Behavioural Equivalence via Modalities for Algebraic Effects

Alex Simpson and Niels Voorneveld(✉)

Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
{Alex.Simpson,Niels.Voorneveld}@fmf.uni-lj.si

Abstract. The paper investigates behavioural equivalence between programs in a call-by-value functional language extended with a signature of (algebraic) effect-triggering operations. Two programs are considered as being behaviourally equivalent if they enjoy the same behavioural properties. To formulate this, we define a logic whose formulas specify behavioural properties. A crucial ingredient is a collection of *modalities* expressing effect-specific aspects of behaviour. We give a general theory of such modalities. If two conditions, *openness* and *decomposability*, are satisfied by the modalities then the logically specified behavioural equivalence coincides with a modality-defined notion of applicative bisimilarity, which can be proven to be a congruence by a generalisation of Howe's method. We show that the openness and decomposability conditions hold for several examples of algebraic effects: nondeterminism, probabilistic choice, global store and input/output.

1 Introduction

The notion of *behavioural equivalence* between programs is a fundamental concept in the theory of programming languages. A conceptually natural approach to defining behavioural equivalence is to consider two programs as being equivalent if they enjoy the same 'behavioural properties'. This can be made precise by specifying a *behavioural logic* whose formulas express behavioural properties. Two programs M, N are then defined to be equivalent if, for all formulas Φ, it holds that $M \models \Phi$ iff $N \models \Phi$ (where $M \models \Phi$ expresses the satisfaction relation: program M enjoys property Φ).

This logical approach to defining behavioural equivalence has been particularly prominent in concurrency theory, where the classic result is that the equivalence defined by Hennessy-Milner logic [4] coincides with bisimilarity [14,17]. The aim of the present paper is to adapt the logical approach to the very different computational paradigm of *applicative programming with effects*.

A. Simpson—Supported by the Slovenian Research Agency, research core funding No. P1–0294.

N. Voorneveld—Supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326, and by EU-MSCA-RISE project 731143 (CID).

A. Ahmed (Ed.): ESOP 2018, LNCS 10801, pp. 300–326, 2018.
https://doi.org/10.1007/978-3-319-89884-1_11

More precisely, we consider a call-by-value functional programming language with *algebraic effects* in the sense of Plotkin and Power [21]. Broadly speaking, effects are those aspects of computation that involve a program interacting with its 'environment'; for example: nondeterminism, probabilistic choice (in both cases, the choice is deferred to the environment); input/output; mutable store (the machine state is modified); control operations such as exceptions, jumps and handlers (which interact with the continuation in the evaluation process); etc. Such general effects collectively enjoy common properties identified in the work of Moggi on monads [15]. Among them, algebraic effects play a special role. They can be included in a programming language by adding effect-triggering operations, whose 'algebraic' nature means that effects act independently of the continuation. From the aforementioned examples of effects, only jumps and handlers are non-algebraic. Thus the notion of algebraic effect covers a broad range of effectful computational behaviour. Call-by-value functional languages provide a natural context for exploring effectful programming. From a theoretical viewpoint, other programming paradigms are subsumed; for example, imperative programs can be recast as effectful functional ones. From a practical viewpoint, the combination of effects with call-by-value leads to the natural programming style supported by impure functional languages such as OCaml.

In order to focus on the main contributions of the paper (the behavioural logic and its induced behavioural equivalence), we instantiate "call-by-value functional language with algebraic effects" using a very simple language. Our language is a simply-typed λ-calculus with a base type of natural numbers, general recursion, call-by-value function evaluation, and algebraic effects, similar to [21]; although, for technical convenience, we adopt the (equivalent) formulation of fine-grained call-by-value [13]. The language is defined precisely in Sect. 2. Following [8,21], an operational semantics is given that evaluates programs to *effect trees*.

Section 3 introduces the behavioural logic. In our impure functional setting, the evaluation of a program of type τ results in a computational process that may or may not invoke effects, and which may or may not terminate with a return *value* of type τ. The key ingredient in our logic is an effect-specific family $\mathcal{O}$ of *modalities*, where each modality $o \in \mathcal{O}$ converts a property ϕ of values of type τ to a property $o\,\phi$ of general programs (called *computations*) of type τ. The idea is that such modalities capture all relevant effect-specific behavioural properties of the effects under consideration.

A main contribution of the paper is to give a general framework for defining such effect modalities, applicable across a wide range of algebraic effects. The general setting is that we have a signature Σ of effect operations, which determines the programming language, and a collection $\mathcal{O}$ of modalities, which determines the behavioural logic. In order to specify the semantics of the logic, we require each modality to be assigned a set of unit-type effect trees, which determines the meaning of the modality. Several concrete examples and a detailed general explanation are given in Sect. 3.

In Sect. 4, we consider the relation of *behavioural equivalence* between programs determined by the logic. A fundamental well-behavedness property is that

any reasonable program equivalence should be a congruence with respect to the syntactic constructs of the programming language. Our main theorem (Theorem 1) is that, under two conditions on the collection $\mathcal{O}$ of modalities, which hold for all the examples of effects we consider, the logically induced behavioural equivalence is indeed a congruence.

In order to prove Theorem 1, we develop an alternative perspective on behavioural equivalence, which is of interest in its own right. In Sect. 5 we show how the modalities $\mathcal{O}$ determine a relation of *applicative $\mathcal{O}$-bisimilarity*, which is an effect-sensitive version of Abramsky's notion of *applicative bisimilarity* [1]. Theorem 2 shows that applicative $\mathcal{O}$-bisimilarity coincides with the logically defined relation of behavioural equivalence.

The proof of Theorem 1 is then concluded in Sect. 6, where we use Howe's method [5,6] to show that applicative $\mathcal{O}$-bisimilarity is a congruence. Although the proof is technically involved, we give only a brief outline, as the details closely follow the recent paper [9], in which Howe's method is applied to an untyped language with general algebraic effects.

In Sect. 7, we present a variation on our behavioural logic, in which we make the syntax of logical formulas independent of the syntax of the programming language.

Finally, in Sect. 8 we discuss related and further work.

2 A Simple Programming Language

As motivated in the introduction, our chosen base language is a simply-typed call-by-value functional language with general recursion and a ground type of natural numbers, to which we add (algebraic) effect-triggering operations. This means that our language is a call-by-value variant of PCF [20], extended with algebraic effects, resulting in a language similar to the one considered in [21]. In order to simplify the technical treatment of the language, we present it in the style of *fine-grained call-by-value* [13]. This means that we make a syntactic distinction between *values* and *computations*, representing the static and dynamic aspects of the language respectively. Furthermore, all *sequencing* of computations is performed using a single language construct, the **let** construct. The resulting language is straightforwardly intertranslatable with the more traditional call-by-value formulation. But the encapsulation of all sequencing within a single construct has the benefit of avoiding redundancy in proofs.

Our types are just the simple types obtained by iterating the function type construction over two base types: $\mathbf{N}$ of natural numbers, and also a unit type $\mathbf{1}$.

Types: $\tau, \rho ::= \mathbf{1} \mid \mathbf{N} \mid \rho \to \tau$
Contexts: $\Gamma ::= \emptyset \mid \Gamma, x : \tau$

As usual, term variables x are taken from a countably-infinite stock of such variables, and the context $\Gamma, x : \tau$ can only be formed if the variable x does not already appear in Γ.

As discussed above, program terms are separated into two mutually defined but disjoint categories: *values* and *computations*.

Values: $V, W ::= * \mid Z \mid S(V) \mid \lambda x.M \mid x$
Computations: $M, N ::= VW \mid \mathbf{return}\ V \mid \mathbf{let}\ \ M \Rightarrow x\ \mathbf{in}\ \ N \mid \mathbf{fix}\ (V) \mid$
$\mathbf{case}\ V\ \ \mathbf{in}\ \ \{Z \Rightarrow M, S(x) \Rightarrow N\}$

Here, $*$ is the unique value of the unit type. The values of the type of natural numbers are the *numerals* represented using zero Z and successor S. The values of function type are the λ-abstractions. And a variable x can be considered a value, because, under the call-by-value evaluation strategy of the language, it can only be instantiated with a value.

The computations are: function application VW; the computation that does nothing but return a value V; a **let** construct for sequencing; a **fix** construct for recursive definition; and a **case** construct that branches according to whether its natural-number argument is zero or positive. The computation $\mathbf{let}\ M \Rightarrow x\ \mathbf{in}\ N$ implements sequencing in the following sense. First the computation M is evaluated. Only in the case that the evaluation of M terminates, with return value V, does the thread of execution continue to N. In this case, the computation $N[V/x]$ is evaluated, and its return value (if any) is the one returned by the **let** construct.

To the pure functional language described above, we add *effect operations*. The collection of effect operations is specified by a set Σ (the *signature*) of such operations, together with, for each $\sigma \in \Sigma$ an associated *arity* which takes one of the four forms below

$$\alpha^n \to \alpha \qquad \mathbf{N} \times \alpha^n \to \alpha \qquad \alpha^{\mathbf{N}} \to \alpha \qquad \mathbf{N} \times \alpha^{\mathbf{N}} \to \alpha.$$

The notation here is chosen to be suggestive of the way in which such arities are used in the typing rules below, viewing α as a type variable. Each of the forms of arity has an associated term constructor, for building additional computation terms, with which we extend the above grammar for computation terms.

Effects: $\sigma(M_0, M_1, \ldots, M_{n-1}) \mid \sigma(V; M_0, M_1, \ldots, M_{n-1}) \mid \sigma(V) \mid \sigma(W; V)$

Motivating examples of effect operations and their computation terms can be found in Examples 0–5 below.

The typing rules for the language are given in Fig. 1 below. Note that the choice of typing rule for an effect operation $\sigma \in \Sigma$ depends on its declared arity.

The terms of type τ are the values and computations generated by the constructors above. Every term has a unique *aspect* as either a value or computation. We write $Val(\tau)$ and $Com(\tau)$ respectively for closed values and computations. So the closed terms of τ are $Term(\tau) = Val(\tau) \cup Com(\tau)$. For $n \in \mathbb{N}$ a natural number, we write $\overline{n}$ for the numeral $S^n(Z)$, hence $Val(\mathbf{N}) := \{\overline{n} \mid n \in \mathbb{N}\}$.

We now consider some standard signatures of computationally interesting effect operations, which will be used as running examples throughout the paper. (We use the same examples as in [8].)

Example 0 (Pure functional computation). This is the trivial case (from an effect point of view) in which the signature Σ of effect operations is empty. The resulting language is a call-by-value variant of PCF [20].

