A Relatively Complete Generic Hoare Logic for Order-Enriched Effects

Sergey Goncharov and Lutz Schröder
Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg
Email: {Sergey.Goncharov, Lutz.Schroeder}@cs.fau.de

Abstract—**Monads are the basis of a well-established method of encapsulating side-effects in semantics and programming. There have been a number of proposals for monadic program logics in the setting of plain monads, while much of the recent work on monadic semantics is concerned with monads on enriched categories, in particular in domain-theoretic settings, which allow for recursive monadic programs. Here, we lay out a definition of order-enriched monad which imposes cpo structure on the monad itself rather than on base category. Starting from the observation that order-enrichment of a monad induces a weak truth-value object, we develop a generic Hoare calculus for monadic side-effecting programs. For this calculus, we prove relative completeness via a calculus of weakest preconditions, which we also relate to strongest postconditions.**

I. Introduction

Side-effects in programming languages come in many shapes and sizes, including e.g. store and heap dependency, I/O, exceptions, and resumptions. As a means of organizing such effects in a uniform manner, thus increasing reusability of semantics, tools, verification logics, and meta-theory, monads have been proposed by Moggi [19].

Almost immediately following the discovery of the monadic programming paradigm, the first monad-based program logics appeared [23], [21]. One recurring question in this context is how to come by truth values and predicates. E.g., one may just assume a hyperdoctrine that determines the predicates, and then, e.g., explicitly impose that predicates in the state monad are state-dependent [23]. Alternatively, one may generate the predicates directly from the monadic structure. The latter approach is pursued in previous work on monad-based Hoare logic and dynamic logic [30], [32], [22], which induces from the underlying monad a canonical notion of predicate that in particular allows for a principled reconstruction of state dependency of predicates in the state monad. However, it still does assume that the truth values are provided by the underlying category, and hence do not relate to the structure of the monad. In the present work, we take this program one step further and generate the truth values from the monad itself.

To this end, we need to impose additional domain-theoretic structure on the monad, which however is needed anyway in order to support iteration; we call such monads *order-enriched* (to avoid terms such as 'bounded-complete-dcpo-enriched'). We emphasize that this structure lives on the monad, not on the underlying category $\mathbf{C}$, which makes our approach applicable, e.g., in the following principal cases.

– $\mathbf{C}$ is the category $\mathbf{Set}$ of sets. In this case, we cover largely the standard examples of effects as long as they account for non-termination (e.g. while the total state monad $TX = S \to (S \times X)$ is not order-enriched, the partial state monad $TX = S \rightharpoonup (S \times X)$ is).

– $\mathbf{C}$ is a category of predomains (e.g. bottomless bounded-complete dcpo's); then a monad on $\mathbf{C}$ is order-enriched if computational types accommodate bottom elements and binding respects existing finite joins on the left.

– $\mathbf{C}$ is a category of presheaves such as $\mathbf{C} = [I, \mathbf{Set}]$ where I is the category of finite sets and injections; one monad of interest here is the *local state monad* [25], which we modify to allow for partiality (and hence order-enrichment):

$$(TX)n = Sn \rightharpoonup \int^{m \in n/I} Sm \times Xm$$

where $S : I^{op} \to \mathbf{Set}$ with $Sn = V^n$ and the integral denotes a so-called coend (the intuitive meaning of the integral here is a sum over $m \supseteq n$ of pairs representing the new state and the output parametrized by the set m of allocated locations; each component of this sum corresponds to allocating $m - n$ new memory cells).

Our notion of predicate is then derived from the underlying order-enriched monad in a uniform manner using a generic notion of *innocence* of programs, a weakening of the generic notion of purity introduced in [31] — informally, an innocent program is a deterministic program that reads but does not write and, unlike a pure program, may fail to terminate. These notions are defined as (in)equational properties in the spirit of [8]; one of our core results characterizes the innocent monads on $\mathbf{Set}$ as submonads of reader monads. Truth values are then simply innocent programs of unit type, thus generalizing the tests of KAT (Kleene algebra with tests) [17]; as such, they have nothing to do with a truth value object that may be present in the base category, and e.g. may a priori be intuitionistic even on $\mathbf{Set}$. In fact, the truth value object need not in general be a Heyting algebra, and our approach works also over domain-like base categories which (due to non-monotonicity of negation) do not support internal Heyting-algebra objects. Based on these concepts, we develop a monadic Hoare calculus which — unlike previous complete monadic program logics [21], [10], [22] — supports loops.

For this calculus, we prove a relative completeness theorem in spirit of Cook [5]. Like in the classical case our proof rests on the concept of *weakest (liberal) preconditions*. Since these are related to dynamic logic, it is clear from previous work [31] that not all monads will provide sufficient support for them (one counterexample being the continuation monad). We show that the technique of weakest preconditions applies, in a precise sense, exactly to those monads that satisfy a mild technical requirement called *sequential compatibility* concerning compatibility of weakest preconditions with monadic binding.

In a closing observation, we relate weakest preconditions to strongest postconditions. It turns out that while weakest preconditions exist in general, strongest postconditions may fail to exist; in natural examples, the existence of strongest postconditions actually characterizes computational feasibility properties of a program such as allocating or writing to only finitely many locations in each execution.

We recall the basics of monad-based side-effects in Section II, and introduce our notions of order-enriched monads and innocence in Sections III and IV. Next, we define an imperative metalanguage with loops (Section V) as the setting for our Hoare calculus (Section VI), which we prove relatively complete via weakest preconditions in Section VII. Finally, we study strongest postconditions in Section VIII.

II. Monads for Computations

We briefly recall the definition of a strong monad over a (locally small) Cartesian category $\mathbf{C}$ (i.e. category with finite products), which we fix throughout. When using monads as models of effects, it is customary to present a monad $\mathbb{T}$ as a *Kleisli triple* $(T, \eta, \text{-}^\dagger)$ consisting of an endomap T over $\operatorname{Ob}\mathbf{C}$ sending an object A to an object TA of *A-valued computations*, a family of morphisms $\eta_A : A \to TA$, and a *Kleisli star* operator that assigns to every morphism $f : A \to TB$ a morphism $f^\dagger : TA \to TB$ lifting f from A to computations over A. One typical example (of many) is the *state monad*, which has $TA = S \to (A \times S)$ for a fixed set S of states, so that morphisms $A \to TB$ may be seen as abstract state-dependent programs with input from A and output in B; we defer further examples to Section III. This data are subject to the equations

$$\eta^\dagger = \mathrm{id} \qquad f^\dagger\eta = f \qquad (f^\dagger g)^\dagger = f^\dagger g^\dagger$$

(for $g : C \to TA$), which ensure that the *Kleisli category* $\mathbf{C}_\mathbb{T}$ of $\mathbb{T}$, i.e. the category that has the same objects as $\mathbf{C}$ and $\mathbf{C}$-morphisms $A \to TB$ as morphisms $A \to B$, is actually a category, with identities η_A and composition $(f, g) \mapsto f^\dagger g$. It is easy to check that this presentation is equivalent to the otherwise more standard one via an endofunctor T and natural transformations $\eta : \mathrm{Id} \to T$ (*unit*) and $\mu : TT \to T$ (*multiplication*). We generally use blackboard capitals $\mathbb{T}$,... to refer to monads and the corresponding Romans T,... to refer to their functorial parts.

A monad $\mathbb{T}$ is *strong* if it comes with a natural transformation $\tau_{A,B} : A \times TB \to T(A \times B)$ called its *strength*, satisfying a number of coherence conditions [20]. Strong monads are precisely those which support programming in terms of more than one variable, and hence are arguably the only computationally relevant ones.

