Completeness of Global Evaluation Logic

Sergey Goncharov[1], Lutz Schröder[1,2], and Till Mossakowski[1,2]

[1] Department of Computer Science, University of Bremen, Germany
[2] DFKI Lab Bremen, Germany

Abstract. Monads serve the abstract encapsulation of side effects in semantics and functional programming. Various monad-based specification languages have been introduced in order to express requirements on generic side-effecting programs. A basic role is played here by global evaluation logic, concerned with formulae which may be thought of as being universally quantified over the state space; this formalism is the fundament of more advanced logics such as monad-based Hoare logic or dynamic logic. We prove completeness of global evaluation logic for models in cartesian categories with a distinguished Heyting algebra object.

Introduction

Monads form the basis of the abstract treatment of side effects in modern functional programming languages and in associated specification languages. The computational significance of monads was first noticed by Moggi [7]. They were subsequently incorporated in the design of the functional programming language Haskell [11] as the principal means of dealing with impure features such as I/O, but also as a convenient mode of expression for state threading, e.g. in parsing [3]. Monads are by now also well-established as a tool in semantics (cf. [15] for a recent example, and [4] for an application to the semantics of Java). The advantage of the abstraction of notions of side effect as monads is twofold: firstly, one obtains a generic framework that can be instantiated with many particular monads, thus promoting the reuse of concepts and indeed programs. Secondly, side effects are cleanly encapsulated and equipped with a clearly delineated scope [9, 2].

The use of monads in programming has led to the design of various specification logics for monadic programs, including Pitts' evaluation logic [12], Moggi's globalized version of evaluation logic [8], a monad-based Hoare logic [13], and a monad-based dynamic logic related to evaluation logic [14]. The basic tool in the semantics of the latter two logics is *global evaluation logic*, concerned with statements of the nature 'after execution of a given program from *any* initial state, it is the case that ...' (referred to as *global dynamic judgements* in [14]). This logic is closely related to Moggi's global semantics of evaluation logic (which is in fact markedly dissimilar from the originally intended *local* semantics of evaluation logic [12]).

Unlike some of the more complex logics, global evaluation logic can be interpreted over arbitrary monads, under only quite rudimentary requirements on

the base category. In this work, we make this semantics explicit, thus detaching the calculus from the more complex framework of the specification language HasCasl (i.e. higher order logic of partial functions) in which it was originally presented [13, 14]. We then prove completeness of global evaluation logic for the said class of models, viz. over cartesian categories with a distinguished Heyting algebra object.

This work forms part of an ongoing effort to establish completeness results for monad-based computational logics. A further result in this direction is the completeness of monad-based dynamic logic [10]. While no completeness result for the local semantics of evaluation logic is given in [12], Moggi does prove completeness for his global version of evaluation logic [8]. The crucial difference between the latter result and the one presented here is that the logic considered in [8] includes the entire equational theory of monads, i.e. the meta-language introduced in [7]. This fact plays a central role in Moggi's completeness proof, which relies on constructing a term model using the equations of the meta-language. In contrast to this, our global evaluation logic talks purely about the observable behaviour of programs, and excludes statements about actual equality of programs (we do admit an underlying equational theory, which however shows up only in rules expressing Leibniz equalities). It may be argued that this amounts to a better encapsulation of program behaviour, and is closer in spirit to program logics such as Hoare logic, which also talk only about properties of the state space affected by program execution rather than about equality of programs. In consequence, the term model construction in our completeness proof is more involved: we cannot rely on treating equality by means of an equational calculus, but need to introduce a Leibniz equality on terms. An important tool in the model construction is a normalization result for program sequences, which may be of independent interest.

The material is organized as follows. We recall the basics of monad-based programming in Sect. 1. In Sect. 2, we define the syntax and semantics of global evaluation logic, indicate how global evaluation logic is used in more advanced program logics, and discuss the relation of our work to [8] in more detail. The proof calculus of the logic is presented in Sect. 3. Section 4 is devoted to the completeness proof of this calculus.

1 Monads for computations

Intuitively, a monad associates to each type A a type TA of computations of type A; a function with side effects that takes inputs of type A and returns values of type B is, then, just a function of type $A \to TB$. This approach abstracts away from particular notions of computation such as store, non-determinism, non-termination etc.; a surprisingly large amount of reasoning can be carried out without commitment to a particular type of side effect.

A monad on a category $\mathbf{C}$ can be defined as a *Kleisli triple* $\mathbb{T} = (T, \eta, _^*)$, where $T : \mathrm{Ob}\,\mathbf{C} \to \mathrm{Ob}\,\mathbf{C}$ is a function, the *unit* η is a family of morphisms $\eta_A : A \to TA$, and $_^*$ assigns to each morphism $f : A \to TB$ a morphism

$f^* : TA \to TB$ such that

$$\eta_A^* = id_{TA}, \quad f^*\eta_A = f, \quad \text{and} \quad g^*f^* = (g^*f)^*.$$

This description is equivalent to the more familiar one via an endofunctor with unit and multiplication [6].

In order to support a language with finitary operations and multi-variable contexts (see below), one needs a further ingredient: a monad is called *strong* if it is equipped with a natural transformation

$$t_{A,B} : A \times TB \to T(A \times B)$$

called *strength*, subject to certain coherence conditions [7].

Example 1. [7] Computationally relevant monads on **Set** include

- Stateful computations with possible non-termination: $TA = (S \to ?\ (A \times S))$, where S is a fixed set of states and $_ \to ?\ _$ denotes the partial function type.
- (Finite) non-determinism: $TA = \mathcal{P}_{fin}(A)$, where $\mathcal{P}_{fin}$ denotes the finite power set functor.
- Exceptions: $TA = A + E$, where E is a fixed set of exceptions.
- Interactive input: TA is the smallest fixed point $\mu\gamma.\, A + (U \to \gamma)$, where U is a set of input values.
- Non-deterministic stateful computations: $TA = \mathcal{P}_{fin}(S \to (A \times S))$, where, again, S is a fixed set of states.
- Continuations: $TA = (A \to R) \to R$, where R is a type of *results*.

