps of $\mathbb{T}_E$ and $\mathbb{P}$, 1 as the terminal object, and $A \to \Omega$ as the Kleisli exponential $([\![P1]\!])^{[\![A]\!]}$.

As usual, contexts are then interpreted as the product of the interpretations of the types of their variables, and a term in context $\Gamma \vdash p : C$ as a morphism $[\![\Gamma]\!] \to [\![C]\!]$. The term constructors for product and sum types are interpreted using the structure of $\mathcal{C}$ as a distributive category in the usual way, and the interpretation of the term constructors for the monadic structure (ret and do) is as in the computational metalanguage [11], instantiated for the exception monad $\mathbb{T}_E$, respectively its submonad $\mathbb{P}$ for innocent computations; $\Gamma \vdash \bot : PA$ is interpreted as the bottom element of $\mathsf{Hom}([\![\Gamma]\!], [\![PA]\!])$. The operations catch and raise are interpreted as the corresponding morphisms for $\mathbb{T}_E$ (Section 2); e.g. $[\![\Gamma \vdash \mathsf{catch}\, p : T_E(A + E)]\!] = \mathsf{catch} \circ [\![p]\!]$.

The interpretation of our coproduct-based loop construct itcase follows the treatment of a previous Boolean loop construct [7]. 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 $c \mapsto \bar{c}$ denote the symmetry isomorphism $A + B \to B + A$, i.e. $\bar{c} = \mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto \mathsf{inr}\, y;\ \mathsf{inr}\, z \mapsto \mathsf{inl}\, z$ for $c : A + B$. We adopt the convention to write $c?$ instead of $(\mathsf{do}\ c?; \mathsf{ret} \star)$ whenever the return type $P1$ is expected.

Lemma 6. *Assume terms* $\Gamma, y : A \vdash p : T_E C$; $\Gamma, z : B \vdash q : T_E C$; *and* $\Gamma \vdash c : A + B$. *Then the join* $(\mathsf{do}\ y \leftarrow c?; p) \sqcup (\mathsf{do}\ z \leftarrow \bar{c}?; q)$ *exists and equals* $\mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto p;\ \mathsf{inr}\, z \mapsto q$.

Consequently, case is Scott-continuous in both program arguments. By Kleene's fixpoint theorem, we can thus 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

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

The full itcase construct is then interpreted as

$$\begin{aligned}&[\![\Gamma \vdash \mathsf{init}\, x \leftarrow p\ \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q;\ \mathsf{inr}\, z \mapsto r : T_E A]\!] = \\ &\quad [\![\Gamma \vdash \mathsf{do}\ x \leftarrow p; (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, y \mapsto q;\ \mathsf{inr}\, z \mapsto r) : T_E A]\!]. \quad (3)\end{aligned}$$

Semantics of assertions. Contexts Γ for assertions may contain propositional variables $X : A \to \Omega$, giving rise to factors $([\![P1]\!])^{[\![A]\!]}$ in the interpretation of Γ. Assertions $\Gamma \vdash \phi : \Omega$ are then interpreted as morphisms $[\![\Gamma]\!] \to [\![P1]\!]$. Logical conjunction and disjunction are interpreted as meets and joins in $\Omega = P1$, respectively (which exist because $P1$ is a distributive lattice object). For lambda abstraction and evaluation of predicates, we inductively define

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

where ϵ is the usual evaluation morphism. Fixpoints $\Gamma \vdash \eta X.\, \phi$ with $\eta \in \{\mu, \nu\}$ are interpreted using the fact that the hom-sets $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$ are complete lattices. In detail, the interpretation of $\Gamma, X : A \to \Omega \vdash \phi : \Omega$ is a morphism $g : [\![\Gamma]\!] \times ([\![P1]\!])^{[\![A]\!]} \to ([\![P1]\!])^{[\![A]\!]}$. We thus define an endomorphism F on $\mathsf{Hom}([\![\Gamma]\!] \times [\![A]\!], [\![\Omega]\!])$ by taking $F(h)$ to be

$$[\![\Gamma]\!] \times [\![A]\!] \xrightarrow{\langle \mathsf{id}, \lambda h \circ \pi_1\rangle} [\![\Gamma]\!] \times [\![A]\!] \times (P1)^{[\![A]\!]} \xrightarrow{\langle g \circ \langle \pi_1, \pi_3\rangle, \pi_2\rangle} (P1)^{[\![A]\!]} \times [\![A]\!] \xrightarrow{\epsilon} [\![\Omega]\!].$$