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{}{\Gamma \vdash * : \mathbf{1}} \qquad \frac{}{\Gamma \vdash Z : \mathbf{N}} \qquad \frac{\Gamma \vdash V : \mathbf{N}}{\Gamma \vdash S(V) : \mathbf{N}}$$

$$\frac{\Gamma \vdash V : \tau}{\Gamma \vdash \mathbf{return}(V) : \tau} \qquad \frac{\Gamma, x : \tau \vdash M : \rho}{\Gamma \vdash (\lambda x : \tau . M) : \tau \to \rho}$$

$$\frac{\Gamma \vdash V : \tau \to \rho \qquad \Gamma \vdash W : \tau}{\Gamma \vdash (VW) : \rho} \qquad \frac{\Gamma \vdash V : (\tau \to \rho) \to (\tau \to \rho)}{\Gamma \vdash \mathbf{fix}(V) : \tau \to \rho}$$

$$\frac{\Gamma \vdash V : \mathbf{N} \qquad \Gamma \vdash M : \tau \qquad \Gamma, x : \mathbf{N} \vdash N : \tau}{\Gamma \vdash \ \mathbf{case}\ V\ \mathbf{of}\ \{Z \Rightarrow M; S(x) \Rightarrow N\} : \tau} \qquad \frac{\Gamma \vdash M : \tau \qquad \Gamma, x : \tau \vdash N : \rho}{\Gamma \vdash \mathbf{let}\ M \Rightarrow x\ \mathbf{in}\ N : \rho}$$

$$\frac{\sigma : \alpha^n \to \alpha \qquad \Gamma \vdash M_i : \tau}{\Gamma \vdash \sigma(M_0, M_1, \ldots, M_{n-1}) : \tau} \qquad \frac{\sigma : \alpha^{\mathbf{N}} \to \alpha \qquad \Gamma \vdash V : \mathbf{N} \to \tau}{\Gamma \vdash \sigma(V) : \tau}$$

$$\frac{\sigma : \mathbf{N} \times \alpha^n \to \alpha \qquad \Gamma \vdash V : \mathbf{N} \qquad \Gamma \vdash M_i : \tau}{\Gamma \vdash \sigma(V; M_0, M_1, \ldots, M_{n-1}) : \tau}$$

$$\frac{\sigma : \mathbf{N} \times \alpha^{\mathbf{N}} \to \alpha \qquad \Gamma \vdash V : \mathbf{N} \qquad \Gamma \vdash W : \mathbf{N} \to \tau}{\Gamma \vdash \sigma(V; W) : \tau}$$

Fig. 1. Typing rules

Example 1 (Error). We take a set of error labels E. For each $e \in E$ there is an effect operator $\mathit{raise}_e : \alpha^0 \to \alpha$ which, when invoked by the computation $\mathit{raise}_e()$, aborts evaluation and outputs e as an error message.

Example 2 (Nondeterminism). There is a binary choice operator $\mathit{or} : \alpha^2 \to \alpha$ which gives two options for continuing the computation. The choice of continuation is under the control of some external agent, which one may wish to model as being cooperative (*angelic*), antagonistic (*demonic*), or *neutral*.

Example 3 (Probabilistic choice). Again there is a single binary choice operator $\mathit{p\text{-}or} : \alpha^2 \to \alpha$ which gives two options for continuing the computation. In this case, the choice of continuation is probabilistic, with a $\frac{1}{2}$ probability of either option being chosen. Other weighted probabilistic choices can be programmed in terms of this fair choice operation.

Example 4 (Global store). We take a set of locations L for storing natural numbers. For each $l \in L$ we have $\mathit{lookup}_l : \alpha^{\mathbf{N}} \to \alpha$ and $\mathit{update}_l : \mathbf{N} \times \alpha \to \alpha$. The computation $\mathit{lookup}_l(V)$ looks up the number at location l and passes it as an argument to the function V, and $\mathit{update}_l(\overline{n}; M)$ stores n at l and then continues with the computation M.

Example 5 (Input/output). Here we have two operators, $\mathit{read} : \alpha^{\mathbf{N}} \to \alpha$ which reads a number from an input channel and passes it as the argument to a function, and $\mathit{write} : \mathbf{N} \times \alpha \to \alpha$ which outputs a number (the first argument) and then continues as the computation given as the second argument.

We next present an operational semantics for our language, under which a computation term evaluates to an *effect tree*: essentially, a coinductively generated term using operations from Σ, and with values and $\perp$ (nontermination) as

the generators. This idea appears in [8,21], and our technical treatment follows approach of the latter, adapted to call-by-value.

We define a single-step reduction relation $\rightarrowtail$ between configurations (S, M) consisting of a stack S and a computation M. The computation M is the term under current evaluation. The stack S represents a continuation computation awaiting the termination of M. First, we define a stack-independent reduction relation on computation terms that do not involve **let** at the top level.

$$
\begin{array}{l}
(\lambda x : \tau.M)V \;\rightsquigarrow\; M[V/x] \\
\textbf{case}\ Z\ \textbf{of}\ \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\} \;\rightsquigarrow\; M_1 \\
\textbf{case}\ S(V)\ \textbf{of}\ \{Z \Rightarrow M_1; S(x) \Rightarrow M_2\} \;\rightsquigarrow\; M_2[V/x] \\
\textbf{fix}(F) \;\rightsquigarrow\; \textbf{return}\ \lambda x : \tau.\ \textbf{let}\ F(\lambda y : \tau.\textbf{let fix}\ F \Rightarrow z\ \textbf{in}\ zy) \Rightarrow w\ \textbf{in}\ wx
\end{array}
$$

The behaviour of **let** is implemented using a system of stacks where:

$$\textbf{Stacks}\ S ::= \ id \mid S \circ (\ \textbf{let}\ \ (-) \Rightarrow x\ \ \textbf{in}\ \ M)$$

We write $S\{N\}$ for the computation term obtained by 'applying' the stack S to N, defined by:

$$
\begin{array}{l}
id\,\{N\} \;=\; N \\
(S \circ (\textbf{let}\ (-) \Rightarrow x\ \ \textbf{in}\ \ M))\,\{N\} \;=\; S\{\ \textbf{let}\ N \Rightarrow x\ \ \textbf{in}\ \ M\}
\end{array}
$$

We write $\mathit{Stack}(\tau, \rho)$ for the set of stacks S such that for any $N \in \mathit{Com}(\tau)$, it holds that $S\{N\}$ is a well-typed expression of type ρ. We define a reduction relation on pairs $\mathit{Stack}(\tau, \rho) \times \mathit{Com}(\tau)$ (denoted $(S_1, M_1) \rightarrowtail (S_2, M_2)$) by:

$$
\begin{array}{ll}
(S,\ \textbf{let}\ N \Rightarrow x\ \ \textbf{in}\ \ M) \;\rightarrowtail\; (S \circ (\ \textbf{let}\ \ (-) \Rightarrow x\ \ \textbf{in}\ \ M), N) & \\
(S, R) \;\rightarrowtail\; (S, R') & \text{if } R \rightsquigarrow R' \\
(S \circ (\ \textbf{let}\ \ (-) \Rightarrow x\ \textbf{in}\ M), \textbf{return}\ V) \;\rightarrowtail\; (S, M[V/x]) &
\end{array}
$$

We define the notion of *effect tree* for an arbitrary set X, where X is thought of as a set of abstract 'values'.

Definition 1. An *effect tree* (henceforth *tree*), over a set X, determined by a signature Σ of effect operations, is a labelled and possibly infinite tree whose nodes have the possible forms.

1. A leaf node labelled with $\bot$ (the symbol for nontermination).
2. A leaf node labelled with x where $x \in X$.
3. A node labelled σ with children $t_0, \ldots, t_{n-1}$, when $\sigma \in \Sigma$ has arity $\alpha^n \to \alpha$.
4. A node labelled σ with children $t_0, t_1, \ldots$, when $\sigma \in \Sigma$ has arity $\alpha^{\mathbf{N}} \to \alpha$.
5. A node labelled σ_m where $m \in \mathbb{N}$ with children $t_0, \ldots, t_{n-1}$, when $\sigma \in \Sigma$ has arity $\mathbf{N} \times \alpha^n \to \alpha$.
6. A node labelled σ_m where $m \in \mathbb{N}$ with children $t_0, t_1, \ldots$, when $\sigma \in \Sigma$ has arity $\mathbf{N} \times \alpha^{\mathbf{N}} \to \alpha$.

We write TX for the set of trees over X. We define a partial ordering on TX where $t_1 \leq t_2$, if t_1 can be obtained by replacing subtrees of t_2 by $\bot$. This forms an *ω-complete* partial order, meaning that every ascending sequence $t_1 \leq t_2 \leq \ldots$ has a least upper bound $\bigsqcup_n t_n$. Let $\mathit{Tree}(\tau) := T\,\mathit{Val}(\tau)$, we will define a reduction relation from computations to trees of values.

Given $f : X \to Y$ and a tree $t \in TX$, we write $t[x \mapsto f(x)] \in TY$ for the tree whose leaves $x \in X$ are renamed to $f(x)$. We have a function $\mu : TTX \to TX$, which takes a tree r of trees and flattens it to a tree $\mu r \in TX$, by taking the labelling tree at each non-$\bot$ leaf of r as the subtree at the corresponding node in μr. The function μ is the multiplication associated with the monad structure of the T operation. The unit of the monad is the map $\eta : X \to TX$ which takes an element $x \in X$ and returns a leaf labelled x.

The operational mapping from a computation $M \in \mathit{Com}(\tau)$ to an effect tree is defined intuitively as follows. Start evaluating the M in the empty stack id, until the evaluation process (which is deterministic) terminates (if this never happens the tree is $\bot$). If the evaluation process terminates at a configuration of the form $(\mathit{id}, \mathbf{return}\ V)$ then the tree is the leaf V. Otherwise the evaluation process can only terminate at a configuration of the form $(S, \sigma(\ldots))$ for some effect operation $\sigma \in \Sigma$. In this case, create an internal node in the tree of the appropriate kind (depending on σ) and continue generating each child tree of this node by repeating the above process by evaluating an appropriate continuation computation, starting from a configuration with the current stack S.

The following (somewhat technical) definition formalises the idea outlined above in a mathematically concise way. We define a family of maps $|-,-|_{(-)} : \mathit{Stack}(\tau,\rho) \times \mathit{Com}(\tau) \times \mathbb{N} \to \mathit{Tree}(\rho)$ indexed over τ, and ρ by:

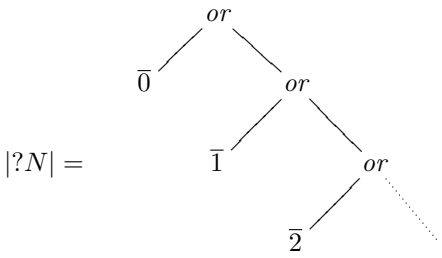
$$|S, M|_0 = \bot$$

$$|S, M|_{n+1} = \begin{cases} V & \text{if } S = \mathit{id} \wedge M = \mathbf{return}\ V \\ |S', M'|_n & \text{if } (S, M) \rightarrowtail (S', M') \\ \sigma(|S, M_0|_n, \ldots, |S, M_{m-1}|_n) & \sigma : \alpha^m \to \alpha,\ M = \sigma(M_0, \ldots, M_{m-1}) \\ \sigma(|S, V\overline{0}|_n, |S, V\overline{1}|_n, \ldots) & \sigma : \alpha^{\mathbf{N}} \to \alpha,\ M = \sigma(V) \\ \sigma_k(|S, M_0|_n, \ldots, |S, M_{m-1}|_n) & \sigma : \mathbf{N} \times \alpha^m \to \alpha,\ M = \sigma(\overline{k}, M_0, \ldots, M_{m-1}) \\ \sigma_k(|S, V\overline{0}|_n, |S, V\overline{1}|_n, \ldots) & \sigma : \mathbf{N} \times \alpha^{\mathbf{N}} \to \alpha,\ M = \sigma(\overline{k}, V) \\ \bot & \text{otherwise} \end{cases}$$

It follows that $|S, M|_n \leq |S, M|_{n+1}$ in the given ordering on trees. We write $|-|_{(-)} : \mathit{Com}(\tau) \times \mathbb{N} \to \mathit{Tree}(\tau)$ for the function defined by $|M|_n = |\mathit{id}, M|_n$. Using this we can give the operational interpretation of computation terms as effect trees by defining $|-| : \mathit{Com}(\tau) \to \mathit{Tree}(\tau)$ by $|M| := \bigsqcup_n |M|_n$.