Reasoning about strong monads is considerably facilitated by using Moggi's *computational metalanguage* [20]. Important language features are the ret operator, which just denotes the monad unit η, and (using Haskell-style do in place of Moggi's let) a binding construct $\mathsf{do}\ x \leftarrow p; q$ which denotes Kleisli composition of $\lambda x.\, q$ with p, with the context propagated using the strength. Intuitively, $\mathsf{ret}\, x$ just returns x as a value without causing side-effects, and $\mathsf{do}\ x \leftarrow p; q$ executes p, binds the result of the computation to x, and then executes q (which may depend on x). We will give a full definition of an extended metalanguage in Section V; for now, we only note that the language is multisorted and as such involves *typed terms in contexts* $\Gamma \rhd p : A$ where Γ, the *context*, is a sequence of pairs $x_i : A_i$ presenting a variable that may appear in p and its type, mentioning every variable at most once. As usual, contexts are concatenated using comma-separated juxtaposition. We will refer to p as a *program* and to A as its *return type*. The contexts and the return types are commonly omitted. In terms of the metalanguage the monad laws can be rewritten as

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

A *monad morphism* is a natural transformation between the underlying monad functors satisfying obvious coherence conditions w.r.t. the unit, the Kleisli star (equivalently, multiplication) and the strength, see e.g. [2] for details. A *submonad* of a monad $\mathbb{T}$ is a monad $\mathbb{S}$ together with a componentwise monic monad morphism $\mathbb{S} \to \mathbb{T}$ called the *inclusion morphism*.

An important issue in this framework is the *construction* of effects. Well-behaved constructions tend to be given in terms of *algebraic operations* as identified by Plotkin and Power [26]:

Definition 1 (Algebraic operation)**.** Given $n \in \mathbb{N}$ and a monad $\mathbb{T}$ over $\mathbf{C}$, a natural transformation $\alpha_X : (TX)^n \to TX$ is an *(n-ary) algebraic operation* if

– for every $f \in \mathbf{C}(A, TB)$, $\alpha_B(f^\dagger)^n = f^\dagger\alpha_A$ and

– for every $A, B \in \operatorname{Ob}\mathbf{C}$, $\tau(\mathrm{id} \times \alpha_B) = \alpha_{A\times B}\tau^n\vartheta_n$

where $\vartheta_n : A \times (TB)^n \to (A \times TB)^n$ is the morphism whose i-th component is $\mathrm{id} \times \pi_i : A \times (TB)^n \to A \times TB$.

Thus, an algebraic operation combines several computations into one in a coherent way. Examples of algebraic operations include exception raising ($T^0 \to T$), binary nondeterministic choice ($T^2 \to T$), reading a value ranging over $\{0, \dots, n-1\}$ ($T^n \to T$) and writing some value from/to a memory location ($T^1 \to T$).

III. Order-Enrichment

The notion of monadic side-effect recalled in the previous section does not provide enough structure to cope with recursion or loops. Conditions ensuring definability of fixpoint operators that effectively impose additional domain-like structure on the base category have been discussed previously [6], [33], [31]. Here, we pursue the alternative approach to enrich only the monad itself over suitable complete partial orders, thus broadening the range of applicability of our results as indicated in the introduction. We call this type of monads *order-enriched*. It will turn out that order-enrichment induces sufficient logical structure on tests for them to serve as truth values; we will base our generic Hoare logic on this this observation (Section VI).

Definition 2 (Order-enriched monad)**.** A strong monad $\mathbb{T}$ over $\mathbf{C}$ is *order-enriched* if the following conditions are met.

– Every hom-set $\mathsf{Hom}(A, TB)$ carries a partial order, denoted $\sqsubseteq$, that has a bottom element denoted by $\bot_{A,B}$ or simply $\bot$, joins of all directed subsets, and joins of all f, g such that $f \sqsubseteq h$, $g \sqsubseteq h$ for some h (bounded completeness).

– For any $h \in \mathsf{Hom}(A', A)$ and any $u \in \mathsf{Hom}(B, TB')$, the maps

$$f \mapsto f \circ h, \qquad f \mapsto u^\dagger \circ f, \qquad f \mapsto \tau\langle \mathrm{id}, f\rangle \tag{1}$$

preserve all existing joins (including the empty join $\bot$).

– Kleisli star is Scott continuous, i.e. if $\mathcal{F}$ is a nonempty directed subset of $\mathsf{Hom}(A, TB)$, then $\bigsqcup_{f\in\mathcal{F}} f^\dagger = \left(\bigsqcup \mathcal{F}\right)^\dagger$.

Most of these conditions, including bounded completeness, descend from the standard definition of Scott domain. Bounded completeness turns out to be a critical property needed to introduce disjunction for our assertion logic in Section V. The continuity conditions above amount to requiring that expressions $\mathsf{do}\ x \leftarrow p; q$ preserve all existing joins in the left argument p, and directed joins in the right argument q. As usual, we denote the meet of $f, g : A \to TB$ by $f \sqcap g$, the join by $f \sqcup g$, and the top element of $\mathsf{Hom}(A, TB)$ by $\top$ if they exist. Next we summarize the most straightforward properties of order-enriched monads.

Proposition 3. *Let $\mathbb{T}$ be an order-enriched monad on $\mathbf{C}$, and let $A, B \in \mathrm{Ob}\,(\mathbf{C})$. Then*

1) *every nonempty subset of $\mathsf{Hom}(A, TB)$ having an upper bound has a least upper bound;*
2) *every nonempty subset of $\mathsf{Hom}(A, TB)$ has a greatest lower bound;*
3) *if $\mathsf{Hom}(A, TB)$ has a top element, then it is a complete lattice.*

It is now apparent that the type of partial orders we involve is obtained from the definition of Scott domain by dropping the algebraicity condition, i.e., essentially, bounded-complete dcpo's. Let $\mathbf{bdCpo}_\bot$ be the category of such partial orders with Scott continuous functions as morphisms. This category is monoidal w.r.t. the standard Cartesian structure induced by the evident forgetful functor to $\mathbf{Set}$. We would like to relate our definition of order-enrichment with the standard notion of enrichment [15]. To that end we first recall a characterization of algebraic operations essentially proved in [26].

Proposition 4. *To give an algebraic operation $T^n \to T$ is the same as to give a $\mathbf{C}^{op} \times \mathbf{C}$-indexed family of maps $\gamma_{A,B}$: $\mathsf{Hom}(A, TB)^n \to \mathsf{Hom}(A, TB)$ satisfying the conditions*

$$\gamma_{A,B}(f_1, \ldots, f_n) \circ h = \gamma_{A',B}(f_1 \circ h, \ldots, f_n \circ h) \tag{2}$$

$$u^\dagger \circ \gamma_{A,B}(f_1, \ldots, f_n) = \gamma_{A,B}(u^\dagger \circ f_1, \ldots, u^\dagger \circ f_n) \tag{3}$$

$$\tau\langle \mathrm{id}, \gamma_{A,B}(f_1, \ldots, f_n)\rangle = \gamma_{A,B}(\tau\langle \mathrm{id}, f_1\rangle, \ldots, \tau\langle \mathrm{id}, f_n\rangle) \tag{4}$$

for any $h \in \mathsf{Hom}(A', A)$ and any $u \in \mathsf{Hom}(B, TB')$.

This suggests the following definition of partial algebraic operation, intended to deal with the fact that join is a partial operation in bounded-complete dcpo's.

Definition 5 (Partial algebraic operation)**.** A *partial algebraic operation* is given by a collection of partial maps of the form $\gamma_{A,B} : \mathsf{Hom}(A, TB)^n \to \mathsf{Hom}(A, TB)$ satisfying the equations (2)–(4) in the sense that whenever the left-hand side of an equation exists then so does the right-hand side, and both sides are equal.

Proposition 6. *A monad $\mathbb{T}$ is order-enriched iff its Kleisli category $\mathbf{C}_\mathbb{T}$ is a $\mathbf{bdCpo}_\bot$-category, strength is Scott continuous in the second argument, and the finite joins (including $\bot$) induced by the order-enrichment are partial algebraic.*

There are two ways of adapting the standard examples of computational monads to the enriched settings: by shifting to an order-enriched base category $\mathbf{C}$ or by modifying the monad. The first approach is embodied in the following proposition. Let $\mathbf{bdCpo}$ be the category of bottomless bounded-complete dcpo's and Scott continuous maps. Recall that a $\mathbf{bdCpo}$-monad is a monad over a $\mathbf{bdCpo}$-category whose underlying functor is $\mathbf{bdCpo}$-enriched (see e.g. [35]; since $\mathbf{bdCpo}$ is concrete, $\mathbf{bdCpo}$-naturality of the involved natural transforms is automatic). Combining Proposition 6 with [35, Theorem 15(b)], we obtain

Proposition 7. *A $\mathbf{bdCpo}$-monad is order-enriched if the finite joins on its Kleisli hom-sets induced by the $\mathbf{bdCpo}$-enrichment are partial algebraic and have algebraic bottom elements.*

Alternatively one can enrich only the monad functor and not the underlying category, hence allowing, in particular, for $\mathbf{C} = \mathbf{Set}$, which is not $\mathbf{bdCpo}$-enriched.