We then take $[\![\Gamma \vdash \eta X.\, \phi]\!]$ to be $\lambda \eta F$; the required fixpoints exist by the Knaster-Tarski theorem as all assertions are monotone in their variables.

Definition 7. A judgment $\Gamma \vdash \phi \sqsubseteq \psi$ is *valid over* $\mathbb{P}$ if $[\![\phi]\!] \sqsubseteq [\![\psi]\!]$.

6 Hoare Calculus

In the exception-free case, a semantics of Hoare triples over order-enriched monads [7] is defined by taking $\{\phi\}\ x \leftarrow p\ \{\psi\}$ (with $p : T_E A$ and assertions ϕ, ψ, where ψ may depend on $x : A$) to abbreviate the equation

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

As indicated previously, this setup is insufficient for the verification of exception-raising programs; e.g. nothing useful can be said about raise beyond $\{\top\}\ \mathsf{raise}\ e\ \{\bot\}$. We thus need an additional postcondition for abnormal termination [9,18]: Intuitively, a *Hoare quadruple*

$$\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$$

with *abnormal postcondition* $\varepsilon : E \to \Omega$ says that if p is executed in a prestate satisfying ϕ and terminates normally with result x, then the poststate and x satisfy the *normal postcondition* ψ, and if p terminates with an exception e then the poststate satisfies $\varepsilon(e)$. Using the previous definition of Hoare triple, we formally interpret $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ by letting it abbreviate

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

We write $\mathbb{T}, \mathbb{P} \vDash \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ if the above equality holds in $(\mathbb{T}, \mathbb{P})$ (under the given interpretation of the basic programs, elided in the notation).

The inference rules for our calculus are shown in Figure 3. The inequalities in **(wk)** refer to the notion of validity introduced above (Definition 7); inequality $\varepsilon \sqsubseteq \varepsilon'$ of abnormal postconditions is understood pointwise. We assume a set Δ of valid axioms for basic programs, and then write $\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ if a Hoare quadruple $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$ is derivable from axioms in Δ and inequalities of assertions valid over $\mathbb{P}$.

The rules **(basic)**, **(ret)**, **(do)** and **(wk)** are derived straightforwardly from the rules for the exception-free case [7]; rule **(⊥)** is clear. We spend some time on discussing the remaining new rules:

Rules for exceptions are taken from [18]. The **(raise)** rule is self-explanatory. The **(catch)** rule is based on the fact that $\mathsf{catch}\, p : T_E(A + E)$ renormalizes the computation p and returns an exception possibly raised by p in the right-hand summand of the normal result.

Case rule. The precondition of a case statement is a disjunction of the preconditions for the two alternatives, with access to the value of the branching condition c within the two summands provided by the $c?$ construct introduced in Section 5. Soundness of the rule is proved by Lemma 6.

Iterated case rule. The full itcase construction, as defined in (3), is actually a sequential composition of two programs. For simplicity, we treat only the special case $p = \mathsf{ret}\, x$ in the Hoare calculus, which we abbreviate as

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

In contrast to the case of **while** loops [7], we cannot expect that after the loop, the value of the test term c contains a right injection, as c could be affected by the program r, which is executed after the loop terminates. The intuition behind the formulation of the **(itcase)** rule is the following: The assertion ψ plays the role of the loop invariant. In every iteration, the loop body q, which depends on a, is executed with the value of a extracted from c, and possibly changes the value of c. As a postcondition of q, we thus require that either the loop continues and the invariant ψ still holds for the next iteration, i.e. $\mathsf{do}\ a \leftarrow c?; \psi$ holds, or the loop terminates and the precondition of r holds, with b replaced by the value contained in c. As the loop terminates with an execution of r, the postcondition of the loop obtained in the conclusion of the rule is then just that of r.