Example 3 (Nondeterminism). Nondeterministically generate a natural number:

$$?N := \mathbf{let}\ \mathbf{fix}(\lambda x : \mathbf{1} \to \mathbf{N}.\ \mathit{or}(\lambda y : \mathbf{1}.\, Z,\ \lambda y : \mathbf{1}.\ \mathbf{let}\ xy \Rightarrow z\ \mathbf{in}\ S(z))) \Rightarrow w\ \mathbf{in}\ w*$$



3 Behavioural Logic and Modalities

The goal of this section is to motivate and formulate a logic for expressing *behavioural properties* of programs. In our language, program means (well-typed) term, and we shall be interested both in properties of *computations* and in properties of *values*. Accordingly, we define a logic that contains both *value formulas* and *computation formulas*. We shall use lower case Greek letters $\phi, \psi, \ldots$ for the former, and upper case Greek letters $\Phi, \Psi, \ldots$ for the latter. Our logic will thus have two satisfaction relations

$$V \models \phi \qquad\qquad M \models \Phi$$

which respectively assert that "value V enjoys the value property expressed by ϕ" and "computation M enjoys the computation property expressed by Φ".

In order to motivate the detailed formulation of the logic, it is useful to identify criteria that will guide the design.

(C1) The logic should express only 'behaviourally meaningful' properties of programs. This guides us to build the logic upon primitive notions that have a direct behavioural interpretation according to a natural understanding of program behaviour.

(C2) The logic should be as expressive as possible within the constraints imposed by criterion (C1).

For every type τ, we define a collection $VF(\tau)$ of *value formulas*, and a collection $CF(\tau)$ of *computation formulas*, as motivated above.

Since boolean logical connectives say nothing themselves about computational behaviour, it is a reasonable general principle that 'behavioural properties' should be closed under such connectives. Thus, in keeping with criterion (C2), which asks for maximal expressivity, we close each set $CF(\tau)$ and $VF(\tau)$, of computation and value formulas, under infinitary propositional logic.

In addition to closure under infinitary propositional logic, each set $VF(\tau)$ contains a collection of *basic* value formulas, from which compound formulas are constructed using (infinitary) propositional connectives.[1] The choice of basic formulas depends on the type τ.

[1] We call such formulas *basic* rather than *atomic* because they include formulas such as $(V \mapsto \Phi)$, discussed below, which are built from other formulas.

In the case of the natural numbers type, we include a basic value formula $\{n\} \in VF(\mathbf{N})$, for every $n \in \mathbb{N}$. The semantics of this formula are given by:

$$V \models \{n\} \quad \Leftrightarrow \quad V = \overline{n}.$$

By the closure of $VF(\mathbf{N})$ under infinitary disjunctions, every subset of $\mathbb{N}$ can be represented by some value formula. Moreover, since a general value formula in $VF(\mathbf{N})$ is an infinitary boolean combination of basic formulas of the form $\{n\}$, the value formulas represent exactly the subsets on $\mathbb{N}$.

For the unit type, we do not require any basic value formulas. The unit type has only one value, $*$. The two subsets of this singleton set of values are defined by the formulas $\bot$ ('falsum', given as an empty disjunction), and $\top$ (the truth constant, given as an empty conjunction).

For a function type $\tau \to \rho$, we want each basic formula to express a fundamental behavioural constraint on values (i.e., λ-abstractions) W of type $\tau \to \rho$. In keeping with the applicative nature of functional programming, the only way in which a λ-abstraction can be used to generate behaviour is to apply it to an argument of type τ, which, because we are in a call-by-value setting, must be a value V. The application of W to V results in a computation WV of type ρ, whose properties can be probed using computation formulas in $CF(\rho)$. Based on this, for every value $V \in Val(\tau)$ and computation formula $\Phi \in CF(\rho)$, we include a basic value formula $(V \mapsto \Phi) \in VF(\tau \to \rho)$ with the semantics:

$$W \models (V \mapsto \Phi) \quad \Leftrightarrow \quad WV \models \Phi.$$

Using this simple construct, based on application to a single argument V, other natural mechanisms for expressing properties of λ-abstractions are definable, using infinitary propositional logic. For example, given $\phi \in VF(\tau)$ and $\Psi \in CF(\rho)$, the definition

$$(\phi \mapsto \Psi) \;:=\; \bigwedge \{(V \mapsto \Psi) \mid V \in Val(\tau),\, V \models \phi\} \tag{1}$$

defines a formula whose derived semantics is

$$W \models (\phi \mapsto \Psi) \quad \Leftrightarrow \quad \forall V \in Val(\tau).\; V \models \phi \text{ implies } WV \models \Psi. \tag{2}$$

In Sect. 7, we shall consider the possibility of changing the basic value formulas in $VF(\tau \to \rho)$ to formulas $(\phi \mapsto \Psi)$.

It remains to explain how the basic computation formulas in $CF(\tau)$ are formed. For this we require a given set $\mathcal{O}$ of *modalities*, which depends on the algebraic effects contained in the language. The basic computation formulas in $CF(\tau)$ then have the form $o\,\phi$, where $o \in \mathcal{O}$ is one of the available modalities, and ϕ is a value formula in $VF(\tau)$. Thus a modality 'lifts' properties of values of type τ to properties of computations of type τ.

In order to give semantics to computation formulas $o\,\phi$, we need a general theory of the kind of modality under consideration. This is one of the main contributions of the paper. Before presenting the general theory, we first consider motivating examples, using our running examples of algebraic effects.

Example 0 (Pure functional computation). Define $\mathcal{O} = \{\downarrow\}$. Here the single modality $\downarrow$ is the *termination modality*: $\downarrow\phi$ asserts that a computation terminates with a return value V satisfying ϕ. This is formalised using effect trees:

$$M \models \downarrow\phi \quad \Leftrightarrow \quad |M| \text{ is a leaf } V \text{ and } V \models \phi.$$

Note that, in the case of pure functional computation, all trees are leaves: either value leaves V, or nontermination leaves $\bot$.

Example 1 (Error). Define $\mathcal{O} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in E\}$. The semantics of the termination modality $\downarrow$ is defined as above. The *error modality* E_e flags error e:

$$M \models \mathsf{E}_e\phi \quad \Leftrightarrow \quad |M| \text{ is a node labelled with } \mathit{raise}_e.$$

(Because raise_e is an operation of arity 0, a raise_e node in a tree has 0 children.) Note that the semantics of $\mathsf{E}_e\phi$ makes no reference to ϕ. Indeed it would be natural to consider E_e as a basic computation formula in its own right, which could be done by introducing a notion of 0-argument modality, and considering E_e as such. In this paper, however, we keep the treatment uniform by always considering modalities as unary operations, with natural 0-argument modalities subsumed as unary modalities with redundant argument.

Example 2 (Nondeterminism). Define $\mathcal{O} = \{\Diamond, \Box\}$ with:

$$M \models \Diamond\phi \quad \Leftrightarrow \quad |M| \text{ has some leaf } V \text{ such that } V \models \phi$$
$$M \models \Box\phi \quad \Leftrightarrow \quad |M| \text{ has finite height and every leaf is a value } V \text{ s.t. } V \models \phi.$$

Including both modalities amounts to a neutral view of nondeterminism. In the case of angelic nondeterminism, one would include just the $\Diamond$ modality; in that of demonic nondeterminism, just the $\Box$ modality. Because of the way the semantic definitions interact with termination, the modalities $\Box$ and $\Diamond$ are not De Morgan duals. Indeed, each of the three possibilities $\{\Diamond, \Box\}, \{\Diamond\}, \{\Box\}$ for $\mathcal{O}$ leads to a logic with a different expressivity.

Example 3 (Probabilistic choice). Define $\mathcal{O} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q < 1\}$ with:

$$M \models \mathsf{P}_{>q}\,\phi \quad \Leftrightarrow \quad \mathbf{P}(|M| \text{ terminates with a value in } \{V \mid V \models \phi\}) > q,$$

where the probability on the right is the probability that a run through the tree $|M|$, starting at the root, and making an independent fair probabilistic choice at each branching node, terminates at a value node with a value V in the set $\{V \mid V \models \phi\}$. We observe that the restriction to rational thresholds q is immaterial, as, for any real r with $0 \leq r < 1$, we can define:

$$\mathsf{P}_{>r}\,\phi \;:=\; \bigvee\{\mathsf{P}_{>q}\,\phi \mid q \in \mathbb{Q}, r < q < 1\}.$$

Similarly, we can define non-strict threshold modalities, for $0 < r \leq 1$, by:

$$\mathsf{P}_{\geq r}\,\phi \;:=\; \bigwedge\{\mathsf{P}_{>q}\,\phi \mid q \in \mathbb{Q}, 0 \leq q < r\}.$$

Also, we can exploit negation to define modalities expressing strict and non-strict upper bounds on probabilities. Notwithstanding the definability of non-strict and upper-bound thresholds, we shall see later that it is important that we include only strict lower-bound modalities in our set $\mathcal{O}$ of primitive modalities.