Example 8 (Order-enriched monad)**.** Most of the standard examples of computational monads on $\mathbf{Set}$ (e.g. exceptions, state, I/O, see [19]) become order-enriched as soon as we allow for explicit non-termination. We look explicitly at variants of the partial state monad, all of which are order-enriched:

1. The partial state monad on $\mathbf{Set}$, with functorial part $TA = S \rightharpoonup (A \times S)$ where S is a fixed set of states and $\rightharpoonup$ denotes partial function space, is order-enriched when equipped with the extension ordering. (Contrastingly, the total state monad $A \mapsto (S \to (A \times S))$ fails to be order-enriched, as it does not have a bottom element.)

2. The *non-deterministic state monad*, with functorial part $TA = S \to \mathcal{P}(A \times S)$, is order-enriched.

3. Applying the exception monad transformer, which maps a monad T to the monad $T(- + E)$ for a fixed set E of exception, to the partial state monad yields the so-called *Java monad* [14]. This monad is order-enriched and as such yields an example of binding not preserving $\bot$ on the right ($\mathsf{do}\ x \leftarrow \mathit{raise}\ e; \bot =$

raise $e \neq \bot$ where *raise* e stands for raising an exception $e \in E$).

4. Similarly, preservation of binary joins in q by $\mathsf{do}\; x \leftarrow p; q$ fails for non-deterministic monads featuring input or resumptions [13], [3], [11], essentially for the same reason that causes the well-known failure of non-deterministic choice to commute with sequential composition from the left for parallel processes modulo bisimulation. Again, these monads do admit order-enrichment.

5. We call the partial state monad $TA = S \to (A \times S)_\bot$ on $\mathbf{bdCpo}$ the *domain-theoretic state monad*. A typical case is $S = L \to V_\bot$ where L and V are (possibly infinite) discrete domains of values and locations, respectively. In this case, Scott continuity of a state transformer f means essentially that the output and the value of $f(s)$ at any given location depend on the values of only finitely many locations under $s : L \to V_\bot$.

6. The *topological state monad* on $\mathbf{Set}$ (!) has a topological state space S, and TA consists of the continuous partial maps $S \rightharpoonup (A \times S)$ with open domain, where A carries the discrete topology. Equivalently, $TA = S \to (A \times S)_\bot$, where for any space Y, the space $Y_\bot = Y \cup \{\bot\}$ carries the topology generated by the topology of Y (i.e. its opens are $Y_\bot$ and the opens of Y). One example of interest is $S = L \to V$ (total function space!) for sets L and V — this set does not carry any natural non-trivial domain structure, but it does carry an interesting topology, the product topology of L copies of the discrete space V. Under this topology, continuity of $f \in TA$ means, again, that the value of a location under $f(s)$ and the output depend only on the values of finitely many locations under s.

7. The *partial local state monad* from the introduction is order-enriched under the obvious extension ordering.

IV. Innocence

Following [29] we next introduce an appropriate notion of (comparatively) well-behaved computations. To begin, we recall the notion of *commutative monad* [16] as based on a requirement of commutation of programs.

Definition 9 (Commutation)**.** Two programs p and q *commute* if the equation

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

holds, where $x, y \notin \mathrm{Vars}(p) \cup \mathrm{Vars}(q)$. If this is true for all p, q then the monad at hand is *commutative*.

The core notion of innocence is then defined as follows.

Definition 10 (Innocence)**.** Given an order-enriched monad $\mathbb{T}$,

- a program p is *copyable* if it satisfies the equation $\mathsf{do}\; x \leftarrow p; y \leftarrow p; \mathsf{ret}\langle x, y\rangle = \mathsf{do}\; x \leftarrow p; \mathsf{ret}\langle x, x\rangle$;
- a program p is *weakly discardable* if it satisfies the inequality $\mathsf{do}\; y \leftarrow p; \mathsf{ret}\star \sqsubseteq \mathsf{ret}\star$;
- $\mathbb{T}$ is *innocent* if it is commutative and all programs in it are copyable and weakly discardable.

The conditions defining innocent monads are slightly less restrictive than those defining *pure computations*, used for similar purposes in [29]. Specifically, a monad is pure if it is innocent and satisfies *discardability*, i.e. for every p, $\mathsf{do}\; y \leftarrow p; \mathsf{ret}\star = \mathsf{ret}\star$.

Intuitively, weakly discardable programs may read but not write the state, and innocent programs are additionally deterministic (due to copyability). Computationally interesting monads typically fail to be innocent but have natural innocent submonads; this situation is illustrated in Example 12. For brevity, we fix the following terminology.

Definition 11 (Predicated Monad)**.** A *predicated monad* is a pair $(\mathbb{T}, \mathbb{P})$ consisting of an order-enriched monad $\mathbb{T}$ and an innocent submonad $\mathbb{P}$ of $\mathbb{T}$.

Example 12 (Innocent/Predicated Monads)**.**

– The partiality monad $\mathbb{P}$ (where $PA = A + 1$) can be mapped into any order-enriched monad $\mathbb{T}$ by $\alpha_A(\mathsf{inl}(x)) = \eta_\mathbb{T}(x)$, $\alpha_A(\mathsf{inr}\star) = \bot$; if η is componentwise monic, then this makes $\mathbb{P}$ an innocent submonad of $\mathbb{T}$. If $\mathbf{C} = \mathbf{Set}$ and $\mathbb{T}$ is *free*, i.e. $TA = \mu\gamma.\, F(\gamma + A)$ for some functor F, then the partiality monad is the only innocent submonad of $\mathbb{T}$. This covers the case of exceptions, input, output and resumptions [19].

– If $\mathbb{T}$ is a partial or non-deterministic state monad (Example 8) or, e.g., the Java monad of [14] ($TA = S \rightharpoonup S \times A + E \times A$), then the partial reader monad ($PA = S \rightharpoonup A$) is an innocent submonad of $\mathbb{T}$.

– If $\mathbb{T}$ is the partial local state monad from the introduction, then the corresponding partial local reader monad whose functorial part is given as

$$(PA)n = Sn \rightharpoonup An$$

is an innocent submonad of $\mathbb{T}$.

Next, we establish that an innocent monad $\mathbb{P}$ yields a truth value object in a weak sense, $P1$, on which we will base our assertion language. Recall that a *frame* is a complete lattice in which finite meets distribute over all joins, with *frame homomorphisms* preserving finite meets and all joins.

Lemma 13. *Let $\mathbb{P}$ be an innocent monad. Then $p \sqcap q = \mathsf{do}\; p; q = \mathsf{do}\; q; p$ for $\Gamma \rhd p, q : P1$.*

Theorem 14. *Let $\mathbb{P}$ be an innocent monad. Then $P1$ is a distributive lattice object in $\mathbf{C}$ 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.*

We emphasize that $P1$ need neither be internally complete nor residuated, i.e. does not in general support implication and quantifiers — e.g. in categories of domains, there will be no sensible objects supporting implication. It is one of the contributions of this work to show that despite its weakness, the arising logic does support a relatively complete Hoare logic, i.e. a weakest precondition calculus. When $P1$ has additional logical structure, the weakest precondition calculus will support this structure as well. We do have

Corollary 15. *Let $\mathbb{P}$ be an innocent monad on* $\mathbf{Set}$. *Then $P1$ is a complete Heyting algebra under the monad ordering.*

Example 16. In case $\mathbb{P}$ is the partial local reader monad, it is easy to check that $P1$ is a Boolean algebra (although the ambient internal logic of the presheaf topos $[I, \mathbf{Set}]$ is intuitionistic).