$$\textbf{(basic)}\ \frac{\{\phi\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}}{\{\phi[t/z]\}\ x \leftarrow f(t)\ \{\psi \mid \varepsilon\}} \qquad (\bot)\ \frac{}{\{\top\}\ \bot\ \{\bot \mid \lambda e.\, \bot\}}$$

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

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

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

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

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

Fig. 3. Hoare calculus for order-enriched effects with exceptions.

Theorem 8 (Soundness). *The Hoare calculus for order-enriched effects with exceptions is sound, i.e.*

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

The main idea in Cook's original proof of relative completeness of ordinary Hoare logic is a calculus of *weakest liberal preconditions* or shorter *weakest preconditions* [5], that is (adapted to our setting), for every program p, postcondition ψ and abnormal postcondition ε, a precondition $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ such that

$$\{\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\} \tag{4}$$

is a valid Hoare quadruple and $\phi \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ whenever $\{\phi\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}$. If we can prove that all quadruples (4) are derivable in the calculus then relative completeness follows by soundness and **(wk)**.

Semantically, we construct weakest preconditions in the obvious way by

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

The join exists because every $\mathsf{Hom}(X, \Omega)$ is a complete lattice. The definition of a weakest precondition $\mathsf{wp}(x \leftarrow p, \psi)$ for $\mathbb{T}$ is recovered by taking $E = 0$. By the continuity requirements for order-enriched monads, we have

Lemma 9. *For all programs p and postconditions $\psi \mid \varepsilon$, $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ is a weakest precondition.*

It remains to show that weakest preconditions are definable in our assertion language under the assumption that the weakest preconditions of all signature symbols exist. To this end we give an inductive definition of *syntactic* weakest preconditions $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$, extending the corresponding calculus for the exception-free case [7], and prove that the definition of wp_s is sound w.r.t. wp. This requires a suitable restriction imposed on the underlying predicated monad, which we first recall for the exception-free case:

Definition 10 (Sequential Compatibility [7]). We call a predicated monad $(\mathbb{T}, \mathbb{P})$ *sequentially compatible* if for all programs p, q and assertions ψ, we have

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi) \sqsubseteq \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi)). \tag{5}$$

Remark 11. The inequality converse to (5) holds automatically, and therefore for sequentially compatible predicated monads

$$\mathsf{wp}(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi) = \mathsf{wp}(y \leftarrow p, \mathsf{wp}(x \leftarrow q, \psi)).$$

An equivalent formulation, called the *interpolation property*, is used by Bloom and Ésik [3]; it can be reformulated in our terms as follows: for every valid Hoare triple $\{\phi\}\ x \leftarrow (\mathsf{do}\ y \leftarrow p; q)\ \{\psi\}$ there exists ξ such that $\{\phi\}\ y \leftarrow p\ \{\xi\}$ and $\{\xi\}\ x \leftarrow q\ \{\psi\}$ (w.l.o.g. we can take ξ to be $\mathsf{wp}(x \leftarrow q, \psi)$).

Given $\phi : X \to \Omega$, we can form $\phi^\circ(p) = \mathsf{wp}(x \leftarrow p, \phi(x))$, which yields an operator $(\text{–})^\circ : \mathsf{Hom}(X, \Omega) \to \mathsf{Hom}(TX, \Omega)$. A predicated monad $(\mathbb{T}, \mathbb{P})$ is sequentially compatible iff the right of the following two conditions is satisfied (the left one is satisfied automatically) for all $\phi : Y \to \Omega$, $f : X \to TY$:

$$\phi^\circ \circ \eta = \phi, \qquad \phi^\circ \circ f^\star = (\phi^\circ \circ f)^\circ. \tag{6}$$