Reasoning about a category equipped with a strong monad is greatly facilitated by the fact that proofs can be conducted in a *computational meta-language* with the following features, introduced in [7]:

- A type constructor T; TA is the type of A-valued *programs* or *computations*;
- a polymorphic operator $\mathrm{ret} : A \to TA$ corresponding to the unit;
- a binding construct, which we denote in Haskell's do style: terms of the form

  $$\mathrm{do}\ \ x \leftarrow p;\ q$$

  are interpreted by means of the tensorial strength and Kleisli composition (cf. [7] for details). Intuitively, $\mathrm{do}\ \ x \leftarrow p;\ q$ first computes p, whose result is assigned to the bound variable x, and then computes q (which may depend on x). Binding is *associative*, i.e. one has

  $$\mathrm{do}\ \ y \leftarrow (\mathrm{do}\ \ x \leftarrow p;\ q);\ r = \mathrm{do}\ \ x \leftarrow p;\ \mathrm{do}\ \ y \leftarrow q;\ r$$

  if r does not contain x. We denote nested do expressions $\mathrm{do}\ \ x \leftarrow p;\ \mathrm{do}\ \ y \leftarrow q;\ \ldots$ by $\mathrm{do}\ \ x \leftarrow p; y \leftarrow q;\ \ldots$. Repeated nestings $\mathrm{do}\ \ x_1 \leftarrow p_1, \ldots, x_n \leftarrow p_n;\ q$ are denoted in the form $\mathrm{do}\ \ \bar{x} \leftarrow \bar{p};\ q$. Term fragments of the form $\bar{x} \leftarrow \bar{p}$ are called *program sequences*. Variables x_i that do not appear later on may be omitted from the notation. Terms that differ only in the names of bound variables are equal (α-equivalence). The scope of a binding extends as far as possible.

– besides the associative law mentioned above, *unit laws* stating that

$$(\text{do } x \leftarrow y;\ \text{ret}\, x) = y \qquad \text{and}$$
$$(\text{do } x \leftarrow \text{ret}\, a;\ p) = p[a/x].$$

Terms are generally formed in a context of variables with assigned types; however, we will omit contexts from the notation.

A program p is called *stateless* if it factors through ret, i.e. if it is just a value inserted into the monad. In [14], we also introduce a notion of *deterministically side-effect free (dsef)* monadic value. All stateless programs are dsef, but not vice versa. For example, in the state monad, a program that reads the state but does not modify it is dsef but not stateless.

2 Global Evaluation Logic

Global evaluation logic is a simple specification language for monadic programs. It is related to (monad-based) dynamic logic in that it features a modality that encapsulates the effect of executing a program; however, unlike in dynamic logic this modality is *global*, i.e. implicitly quantified over all states. Thus, the formulae of global evaluation logic are of the form

$$[\![x \leftarrow p]\!]\, \phi,$$

where $p : TA$ is a monadic program and ϕ is a stateless formula possibly containing the free variable x, to be read informally as *for all states s, execution of p in state s yields a result x satisfying ϕ.*

Global evaluation logic is used chiefly as a means of encoding or defining more advanced logics. In the monad-based Hoare calculus [13], Hoare triples are defined by

$$\{\varphi\}\ x \leftarrow p\ \{\psi\} :\equiv [\![a \leftarrow \varphi; x \leftarrow p; b \leftarrow \psi]\!]\, (a \to b).$$

Here ψ may contain x, and φ and ψ are *dynamic formulae*, i.e. computations of truth values which are dsef, in particular may access but not modify the state. The Hoare triple $\{\varphi\}\ x \leftarrow p\ \{\psi\}$ has, in typical examples, the expected meaning *if program p is executed in a state satisfying φ, terminating in some state s, then s and the result x of p satisfy ψ*. Further examples of encodings of program logics in a logic related to global evaluation logic are given in [8]. Moreover, the necessity operators $[x \leftarrow p]\, \varphi$ of monad-based dynamic logic [14], where φ is a dynamic formula, are uniquely determined by being dsef and satisfying the equivalence

$$[\![y \leftarrow q]\!]\, (\psi \to [x \leftarrow p]\, \varphi) \text{ iff } [\![y \leftarrow q; x \leftarrow p]\!]\, (\psi \to \varphi)$$

in global evaluation logic (with some syntactic sugar), where q ranges over programs and ψ ranges over dynamic formulae possibly containing y (but not x). If $[x \leftarrow p]\, \varphi$ exists, then its interpretation in typical examples may be thought of as a state-dependent predicate, satisfied in state s if *all possible poststates*

and results for terminating execution of p from state s satisfy φ. Completeness of monad-based dynamic logic is shown in [10] without recourse to more basic completeness results. The pending completeness proof for the monad-based Hoare calculus, however, is expected to make use of completeness of global evaluation logic.

To begin, we separate the definition of the syntax and semantics of global evaluation logic from the HasCasl framework originally used in its formulation [14]. The presentation of the logic is slightly more general than in [14] in that we admit an underlying equational theory.

Given a set S of basic types, the set of *types* A is generated by the grammar

$$A ::= 1 \mid \Omega \mid TA \mid A \times A \mid S.$$

A type of the form TA is called a *computational type.* A type is called T*-free* if it does not contain occurrences of T. A signature $\Sigma = (S, F)$ consists of a set S of basic types and a set F of operation symbols $f : A \to B$, where A and B are types over S, *with* A T*-free.* The latter condition means that we exclude user-defined control structures; the design of a complete logic that includes such control structures is the subject of future research. The type Ω serves as a type of truth values; we assume that Σ contains operations on Ω representing the logical connectives.

A signature gives rise to a notion of *term* (in a context $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ of type assignments for variables, which we mostly omit from the notation) in the usual way, with term formation rules including variable introduction, application of basic operators, formation of pairs $\langle s, t\rangle : A \times B$, projection terms $\mathit{fst}(t)$, $\mathit{snd}(t)$, the distinguished term $*$ inhabiting the unit type 1, and the return and binding operations of the computational metalanguage. We denote the set of free variables occurring in a term t (or, later, a formula) by $FV(t)$. An *equation* is a pair of terms of the same type (in a common context).