Example 4 (Global store). For a set of locations L, define the set of states by $State = \mathbb{N}^L$. The modalities are $\mathcal{O} = \{(s \rightarrowtail r) \mid s, r \in State\}$, where informally:

$$M \models (s \rightarrowtail r)\,\phi \quad \Leftrightarrow \quad \text{the execution of } M, \text{ starting in state } s, \text{ terminates in final state } r \text{ with return value } V \text{ such that } V \models \phi.$$

We make the above definition precise using the effect tree of M. Define

$$exec : TX \times State \to X \times State,$$

for any set X, to be the least partial function satisfying:

$$exec(t,s) = \begin{cases} (x,s) & \text{if } t \text{ is a leaf labelled with } x \in X \\ exec(t_{s(l)}, s) & \text{if } t = lookup_l(t_0, t_1, \cdots) \text{ and } exec(t_{s(l)}, s) \text{is defined} \\ exec(t', s[l := n]) & \text{if } t = update_{l,n}(t') \text{ and } exec(t', s[l := n]) \text{ is defined,} \end{cases}$$

where $s[l := n]$ is the evident modification of state s. Intuitively, $exec(t, s)$ defines the result of "executing" the tree of commands in effect tree t starting in state s, whenever this execution terminates. In terms of operational semantics, it can be viewed as defining a 'big-step' semantics for effect trees (in the signature of global store). We can now define the semantics of the $(s \rightarrowtail r)$ modality formally:

$$M \models (s \rightarrowtail r)\,\phi \quad \Leftrightarrow \quad exec(|M|, s) = (V, r) \text{ where } V \models \phi.$$

Example 5 (Input/output). Define an *i/o-trace* to be a word w over the alphabet

$$\{?n \mid n \in \mathbb{N}\} \cup \{!n \mid n \in \mathbb{N}\}.$$

The idea is that such a word represents an input/output sequence, where $?n$ means the number n is given in response to an input prompt, and $!n$ means that the program outputs n. Define the set of modalities

$$\mathcal{O} = \{\langle w\rangle\!\downarrow, \langle w\rangle_{\ldots} \mid w \text{ an i/o-trace}\}.$$

The intuitive semantics of these modalities is as follows.

$$\begin{aligned} M \models \langle w\rangle\!\downarrow \phi \quad &\Leftrightarrow \quad w \text{ is a complete i/o-trace for the execution of } M \\ &\qquad \text{resulting in termination with } V \text{ s.t. } V \models \phi \\ M \models \langle w\rangle_{\ldots}\,\phi \quad &\Leftrightarrow \quad w \text{ is an initial i/o-trace for the execution of } M. \end{aligned}$$

In order to define the semantics of formulas precisely, we first define relations $t \models \langle w\rangle\!\downarrow P$ and $t \models \langle w\rangle_{\ldots}$, between $t \in TX$ and $P \subseteq X$, by induction on words

$$\frac{n \in \mathbb{N}}{\{n\} \in VF(\mathbf{N})}(1) \qquad \frac{V : \tau \qquad \Phi \in CF(\rho)}{(V \mapsto \Phi) \in VF(\tau \to \rho)}(2) \qquad \frac{\phi \in VF(\tau) \qquad o \in \mathcal{O}}{o\,\phi \in CF(\tau)}(3)$$

$$\frac{\phi : I \to VF(\tau)}{\bigvee_I \phi \in VF(\tau)}(4) \qquad \frac{\phi : I \to VF(\tau)}{\bigwedge_I \phi \in VF(\tau)}(5) \qquad \frac{\phi \in VF(\tau)}{\neg\phi \in VF(\tau)}(6)$$

$$\frac{\Phi : I \to CF(\tau)}{\bigvee_I \Phi \in CF(\tau)}(7) \qquad \frac{\Phi : I \to CF(\tau)}{\bigwedge_I \Phi \in CF(\tau)}(8) \qquad \frac{\Phi \in CF(\tau)}{\neg\Phi \in CF(\tau)}(9)$$

Fig. 2. The logic $\mathcal{V}$

(Note that we are overloading the $\models$ symbol.) In the following, we write ε for the empty word, and we use textual juxtaposition for concatenation of words.

$$\begin{aligned}
t \;&\models\; \langle\varepsilon\rangle{\downarrow}\, P &&\Leftrightarrow\quad t \text{ is a leaf } x \text{ and } x \in P\\
t \;&\models\; \langle(?n)\, w\rangle{\downarrow}\, P &&\Leftrightarrow\quad t = \mathit{read}(t_0, t_1, \dots) \text{ and } t_n \;\models\; \langle w\rangle{\downarrow}\, P\\
t \;&\models\; \langle(!n)\, w\rangle{\downarrow}\, P &&\Leftrightarrow\quad t = \mathit{write}_n(t') \text{ and } t' \models\; \langle w\rangle{\downarrow}\, P\\
t \;&\models\; \langle\varepsilon\rangle_{\dots} &&\Leftrightarrow\quad \text{true}\\
t \;&\models\; \langle(?n)\, w\rangle_{\dots} &&\Leftrightarrow\quad t = \mathit{read}(t_0, t_1, \dots) \text{ and } t_n \;\models \langle w\rangle_{\dots}\\
t \;&\models\; \langle(!n)\, w\rangle_{\dots} &&\Leftrightarrow\quad t = \mathit{write}_n(t') \text{ and } t' \;\models\; \langle w\rangle_{\dots}
\end{aligned}$$

The formal semantics of modalities is now easily defined by:

$$\begin{aligned}
M \;&\models\; \langle w\rangle{\downarrow}\, \phi &&\Leftrightarrow\quad |M| \;\models\; \langle w\rangle{\downarrow}\, \{V \mid V \;\models\; \phi\}\\
M \;&\models\; \langle w\rangle_{\dots}\, \phi &&\Leftrightarrow\quad |M| \;\models\; \langle w\rangle_{\dots}.
\end{aligned}$$

Note that, as in Example 1, the formula argument of the $\langle w\rangle_{\dots}$ modality is redundant. Also, note that our modalities for input/output could naturally be formed by combining the termination modality $\downarrow$, which lifts value formulas to computation formulas, with sequences of atomic modalities $\langle ?n\rangle$ and $\langle !n\rangle$ acting directly on computation formulas. In this paper, we do not include such modalities, acting on computation formulas, in our general theory. But this is a natural avenue for future consideration.

We now give a formal treatment of the logic and its semantics, in full generality. We assume given a signature Σ of effect operations, as in Sect. 2. And we assume given a set $\mathcal{O}$, whose elements we call *modalities*.

We call our main behavioural logic $\mathcal{V}$, where the letter $\mathcal{V}$ is chosen as a reference to the fact that the basic formula at function type specifies function behaviour on individual value arguments V.

Definition 2 (The logic $\mathcal{V}$). The classes $VF(\tau)$ and $CF(\tau)$ of *value* and *computation formulas*, for each type τ, are mutually inductively defined by the rules in Fig. 2. In this, I can be instantiated to any set, allowing for arbitrary conjunctions and disjunctions. When I is $\emptyset$, we get the special formulas $\top = \bigwedge_\emptyset$ and $\bot = \bigvee_\emptyset$. The use of arbitrary index sets means that formulas, as defined, form a proper class. However, we shall see below that countable index sets suffice.

In order to specify the semantics of modal formulas, we require a connection between modalities and effect trees, which is given by an interpretation function

$$[\![\cdot]\!] : \mathcal{O} \to \mathcal{P}(T\mathbf{1}).$$

That is, every modality $o \in \mathcal{O}$ is mapped to a subset $[\![o]\!] \subseteq T\mathbf{1}$ of unit-type effect trees. Given a subset $P \subseteq X$ (e.g. given by a formula) and a tree $t \in TX$ we can define a unit-type tree $t[\in P] \in T\mathbf{1}$ as the tree created by replacing the leaves of t that belong to P by $*$ and the others by $\bot$. In the case that P is the subset $\{V \mid V \models \phi\}$ specified by a formula $\phi \in VF(\tau)$, we also write $t[\models \phi]$ for $t[\in P]$.

We can now formally define the two satisfaction relations $\models \subseteq Val(\tau) \times VF(\tau)$ and $\models \subseteq Com(\tau) \times CF(\tau)$, mutually inductively, by:

$$\begin{array}{rcl} \overline{m} \models \{n\} & \Leftrightarrow & m = n \\ W \models (V \mapsto \Phi) & \Leftrightarrow & WV \models \Phi \\ M \models o\,\phi & \Leftrightarrow & |M|\,[\models \phi] \in [\![o]\!] \\ W \models \neg\phi & \Leftrightarrow & \neg(W \models \phi). \end{array}$$

We omit the evident clauses for the other propositional connectives. We remark that all conjunctions and disjunctions are semantically equivalent to countable ones, because value and computation formulas are interpreted over sets of terms, $Val(\tau)$ and $Com(\tau)$, which are countable.

We end this section by revisiting our running examples, and showing, in each case, that the example modalities presented above are all specified by suitable interpretation functions $[\![\cdot]\!] : \mathcal{O} \to \mathcal{P}(T\mathbf{1})$.

Example 0 (Pure functional computation). We have $\mathcal{O} = \{\downarrow\}$. Define:

$$[\![\downarrow]\!] \;=\; \{\,*\,\} \quad (\text{where } * \text{ is the tree with single node } *)$$

Example 1 (Error). We have $\mathcal{O} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in E\}$. Define:

$$[\![\mathsf{E}_e]\!] \;=\; \{\, raise_e \,\}.$$

Example 2 (Nondeterminism). We have $\mathcal{O} = \{\Diamond, \Box\}$. Define:

$$\begin{array}{rcl} [\![\Diamond]\!] & = & \{t \mid t \text{ has some * leaf}\} \\ [\![\Box]\!] & = & \{t \mid t \text{ has finite height and every leaf is a *}\}. \end{array}$$

Example 3 (Probabilistic choice). $\mathcal{O} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \le q < 1\}$. Define:

$$[\![\mathsf{P}_{>q}]\!] \;=\; \{t \mid \mathbf{P}(\,t \text{ terminates with a * leaf}\,) > q\}.$$

Example 4 (Global store). $\mathcal{O} = \{(s \rightarrowtail r) \mid s, r \in State\}$. Define:

$$[\![(s \rightarrowtail r)]\!] \;=\; \{t \mid exec(t, s) = (*, r)\}.$$

Example 5 (Input/output). $\mathcal{O} = \{\langle w\rangle\downarrow, \langle w\rangle_{\dots} \mid w \text{ an i/o-trace}\}$. Define:

$$\begin{array}{rcl} [\![\langle w\rangle\downarrow]\!] & = & \{t \mid t \models \langle w\rangle\downarrow\{*\}\} \\ [\![\langle w\rangle_{\dots}]\!] & = & \{t \mid t \models \langle w\rangle_{\dots}\}. \end{array}$$

4 Behavioural Equivalence

The goal of this section is to precisely formulate our main theorem: under suitable conditions, the behavioural equivalence determined by the logic $\mathcal{V}$ of Sect. 3 is a congruence. In order to achieve this, it will be useful to consider the *positive fragment* $\mathcal{V}^+$ of $\mathcal{V}$.

Definition 3 (The logic $\mathcal{V}^+$). The logic $\mathcal{V}^+$ is the fragment of $\mathcal{V}$ consisting of those formulas in $VF(\tau)$ and $CF(\tau)$ that do not contain negation.

Whenever we have a logic $\mathcal{L}$ whose value and computation formulas are given as subcollections $VF_{\mathcal{L}}(\tau) \subseteq VF(\tau)$ and $CF_{\mathcal{L}}(\tau) \subseteq CF(\tau)$, then $\mathcal{L}$ determines a preorder (and hence also an equivalence relation) between terms of the same type and aspect.

Definition 4 (Logical preorder and equivalence). Given a fragment $\mathcal{L}$ of $\mathcal{V}$, we define the *logical preorder* $\sqsubseteq_{\mathcal{L}}$, between well-typed terms of the same type and aspect, by:

$$\begin{array}{lcl} V \sqsubseteq_{\mathcal{L}} W & \Leftrightarrow & \forall \phi \in VF_{\mathcal{L}}(\tau),\ V \models \phi \Rightarrow W \models \phi \\ M \sqsubseteq_{\mathcal{L}} N & \Leftrightarrow & \forall \Phi \in CF_{\mathcal{L}}(\tau),\ M \models \Phi \Rightarrow N \models \Phi \end{array}$$

The *logical equivalence* $\equiv_{\mathcal{L}}$ on terms is the equivalence relation induced by the preorder (the intersection of $\sqsubseteq_{\mathcal{L}}$ and its converse).