It follows that $(\text{–})^\circ$ is natural in the domain of its argument, for $\phi^\circ \circ (Tf) = \phi^\circ \circ (\eta \circ f)^\star = (\phi^\circ \circ \eta \circ f)^\circ = (\phi \circ f)^\circ$, and therefore, by the Yoneda lemma, $(\text{–})^\circ$ is uniquely induced by a map $\Box : T\Omega \to \Omega$. The axioms (6) then amount to saying that $(\Box, \Omega)$ is a $\mathbb{T}$-algebra. This relates sequential compatibility, introduced in [7], to later work by Hasuo [8] who called such an algebra a *modality*, under the running assumption that $\mathbb{T} = \mathbb{P}$, which we do not assume in general.

The inductive definition of syntactic weakest preconditions is given in Figure 4.

Sequential compatibility of $(\mathbb{T}, \mathbb{P})$ generalizes to $(\mathbb{T}_E, \mathbb{P})$:

Lemma 12. *Given a sequentially compatible monad $(\mathbb{T}, \mathbb{P})$, we have*

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

To understand the definition of wp_s for catch, first note that catch itself never terminates abnormally; by unfolding the definitions of Hoare quadruples and catch, we thus have that $\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi \mid \varepsilon\}$ is equivalent to the Hoare triple $\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\}$. We can now eliminate catch by decomposing ψ into a case

$$\mathsf{wp}_s(x \leftarrow f(t), \psi \mid \varepsilon) = \mathsf{wp}(x \leftarrow f(z), \psi \mid \varepsilon)[t/z] \tag{7}$$
$$\mathsf{wp}_s(x \leftarrow \mathsf{ret}\, t, \psi \mid \varepsilon) = \psi[t/x] \tag{8}$$
$$\mathsf{wp}_s(x \leftarrow \bot, \psi \mid \varepsilon) = \top \tag{9}$$
$$\mathsf{wp}_s(y \leftarrow (\mathsf{do}\ x \leftarrow p; q), \psi \mid \varepsilon) = \mathsf{wp}_s(x \leftarrow p, \mathsf{wp}_s(y \leftarrow q, \psi \mid \varepsilon) \mid \varepsilon) \tag{10}$$
$$\begin{aligned}&\mathsf{wp}_s(x \leftarrow (\mathsf{case}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \\ &\quad \mathsf{do}\ a \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, \psi \mid \varepsilon) \vee \mathsf{do}\ a \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\end{aligned} \tag{11}$$
$$\begin{aligned}&\mathsf{wp}_s(x \leftarrow (\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r), \psi \mid \varepsilon) = \\ &\quad \big(\nu X.\, \lambda x.\ \mathsf{do}\ a\ \leftarrow c?; \mathsf{wp}_s(x \leftarrow q, X(x) \mid \varepsilon) \vee \\ &\qquad\qquad \mathsf{do}\ b\ \leftarrow \bar{c}?; \mathsf{wp}_s(x \leftarrow r, \psi \mid \varepsilon)\big)(x)\end{aligned} \tag{12}$$
$$\mathsf{wp}_s(x \leftarrow \mathsf{raise}\, e, \psi \mid \varepsilon) = \epsilon(e) \tag{13}$$
$$\mathsf{wp}_s(y \leftarrow \mathsf{catch}\, p, \psi \mid \varepsilon) = \mathsf{wp}_s(x \leftarrow p, \psi[\mathsf{inl}\, x/y] \mid \lambda e.\psi[\mathsf{inr}\, e/y]) \tag{14}$$

Fig. 4. Syntactic definition of weakest preconditions.

construction and applying the definition of Hoare quadruple:

$$\begin{aligned}&\{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\psi\} \\ &\Leftrightarrow \{\phi\}\ y \leftarrow \mathsf{catch}\, p\ \{\mathsf{case}\, y\, \mathsf{of}\, \mathsf{inl}\, x \mapsto \psi[\mathsf{inl}\, x/y];\ \mathsf{inr}\, e \mapsto \psi[\mathsf{inr}\, e/y]\} \qquad (15)\\ &\Leftrightarrow \{\phi\}\ x \leftarrow p\ \{\psi[\mathsf{inl}\, x/y] \mid \lambda e.\, \psi[\mathsf{inr}\, e/y]\}, \qquad (16)\end{aligned}$$

immediately leading to the definition of $\mathsf{wp}_s(y \leftarrow \mathsf{catch}\, p, \psi \mid \epsilon)$ shown in Figure 4.