If $P1$ is even a Boolean algebra. we say that $(\mathbb{T}, \mathbb{P})$ is *classical*. However, even on $\mathbf{Set}$, not all monads are classical:

Example 17. The partial state monad has a natural innocent submonad (in fact, the largest such), the *partial reader monad* given by the expression $PA = S \rightharpoonup A$, so that $\Omega = P1$ can be identified with the powerset of S, a Boolean algebra. A simple example where Ω is, in general, non-classical is the topological state monad. Here, $P1$ consists essentially of the continuous functions $S \to 1_\perp$. These are in bijection with the open subsets of S, and hence in general form a proper Heyting algebra. Finally, the largest innocent submonad of the domain-theoretic state monad maps A to the continuous function space $S \to A_\perp$, so that in this case, Ω consists essentially of the Scott open subsets of S, in particular is again in general non-classical (not even a Heyting algebra). These notions of predicate are in accordance with the suggestions of [34].

Examples 17 and 12 intimate that an innocent monad might always be a submonad of some partial reader monad. As the main result of this section, we establish this for ranked monads on $\mathbf{Set}$.

Theorem 18. *Let $\mathbb{P}$ be a ranked innocent monad on* $\mathbf{Set}$. *Then $\mathbb{P}$ is a submonad of a partial reader monad.*

V. A Simple Imperative Metalanguage

Here we consider a first order version of a simple computational metalanguage for side-effecting programs [20], which is also reminiscent of the simple imperative language **Imp** from [27], [36] and therefore called shortly the *imperative meta-language*. Our decision to drop high-order types, analogously to the previous decision to drop the Scott's algebraicity condition for dcpo's is entirely due to their irrelevance for any of our results — as such they would only amount to unnecessary restriction of generality.

Like Moggi's original meta-language, our language is generic in the underlying side-effect and in the choice of basic statements. In a significant step beyond this, it features unbounded loops.

Let $\mathcal{W}$ be a set of basic types. The sets of *value types* A and *types* C are defined by the grammar

$$A ::= W \mid 1 \mid 2 \mid A \times A \qquad C ::= A \mid \Omega \mid TA \mid PA$$

where W ranges over $\mathcal{W}$. The intended reading is that TA and PA are types of computations and innocent computations, respectively, over A; Ω abbreviates $P1$ and serves as a type of truth values. Contrastingly, 2 is the type $1 + 1$ of Booleans. Finally, a *predicate type* has the form $A \to \Omega$ where A is a value type; predicates may be thought of as innocently side-effecting (intuitively: state-dependent) truth-valued functions.

Term formation rules for simple programs are given in the top section of Figure 1. They derive *typed terms in context* $\Gamma \rhd t : A$ as explained in Section II; however, we restrict Γ to contain only variables of value types and predicate types. Application of ret is restricted to terms of value types to ensure well-formedness of the resulting type. We fix a signature Σ of typed functional symbols of the form $f : A \to C$ where A is a value type and C is a type. When C is not a value type, then f is a *basic program*. Typical examples are read and write operations in a store-based state monad as in Example 8, or non-deterministic assignment in a non-deterministic state monad. We assume that Σ contains the usual Boolean operations on 2.

At the same time, we introduce the language of assertions to be used in our verification logics in the bottom section of Figure 1. An *assertion* is a program of return type Ω. Note that the identification of Ω with $P1$ entails that the **(do)** rule can be applied to show that when $\Gamma \rhd p : PA$ and $\Gamma, x : A \rhd \phi : \Omega$, then $\Gamma \rhd \mathsf{do}\ x \leftarrow p; \phi : \Omega$. We include quantifiers and implication in the language but emphasize these are supported only when $P1$ has sufficient structure, e.g. lives over $\mathbf{Set}$. The intuitive meaning of the iteration constructor $\mathsf{init}\, x \leftarrow p\ \mathsf{while}\ \phi\ \mathsf{do}\ q$ is to initialize the variable x by p and then pass it iteratively though the loop updating it by q (which can itself depend on x) at every iteration.

Our language offers conjunction and disjunction (but not in general implication) as well as universal and existential quantification. Moreover, we allow for *fixpoint predicate constructors* $\mu X. \phi$ and $\nu X. \phi$ where X has a predicate type $A \to \Omega$, omitted in the notation for brevity.

We parametrize the semantics of the simple imperative metalanguage over a predicated monad $(\mathbb{T}, \mathbb{P})$, as well as over an interpretation of basic types and operations. From now on, we assume the underlying category $\mathbf{C}$ to be distributive [4] (but not necessarily Cartesian closed); i.e. $\mathbf{C}$ has binary coproducts, and finite products distribute over finite coproducts. Every basic type W is interpreted as an object $[\![W]\!]$ in $\mathbf{C}$; this interpretation is inductively extended to all types by $[\![1]\!] = 1$, $[\![2]\!] = 1+1$, $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$, $[\![\Omega]\!] = P1$, $[\![TA]\!] = T[\![A]\!]$, $[\![PA]\!] = P[\![A]\!]$. Signature symbols $f : A \to C \in \Sigma$ are interpreted as morphisms $[\![f]\!] : [\![A]\!] \to [\![B]\!]$.

From now on, we fix a predicated monad $(\mathbb{T}, \mathbb{P})$ and an interpretation of Σ as above.

Since we do not assume function objects in the underlying category, the interpretation of predicate variables in contexts requires some care: a context $\Gamma = (x_1 : A_1; \ldots; x_n : A_n; X_1 : B_1 \to \Omega; \ldots; X_k : B_k \to \Omega)$ is interpreted as the pair (C, H) where $C = [\![A_1]\!] \times \cdots \times [\![A_n]\!]$ (a $\mathbf{C}$-object) and $H = \mathsf{Hom}(C \times [\![B_1]\!], \Omega) \times \cdots \times \mathsf{Hom}(C \times [\![B_k]\!], \Omega))$ (a set). Programs and assertions $\Gamma \rhd t : A$ are then interpreted as maps $[\![t]\!] : H \to \mathsf{Hom}(C, [\![A]\!])$, while second order terms $\Gamma \rhd \phi : A \to \Omega$ are interpreted as maps $[\![\phi]\!] : H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$. The interpretation of programs and assertions is largely standard (see e.g. [19], [4], [10]):

– variables, $\star$, pairing, and projections are interpreted using the Cartesian structure of $\mathbf{C}$;

– 0 and 1 are coproduct injections and if-then-else is a

$$\frac{x : A \text{ in } \Gamma}{\Gamma \rhd x : A} \qquad \frac{f : A \to C \in \Sigma \quad \Gamma \rhd t : A}{\Gamma \rhd f(t) : C} \qquad \frac{}{\Gamma \rhd \star : 1} \qquad \frac{\Gamma \rhd t : A \quad \Gamma \rhd u : B}{\Gamma \rhd \langle t, u \rangle : A \times B}$$

$$\frac{\Gamma \rhd t : A \times B}{\Gamma \rhd \mathsf{pr}_1\, t : A} \qquad \frac{\Gamma \rhd t : A \times B}{\Gamma \rhd \mathsf{pr}_2\, t : B} \qquad \frac{}{\Gamma \rhd 0 : 2} \qquad \frac{}{\Gamma \rhd 1 : 2}$$

$$\frac{\Gamma \rhd p : FA \quad \Gamma, x : A \rhd q : FB}{\Gamma \rhd \mathsf{do}\ x \leftarrow p; q : FB}\ (F \in \{P, T\}) \qquad \frac{\Gamma \rhd p : A}{\Gamma \rhd \mathsf{ret}\, p : PA}\ (A \text{ a value type}) \qquad \frac{\Gamma \rhd p : PA}{\Gamma \rhd p : TA}$$

$$\frac{\Gamma \rhd b : 2 \quad \Gamma \rhd s : C \quad \Gamma \rhd t : C}{\Gamma \rhd \mathsf{if}\, b\, \mathsf{then}\, s\, \mathsf{else}\, t : C} \qquad \frac{\Gamma, x : A \rhd b : 2 \quad \Gamma \rhd p : TA \quad \Gamma, x : A \rhd q : TA}{\Gamma \rhd \mathsf{init}\, x \leftarrow p\ \mathsf{while}\ b\ \mathsf{do}\ q : TA}$$