An *equational theory* $\mathcal{E}$ is a pair (Σ, E), where Σ is a signature and E is a set of equations over Σ. We assume that $\mathcal{E}$ contains the Heyting algebra axioms for Ω (if classical reasoning is required, one may extend these to the axioms for a Boolean algebra). We write $\mathcal{E} \vdash s = t$ if $s = t$ can be derived from $\mathcal{E}$ using standard equational reasoning (this includes equations for product types involving terms of the computational metalanguage such as $\mathit{snd}\langle(\mathrm{do}\ \ x \leftarrow p;\ q), \mathrm{ret}\, a\rangle = \mathrm{ret}\, a$, but *excludes* the equations of the computational metalanguage, i.e. associativity and the two unit laws).

A *model* of a signature consists of a cartesian category $\mathbf{C}$ (i.e. a category with finite products) equipped with a distinguished object Ω and a strong monad T, together with interpretations of the basic types as objects in $\mathbf{C}$ and interpretations of the basic operations as morphisms in $\mathbf{C}$, referred to slightly inaccurately just as T. This extends to an interpretation $[\![A]\!]_T \in \mathrm{Ob}\,(\mathbf{C})$ for every type A by interpreting product types and unit types by the cartesian structure of $\mathbf{C}$, Ω as the distinguished object Ω, and computational types TA by applying the monad T. Then, a context $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ is interpreted as the product $[\![\Gamma]\!]_T = \prod_{i=1}^{n} [\![A_i]\!]_T$. The interpretation of a basic operation $f : A \to B$ is required to be a morphism $[\![A]\!]_T \to [\![B]\!]_T$.

We then obtain, for every term $t : A$ in context Γ, an interpretation as a morphism $[\![t]\!]_T : [\![\Gamma]\!]_T \to [\![A]\!]_T$, where variables are interpreted as projections, pairing, projections, and $*$ are interpreted using the cartesian structure, application of basic operations is composition, and return and binding are interpreted as prescribed by T. A *model* of an equational theory $\mathcal{E} = (\Sigma, E)$ is a model of Σ that *satisfies* every equation $s = t$ in E in the sense that $[\![s]\!]_T = [\![t]\!]_T$. The equations in $\mathcal{E}$ imply that in every model of $\mathcal{E}$, the object Ω is a *Heyting algebra object*, i.e. its hom-functor $hom(_, \Omega)$ factors through Heyting algebras.

Definition 2. A term is called *product β-normal* if it does not contain subterms of the form $fst(u, v)$ or $snd(u, v)$.

Lemma 3. *Every term t has a* product β-normal form*, i.e. a product β-normal term t' such that $\mathcal{E} \vdash t = t'$.*

As indicated above, a *formula* of *global evaluation logic* is an expression of the form $[\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$, where $p_i : TA_i$ and $\phi : \Omega$ are terms. We say that a model T *satisfies* $[\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$, and write $T \models [\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$, if

$$[\![\mathrm{do}\ \bar{x} \leftarrow \bar{p};\ \mathrm{ret}\langle \bar{x}, \phi \rangle]\!]_T = [\![\mathrm{do}\ \bar{x} \leftarrow \bar{p};\ \mathrm{ret}\langle \bar{x}, \top \rangle]\!]_T.$$

Moreover, T satisfies a set Φ of formulae, written $T \models \Phi$, if $T \models P$ for every $P \in \Phi$.

Definition 4. We say that the monad (or model) T is *simple* if satisfaction of $[\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$ is equivalent to

$$[\![\mathrm{do}\ \bar{x} \leftarrow \bar{p};\ \mathrm{ret}\, \phi]\!]_T = [\![\mathrm{do}\ \bar{x} \leftarrow \bar{p};\ \mathrm{ret}\, \top]\!]_T.$$

The calculus of Section 3, for which we prove completeness, is sound only for simple models.

Example 5. In the monads of Example 1, satisfaction of $[\![x \leftarrow p]\!]\, \phi$, where $p : TA$, amounts to the following (we freely omit semantic brackets from the notation):

- *stateful computation*: terminating execution of p from any initial state yields a result x satisfying ϕ;
- *non-determinism*: all values x in $p \in \mathcal{P}(A)$ satisfy ϕ;
- *exceptions*: if p terminates normally, then its result x satisfies ϕ;
- *interactive input*: the value x eventually produced by p after some combination of inputs always satisfies ϕ;
- *non-deterministic stateful computation*: all possible results x obtained by execution of p from any initial state satisfy ϕ;
- *continuations*: for $k : A \to R$, $p\ k$ depends only on the restriction of k to the set of values $x : A$ satisfying ϕ.

All these monads except the continuation monad are simple [14].

Remark 6. A particularly troublesome case w.r.t. monad-based specification logics is the continuation monad (Example 1). The main problem is that the continuation monad has too few deterministically side-effect free computations — e.g. if the base category is **Set**, all dsef computations are already stateless values. Consequently, the continuation monad does not admit dynamic logic [14], and Hoare logic over the continuation monad, while sound, does not express any interesting properties, as pre- and postconditions are required to be dsef. Global evaluation logic, however, has a rather sensible interpretation over the continuation monad (Example 5). Thus, it appears feasible to take global evaluation logic as the starting point for the design of a program logic for the continuation monad. As the continuation monad fails to be simple, this means that there is some interest in the pending issue of finding a complete proof system for our global evaluation logic without the simplicity assumption.

Related Work

Global evaluation logic in the above sense is strongly related to Moggi's global semantics of evaluation logic [8] (although Moggi uses the term evaluation logic, the formulae denoted $[x \Leftarrow p]\,\phi$ in loc. cit. correspond essentially to $[\![x \leftarrow p]\!]\,\phi$ above, rather than to the modal formulae $[x \leftarrow p]\,\phi$ of evaluation logic in the original sense [12] or monad-based dynamic logic). In [8], it is assumed that T preserves monos, so that a predicate on A, represented as an admissible subobject $Q \hookrightarrow A$, lifts to a predicate TQ on TA, in order to define $[\![x \leftarrow p]\!]\,\phi$ as the formula

$$(\mathrm{do}\ \ x \leftarrow p;\ \mathrm{ret}\langle y, x\rangle) \in T\phi$$

(expressed for the sake of readability in the computational metalanguage). Here, $y : B$ is the context, so that ϕ is a predicate on $B \times A$. Roughly speaking, one can show that this semantics agrees with ours in case the base category has equality, i.e. diagonals are admissible subobjects, and T preserves admissible monos and their inverse images (so that T is simple [8]). In cases where these conditions fail, we have the impression that the semantics used here is slightly easier to handle (as well as more general as it does not require preservation of monos — preservation of monos may fail e.g. for the continuation monad over base categories other than **Set** [8]).