In the case that formulas in $\mathcal{L}$ are closed under negation, it is trivial that the preorder $\sqsubseteq_{\mathcal{L}}$ is already an equivalence relation, and hence coincides with $\equiv_{\mathcal{L}}$. Thus we shall only refer specifically to the preorder $\sqsubseteq_{\mathcal{L}}$, for fragments, such as $\mathcal{V}^+$, that are not closed under negation.

The two main relations of interest to us in this paper are the primary relations determined by $\mathcal{V}$ and $\mathcal{V}^+$: full *behavioural equivalence* $\equiv_{\mathcal{V}}$; and the *positive behavioural preorder* $\sqsubseteq_{\mathcal{V}^+}$ (which induces *positive behavioural equivalence* $\equiv_{\mathcal{V}^+}$).

We next formulate the appropriate notion of (pre)congruence to apply to the relations $\equiv_{\mathcal{V}}$ and $\sqsubseteq_{\mathcal{V}^+}$. These two preorders are examples of *well-typed relations* on closed terms. Any such relation can be extended to a relation on open terms in the following way. Given a well-typed relation $\mathcal{R}$ on closed terms, we define the *open extension* $\mathcal{R}^\circ$ where $\Gamma \vdash M \mathcal{R}^\circ N : \tau$ precisely when, for every well-typed vector of closed values $\overrightarrow{V} : \Gamma$, it holds that $M[\overrightarrow{V}]\,\mathcal{R}\,N[\overrightarrow{V}]$. The correct notion of precongruence for a well-typed preorder on closed terms, is to ask for its open extension to be *compatible* in the sense of the definition below; see, e.g., [10,19] for further explanation.

Definition 5 (Compatibility). A well-typed open relation $\mathcal{R}$ is said to be *compatible* if it is closed under the rules in Fig. 3.

We now state our main congruence result, although we have not yet defined the conditions it depends upon.

$$\frac{}{\Gamma, x:\tau \vdash x \,\mathcal{R}\, x:\tau} \qquad \frac{}{\Gamma \vdash Z \,\mathcal{R}\, Z : \mathbf{N}} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N}}{\Gamma \vdash S(V) \,\mathcal{R}\, S(V') : \mathbf{N}}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \tau}{\Gamma \vdash \mathbf{return}(V) \,\mathcal{R}\, \mathbf{return}(V') : \tau} \qquad \frac{\Gamma, x:\tau \vdash M \,\mathcal{R}\, M' : \rho}{\Gamma \vdash (\lambda x:\tau.M) \,\mathcal{R}\, (\lambda x:\tau.M') : \tau \to \rho}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \tau \to \rho \qquad \Gamma \vdash W \,\mathcal{R}\, W' : \tau}{\Gamma \vdash (VW) \,\mathcal{R}\, (V'W') : \rho} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : (\tau \to \rho) \to (\tau \to \rho)}{\Gamma \vdash \mathbf{fix}(V) \,\mathcal{R}\, \mathbf{fix}(V') : \tau \to \rho}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N} \qquad \Gamma \vdash M \,\mathcal{R}\, M' : \tau \qquad \Gamma, x:\mathbf{N} \vdash N \,\mathcal{R}\, N' : \tau}{\Gamma \vdash \ \mathbf{case}\ V\ \mathbf{of}\ \{Z \Rightarrow M; S(x) \Rightarrow N\} \,\mathcal{R}\, \mathbf{case}\ V'\ \mathbf{of}\ \{Z \Rightarrow M'; S(x) \Rightarrow N'\} : \tau}$$

$$\frac{\Gamma \vdash M \,\mathcal{R}\, M' : \tau \qquad \Gamma, x:\tau \vdash N \,\mathcal{R}\, N' : \rho}{\Gamma \vdash \mathbf{let}\ M \Rightarrow x\ \mathbf{in}\ N \,\mathcal{R}\, \mathbf{let}\ M' \Rightarrow x\ \mathbf{in}\ N' : \rho}$$

$$\frac{\Gamma \vdash M_i \,\mathcal{R}\, M_i' : \tau}{\Gamma \vdash \sigma(M_0, M_1, ...) \,\mathcal{R}\, \sigma(M_0', M_1', ...) : \tau} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N} \qquad \Gamma \vdash M_i \,\mathcal{R}\, M_i' : \tau}{\Gamma \vdash \sigma(V; M_0, M_1, ...) \,\mathcal{R}\, \sigma(V'; M_0', M_1', ...) : \tau}$$

$$\frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N} \to \tau}{\Gamma \vdash \sigma(V) \,\mathcal{R}\, \sigma(V') : \tau} \qquad \frac{\Gamma \vdash V \,\mathcal{R}\, V' : \mathbf{N} \qquad \Gamma \vdash W \,\mathcal{R}\, W' : \mathbf{N} \to \tau}{\Gamma \vdash \sigma(V; W) \,\mathcal{R}\, \sigma(V'; W') : \tau}$$

Fig. 3. Rules for compatibility

Theorem 1. *If $\mathcal{O}$ is a decomposable set of Scott-open modalities then the open extensions of $\equiv_{\mathcal{V}}$ and $\sqsubseteq_{\mathcal{V}^+}$ are both compatible. (It is an immediate consequence that the open extension of $\equiv_{\mathcal{V}^+}$ is also compatible.)*

The Scott-openness condition refers to the *Scott topology* on $T\mathbf{1}$.

Definition 6. We say that $o \in \mathcal{O}$ is *upwards closed* if $[\![o]\!]$ is an upper-closed subset of $T\mathbf{1}$; i.e., if $t \in [\![o]\!]$ implies $t' \in [\![o]\!]$ whenever $t \leq t'$.

Definition 7. We say that $o \in \mathcal{O}$ is *Scott-open* if $[\![o]\!]$ is an open subset in the Scott topology on $T\mathbf{1}$; i.e., $[\![o]\!]$ is upper closed and, whenever $t_1 \leq t_2 \leq \ldots$ is an ascending chain in $T\mathbf{1}$ with supremum $\sqcup_i t_i \in [\![o]\!]$, we have $t_n \in [\![o]\!]$ for some n.

Before formulating the property of *decomposability*, we make some simple observations about the positive preorder $\sqsubseteq_{\mathcal{V}^+}$.

Lemma 8. *For any $V_0, V_1 \in Val(\rho \to \tau)$, we have $V_0 \sqsubseteq_{\mathcal{V}^+} V_1$ if and only if:*

$$\forall W \in Val(\rho),\ \forall \Psi \in CF_{\mathcal{V}^+}(\tau),\ V_0 \ \models\ (W \mapsto \Psi) \text{ implies } V_1 \ \models\ (W \mapsto \Psi).$$

Lemma 9. *For any $M_0, M_1 \in Com(\tau)$, we have $M_0 \sqsubseteq_{\mathcal{V}^+} M_1$ if and only if:*

$$\forall o \in \mathcal{O},\ \forall \phi \in VF_{\mathcal{V}^+}(\tau),\ M_0 \ \models\ o\,\phi \text{ implies } M_1 \ \models\ o\,\phi.$$

Similar characterisations, with appropriate adjustments, hold for behavioural equivalence $\equiv_{\mathcal{V}}$.

The decomposability property is formulated using an extension of the positive preorder $\sqsubseteq_{\mathcal{V}^+}$, at unit type, from a relation on computations to a relation on arbitrary effect trees. Accordingly, we define a preorder $\preceq$ on $T\mathbf{1}$ by:

$$t \preceq t' \quad \Leftrightarrow \quad \forall o \in \mathcal{O},\ (t \in [\![o]\!] \Rightarrow t' \in [\![o]\!]) \wedge (t[\in \emptyset] \in [\![o]\!] \Rightarrow t'[\in \emptyset] \in [\![o]\!]).$$

Proposition 10. *For computations $M, N \in Com(\mathbf{1})$, it holds that $|M| \preceq |N|$ if and only if $M \sqsubseteq_{\mathcal{V}^+} N$.*

Proof. The defining condition for $|M| \preceq |N|$ unwinds to:

$$\forall o \in \mathcal{O},\ (M \models o\top \text{ implies } N \models o\top) \wedge (M \models o\bot \text{ implies } N \models o\bot).$$

This coincides with $M \sqsubseteq_{\mathcal{V}^+} N$ by Lemma 9. □

We now formulate the required notion of decomposability. We first give the general definition, and then follow it with a related notion of *strong decomposability*, which can be more convenient to establish in examples. Both definitions are unavoidably technical in nature.

For any relation $\mathcal{R} \subseteq X \times Y$ and subset $A \subseteq X$, we write $\mathcal{R}^{\uparrow} A$ for the right set $\{y \in Y \mid \exists x \in A, x\mathcal{R}y\}$. This allows use to easily define our required notion.

Definition 11 (Decomposability). We say that $\mathcal{O}$ is *decomposable* if, for all $r, r' \in TT\mathbf{1}$, we have:

$$(\forall A \subseteq T\mathbf{1},\ r[\in A] \preceq r'[\in \preceq^{\uparrow} A]) \quad \Rightarrow \quad \mu r \preceq \mu r'.$$

Corollary 22 in Sect. 5, may help to motivate the formulation of the above property, which might otherwise appear purely technical. The following stronger version of decomposability, which suffices for all examples considered in the paper, is perhaps easier to understand in its own right.

Definition 12 (Strong decomposability). We say that $\mathcal{O}$ is *strongly decomposable* if, for every $r \in TT\mathbf{1}$ and $o \in \mathcal{O}$ for which $\mu r \in [\![o]\!]$, there exists a collection $\{(o_i, o_i')\}_{i \in I}$ of pairs of modalities such that:

1. $\forall i \in I,\ r[\in [\![o_i']\!]] \in [\![o_i]\!]$; and
2. for every $r' \in TT\mathbf{1}$, $(\forall i \in I,\ r'[\in [\![o_i']\!]] \in [\![o_i]\!])$ implies $\mu r' \in [\![o]\!]$.

Proposition 13. *If $\mathcal{O}$ is a strongly decomposable then it is decomposable.*

Proof. Suppose that $r[\in A] \preceq r'[\in (\preceq^{\uparrow} A)]$ holds for every $A \subseteq T\mathbf{1}$. Assume that $\mu r \in [\![o]\!] \in \mathcal{O}$. Then strong decomposability gives a collection $\{(o_i, o_i')\}_I$. By the definition of $\preceq$, for each o_i' we have $\preceq^{\uparrow} [\![o_i']\!] = [\![o_i']\!]$. By the initial assumption, $r[\in [\![o_i']\!]] \in [\![o_i]\!]$ implies $r'[\in (\preceq^{\uparrow} [\![o_i']\!])] \in [\![o_i]\!]$, and hence $r'[\in [\![o_i']\!]] \in [\![o_i]\!]$. This holds for every i, so by strong decomposability $\mu r' \in [\![o]\!]$. We have shown that $\mu r \in [\![o]\!]$ implies $\mu r' \in [\![o]\!]$. One can prove similarly that $\mu r[\in \emptyset] \in [\![o]\!]$ implies that $\mu r'[\in \emptyset] \in [\![o]\!]$ by observing that $\preceq^{\uparrow} \{x \mid x[\in \emptyset] \in [\![o_i']\!]\} = \{x \mid x[\in \emptyset] \in [\![o_i']\!]\}$. Thus it holds that $\mu r \preceq \mu r'$ and hence $\mathcal{O}$ is decomposable. □

We end this section by again looking at our running examples, and showing, in each case, that the identified collection $\mathcal{O}$ of modalities is Scott-closed (hence upwards closed) and strongly decomposable (hence decomposable). For any of the examples, upwards closure is easily established, so we will not show it here.