$$\frac{}{\Gamma \rhd a : \Omega}\ (a \in \{\top, \bot\}) \qquad \frac{\Gamma, x : A \rhd \phi : \Omega}{\Gamma \rhd Qx.\, \phi : \Omega}\ (Q \in \{\exists, \forall\}) \qquad \frac{\Gamma, X : A \to \Omega \rhd \phi : A \to \Omega}{\Gamma \rhd \eta X.\, \phi : A \to \Omega}\ (\eta \in \{\mu, \nu\})$$

$$\frac{\Gamma \rhd \phi : \Omega \quad \Gamma \rhd \psi : \Omega}{\Gamma \rhd \phi \odot \psi : \Omega}\ (\odot \in \{\wedge, \vee, \to\}) \qquad \frac{X : A \to \Omega \text{ in } \Gamma}{\Gamma \rhd X : A \to \Omega} \qquad \frac{\Gamma, x : A \rhd t : \Omega}{\Gamma \rhd \lambda x.\, t : A \to \Omega} \qquad \frac{\Gamma \rhd t : A \quad \Gamma \rhd s : A \to \Omega}{\Gamma \rhd s(t) : \Omega}$$

Fig. 1. Term formation rules for the simple imperative metalanguage ($\to, \forall, \exists$ supported only over $\mathbf{Set}$).

case distinction over the summands of $2 = 1 + 1$, interpreted using the distributive structure of $\mathbf{C}$ [4];

– do and ret are interpreted using Kleisli star, strength, and unit of $\mathbb{T}$ and $\mathbb{P}$, respectively [19];

– Boolean connectives and, if present, quantifiers are interpreted using the lattice structure of $[\![\Omega]\!] = P1$ (Theorem 14, Corollary 15).

We treat the semantics of predicate terms and fixpoints in more detail. Let $[\![\Gamma]\!] = (C, H)$ as above.

– A variable $\Gamma \rhd X : A \to \Omega$ is interpreted as a projection map $H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$.

– Let $\Gamma \rhd t : A$, $\Gamma \rhd s : A \to \Omega$. Then $s(t)$ is interpreted as the map $[\![s(t)]\!] : H \to \mathsf{Hom}(C, [\![\Omega]\!])$ obtained from $[\![s]\!] : H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$ and $[\![t]\!] : H \to \mathsf{Hom}(C, [\![A]\!])$ by putting $[\![s(t)]\!](h) = [\![s]\!](h) \circ \langle \mathrm{id}_C, [\![t]\!](h) \rangle$ where $\langle -, - \rangle$ denotes pairing in $\mathbf{C}$.

– Since we do not assume function objects in $\mathbf{C}$ and hence represent predicates $\Gamma \rhd \phi : A \to \Omega$ in uncurried form, i.e. as maps $H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$, λ-abstraction semantically does nothing: the interpretation of $\Gamma, x : A$ is $(C \times [\![A]\!], H)$, so when $\Gamma, x : A \rhd t : \Omega$, then $[\![t]\!]$ is a map $H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$; we take $[\![\lambda x.\, t]\!]$ to be the same map.

– The interpretation of $\Gamma, X : A \to \Omega \rhd \phi : A \to \Omega$ is a map $[\![\phi]\!] : H \times \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!]) \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$. We define the interpretation $[\![\nu X.\, \phi]\!] : H \to \mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$ by taking $[\![\nu X.\, \phi]\!](h)$ to be the greatest fixpoint of the endomap on $\mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$ taking g to $[\![\phi]\!](h, g)$, which exists because by the restrictions on ϕ this map is monotone and $\mathsf{Hom}(C \times [\![A]\!], [\![\Omega]\!])$ is a complete lattice; correspondingly for least fixpoints $\mu X.\, \phi$.

Definition 19. Let $[\![\Gamma]\!] = (C, H)$. An assertion $\Gamma \rhd \phi : \Omega$ is *valid* (in $(\mathbb{T}, \mathbb{P})$) if $[\![\phi]\!](h) = \top$ for all $h \in H$.

Remark 20. Predicate variables have the context as an implicit argument in the semantics. This is necessitated by the absence of function objects in $\mathbf{C}$. Somewhat informally, it is justified by the equality

$$\begin{aligned} &\mu X : A \to \Omega.\, F(c, X) = \\ &\qquad (\mu X : C \times A \to \Omega.\, \lambda d : C.\, F(d, \lambda x : A.\, X(d, x)))(c) \end{aligned}$$

for $c : C$ representing the context (using full higher order syntax for brevity), correspondingly for ν.

The definition of the semantics of the while loop requires some preliminaries. First, we introduce an operator $? : 2 \to \Omega$ by the equation $\phi? = \mathsf{if}\, \phi\, \mathsf{then} \top \mathsf{else} \bot$. It is easy to see that 2 carries a natural Boolean algebra structure, with the complement of $b : 2$ denoted $\bar{b}$.

Lemma 21. *The operator ? defines a homomorphism of distributive lattices.*

Lemma 22. *Let $p, q : TA$. Then the join* $(\mathsf{do}\ b?; p) \sqcup (\mathsf{do}\ \bar{b}?; q)$ *exists and equals* $(\mathsf{if}\, b\, \mathsf{then}\, p\, \mathsf{else}\, q)$.

One consequence of this lemma is that if b then p else q is Scott continuous in p and q. We can therefore define the semantics of $\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{while}\ b\ \mathsf{do}\ q$ (which should be read as an informal way to denote $\mathsf{init}\, y \leftarrow \mathsf{ret}\, x\ \mathsf{while}\ b[y/x]\ \mathsf{do}\ q[y/x]$ for some fresh y) as expected, namely as the least fixpoint of the map $p \mapsto \mathsf{if}\, b\, \mathsf{then}\ \mathsf{do}\ x \leftarrow q; p\ \mathsf{else}\ \mathsf{ret}\, x$ (where we abuse the metasymbol p as a function symbol), which exists by Scott continuity of the expression on the right. Generally:

$$\begin{aligned} &[\![\Gamma \rhd \mathsf{init}\, x \leftarrow p\ \mathsf{while}\ b\ \mathsf{do}\ q : TA]\!] \\ &\quad = [\![\Gamma \rhd \mathsf{do}\ x \leftarrow p; \mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{while}\ b\ \mathsf{do}\ q : TA]\!]. \qquad (5) \end{aligned}$$

We will prefer the notation $(\mathsf{while}\ b\ \mathsf{do}\ x \leftarrow p)$ as a shorthand for $(\mathsf{init}\, x \leftarrow \mathsf{ret}\, x\ \mathsf{while}\ b\ \mathsf{do}\ p)$ in the sequel. As seen in (5), the general form is expressible using the shortened one. The only drawback of the short syntax is that it is not stable under variable substitution, but this will play no particular role below.

Remark 23. Unlike in the classical case where fixpoints are eliminated using Gödel's β-function, we include them in the language. Fixpoints naturally arise in standard verification scenarios, e.g. when reasoning over data structures. For instance, a monad for singly linked lists can be thought of in terms of innocent operations $\mathit{emp}(x)$ (test for emptiness of a reference x), $\mathit{node}(x)$ (data element stored at x), and $\mathit{next}(x)$ (next location stored at x). We want to relate this type of structure to abstract lists, e.g specified by a Lisp-like interface $\mathit{nil} : L$ and $\mathit{cons} : A \times L \to L$ with appropriate axioms. If we have implication and quantification (e.g. Corollary 15), then the type of lists can be represented using the predicate $\mu X. (X(\mathit{nil}) \wedge (\forall a\colon A, l\colon L. (X(l) \Rightarrow X(\mathit{cons}(a, l)))))$. Assuming sufficient syntactic sugar, we may then define the predicate $\mathit{list}(\alpha, x)$, stating that reference x points to a linked list representing the abstract list α, by a least fixpoint:

$$\begin{aligned}\mathit{list}(\alpha, x) = (\mu X. \lambda \alpha, x. \, & \mathit{emp}(x) \wedge \alpha = \mathrm{nil} \vee \\ & \exists a, \beta. \, \alpha = \mathit{cons}(a, \beta) \wedge \mathit{node}(x) = a \wedge \\ & \exists y. \, \mathit{next}(x) = y \wedge X(\beta, y))(\alpha, x).\end{aligned}$$

Similarly, greatest fixpoints provide support for coinductive data, such as streams.