These remarks aside, the main difference between global evaluation logic and the logic considered in [8] is that the latter includes the full equational logic of the computational metalanguage of [7], that is, equality of programs is part of the logic. This fact plays a crucial technical role in the completeness proof presented in [8], where term models for a theory $\mathcal{T}$ are equipped with an equality determined by the equations derivable from $\mathcal{T}$. Contrastingly, global evaluation logic is restricted to formulae of the form $[\![x \leftarrow p]\!]\,\phi$, which talk only about results of programs rather than equalities between programs; thus, global evaluation logic provides a black-box view of programs, while the logic of [8] provides a glass-box view.

3 A Proof Calculus for Global Evaluation Logic

We now present a proof system for global evaluation logic *over simple monads* (finding a complete calculus for the general case remains an open problem). The calculus is shown in Fig. 1. Double lines indicate that a rule may be applied in both directions. The calculus is slightly modified w.r.t the one presented in [14]: the structural rules have been gathered in a slightly stronger rule (derivability of the original rules will be shown below), rules have been added to support the underlying equational theory, and a rule concerning equality under the global modality (which is not supported in the present framework, but could be added in later extensions) has been removed. Importantly, we allow rule (ctr) to be applied backwards, which is sound only for simple monads. We write $\Phi \vdash_{\mathcal{E}} [\bar{x} \leftarrow \bar{p}]\,\phi$ if $[\bar{x} \leftarrow \bar{p}]\,\phi$ is derivable from a set Φ of formulae by means of the calculus. We will mostly omit the subscript $\mathcal{E}$. Instead of $\emptyset \vdash [\bar{x} \leftarrow \bar{p}]\,\phi$, we write $\vdash [\bar{x} \leftarrow \bar{p}]\,\phi$. Moreover, if $\Phi, [\bar{x} \leftarrow \bar{p}]\,\phi \vdash [\bar{y} \leftarrow \bar{q}]\,\psi$ and $\Phi, [\bar{y} \leftarrow \bar{q}]\,\psi \vdash [\bar{x} \leftarrow \bar{p}]\,\phi$, then we say that $[\bar{x} \leftarrow \bar{p}]\,\phi$ and $[\bar{y} \leftarrow \bar{q}]\,\psi$ are provably equivalent relative to Φ, and write $[\bar{x} \leftarrow \bar{p}]\,\phi \vdash\!\dashv^{\Phi} [\bar{y} \leftarrow \bar{q}]\,\psi$, again omitting the mention of Φ if Φ is empty.

$$(\top)\ \frac{\mathcal{E} \vdash \xi = \top}{[\bar{x} \leftarrow \bar{p}]\,\xi} \qquad \textbf{(cong)}\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\,(\xi \leftrightarrow \chi)\\ [\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\xi/y]]\,\eta[\xi/y]\end{array}}{[\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\chi/y]]\,\eta[\chi/y]}$$

$$\textbf{(pre)}\ \frac{\begin{array}{c}x \text{ not free in undischarged assumptions}\\ [\bar{y} \leftarrow \bar{q}]\,\phi\end{array}}{[x \leftarrow p; \bar{y} \leftarrow \bar{q}]\,\phi} \qquad (\mathcal{E})\ \frac{\begin{array}{c}\mathcal{E} \vdash t = t'\\ [\ldots x; \leftarrow \mathrm{ret}\, t; \ldots]\,\eta\end{array}}{[\ldots x; \leftarrow \mathrm{ret}\, t'; \ldots]\,\eta}$$

$$(\eta)\ \frac{[\bar{x} \leftarrow \bar{p}; y \leftarrow \mathrm{ret}\, r; \bar{z} \leftarrow \bar{q}]\,\eta}{[\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{q}[r/y]]\,\eta[r/y]} \qquad \textbf{(ctr)}\ \frac{\begin{array}{c}x \notin FV(\eta) \cup FV(\bar{r})\\ [\ldots; x \leftarrow p; y \leftarrow q; \bar{z} \leftarrow \bar{r}]\,\eta\end{array}}{[\ldots; y \leftarrow (\mathrm{do}\ \ x \leftarrow p;\ \ q); \bar{z} \leftarrow \bar{r}]\,\eta}$$

Fig. 1. Proof calculus for global evaluation logic

Lemma 7. *The rules*

$$(\wedge\mathbf{I})\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\,\phi\\ [\bar{x} \leftarrow \bar{p}]\,\psi\end{array}}{[\bar{x} \leftarrow \bar{p}]\,(\phi \wedge \psi)} \qquad \textbf{(wk)}\ \frac{\begin{array}{c}\bar{x} \text{ not free in undis-}\\ \text{charged assumptions}\\ \phi \rightarrow \psi\\ [\bar{x} \leftarrow \bar{p}]\,\phi\end{array}}{[\bar{x} \leftarrow \bar{p}]\,\psi} \qquad \textbf{(app)}\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\,\phi\\ y \notin FV(\phi)\end{array}}{[\bar{x} \leftarrow \bar{p}; y \leftarrow q]\,\phi}$$

are derivable.

Moreover, note that rules $(\mathcal{E})$ and (η) allow exchanging t for t' at any place in a formula whenever $\mathcal{E} \vdash t = t'$.

Remark 8. Under the additional restriction on signatures that an operation $f \in \Sigma$ has an argument of type Ω only if its result is of type Ω as well, rule (cong) can in fact be equivalently replaced by rules ($\wedge$I), (wk), and (app).