Example 0 (Pure functional computation). We have $\mathcal{O} = \{\downarrow\}$ and $[\![\downarrow]\!] = \{*\}$. Scott openness holds since if $\sqcup_i t_i = *$ then for some i we must already have $t_i = *$. It is strongly decomposable since: $\mu r \in [\![\downarrow]\!] \Leftrightarrow r[\in [\![\downarrow]\!]] \in [\![\downarrow]\!]$, which means r returns a tree t which is a leaf $*$.

Example 1 (Error). We have $\mathcal{O} = \{\downarrow\} \cup \{\mathsf{E}_e \mid e \in E\}$ and $[\![\mathsf{E}_e]\!] = \{ raise_e \}$. Scott-openness holds for both modalities for the same reason as in the previous example, and its strongly decomposable since:

$$\mu r \in [\![\downarrow]\!] \quad \Leftrightarrow \quad r[\in [\![\downarrow]\!]] \in [\![\downarrow]\!].$$

Which means r returns a tree t which returns $*$.

$$\mu r \in [\![\mathsf{E}_e]\!] \quad \Leftrightarrow \quad r[\in [\![\mathsf{E}_e]\!]] \in [\![\mathsf{E}_e]\!] \vee r[\in [\![\mathsf{E}_e]\!]] \in [\![\downarrow]\!].$$

Which means r raises an error, or returns a tree that raises an error.

Example 2 (Nondeterminism). We have $\mathcal{O} = \{\Diamond, \Box\}$. The Scott-openness of $[\![\Diamond]\!] = \{t \mid \ t$ has some $*$ leaf$\}$ is because if $\sqcup_i t_i$ has a $*$ leaf, then that leaf must already be contained in t_i for some i. Similarly, if $\sqcup_i t_i \in [\![\Box]\!]$ then, because $[\![\Box]\!] = \{t \mid$t has finite height and every leaf is a$*\}$, the tree $\sqcup_i t_i$ has finitely many leaves and all must be contained in t_i for some i. Hence $t_i \in [\![\Box]\!]$. Strong decomposability holds because:

$$\mu r \in [\![\Diamond]\!] \Leftrightarrow r[\in [\![\Diamond]\!]] \in [\![\Diamond]\!] \quad \text{and} \quad \mu r \in [\![\Box]\!] \Leftrightarrow r[\in [\![\Box]\!]] \in [\![\Box]\!].$$

The right-hand-side of the former states that r has as a leaf a tree t, which itself has a leaf $*$. That of the latter states that r is finite and all leaves are finite trees t that have only $*$ leaves. The same arguments show that $\{\Diamond\}$ and $\{\Box\}$ are also decomposable sets of Scott open modalities.

Example 3 (Probabilistic choice). $\mathcal{O} = \{\mathsf{P}_{>q} \mid q \in \mathbb{Q}, 0 \leq q < 1\}$. For the Scott-openness of $[\![\mathsf{P}_{>q}]\!] = \{t \mid \mathbf{P}($ t terminates with a * leaf$) \ > \ q\}$, note that $\mathbf{P}(\sqcup_i t_i$ terminates with a $*$ leaf$)$ is determined by some countable sum over the leaves of t_i. If this sum is greater than a rational q, then some finite approximation of the sum must already be above q. The finite sum is over finitely many leaves from $\sqcup_i t_i$, all of which will be present in t_i for some i. Hence $t_i \in [\![\mathsf{P}_{>q}]\!]$.

We have strong decomposability, since $\mathbf{P}(\mu r$ terminates with a $*$ leaf$)$ equals the integral of the function $f_r(x) = sup\{y \in [0,1] \mid r[[\![\mathsf{P}_{>x}]\!]] \in [\![\mathsf{P}_{>y}]\!]\}$ from $[0,1]$ to $[0,1]$. Indeed, $f_r(x)$ gives the probability that r return a tree $t \in [\![\mathsf{P}_{>x}]\!]$. So we know that if $\forall x, y, r[[\![\mathsf{P}_{>x}]\!]] \in [\![\mathsf{P}_{>y}]\!] \Rightarrow r'[[\![\mathsf{P}_{>x}]\!]] \in [\![\mathsf{P}_{>y}]\!]$, then $f_{r'}(x) \geq f_r(x)$ for any x. Hence if $\mu r \in [\![\mathsf{P}_{>q}]\!]$ then $\int f_r > q$, whence also $\int f_{r'} > q$, which means $\mu r' \in [\![\mathsf{P}_{>q}]\!]$.

Example 4 (Global store). We have $\mathcal{O} = \{(s \rightarrowtail s') \mid s, s' \in State\}$. For the Scott-openness of $[\![(s \rightarrowtail s')]\!] = \{t \mid exec(t, s) = (*, r)\}$, note that if $exec(\sqcup_i t_i, s) = (*, s')$, there is a single finite branch of t that follows the path the recursive function *exec* took. This branch must already be contained in t_i for some i. We also have strong decomposability since:

$$\mu r \in [\![s \rightarrowtail s']\!] \quad \Leftrightarrow \quad \exists s'' \in State,\, r[\in [\![s'' \rightarrowtail s']\!]] \in [\![s \rightarrowtail s'']\!].$$

Which just means that $exec(r, s) = (t, s'')$ and $exec(t, s'') = (*, s')$ for some s''.

Example 5 (Input/output). We have $\mathcal{O} = \{\langle w\rangle\downarrow, \langle w\rangle_{\ldots} \mid w \text{ an i/o-trace}\}$. For the Scott-openness of $[\![\langle w\rangle\downarrow]\!] = \{t \mid t \models \langle w\rangle\downarrow\{*\}\}$, note that the i/o-trace $\langle w\rangle\downarrow$ is given by some finite branch, which if in $\sqcup_i t_i$ must be in t_i for some i. The Scott-openness of $[\![\langle w\rangle_{\ldots}]\!] = \{t \mid t \models \langle w\rangle_{\ldots}\}$ holds for similar reasons. We have strong decomposability because of the implications:

$$\mu r \in [\![\langle w\rangle\downarrow]\!] \quad \Leftrightarrow \quad \exists v, u \text{ i/o-traces}, vu = w \wedge r[\in [\![\langle u\rangle\downarrow]\!]] \in [\![\langle v\rangle\downarrow]\!].$$

Which means r follows trace v returning t, and t follows trace u returning $*$.

$$\mu r \in [\![\langle w\rangle\ldots]\!] \Leftrightarrow \quad r[\in [\![\downarrow]\!]] \in [\![\langle w\rangle\ldots]\!] \vee \exists v, u, vu = w \wedge r[\in [\![\langle u\rangle\ldots]\!]] \in [\![\langle v\rangle\downarrow]\!].$$

Which means either r follows trace w immediately, or it follows v returning a tree that follows u.

5 Applicative $\mathcal{O}$-(bi)similarity

In this section we look at an alternative description of our logical pre-order. Central to such a definition lies the concept of a *relator* [12,25], which we use to lift a relation on value terms to a relation on computation terms. With our family of modalities $\mathcal{O}$ we can define a relator which takes a relation $\mathcal{R} \subseteq X \times Y$ and returns the relation $\mathcal{O}(\mathcal{R}) \subseteq TX \times TY$, defined by:

$$t\,\mathcal{O}(\mathcal{R})\,t' \quad \Leftrightarrow \quad \forall A \subseteq X, \forall o \in \mathcal{O},\, t[\in A] \in [\![o]\!] \Rightarrow t'[\in (\mathcal{R}^{\uparrow} A)] \in [\![o]\!].$$

Note that $\mathcal{O}(id_{\mathbf{1}}) = (\preceq)$. Following [9], we use this relation-lifting operation to define notions of applicative similarity and bisimilarity.

Definition 14. An *applicative $\mathcal{O}$-simulation* is given by a pair of relations $\mathcal{R}^v_\tau$ and $\mathcal{R}^c_\tau$ for each type τ, where $\mathcal{R}^v_\tau \subseteq Val(\tau)^2$ and $\mathcal{R}^c_\tau \subseteq Com(\tau)^2$, such that:

1. $V \mathcal{R}^v_{\mathbf{N}} W \;\Rightarrow\; (V = W)$
2. $M \mathcal{R}^c_\tau N \;\Rightarrow\; |M|\; \mathcal{O}(\mathcal{R}^v_\tau)\; |N|$
3. $V \mathcal{R}^v_{\rho\to\tau} W \;\Rightarrow\; \forall U \in Val(\rho),\; VU\; \mathcal{R}^c_\tau\; WU$

Applicative $\mathcal{O}$-similarity is the largest applicative $\mathcal{O}$-simulation, which is equal to the union of all applicative $\mathcal{O}$-simulations.

Definition 15. An *applicative $\mathcal{O}$-bisimulation* is a symmetric $\mathcal{O}$-simulation. The relation of *$\mathcal{O}$-bisimilarity* is the largest applicative $\mathcal{O}$-bisimulation.

Lemma 16. *Applicative $\mathcal{O}$-bisimilarity is identical to the relation of applicative $(\mathcal{O} \cap \mathcal{O}^{\mathsf{op}})$-similarity, where $t(\mathcal{O} \cap \mathcal{O}^{\mathsf{op}})(\mathcal{R})r \Leftrightarrow t\mathcal{O}(\mathcal{R})r \wedge r\mathcal{O}(\mathcal{R}^{\mathsf{op}})t$.*

Proof. Let $\mathcal{R}$ be the $\mathcal{O}$-bisimilarity, then by symmetry we have $\mathcal{R}^{\mathsf{op}} = \mathcal{R}$. So if $M\mathcal{R}N$ we have $N\mathcal{R}M$, and by the simulation rules we derive $|M|\mathcal{O}(\mathcal{R})|N|$ and $|N|\mathcal{O}(\mathcal{R})|M|$ which is what we needed.

Let $\mathcal{R}$ be the $\mathcal{O} \cap \mathcal{O}^{\mathsf{op}}$-similarity. If $M\mathcal{R}^{\mathsf{op}}N$ then $|N|(\mathcal{O} \cap \mathcal{O}^{\mathsf{op}})(\mathcal{R})|M|$ so $|N|\mathcal{O}(\mathcal{R})|M| \wedge |M|\mathcal{O}(\mathcal{R}^{\mathsf{op}})|N|$ which results in $|M|(\mathcal{O} \cap \mathcal{O}^{\mathsf{op}})(\mathcal{R}^{\mathsf{op}})|N|$. Verifying the other simulation conditions as well, we can conclude that the symmetric closure $\mathcal{R} \cup \mathcal{R}^{\mathsf{op}}$ is also a $\mathcal{O} \cap \mathcal{O}^{\mathsf{op}}$-simulation. So $\mathcal{R}$ must, as the largest such simulation, be symmetric. Hence $\mathcal{R}$ is a symmetric $\mathcal{O}$-simulation as well.