For basic programs $f \in \Sigma$, we need to assume that weakest preconditions $\mathsf{wp}(x \leftarrow f(z), \psi)$ for every ψ are expressible in the assertion language, and then include $\{\mathsf{wp}(x \leftarrow f(z), \psi)\}\ x \leftarrow f(z)\ \{\psi \mid \varepsilon\}$ as an axiom in Δ.

Lemma 13. *Given a sequentially compatible predicated monad, we have for all programs p and postconditions ψ*

$$\Delta \vdash_{\mathbb{P}} \{\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)\}\ x \leftarrow p\ \{\psi \mid \varepsilon\}.$$

Proof (Sketch). The proof is via induction over p. We carry out the interesting cases of the inductive step in detail:

Case $p = \mathsf{catch}\, q$. Just note that the rule **(catch)** essentially connects (15) and (16) in the above chain of equivalences explaining wp_s for catch, decorated with an (irrelevant) abnormal postcondition.

Case $p = \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r$. Let ξ denote the right-hand side of (12). By the induction hypothesis, $\{\mathsf{wp}_s(x \leftarrow q, \xi \mid \varepsilon)\}\ x \leftarrow q\ \{\xi \mid \varepsilon\}$ is derivable. This quadruple is equivalent to

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

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

By Lemma 13 and soundness, $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ for all p, ψ, ϵ. Conversely, we have

Lemma 14. *For all p, ψ, ϵ, $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon) \sqsubseteq \mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$.*

Proof (Sketch). Induction over p. Again, we only detail the interesting cases:

Case $p = \mathsf{itercase}\, c\, \mathsf{of}\, \mathsf{inl}\, a \mapsto q;\ \mathsf{inr}\, b \mapsto r$. Let ξ denote the right-hand side of (12). Following [7], we need to prove that $\phi_0 := \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ is a postfixed point of the functional defining ξ.

By the definition of weakest preconditions, 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.

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

Therefore, by case distinction, we need to show that

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

– *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

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

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

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

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

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

Case $p = \mathsf{catch}\, q$. As seen in the derivation 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, $\phi \sqsubseteq \mathsf{wp}(x \leftarrow q, \psi[\mathsf{inl}\, x/y] \mid \lambda e.\, \psi[\mathsf{inr}\, e/y])$ which is smaller than $\mathsf{wp}_s(x \leftarrow p, \psi \mid \varepsilon)$ by induction.

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

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

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

We are now ready to prove our main result:

Theorem 15 (Relative completeness). *Let* $(\mathbb{T}_E, \mathbb{P})$ *be a sequentially compatible predicated monad and let the weakest preconditions for basic programs be expressible in the assertion language. Then*

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

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

Proof. By Lemma 13 and Lemma 14, we can express the weakest precondition $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$ in the assertion language, and

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

By the definition of $\mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$, we thus have $\phi \sqsubseteq \mathsf{wp}(x \leftarrow p, \psi \mid \varepsilon)$, so we can use **(wk)** to derive the required quadruple. □

7 Conclusion and Further Work

We have extended a previous monad-based generic Hoare calculus [7] to cover computations raising exceptions, modelled in terms of the exception monad transformer [4]. To this end, we have added *abnormal postconditions* [9] to the system, which govern poststates reached by the program in case it terminates with an exception. Our framework is based on *order-enriched monads*, which provide the requisite semantic infrastructure to support loops while avoiding the assumption that the underlying *category of data types* is enriched over complete partial orders; consequently, our generic logic applies also to monads that live on categories with less structure, in particular the category of sets. Moreover, we equip the underlying monad with the choice of a submonad of *innocent* computations causing only restricted side-effects; this choice induces an object of truth values, in the shape of the type of innocent computations of unit type, that supports a form of geometric logic in which we phrase our assertion language.