Lemma 9 (Substitution rule). *If σ is a substitution and $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\,\phi$, then $\Phi\sigma \vdash ([\bar{x} \leftarrow \bar{p}]\,\phi)\sigma$, where application of σ is by capture-avoiding substitution of free variables.* □

Theorem 10 (Soundness for simple models). *Let T be a simple model such that $T \models \Phi$ for a set Φ of formulae. Then $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\,\phi$ implies $T \models [\bar{x} \leftarrow \bar{p}]\,\phi$.*

4 Completeness

We will now prove strong completeness of global evaluation logic over simple monads. We explicitly fix the relevant notion of semantic consequence:

Definition 11. We write $\Phi \models [\bar{x} \leftarrow \bar{p}]\,\phi$ for a formula $[\bar{x} \leftarrow \bar{p}]\,\phi$ and a set Φ of formulae if for every simple model T, $T \models \Phi$ implies $T \models [\bar{x} \leftarrow \bar{p}]\,\phi$.

Theorem 12 (Strong completeness for simple models). *Let Φ be a set of formulae. Then $\Phi \models [\bar{x} \leftarrow \bar{p}]\,\phi$ implies $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\,\phi$.*

The proof requires a number of lemmas, the most important of which is a normalization result for global evaluation formulae (Lemma 15).

Lemma 13. *Product β-normal terms of T-free type do not contain subterms of computational type.*

Proof. (Sketch) Induction over terms, using the restriction on argument types of operation symbols. □

Lemma 14. *Let t be product β-normal. Then there exists a decomposition $t = s[t_1/x_1, \ldots t_n/x_n]$ such that*

- *s does not contain signature symbols and*
- *the t_i do not contain return or binding operators.*

Proof. (Sketch) Induction over the height of t, using Lemma 13 for the case that t is an application $f(q)$ of an operation symbol f. □

Lemma 15. *Given a formula $[\bar{x} \leftarrow \bar{p}]\,\phi$ and a variable $y : TA \in FV(\bar{p})$ there exist a formula $[\bar{z} \leftarrow \bar{q}]\,\psi$ provably equivalent to $[\bar{x} \leftarrow \bar{p}]\,\phi$, with $y \notin FV(\psi)$, and $y \in FV(q_j)$ only in case $q_j = y$.*

Proof. By successive reduction of $[\bar{x} \leftarrow \bar{p}]\,\phi$ to the required form. At the beginning of every step, reduce ϕ and the p_i to product β-normal form (Lemma 3), so that in particular $y \notin FV(\phi)$ by Lemma 13. Stop the process if none of the p_i contains return or binding. Otherwise, let p_i be the rightmost item containing return or binding. By Lemma 14, p_i is either a binding or an application of ret.

Accordingly, perform at p_i a reduction step (thus following a rightmost strategy) to a provably equivalent formula using rule (ctr) in the 'up' direction or rule (η) in the 'down' direction, respectively, and continue.

At the end of this process, we have reached the required form $[\bar{z} \leftarrow \bar{q}]\,\psi$: This is clear if q_j is a variable. If q_j is of the form $e_1(\dots e_k(t)\dots)$, where $e_k \in \{fst, snd\}$, $k \geq 1$, with t not an application of a projection, then t is either a variable or an application $f(u)$ of an operation symbol f. In the former case, $t \not\equiv y$, since t is of product type, so that $y \notin FV(q_j)$. In the latter case, u is of T-free type by the restriction on signatures; by Lemma 14, $y \notin FV(q_j)$. The same applies if q_j is an application of an operation symbol. By product β-normality, no other cases occur.

It remains to prove termination. We introduce a well-founded strict ordering $\succ$ on term sequences and prove that the reductions are decreasing under $\succ$. Let $Rets(t)$ be the maximal depth of nested occurrences of the return operator in a term t. Let $\mu(\bar{p})$ be the multiset $\{Rets(p_i)\}$ for a term sequence $\bar{p}$, and let $h(\bar{p})$ be the multiset of heights of the p_i, where we count sequential do-bindings do $\bar{x} \leftarrow \bar{p};\ q$ as *one* application (rather than n applications) of do. We define the strict well-founded ordering $\gg$ on multisets of integers as the multiset extension [1, 5] of the standard ordering $>$ of the integers, generated by closure under multiset union and transitivity from the clauses

$$\{a\} \gg \{b_1, \dots, b_r\} \text{ if } a > b_i \text{ for all } i.$$

Then we put $\bar{p} \succ \bar{p}'$ iff $\langle \mu(\bar{p}), h(\bar{p})\rangle$ is lexicographically greater than $\langle \mu(\bar{p}'), h(\bar{p}')\rangle$. Upwards reduction by (ctr) at p_i is decreasing: the first component stays unchanged and the second decreases, as p_i is replaced by two terms of lesser height. For reduction by (η), recall that p_j does not contain ret for $j > i$, so that $Rets(p_j[q/x_i]) < Rets(\mathrm{ret}\, q)$. Hence $\mu(\bar{p}) \gg \mu(\bar{p}')$, and thus $\bar{p} \succ \bar{p}'$. Finally, the intermediate reductions to product β-normal form are height-decreasing, and thus decreasing w.r.t. $\succ$. □

Note that the previous proof essentially establishes a form of weak normalization for program sequences, for a rightmost reduction strategy.

As indicated above, a form of Leibniz equality will be imposed on the term model to be constructed below:

Definition 16. Let $t, s : TA$, and let Φ be a set of formulae. We say that t and s are *Leibniz equal* (under Φ), and write $t \sim_\Phi s$, if

$$[\bar{x} \leftarrow \bar{p}[t/y]]\,\phi \vdash\!\dashv^{\Phi} [\bar{x} \leftarrow \bar{p}[s/y]]\,\phi$$

for all $\phi : \Omega$ and all program sequences $\bar{p}$.

As a direct consequence of Lemma 15, we have

Lemma 17. *Let $t, s : TA$, and let Φ be a set of formulae. Then $t \sim_\Phi s$ iff t and s are* weakly Leibniz equal *(under Φ), i.e.*

$$[\bar{v} \leftarrow \bar{u}; x \leftarrow t; \bar{z} \leftarrow \bar{r}]\,\phi \vdash\!\dashv^{\Phi} [\bar{v} \leftarrow \bar{u}; x \leftarrow s; \bar{z} \leftarrow \bar{r}]\,\phi$$

for all $\phi : \Omega$ and all program sequences $\bar{v} \leftarrow \bar{u}$, $\bar{z} \leftarrow \bar{r}$.