For brevity, we will leave out the word "applicative" from here on, and write o to mean its denotation $[\![o]\!]$. We also introduce brackets, writing $o[\phi]$ for $o\,\phi$. The key result now is that the maximal relation, the $\mathcal{O}$-similarity is in most cases the same object as our logical preorder. We first give a short Lemma.

Lemma 17. *For any fragment $\mathcal{L}$ of $\mathcal{V}$ closed under countable conjunction, it holds that for each value V there is a formula $\chi_V \in \mathcal{L}$ s.t. $W \models_{\mathcal{L}} \chi_V \Leftrightarrow V \sqsubseteq_{\mathcal{L}} W$.*

Proof. For each U such that $(V \not\sqsubseteq_{\mathcal{L}} U)$, choose a formula $\phi^U \in \mathcal{L}$ such that $V \models_{\mathcal{L}} \phi^U$ and $(U \not\models \phi^U)$. Then if we define $\chi_V := \bigwedge_{\{U \mid V \not\sqsubseteq_{\mathcal{L}} U\}} \phi^U$ it holds that $V \not\sqsubseteq_{\mathcal{L}} U \Leftrightarrow U \not\models \chi_V$, which is what we want.

Theorem 2 (a). *For any family of upwards closed modalities $\mathcal{O}$, we have that the logical preorder $\sqsubseteq_{\mathcal{V}^+}$ is identical to $\mathcal{O}$-similarity.*

Proof. We write $\sqsubseteq$ instead of $\sqsubseteq_{\mathcal{V}^+}$ to make room for other annotations. We first prove that our logical preorder $\sqsubseteq$ is an $\mathcal{O}$-simulation by induction on types.

1. Values of $\mathbf{N}$. If $\overline{n} \sqsubseteq^v_{\mathbf{N}} \overline{m}$, then since $\overline{n} \models \{n\}$ we have that $\overline{m} \models \{n\}$, hence $m = n$.
2. Computations of τ. Assume $M \sqsubseteq^c_\tau N$, we prove that $|M|\mathcal{O}(\sqsubseteq^v_\tau)|N|$. Take $A \subseteq \mathit{Val}(\tau)$ and $o \in \mathcal{O}$ such that $|M|[\in A] \in o$. Taking the following formula $\phi := \bigvee_{a \in A} \chi_a$ (where χ_a as in Lemma 17), then $b \models \phi \Leftrightarrow \exists a \in A, a \sqsubseteq^v_\tau b$ and $a \in A \Rightarrow a \models \phi$. So $|M|[\models \phi] \geq |M|[\in A]$, hence since o is upwards closed, $|M|[\models \phi] \in o$. By $M \sqsubseteq^c_\tau N$ we have $|N|[\in \{b \in \mathit{Val}(\tau) \mid \exists a \in A, a \sqsubseteq^v_\tau b\}] = |N|[\models \phi] \in o$. Hence we can conclude that $|M|\mathcal{O}(\sqsubseteq^v_\tau)|N|$.
3. Function values of $\rho \to \tau$, this follows from Lemma 8 and the Induction Hypothesis.

We can conclude that $\sqsubseteq$ is an $\mathcal{O}$-simulation. Now take an arbitrary $\mathcal{O}$-simulation $\mathcal{R}$. We prove by induction on types that $\mathcal{R} \subseteq (\sqsubseteq)$.

1. Values of $\mathbf{N}$. If $V \mathcal{R}^v_{\mathbf{N}} W$ then $V = W$, hence by reflexivity we get $V \sqsubseteq^v_{\mathbf{N}} W$.
2. Computations of τ. Assume $M \mathcal{R}^c_\tau N$, we prove that $M \sqsubseteq^c_\tau N$ using the characterisation from Lemma 9. Say for $o \in \mathcal{O}$ and $\phi \in \mathit{VF}(\tau)$ we have $M \models o[\phi]$. Let $A_\phi := \{a \in \mathit{Val}(\tau) \mid a \models \phi\} \subseteq \mathit{Val}(\tau)$, then $|M|[\in A_\phi] = |M|[\models \phi] \in o$ hence by $M \mathcal{R}^c_\tau N$ we derive $|N|[\in \{b \in \mathit{Val}(\tau) \mid \exists a \in A_\phi, a \mathcal{R}^v_\tau b\}] \in o$. By Induction Hypothesis on values of τ, we know that $\mathcal{R}^v_\tau \subseteq (\sqsubseteq^v_\tau)$, hence '$\exists a \in A_\phi, a \mathcal{R}^v_\tau b$' implies $b \models \phi$. We get that $|N|[\models \phi] \geq |N|[\in \{b \in \mathit{Val}(\tau) \mid \exists a \in A_\phi, a \mathcal{R}^v_\tau b\}]$, so by upwards closure of o we have $|N|[\models \phi] \in o$ meaning $N \models o[\phi]$. We conclude that $M \sqsubseteq^c_\tau N$.
3. Function values of $\rho \to \tau$, assume $V \mathcal{R}^v_{\rho \to \tau} W$. We prove $V \sqsubseteq^v_{\rho \to \tau} W$ using the characterisation from Lemma 8. Assume $V \models (U \mapsto \Phi)$ where $U \in \mathit{Val}(\rho)$ and $\Phi \in \mathit{CF}(\tau)$, so $VU \models \Phi$. By $V \mathcal{R}^v_{\rho \to \tau} W$ we have $VU \, \mathcal{R}^c_\tau \, WU$ and by Induction Hypothesis we have $\mathcal{R}^c_\tau \subseteq (\sqsubseteq^c_\tau)$, so $VU \sqsubseteq^c_\tau WU$. Hence $WU \models \Phi$ meaning $W \models (U \mapsto \Phi)$. We can conclude that $V \sqsubseteq^v_{\rho \to \tau} W$.

4. Values of $\mathbf{1}$. If $V \mathcal{R}^v_{\mathbf{1}} W$ then $V = * = W$ hence $V \sqsubseteq^v_{\mathbf{1}} W$.

In conclusion: any $\mathcal{O}$-simulation $\mathcal{R}$ is a subset of the $\mathcal{O}$-simulation $\sqsubseteq_{\mathcal{V}^+}$. So $\sqsubseteq_{\mathcal{V}^+}$ is $\mathcal{O}$-similarity. □

Alternatively, we can look at the variation of our logic with negation. This is related to applicative bisimulations.

Theorem 2 (b). *For any family of upwards closed modalities $\mathcal{O}$, we have that the logical equivalence $\equiv_{\mathcal{V}}$ is identical to $\mathcal{O}$-bisimilarity.*

Proof. Note first that $\equiv_{\mathcal{V}}$ is symmetric.

Secondly, note that since $\equiv_{\mathcal{V}} = \sqsubseteq_{\mathcal{V}}$ we know by Lemma 17, that for any V, there is a formula χ_V such that $W \models \chi_V \Leftrightarrow V \equiv_{\mathcal{V}} W$.

Using these special formulas χ_V, the rest of the proof is very similar to the proof in Theorem 2(a). Here follow the non-trivial parts of the proof, different from the previous lemma. For proving $\equiv_{\mathcal{V}}$ is an $\mathcal{O}$-simulation:

1. Computations of τ. Assume $M \equiv^c_\tau N$ and $|M|[\in A] \in o \in \mathcal{O}$. Then $M \models o[\bigvee_{V \in A} \chi_V]$ hence $N \models o[\bigvee_{V \in A} \chi_V]$ meaning $|N|[\in \{W \mid \exists V \in A, V \equiv^c_\tau W\}]$. So $|M|\mathcal{O}(\equiv^v_\tau)|N|$.
2. Functions of $\rho \to \tau$, if $V \equiv^v_{\rho \to \tau} W$ and $U \in \mathit{Val}(\rho)$. If $VU \models \Phi$, then $V \models U \mapsto \Phi$ hence $W \models U \mapsto \Phi$ so $WU \models \Phi$. Same vice versa, so $VU \equiv^c_\tau WU$.

So $\equiv_{\mathcal{V}}$ is an $\mathcal{O}$-bisimulation. Now take any $\mathcal{O}$-bisimulation $\mathcal{R}$.

1. Computations of τ, if $M \mathcal{R} N$ and $M \models o[\phi]$ then $|M|[\models \phi] \in o$ hence $|N|[\in \{W \mid \exists V \models \phi, V \mathcal{R}^v_\tau W\}] \in o$. By Induction Hypothesis, $(\mathcal{R}^v_\tau) \subseteq (\equiv^v_\tau)$ so $\{W \mid \exists V \models \phi, V \mathcal{R}^v_\tau W\} \subseteq \{W \mid \exists V \models \phi, V \equiv^v_\tau W\}$. So by upwards closure of o we get that $|N|[\in \{W \mid \exists V \models \phi, V \equiv^v_\tau W\}] \in o$ and further that $N \models o[\phi]$. We can conclude $M \equiv_{\mathcal{V}} N$.
2. Values of $\rho \to \tau$, if $V \mathcal{R} W$ and $V \models U \mapsto \Phi$, then $VU \models \Phi$ and $VU \mathcal{R} WU$ hence by Induction Hypothesis, $VU \equiv WU$ meaning $WU \models \Phi$ so $W \models U \mapsto \Phi$. If $V \models \neg(U \mapsto \Phi)$ then $\neg(VU \models \Phi)$ hence by $VU \equiv WU$ we have $\neg(WU \models \Phi)$ so $W \models \neg(U \mapsto \Phi)$. For the $\bigvee$ and $\bigwedge$ constructors, a simple Induction Step would suffice, and for higher level negation note that $\neg \bigvee \phi \Leftrightarrow \bigwedge \neg\phi$ and $\neg \bigwedge \phi \Leftrightarrow \bigvee \neg\phi$.