We have proved soundness and relative completeness, the latter by giving a calculus of weakest (liberal) preconditions, which turn out to be expressible in spite of the fact that the assertion language is quite weak. Here, a key role is played by fixpoint constructs in the assertion language.

Exceptions can be regarded as a very simple case of *uninterpreted operations*; in future research, we will explore the possibility of extending the framework to cover more general uninterpreted operations, such as process-algebraic action prefixing. Moreover, we will investigate a more general setup where loops are interpreted by axiomatic iteration in the spirit of *complete Elgot monads* [1,6], thus in particular covering also coinductive resumption monads [14].

References

1. Adámek, J., Milius, S., Velebil, J.: Equational properties of iterative monads. Information and Computation 208(12), 1306 – 1348 (2010)
2. Barr, M., Wells, C. (eds.): Category theory for computing science, 2nd ed. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1995)
3. Bloom, S.L., Ésik, Z.: Iteration theories: the equational logic of iterative processes. Springer-Verlag New York, Inc. (1993)
4. Cenciarelli, P., Moggi, E.: A syntactic approach to modularity in denotational semantics. In: Category Theory and Computer Science, CTCS 1993 (1993)
5. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM 18(8), 453–457 (Aug 1975)
6. Goncharov, S., Rauch, C., Schröder, L.: Unguarded recursion on coinductive resumptions. In: Mathematical Foundations of Programming Semantics, MFPS 2015. ENTCS (2015)
7. Goncharov, S., Schröder, L.: A relatively complete Hoare logic for order-enriched effects. In: Logic in Computer Science, LICS 2013. pp. 273–282. IEEE (2013)
8. Hasuo, I.: Generic weakest precondition semantics from monads enriched with order. Theoretical Computer Science 604, 2 – 29 (2015)
9. Huisman, M., Jacobs, B.: Java program verification via a Hoare logic with abrupt termination. In: Fundamental Approaches to Software Engineering, FASE 2000. LNCS, vol. 1783, pp. 284–303. Springer (2000)
10. Moggi, E.: A modular approach to denotational semantics. In: Category Theory and Computer Science, CTCS 1991. LNCS, vol. 530, pp. 138–139. Springer (1991)
11. Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)
12. Nordio, M., Calcagno, C., Müller, P., Meyer, B.: A sound and complete program logic for Eiffel. In: Objects, Components, Models and Patterns, TOOLS EUROPE 2009. Lect. Notes Business Inf. Proc., vol. 33, pp. 195–214. Springer (2009)
13. von Oheimb, D.: Hoare logic for Java in Isabelle/HOL. Concurrency and Computation: Practice and Experience 13, 1173–1214 (2001)
14. Piróg, M., Gibbons, J.: The coinductive resumption monad. In: Mathematical Foundations of Programming Semantics, MFPS 2014. ENTCS, vol. 308, pp. 273–288 (2014)
15. Plotkin, G., Power, J.: Algebraic operations and generic effects. Appl. Cat. Struct. 11, 69–94 (2003)
16. Poetzsch-Heffter, A., Rauch, N.: Soundness and relative completeness of a programming logic for a sequential Java subset. Tech. rep., TU Kaiserlautern (2004)
17. Schröder, L., Mossakowski, T.: Monad-independent Hoare logic in HasCASL. In: Fundamental Aspects of Software Engineering, FASE 2003. LNCS, vol. 2621, pp. 261–277 (2003)
18. Schröder, L., Mossakowski, T.: Generic exception handling and the Java monad. In: Algebraic Methodology and Software Technology, AMAST 2004. LNCS, vol. 3116, pp. 443–459. Springer (2004)
19. Simpson, A.K.: Recursive types in Kleisli categories. Tech. rep., MFPS Tutorial, April 2007 (1992)
20. Vickers, S.: Topology via Logic. Cambridge University Press (1989)
21. Wadler, P.: How to declare an imperative. ACM Comput. Surveys 29, 240–263 (1997)