This now allows us to prove the non-trivial direction ('only if') of what will turn out to be the truth lemma for our term model:

Lemma 18 (Truth lemma). *For every set Φ of formulae, $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, \psi$ iff* $\mathrm{do}\ \bar{y} \leftarrow \bar{q};\ \mathrm{ret}\langle \bar{y}, \psi \rangle \sim_{\Phi} \mathrm{do}\ \bar{y} \leftarrow \bar{q};\ \mathrm{ret}\langle \bar{y}, \top \rangle$.

The proof of the completeness theorem is now by a term model construction which proceeds similarly as in [10]. Given an equational theory $\mathcal{E} = (\Sigma, E)$ and a set Φ of Σ-formulae, we construct a model of $\mathcal{E}$ as follows. Let $\mathbf{C}_{\mathcal{E},\Phi}$ be the category with objects being all types over Σ, and morphisms $A \to B$ being terms in context

$$(x : A \rhd t : B)$$

modulo Leibniz equality under Φ (we omit explicit notation for equivalence classes). It is easy to check that Leibniz equality is a congruence for the computational metalanguage. Identities in $\mathbf{C}_{\mathcal{E},\Phi}$ are given by variables $x : A \rhd x : A$, and composition is given by substitution. The category $\mathbf{C}_{\mathcal{E},\Phi}$ comes with a canonical cartesian structure, which on objects is just given by $A \times B$ and 1, with projections, pairing, and unique morphisms into 1 given in the obvious way. Axiom $(\mathcal{E})$ ensures that this does define a cartesian category satisfying $\mathcal{E}$.

Now we turn the type constructor T into a simple strong monad on $\mathbf{C}_{\mathcal{E},\Phi}$, with the unit η given by ret, the Kleisli star defined by

$$(x : A \rhd q : TB)^* := (p : TA \rhd \mathrm{do}\ \ x \leftarrow p;\ \ q : TB),$$

and the strength given by

$$t_{A,B} := (p : A \times TB \rhd \mathrm{do}\ \ x \leftarrow \mathit{snd}(p);\ \ \mathrm{ret}\langle \mathit{fst}(p), x \rangle : T(A \times B)).$$

In the verification of the monadic laws and simplicity, crucial use is made of the fact that by Lemma 17, it suffices to prove weak Leibniz equality.

The completeness proof is finished by noting that the Truth Lemma states that $T \models [\bar{y} \leftarrow \bar{q}]\, \psi$ iff $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, \psi$.

5 Conclusion

We have defined the syntax, semantics, and proof calculus of global evaluation logic over a simple monad on a cartesian base category with a distinguished Heyting algebra object. We have proved completeness of the logic using a term model in which, due to the absence of program equality from the logic, equality of terms is defined as Leibniz equality. The crucial point in the model construction is the reduction of Leibniz equality to a simplified form, which requires a normalization procedure on program sequences that uses a multiset termination measure. Two improvements of this result are left for future investigation: the extension of the calculus to sum types, with models over distributive categories, and support for user-defined control structures, i.e. basic operations with arguments of computational type.

The completeness of global evaluation logic forms the basis for investigations into the completeness of more advanced program logics defined in terms of it, in particular monad-based Hoare logic [13]. Moreover, global evaluation logic is expected to play a role in the design of a specification logic for the continuation monad: the latter has been shown not to admit an interpretation of dynamic logic [14], and Hoare logic over it lacks expressivity, while the much simpler global evaluation logic does make good sense over the continuation monad. As the continuation monad fails to be simple, this puts on the agenda the design of a complete calculus for the general case, without the simplicity assumption.

Acknowledgements

This work forms part of the DFG-funded project HasCASL (KR 1191/7-1 and KR 1191/7-2) and the project HOL-MDL funded by the FNK of the University of Bremen.

References

[1] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Comm. ACM*, 22:465–476, 1979.
[2] M. Fluet and G. Morrisett. Monadic regions. In *International Conference on Functional Programming*, pages 103–114. ACM, 2004.
[3] G. Hutton and E. Meijer. Monadic Parsing in Haskell. *J. Functional Programming*, 8:437–444, 1998.
[4] B. Jacobs and E. Poll. Coalgebras and Monads in the Semantics of Java. *Theoret. Comput. Sci.*, 291:329–349, 2003.
[5] J. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–117. Oxford, 1992.
[6] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1997.
[7] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.
[8] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22:117–152, 1995.
[9] E. Moggi and A. Sabry. Monadic encapsulation of effects: A revised approach (extended version). *J. Funct. Programming*, 11:591–627, 2001.
[10] T. Mossakowski, L. Schröder, and S. Goncharov. Completeness of monad-based dynamic logic. Technical report, University of Bremen, 2006.
[11] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries — The Revised Report*. Cambridge, 2003. Also: J. Funct. Programming **13** (2003).
[12] A. Pitts. Evaluation logic. In *Higher Order Workshop*, Workshops in Computing, pages 162–189. Springer, 1991.
[13] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in HasCasl. In *Fundamental Aspects of Software Engineering*, volume 2621 of *LNCS*, pages 261–277, 2003.
[14] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HasCasl. *J. Logic Comput.*, 14:571–619, 2004.
[15] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoret. Comput. Sci.*, 342:28–55, 2005.

A Appendix: Omitted Proofs

(Not for inclusion in the final version.)

Proof of Lemma 7 Rule ($\wedge$I) can be obtained from rule (cong) with empty sequence $\bar{r}$ by taking $\eta = y \wedge \psi, \chi = \phi, \xi = \top$ and applying ($\mathcal{E}$) and (η). Likewise, (wk) can be obtained from (cong) with empty sequence $\bar{r}$ by taking $\eta = y \to \psi, \chi = \top, \xi = \phi$ and applying ($\mathcal{E}$) and (η). Here and in the sequel, (intuitionistic) tautological reasoning is done tacitly, using ($\top$).