VI. The Hoare Calculus

To support verification of programs in the imperative meta-language, we introduce *Hoare triples* of the form

$$\Gamma \rhd \{\phi\} x \leftarrow p \{\psi\}$$

where $\Gamma \rhd \phi : \Omega$, $\Gamma \rhd p : A$ and $\Gamma, x : A \rhd \psi : \Omega$. As usual we tend to omit contexts. Formally, the semantics is given by the equivalence $\{\phi\}\ x \leftarrow p\ \{\psi\} \iff [x \leftarrow (\mathsf{do}\ \phi; p)]\ \psi$. Here, $[x \leftarrow p]\ \psi$ is a *global evaluation formula* [10] defined by the equivalence

$$[x \leftarrow p]\ \psi \iff \mathsf{do}\ x \leftarrow p; \psi; \mathsf{ret}\, x = p \qquad (6)$$

and saying informally that after every terminating execution of $x \leftarrow p$, ψ holds for the result x in the poststate. We write $\mathbb{T}, \mathbb{P} \models \{\phi\}\ x \leftarrow q\ \{\psi\}$ if the corresponding equation holds in $\mathbb{T}, \mathbb{P}$ under the given interpretation of basic programs (which we elide in the notation).

Note that this treatment contrasts with the definition from [29], [32], which is based on assuming a truth value object in the base category as opposed to extracting an innocent submonad of $\mathbb{T}$. We do however have an analogous calculus of Hoare triples as shown in Figure 2. As expected, the rules are syntax-directed except for **(wk)**. Hoare triples $\{\phi\}\ x \leftarrow f(z)\ \{\psi\}$ for basic programs $f : A \to TB$ cannot be further reduced by the calculus (except by **(wk)**) and are expected to be taken from a suitable axiomatization of f instead.

Syntactic differences with the standard calculus are comparatively minor; they are related to the fact that monadic programs have result values (while programs have only side effects in the classical setup), to the presence of ret, and to the replacement of assignment with general basic programs. The semantics of the calculus, on the other hand, is a broadly generalized version of the classical setup as already discussed. We will elucidate later how the classical calculus can be obtained by instantiation from the generic calculus.

We regard the proof of *inequality assertions* $\phi \sqsubseteq \psi$, needed for application of **(wk)**, as discharged outside the calculus. We thus write $\Delta \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow q\ \{\psi\}$ whenever $\{\phi\}\ x \leftarrow q\ \{\psi\}$ is derivable in the calculus from axioms Δ (necessary to deal with basic programs) and the set of all inequality assertions $\phi \sqsubseteq \psi$ valid in $(\mathbb{T}, \mathbb{P})$, where in context Γ, $\phi \sqsubseteq \psi$ is valid if $[\![\Gamma \rhd \phi]\!] \sqsubseteq [\![\Gamma \rhd \psi]\!]$.

Theorem 24. *The calculus of Figure 2 is sound, i.e.* $\vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow q\ \{\psi\}$ *implies* $\mathbb{T}, \mathbb{P} \models \{\phi\}\ x \leftarrow q\ \{\psi\}$.

Proof: Straightforward; soundness of **(while)** is by fixpoint induction, using the fact that while is a fixpoint of a continuous functional. ■

Remark 25. An alternative encoding of Hoare triples into equational logic is suggested by their treatment in Kleene algebra with tests (KAT) [18]. Adapted to our setting, this would lead to the definition of $[x \leftarrow p]\ \psi$ as

$$\mathsf{do}\ x \leftarrow p; \neg\psi; \mathsf{ret}\, x = \mathsf{do}\ x \leftarrow p; \bot; \mathsf{ret}\, x. \qquad (7)$$

While in KAT, $p; \neg\psi = p; 0$ (i.e. $p; \neg\psi = 0$) is equivalent to $p; \psi = p$, (7) fails to be equivalent to (6) in our setting unless ψ is classical (i.e. $\psi \vee \neg\psi$ holds). We choose definition (6) for its better properties; e.g. it makes weakest preconditions commute with finite conjunctions and does not involve negation.

VII. Relative Completeness

We proceed to establish a generic relative completeness result for our Hoare calculus. This result is *parametrized* over the base monad, i.e. it holds for a fixed (but, up to mild side conditions, arbitrary) predicated monad $(\mathbb{T}, \mathbb{P})$ and interpretation of the basic types and programs. This is in keeping with Cook's seminal relative completeness theorem [5] for the classical case, which also works with a fixed interpretation. Like in [5] the core idea of the our proof is based on the notion of a *weakest (liberal) precondition*, which we introduce semantically by

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

for a program p and an assertion ψ. By construction, $\phi \sqsubseteq \mathit{wp}(x \leftarrow p, \psi)$ whenever $\{\phi\}\ x \leftarrow p\ \{\psi\}$. Moreover, since do preserves existing joins in the left argument, wp really yields a precondition, i.e. we have

Lemma 26. *For all* p, ψ, $\{\mathit{wp}(x \leftarrow p, \psi)\}\ x \leftarrow p\ \{\psi\}$.

A crucial ingredient of Cook's proof in the classical case is the fact that $\mathit{wp}(x \leftarrow p, \psi)$ is syntactically definable by induction over p. Of course, syntactic weakest preconditions for *basic* programs f need to be assumed in our setting; that is, we assume that for every ψ,

– $\mathit{wp}(x \leftarrow f(z), \psi)$ is expressible as an assertion;

$$\textbf{(ret)}\ \frac{}{\{\phi[t/x]\}\ x \leftarrow \mathsf{ret}(t)\ \{\phi\}} \qquad \textbf{(basic)}\ \frac{\{\phi\}\ x \leftarrow f(z)\ \{\psi\}}{\{\phi[t/x]\}\ x \leftarrow f(t)\ \{\psi\}}\ (f \in \Sigma)$$

$$\textbf{(do)}\ \frac{\{\phi\}\ x \leftarrow p\ \{\psi\} \quad \{\psi\}\ y \leftarrow q\ \{\chi\}}{\{\phi\}\ y \leftarrow \mathsf{do}\ x \leftarrow p; q\ \{\chi\}} \qquad \textbf{(wk)}\ \frac{\phi' \sqsubseteq \phi \quad \{\phi\}\ x \leftarrow p\ \{\psi\} \quad \psi \sqsubseteq \psi'}{\{\phi'\}\ x \leftarrow p\ \{\psi'\}}$$

$$\textbf{(if)}\ \frac{\{\phi \wedge b?\}\ x \leftarrow p\ \{\psi\} \quad \{\phi \wedge \bar{b}?\}\ x \leftarrow q\ \{\psi\}}{\{\phi\}\ x \leftarrow (\mathsf{if}\, b\, \mathsf{then}\, p\, \mathsf{else}\, q)\ \{\psi\}} \qquad \textbf{(while)}\ \frac{\{\psi \wedge b?\}\ x \leftarrow p\ \{\psi\}}{\{\psi\}\ x \leftarrow (\mathsf{while}\, b\, \mathsf{do}\, x \leftarrow p)\ \{\psi \wedge \bar{b}?\}}$$

Fig. 2. The generic Hoare calculus.

– we have an axiom $\{wp(x \leftarrow f(z), \psi)\}\ x \leftarrow f(z)\ \{\psi\}$, which we call $\textsc{wp}_f$.