It remains to derive rule (app). First, note that by rule ($\top$), $\vdash [\![\bar{x} \leftarrow \bar{p}; \bar{y} \leftarrow \bar{q}]\!]\, \top$, from which we may derive $[\![\bar{x} \leftarrow \bar{p}; z \leftarrow \mathrm{ret}\langle \bar{x}, \top\rangle; \bar{y} \leftarrow \bar{q}]\!]\, \mathit{snd}(z)$. Applying (cong) to $[\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$ and $[\![\bar{x} \leftarrow \bar{p}; z \leftarrow \mathrm{ret}\langle \bar{x}, \top\rangle; \bar{y} \leftarrow \bar{q}]\!]\, \mathit{snd}(z)$, we obtain $[\![\bar{x} \leftarrow \bar{p}; z \leftarrow \mathrm{ret}\langle \bar{x}, \phi\rangle; \bar{y} \leftarrow \bar{q}]\!]\, \mathit{snd}(z)$, from which we obtain $[\![\bar{x} \leftarrow \bar{p}; \bar{y} \leftarrow \bar{q}]\!]\, \phi$. □

Proof of Theorem 10 (Soundness) Soundness of rules (η), (ctr) and (pre) has already been proved in [14]. Soundness of rules ($\top$) and ($\mathcal{E}$) is clear. Soundness of rule (cong) is seen as follows. Using only rules already proved sound, we have the derivation

$$\begin{aligned} & [\![\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\xi/y]]\!]\, \eta[\xi/y] \\ \vdash\dashv\ & [\![\bar{x} \leftarrow \bar{p}; v \leftarrow \mathrm{ret}\langle \bar{x}, \xi\rangle; \bar{z} \leftarrow \bar{r}']\!]\, \eta' \\ \vdash\dashv\ & [\![\bar{x} \leftarrow \bar{p}; u \leftarrow \mathrm{ret}\langle \bar{x}, \top\rangle; v \leftarrow \mathrm{ret}\langle \mathit{fst}(u), \mathit{snd}(u) \wedge \xi\rangle; \bar{z} \leftarrow \bar{r}']\!]\, \eta' \end{aligned}$$

where $\bar{r}' = \bar{r}[\mathit{snd}(v)/y, \mathit{fst}(v)/\bar{x}], \eta' = \eta[\mathit{snd}(v)/y, \mathit{fst}(v)/\bar{x}]$. Since do $\bar{x} \leftarrow \bar{p};\ \mathrm{ret}\langle \bar{x}, \xi \leftrightarrow \chi\rangle = \mathrm{do}\ \bar{x} \leftarrow \bar{p};\ \mathrm{ret}\langle \bar{x}, \top\rangle$, we can continue as follows:

$$\begin{aligned} & [\![\bar{x} \leftarrow \bar{p}; u \leftarrow \mathrm{ret}\langle \bar{x}, \xi \leftrightarrow \chi\rangle; v \leftarrow \mathrm{ret}\langle \mathit{fst}(u), \mathit{snd}(u) \wedge \xi\rangle; \bar{z} \leftarrow \bar{r}']\!]\, \eta' \\ \vdash\dashv\ & [\![\bar{x} \leftarrow \bar{p}; v \leftarrow \mathrm{ret}\langle \bar{x}, (\xi \leftrightarrow \chi) \wedge \xi\rangle; \bar{z} \leftarrow \bar{r}']\!]\, \eta'. \end{aligned}$$

Using the (intuitionistically valid) propositional equivalence of the formulae $(\xi \leftrightarrow \chi) \wedge \xi$ and $(\chi \leftrightarrow \xi) \wedge \chi$, we rewrite the last formula to

$$[\![\bar{x} \leftarrow \bar{p}; v \leftarrow \mathrm{ret}\langle \bar{x}, (\chi \leftrightarrow \xi) \wedge \chi\rangle; \bar{z} \leftarrow \bar{r}']\!]\, \eta'$$

and applying all the reasoning above backwards, with ξ and χ exchanged, finally obtain $[\![\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\chi/y]]\!]\, \eta[\chi/y]$. □

Proof of Lemma 13 Let t be product β-normal. We proceed by induction on the height n of t. The base case ($n = 1$) is clear. So let $n > 1$. Since the only way to form terms of T-free type from terms of non-T-free type is by application of *fst* or *snd*, we are done by induction unless t is such an application. By assumption, only two cases are possible: $t = e_1(\ldots(e_n(v))\ldots)$ or $t = e_1(\ldots(e_n(f(s)))\ldots)$, where $e_i \in \{\mathit{fst}, \mathit{snd}\}$, v is a variable, and f is a function symbol. In the first case, all the subterms, including v, have product types, which are not computational. In the second case, due to the general restriction on signature symbols, s must be of T-free type, so the induction hypothesis applies to s and we are done. □

Proof of Lemma 14 Structural induction on t.

- If t is a variable, put $s = t$ and $n = 0$.
- $t = \text{do } z \leftarrow q;\ r$. By induction, we have decompositions $q = s_1[t_1/x_1, \ldots t_k/x_k]$ and $r = s_2[t_{k+1}/x_{k+1}, \ldots t_n/x_n]$ as in the claim, and we can put $s = \text{do } z \leftarrow s_1;\ s_2$.
- The cases where t is a pair or an application of a projection or the return operator are similar.
- $t = f(q)$, where $f \in \Sigma$. By the general restriction on signatures, q must have T-free type. By Lemma 13, q does not contain return or binding operators. Therefore we can put $s = x_1$, $t_1 = f(q)$, and $n = 1$. □

Proof of Lemma 17 The 'only if' direction is trivial; we prove 'if'. Let $\phi : \Omega$, and let $\bar{x} \leftarrow \bar{p}$ be a program sequence. Assume w.l.o.g. that $y \notin FV(\Phi, \phi)$. Let $[\![\bar{z} \leftarrow \bar{q}]\!]\, \psi$ be the normalization of $[\![\bar{x} \leftarrow \bar{p}]\!]\, \phi$ according to Lemma 15. By the substitution rule, it now suffices to prove

$$[\![\bar{z} \leftarrow \bar{q}[t/y]]\!]\, \psi \vdash^{\Phi}\!\dashv [\![\bar{z} \leftarrow \bar{q}[s/y]]\!]\, \psi.$$

Since for all i, we have either $q_i = y$ or $y \notin q_i$, this can be achieved by several applications of the assumption — one for each i with $q_i = y$. □

Proof of the Truth Lemma (Lemma 18) *If:* Apply the assumption to the formula $[\![x \leftarrow z]\!]\, snd(x)$, where we can substitute $\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \psi \rangle$ and $\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \top \rangle$, respectively, for z.

Only if: By Lemma 17 it is enough to show

$$[\![\bar{w} \leftarrow \bar{s}; x \leftarrow (\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \psi \rangle); \bar{v} \leftarrow \bar{r}]\!]\, \phi \vdash^{\Phi}\!\dashv [\![\bar{w} \leftarrow \bar{s}; z \leftarrow (\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \top \rangle); \bar{v} \leftarrow \bar{r}]\!]\, \phi$$

for all sequences $\bar{w} \leftarrow \bar{s}$, $\bar{v} \leftarrow \bar{r}$ and all $\phi : \Omega$. By (pre), $\Phi \vdash [\![\bar{y} \leftarrow \bar{q}]\!]\, \psi$ implies $\Phi \vdash [\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}]\!]\, \psi$. Now the above claim is proved as follows.

$$\begin{aligned}
& [\![\bar{w} \leftarrow \bar{s}; x \leftarrow (\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \psi \rangle); \bar{v} \leftarrow \bar{r}]\!]\, \phi \\
\vdash^{\Phi}\!\dashv\ & [\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}; x \leftarrow \text{ret}\langle \bar{y}, \psi \rangle; \bar{v} \leftarrow \bar{r}]\!]\, \phi \\
\vdash^{\Phi}\!\dashv\ & [\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}; \bar{v} \leftarrow \bar{r}[\langle \bar{y}, \psi \rangle / x]]\!]\, \phi[\langle \bar{y}, \psi \rangle / x] \\
\vdash^{\Phi}\!\dashv\ & [\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}; \bar{v} \leftarrow \bar{r}[\langle \bar{y}, \top \rangle / x]]\!]\, \phi[\langle \bar{y}, \top \rangle / x] \\
\vdash^{\Phi}\!\dashv\ & [\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}; x \leftarrow \text{ret}\langle \bar{y}, \top \rangle; \bar{v} \leftarrow \bar{r}]\!]\, \phi \\
\vdash^{\Phi}\!\dashv\ & [\![\bar{w} \leftarrow \bar{s}; z \leftarrow (\text{do } \bar{y} \leftarrow \bar{q};\ \text{ret}\langle \bar{y}, \top \rangle); \bar{v} \leftarrow \bar{r}]\!]\, \phi.
\end{aligned}$$

The main point of this derivation is in the middle, where we apply rule (cong) in conjunction with $[\![\bar{w} \leftarrow \bar{s}; \bar{y} \leftarrow \bar{q}]\!]\, (\psi \leftrightarrow \top)$. □

Proof details of the completeness theorem It remains to prove that T is a simple monad. First we prove associativity — the unit laws can be proved analogously. We need to show Leibniz equality of terms do $u \leftarrow (\text{do } v \leftarrow q;\ s);\ r$ and do $v \leftarrow q; u \leftarrow s;\ r$. By Lemma 17, it suffices to prove

$$\begin{aligned} &[\bar{v} \leftarrow \bar{u}; x \leftarrow (\text{do } u \leftarrow (\text{do } v \leftarrow q;\ s);\ r); \bar{z} \leftarrow \bar{r}]\, \phi \\ \vdash^{\Phi}\!\dashv\ &[\bar{v} \leftarrow \bar{u}; x \leftarrow (\text{do } v \leftarrow q; u \leftarrow s;\ r); \bar{z} \leftarrow \bar{r}]\, \phi, \end{aligned}$$

for all $\phi : \Omega$ and program sequences $\bar{v} \leftarrow \bar{u}$, $\bar{z} \leftarrow \bar{r}$. Indeed, by (ctr):

$$\begin{aligned} &[\ldots; x \leftarrow (\text{do } u \leftarrow (\text{do } v \leftarrow q;\ s);\ r); \ldots]\, \phi \\ \vdash^{\Phi}\!\dashv\ &[\ldots; u \leftarrow (\text{do } v \leftarrow q;\ s); x \leftarrow r; \ldots]\, \phi \\ \vdash^{\Phi}\!\dashv\ &[\ldots; v \leftarrow q; u \leftarrow s; x \leftarrow r; \ldots]\, \phi \\ \vdash^{\Phi}\!\dashv\ &[\ldots; v \leftarrow q; x \leftarrow (\text{do } u \leftarrow s;\ r); \ldots]\, \phi \\ \vdash^{\Phi}\!\dashv\ &[\ldots; x \leftarrow (\text{do } v \leftarrow q; u \leftarrow s;\ r); \ldots]\, \phi \end{aligned}$$

In order to show that the monad is simple, suppose do $\bar{y} \leftarrow \bar{p};\ \text{ret}\,\psi \sim_{\Phi}$ do $\bar{y} \leftarrow \bar{p};\ \text{ret}\,\top$. By definition, we have in particular:

$$[x \leftarrow (\text{do } \bar{y} \leftarrow \bar{p};\ \text{ret}\,\psi)]\, x \vdash^{\Phi}\!\dashv [x \leftarrow (\text{do } \bar{y} \leftarrow \bar{p};\ \text{ret}\,\top)]\, x,$$

from which we obtain $[\bar{y} \leftarrow \bar{p}]\, \psi \vdash^{\Phi}\!\dashv [\bar{y} \leftarrow \bar{p}]\, \top$ by rules (ctr) and (η). Since $\vdash [\bar{y} \leftarrow \bar{p}]\, \top$, we arrive at $\Phi \vdash [\bar{y} \leftarrow \bar{p}]\, \psi$. The claim then follows by the Truth Lemma.