We denote the set of axioms $\textsc{wp}_f$, $f \in \Sigma$, by Δ_Σ. We then define a syntactic version wp_s of wp by

$$\begin{aligned}
&wp_s(x \leftarrow f(t), \psi) = wp(x \leftarrow f(z), \psi)[t/z] \ \ (f \in \Sigma, z \text{ fresh});\\
&wp_s(x \leftarrow \mathsf{ret}\, t, \psi) = \psi[t/x];\\
&wp_s(x \leftarrow (\mathsf{do}\ y \leftarrow p; q), \psi) = wp_s(y \leftarrow p, wp_s(x \leftarrow q, \psi));\\
&wp_s(x \leftarrow (\mathsf{if}\, b\, \mathsf{then}\, p\, \mathsf{else}\, q), \psi) =\\
&\qquad \mathsf{if}\, b\, \mathsf{then}\ wp_s(x \leftarrow p, \psi)\ \mathsf{else}\ wp_s(x \leftarrow q, \psi);\\
&wp_s(x \leftarrow (\mathsf{while}\, b\, \mathsf{do}\, x \leftarrow p), \psi) =\\
&\qquad (\nu X. \lambda x.\ \mathsf{if}\, b\, \mathsf{then}\ wp_s(x \leftarrow p, X(x))\ \mathsf{else}\, \psi)(x).
\end{aligned}$$

Thus defined, $wp_s(x \leftarrow q, \psi)$ is derivably a precondition:

Lemma 27. *For all p, ψ,*

$$\Delta_\Sigma \vdash_{\mathbb{P}} \{wp_s(x \leftarrow p, \psi)\}\ x \leftarrow p\ \{\psi\}.$$

Hence, $wp_s(x \leftarrow p, \psi) \sqsubseteq wp(x \leftarrow p, \psi)$.

To prove that wp_s is actually the same as wp, we essentially need to show that wp satisfies the recursive definition of wp_s, where the right-to-left implication is shown in the same way as Lemma 27. For the left-to-right implications, the only problematic case is do. Indeed, we need to postulate compatibility of preconditions with sequential composition as an additional requirement on the underlying monad:

Lemma and Definition 28 (Sequential Compatibility)**.** *For programs p, q and an assertion ψ, we have*

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

iff

$$[x \leftarrow (\mathsf{do}\ y \leftarrow p; q)]\, \psi \ \ \textit{implies} \ \ [y \leftarrow p]\ wp(x \leftarrow q, \psi)$$

If these conditions hold for all p, q, ψ, we say that $(\mathbb{T}, \mathbb{P})$ is sequentially compatible.

Remark 29. Sequential compatibility is reminiscent of the definition of a monad admitting dynamic logic [31], a notable difference being that in order-enriched monads, wp becomes definable, so that only its properties instead of its existence need to be postulated. *Mutatis mutandis*, essentially the same counterexample as in [31], a continuation monad, shows that not all monads are sequentially compatible.

Lemma 30. *Let $(\mathbb{T}, \mathbb{P})$ be sequentially compatible. Then $wp_s(x \leftarrow p, \psi) \sqsupseteq wp(x \leftarrow p, \psi)$ for all p, ψ.*

Proof: Induction over p. ∎

The generic relative completeness theorem is now immediate:

Theorem 31 (Generic Relative Completeness)**.** *Let $(\mathbb{T}, \mathbb{P})$ be sequentially compatible. Then $\mathbb{T}, \mathbb{P} \models \{\phi\}\ x \leftarrow p\ \{\psi\}$ implies $\Delta_\Sigma \vdash_{\mathbb{P}} \{\phi\}\ x \leftarrow p\ \{\psi\}$.*

Proof: Let $\mathbb{T}, \mathbb{P} \models \{\phi\}\ x \leftarrow p\ \{\psi\}$. Then the inequality assertion $\phi \sqsubseteq wp(x \leftarrow p, \psi)$, which is expressible by Lemmas 27 and 30, is valid in $\mathbb{P}$ and hence can be used to derive $\{\phi\}\ x \leftarrow p\ \{\psi\}$ by **(wk)** from $\{\mathsf{wp}(x \leftarrow p, \psi)\}\ x \leftarrow p\ \{\psi\}$. The latter is derivable by Lemma 27. ∎

Example 32. The generic relative completeness theorem instantiates to relative completeness of our Hoare calculus over any of the monads listed in Example 8 (as all these monads are easily shown to be sequentially compatible). The concrete instantiation depends, of course, on the choice of basic programs. We consider some examples in more detail:

1. *Partial state monad:* Working with a state set $S = L \to V$ where L is a set of locations and V is a set of values (say, natural numbers), we can introduce a type V to represent the set of values, and basic programs $write_l : V \to T1$ and $read_l : PV$ with the expected interpretation for each $l \in L$. We thus recover the standard Hoare calculus by introducing axioms capturing the usual weakest preconditions,

$$wp(write_l(z), \psi) = \psi'$$

where ψ' is obtained from ψ by replacing all occurrences of $read_l$ with $\mathsf{ret}(z)$. In this case, our generic completeness result instantiates exactly to Cook's relative completeness theorem [5].

2. *Non-deterministic state monad:* In otherwise the same setup as above, we can introduce non-deterministic basic programs, such as the *havoc* construct found in Boogie [1] and ESC/Java [7], which assigns an arbitrary value to a location. In this case, we have to specify $wp(havoc_l, \psi) = \forall z.\, \psi'$ where as above, ψ' is obtained from ψ by replacing all occurrences of

$read_l$ with $\mathsf{ret}(z)$. Similarly, we can specify a non-deterministic coin $toss : T2$ with $wp(x \leftarrow toss, \psi) = \psi[\bot/x] \wedge \psi[\top/x]$, which then allows coding binary nondeterministic choice as $p + q = \mathsf{if}\ toss\ \mathsf{then}\ p\ \mathsf{else}\ q$, and non-deterministic iteration as $\mathsf{init}\ x \leftarrow \mathsf{ret}\ x\ \mathsf{in}\ x \leftarrow p^\star = \mathsf{while}\ toss\ \mathsf{do}\ x \leftarrow p$. Our framework yields a relatively complete Hoare calculus for such languages.

3. *Additional computational features:* Our calculus remains sound and relatively complete when features such as exceptions, resumptions (used for modelling interleaving parallelism) or local state are added. In order to verify properties of interest for programs using these features, one does need additional logical scaffolding beyond the simplistic before-after of Hoare logic — e.g. abnormal postconditions to deal with exceptions, and stepwise invariants for resumptions. However, the basic Hoare calculus remains a crucial ingredient in the verification of such programs, and in fact some of the extended features required in the full verification logic can be encoded into the base calculus [30], [9].

4. *Domain-theoretic and topological state monads:* For these monads, we obtain the same results as indicated above but now for a non-classical assertion language. In other words, we show that classicality of the assertion logic is inessential for purposes of relative completeness; an added benefit of the non-classical setup is that we now know more about assertions — specifically that they are open subsets of the state space, which in the cases discussed in Example 8 means they are determined locally by finite information.

VIII. Strongest Postconditions

In the classical setting, one has a dual calculus of strongest postconditions complementing weakest preconditions. In the general case, it turns out the situation is more complicated. Since Ω is a frame, we can, of course, put

$$sp(x,p) = \bigsqcap\{\phi \mid [x \leftarrow p]\ \phi\}.$$

We can then accommodate preconditions by precomposing them with programs: $sp(\phi, x, p) := sp(x, \mathsf{do}\ \phi; p)$. However, unlike in the dual case of weakest preconditions, it will turn out that $sp(x,p)$ is not always a postcondition of p. We therefore introduce specific terminology for the positive case:

Definition 33 (Admitting Strongest Postconditions)**.** Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad. We say that $(\mathbb{T}, \mathbb{P})$ *admits strongest postconditions* if

$$[x \leftarrow p]\ sp(x,p)$$

for every program p. A program p *admits strongest postconditions* if $\{\phi\}\ x \leftarrow p\ \{sp(\phi, x, p)\}$ for all ϕ.

Proposition 34. *Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad that admits strongest postconditions. Then the following are equivalent.*

1) *$(\mathbb{T}, \mathbb{P})$ is sequentially compatible.*
2) *For all p, q, ψ, $[x \leftarrow (\mathsf{do}\ y \leftarrow p; q)]\,\psi$ implies that $sp(y,p) \sqsubseteq wp(x \leftarrow q, \psi)$ is valid in $\mathbb{P}$.*
3) *For all p, q, $\{sp(y,p)\}\ x \leftarrow q\ \{sp(x, \mathsf{do}\ y \leftarrow p; q)\}$.*

In the classical case, strongest postconditions always exist:

Proposition 35. *Let $(\mathbb{T}, \mathbb{P})$ be a predicated monad. If $P1$ is classical then $(\mathbb{T}, \mathbb{P})$ admits strongest postconditions.*

The proof makes use of the 'drinker's paradox' in the form $\bigvee_i (a_i \Rightarrow \bigwedge_i a_i)$, which depends on classicality. If $P1$ is not classical, it may happen that not all programs admit strongest postconditions; the following examples suggest that admitting strongest postconditions relates to computational feasibility.

Example 36. 1. *Domain-theoretic state monad:* A program $p : S \to (A \times S)_\bot$ has a strongest postcondition iff its image has a Scott open hull. Thus, *for algebraic S, a Scott continuous state transformer $f : S \to (A \times S)_\bot$ admits strongest postconditions iff f is a compact element of $S \to (A \times S)_\bot$*, i.e. preserves compact elements. E.g. for $S = L \to V_\bot$ as in Example 8, the compact elements of S are the *finite* states, i.e. those with only finitely many allocated locations. Then, a state transformer $f : S \to (A \times S)_\bot$ admits strongest postconditions, i.e. is compact, iff it allocates only finitely many locations when run from a finite state s.

2. *Topological State Monad:* When the state set S is T_1, then only open sets have open hulls, so that a continuous state transformer $f : S \to (A \times S)_\bot$ admits strongest postconditions iff f is open. (In particular, it is not generally case that compact elements of TA admit strongest postconditions. Comparing this to the above, note that the Scott topology is only T_0). When S is Stone, i.e. compact Hausdorff with a clopen base, then this means that f preserves clopens. E.g., in the case $S = L \to V$ as in Example 8 (which is a Stone space), the clopens are precisely the subsets of S defined by constraints on finitely many locations — that is, admitting strongest postconditions means writing only to finitely many locations.

IX. Conclusion and Related Work

We have introduced a Hoare logic for programs with order-enriched effects encapsulated as monads. For this logic, we have proved relative completeness. This result is formulated as a *generic* completeness theorem, instantiated to completeness results for numerous monadic models; e.g. it reproduces Cook's original completeness result but also a range of further completeness theorems for programs with additional or different computational effects, with more powerful basic operations, and with different assertion logics including logics of (Scott) open sets. Our formalisation utilizes the approach of domain theory in combination with recent developments on algebraic operations for computational effects [24], [25], [26]. This allows for a seamless integration of the assertion language with the programing language. In particular, we have shown that appropriate enrichment of a monad yields a natural frame structure on a submonad of innocent computations.

The way of enriching the monad we use is more general than the standard approach by enriching the underlying category. Instead we essentially enrich the corresponding Kleisli category. A form of monad enrichment similar to ours appears in [12] for the entirely different purpose of defining trace semantics of coalgebraic languages.

X. Further Work

Although our assertion language extends intuitionistic first order logic in case the underlying category is $\mathbf{Set}$, it is quite weak in the general case, and, e.g., does not in general have implication; we emphasize that this actually lends additional strength to our relative completeness result. General criteria under which the assertion language does support full intuitionistic first-order logic are under investigation. Whereas we currently do not have a perspective to induce such a structure solely from properties of the base category in any interesting cases other than $\mathbf{Set}$, in view of Theorem 18 there is hope to obtain a strong logic of assertions by using properties of the category in combination with the innocence condition. The direction of primary interest in this respect is the case when the base category is a topos. We expect that the chances to succeed in endowing $P1$ with a complete Heyting algebra critically depend the enrichment of P being compatible with the notion of partiality offered by the topos. E.g. one can argue that assertions over the partial state monad $S \rightharpoonup (X \times S)$ considered over a topos support intuitionistic first-order logic once the partial function type is interpreted in such a way that $X \rightharpoonup 1$ is isomorphic to $\mathcal{P}(X)$.

Additional perspectives for further work include coverage for computational effects whose full specification escapes the basic before/after paradigm that underlies Hoare logic, such as I/O, exceptions, and numerical probabilities — our calculus is sound and complete for monads with such features but does not express all of the requisite properties, such as abnormal termination. We expect some of these features to be encodable into the basic setting by means of additional observational operations in the spirit of [30]; a true conceptual unification of verification logics for these features however remains the subject of ongoing investigations. Orthogonally to these efforts, we are also working on an extension of our calculus with separation logic features [28] using the tensor product of monads as a generic counterpart of effect separation.

References

[1] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *Formal Methods for Components and Objects, FMCO 2005*, volume 4111 of *LNCS*, pages 364–387. Springer, 2006.

[2] M. Barr and C. Wells. *Toposes, Triples and Theories*, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1985.

[3] P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *Category Theory and Computer Science, CTCS 1993*, 1993.

[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] R. Crole and A. Pitts. New foundations for fixpoint computations. In *Logic in Computer Science, LICS 1990*, pages 489–497. IEEE Computer Society, 1990.

[7] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Principles of Programming Languages, POPL 2002*, pages 191–202. ACM, 2002.

[8] C. Führmann. Varieties of effects. In *Foundations of Software Science and Computation Structures*, volume 2303 of *LNCS*, pages 144–158. Springer, 2002.

[9] 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.

[10] S. Goncharov, L. Schröder, and T. Mossakowski. Completeness of global evaluation logic. In *Mathematical Foundations of Computer Science, MFCS 2006*, volume 4162 of *LNCS*, pages 447–458. Springer, 2006.

[11] W. Harrison. The essence of multitasking. In *Algebraic Methodology and Software Technology, AMAST 2006*, volume 4019 of *LNCS*, pages 158–172. Springer, 2006.

[12] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. In *Logical Methods in Comp. Sci*, page 2007, 2007.

[13] M. Hennessy and G. Plotkin. Full abstraction for a simple parallel programming language. In *MFCS*, pages 108–120, 1979.

[14] B. Jacobs and E. Poll. Coalgebras and Monads in the Semantics of Java. *Theoret. Comput. Sci.*, 291:329–349, 2003.

[15] M. Kelly. *Basic Concepts of Enriched Category Theory*. Number 64 in London Mathematical Society Lecture Notes. Cambridge University Press, 1982.

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

[17] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[18] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Logic*, 1(1):60–76, July 2000.

[19] 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.

[20] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93:55–92, 1991.

[21] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22:117–152, 1995.

[22] T. Mossakowski, L. Schröder, and S. Goncharov. A generic complete dynamic logic for reasoning about purity and effects. *Formal Asp. Comput.*, 22:363–384, 2010.

[23] A. Pitts. Evaluation logic. In *Higher Order Workshop*, Workshops in Computing, pages 162–189. Springer, 1991.

[24] G. Plotkin and J. Power. Adequacy for algebraic effects. In *Foundations of Software Science and Computation Structures, FoSSaCS 2001*, volume 2030 of *LNCS*, pages 1–24. Springer, 2001.

[25] G. Plotkin and J. Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures, FoSSaCS 2002*, volume 2303 of *LNCS*, pages 342–356. Springer, 2002.

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

[27] J. Reynolds. *Theories of Programming Languages.* Cambridge University Press, 1998.

[28] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science, LICS 2002*, pages 55–74. IEEE Computer Society, 2002.

[29] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in HasCASL. In *Fundamental Aspects of Software Engineering, FASE 2003*, volume 2621 of *LNCS*, pages 261–277, 2003.

[30] 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.

[31] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HasCASL. *J. Logic Comput.*, 14:571–619, 2004.

[32] L. Schröder and T. Mossakowski. Hascasl: Integrated higher-order specification and program development. *Theoret. Comput. Sci.*, 410:1217–1260, 2009.

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

[34] M. Smyth. Power domains and predicate transformers: A topological view. In *Automata, Languages and Programming, ICALP 1983*, volume 154 of *LNCS*, pages 662–675. Springer, 1983.

[35] R. Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2:149–168, 1972.

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