We can conclude that $(\mathcal{R}) \subseteq (\equiv_{\mathcal{V}})$, so $\equiv_{\mathcal{V}}$ is indeed $\mathcal{O}$-bisimilarity. □

We end this section by stating the abstract properties of our relational lifting $\mathcal{O}(\mathcal{R})$ required for the proof by Howe's method in Sect. 6 to go through. The necessary properties were identified in [9]. The contribution of this paper is that all the required properties follow from our modality-based definition of $\mathcal{O}(\mathcal{R})$. The first set of properties tell us that $\mathcal{O}(-)$ is a relator in the sense of [12]:

Lemma 18. *If the modalities from $\mathcal{O}$ are upwards closed, then $\mathcal{O}(-)$ is a relator, meaning that:*

1. *If* $\mathcal{R} \subseteq X \times X$ *is reflexive, then so is* $\mathcal{O}(\mathcal{R})$.
2. $\forall \mathcal{R}, \forall \mathcal{S}, \quad \mathcal{O}(\mathcal{R})\mathcal{O}(\mathcal{S}) \subseteq \mathcal{O}(\mathcal{R}\mathcal{S})$, *where* $\mathcal{R}\mathcal{S}$ *is relation composition.*
3. $\forall \mathcal{R}, \forall \mathcal{S}, \quad \mathcal{R} \subseteq \mathcal{S} \Rightarrow \mathcal{O}(\mathcal{R}) \subseteq \mathcal{O}(\mathcal{S})$.
4. $\forall f : X \to Z, g : Y \to W, \mathcal{R} \subseteq Z \times W, \mathcal{O}((f \times g)^{-1}\mathcal{R}) = (Tf \times Tg)^{-1}\mathcal{O}(\mathcal{R})$ *where* $(f \times g)^{-1}(\mathcal{R}) = \{(x, y) \in X \times Y \mid f(x)\mathcal{R}g(y)\}$.

The next property together with the previous lemma establishes that $\mathcal{O}(-)$ is a *monotone relator* in the sense of [25].

Lemma 19. *If the modalities from* $\mathcal{O}$ *are upwards closed, then* $\mathcal{O}(-)$ *is* monotone*, meaning for any* $f : X \to Z$, $g : Y \to W$, $\mathcal{R} \subseteq X \times Y$ *and* $\mathcal{S} \subseteq Z \times W$*:*

$$(\forall x, y, x\mathcal{R}y \Rightarrow f(x)\,\mathcal{S}\,g(y)) \wedge t\mathcal{O}(\mathcal{R})r \Rightarrow t[x \mapsto f(x)]\,\mathcal{O}(\mathcal{S})\,r[y \mapsto g(y)]$$

The relator also interacts well with the monad structure on T.

Lemma 20. *If* $\mathcal{O}$ *is a decomposable set of upwards closed modalities, then:*

1. $x\mathcal{R}y \;\Rightarrow\; \eta(x)\mathcal{O}(\mathcal{R})\eta(y)$;
2. $t\mathcal{O}(\mathcal{O}(\mathcal{R}))r \;\Rightarrow\; \mu t\mathcal{O}(\mathcal{R})\mu r$.

Finally, the following properties show that relator behaves well with respect to the order on trees.

Lemma 21. *If* $\mathcal{O}$ *only contains Scott open modalities, then:*

1. *If* $\mathcal{R}$ *is reflexive, then* $t \leq r \Rightarrow t\mathcal{O}(\mathcal{R})r$.
2. *For any two sequences* $u_0 \leq u_1 \leq u_2 \leq \dots$ *and* $v_0 \leq v_1 \leq v_2 \leq \dots$*:* $\forall n, (u_n\mathcal{O}(\mathcal{R})v_n) \;\Rightarrow\; (\sqcup_n u_n)\mathcal{O}(\mathcal{R})(\sqcup_n v_n)$

The lemmas above list the core properties of the relator, which are satisfied when our family $\mathcal{O}$ is decomposable and contains only Scott open modalities. The results below follow from those above.

Corollary 22. *If* $\mathcal{O}$ *contains only upwards closed modalities, then:*

$$\mathcal{O} \text{ is decomposable} \quad \Leftrightarrow \quad \forall \mathcal{R} \subseteq X \times Y, \forall t, r \in TT\mathbf{1}, (t\mathcal{O}(\mathcal{O}(\mathcal{R}))r \Rightarrow \mu t\,\mathcal{O}(\mathcal{R})\,\mu r)$$

Corollary 23. *If* $\mathcal{O}$ *is a decomposable family of upwards closed modalities, then lifted relations are preserved by Kleisli lifting and effect operators:*

1. *Given* $f : X \to Z$, $g : Y \to W$, $\mathcal{R} \subseteq X \times Y$ *and* $\mathcal{S} \subseteq Z \times W$*, if for all* $x \in X$ *and* $y \in Y$ *we have* $x\mathcal{R}y \Rightarrow f(x)\,\mathcal{O}(\mathcal{S})\,g(y))$ *and if* $t\mathcal{O}(\mathcal{R})r$ *then* $\mu(t[x \mapsto f(x)])\,\mathcal{O}(\mathcal{S})\,\mu(r[y \mapsto g(y)])$
2. $(\forall k, u_k\mathcal{O}(\mathcal{S})v_k) \;\Rightarrow\; \sigma(u_0, u_1, \dots)\mathcal{O}(\mathcal{S})\sigma(v_0, v_1, \dots)$

Point 2 of Corollary 23 has been stated in such a way that it contains both the infinite arity case $\alpha^{\mathbf{N}} \to \alpha$ and the finite arity case $\alpha^n \to \alpha$. So it states that any lifted relation is preserved under any of the predefined algebraic effects.

6 Howe's Method

In this section, we apply Howe's method, first developed in [5,6], to establish the compatibility of applicative (bi)similarity, and hence of the behavioural preorders. Given a relation $\mathcal{R}$ on terms, one defines its *Howe closure* $\mathcal{R}^\bullet$, which is compatible and contains the open extension $\mathcal{R}^\circ$. Our proof makes fundamental use of the relator properties from Sect. 5, closely following the approach of [9].

Proposition 24. *If $\mathcal{O}$ is a decomposable set of Scott open modalities, then for any $\mathcal{O}$-simulation preorder $\sqsubseteq$, the restriction of its Howe closure $\sqsubseteq^\bullet$ to closed terms is an $\mathcal{O}$-simulation.*

In the proof of the proposition, the relator properties are mainly used to show that $\sqsubseteq^\bullet$ satisfies condition (2) in Definition 14.

We can now establish the compatibility of applicative $\mathcal{O}$-similarity.

Theorem 3 (a). *If $\mathcal{O}$ is a decomposable set of Scott open modalities, then the open extension of the relation of $\mathcal{O}$-similarity is compatible.*

Proof (sketch). We write $\sqsubseteq_s$ for the relation of $\mathcal{O}$-similarity. Since $\sqsubseteq_s$ is an $\mathcal{O}$-simulation, we know by Proposition 24 that $\sqsubseteq_s^\bullet$ limited to closed terms is one as well, and hence is contained in the largest $\mathcal{O}$-simulation $\sqsubseteq_s$. Since $\sqsubseteq_s^\bullet$ is compatible, it is contained in the open extension $\sqsubseteq_s^\circ$. We can conclude that $\sqsubseteq_s^\circ$ is equal to the Howe closure $\sqsubseteq_s^\bullet$, which is compatible. □

To prove that $\mathcal{O}$-bisimilarity is compatible, we use the following result from [10] (where we write $\mathcal{S}^*$ for the transitive-reflexive closure of a relation $\mathcal{S}$).

Lemma 25. *If $\mathcal{R}^\circ$ is symmetric and reflexive, then $\mathcal{R}^{\bullet *}$ is symmetric.*

Theorem 3 (b). *If $\mathcal{O}$ is a decomposable set of Scott open modalities, then the open extension of the relation of $\mathcal{O}$-bisimilarity is compatible.*

Proof (sketch). We write $\mathcal{O}$-bisimilarity as $\sqsubseteq_b$. From Proposition 24 we know that $\sqsubseteq_b^\bullet$ on closed terms is an $\mathcal{O}$-simulation, and so we know $\sqsubseteq_b^{\bullet *}$ is an $\mathcal{O}$-simulation as well (using Lemma 18). Since $\sqsubseteq_b$ is reflexive and symmetric, we know by the previous lemma that $\sqsubseteq_b^{\bullet *}$ is symmetric. Hence $\sqsubseteq_b^{\bullet *}$ is an $\mathcal{O}$-bisimulation, implying $(\sqsubseteq_b^{\bullet *}) \subseteq (\sqsubseteq_b^\circ)$ by compatibility of $\sqsubseteq_b^{\bullet *}$. Since $(\sqsubseteq_b^\circ) \subseteq (\sqsubseteq_b^\bullet) \subseteq (\sqsubseteq_b^{\bullet *})$ we have that $(\sqsubseteq_b^{\bullet *}) = (\sqsubseteq_b^\circ)$, and we can conclude that $\sqsubseteq_b^\circ$ is compatible. □

Theorem 1 is an immediate consequence of Theorems 2 and 3.

7 Pure Behavioural Logic

In this section, we briefly explore an alternative formulation of our logic. This has both conceptual and practical motivations. Our very approach to behavioural logic, fits into the category of *endogenous* logics in the sense of Pnueli [24]. Formulas (ϕ and Φ) express properties of individual programs, through satisfaction

relations ($V \models \phi$ and $M \models \Phi$). Programs are thus considered as 'models' of the logic, with the satisfaction relation being defined via program behaviour.

It is conceptually appealing to push the separation between program and logic to its natural conclusion, and ask for the syntax of the logic to be independent of the syntax of the programming language. Indeed, it seems natural that it should be possible to express properties of program behaviour without knowledge of the syntax of the programming language. Under our formulation of the logic $\mathcal{V}$, this desideratum is violated by the value formula $(V \mapsto \Psi)$ at function type, which mentions the programming language value V.

This issue can be addressed, by replacing the basic value formula $(V \mapsto \Psi)$ with the alternative $(\phi \mapsto \Psi)$, already mentioned in Sect. 3. Such a change also has a practical motivation. The formula $(\phi \mapsto \Psi)$ declares a precondition and postcondition for function application, supporting a useful specification style.

Definition 26. The *pure behavioural logic* $\mathcal{F}$ is defined by replacing rule (2) in Fig. 2 with the alternative:

$$\frac{\phi \in VF(\rho) \qquad \Psi \in CF(\tau)}{(\phi \mapsto \Psi) \in VF(\rho \to \tau)}\,(2^*)$$

The semantics is modified by defining $V \models (\phi \mapsto \Psi)$ using formula (2) of Sect. 3.

Proposition 27. *If the open extension of $\equiv_{\mathcal{V}}$ is compatible then the logics $\mathcal{V}$ and $\mathcal{F}$ are equi-expressive. Similarly, if the open extension of $\sqsubseteq_{\mathcal{V}^+}$ is compatible then the positive fragments $\mathcal{V}^+$ and $\mathcal{F}^+$ are equi-expressive.*

Proof. The definition of $(\phi \mapsto \Psi)$ within $\mathcal{V}$, given in (1) of Sect. 3, can be used as the basis of an inductive translation from $\mathcal{F}$ to $\mathcal{V}$ (and from $\mathcal{F}^+$ to $\mathcal{V}^+$).

For the reverse translation, whose correctness proof is more interesting, we give a little more detail. Every value/computation formula, ϕ/Φ, of $\mathcal{V}$ is inductively translated to a corresponding formula $\widehat{\phi}/\widehat{\Phi}$ of $\mathcal{F}$. The interesting case is:

$$\widehat{(V \mapsto \Phi)} \;:=\; (\psi_V \mapsto \widehat{\Phi}),$$

where ψ_V is a formula such that: $V \models_{\mathcal{F}} \psi_V$; and, for any ψ, if $V \models_{\mathcal{F}} \psi$ then $\psi_V \to \psi$ (meaning that $V' \models_{\mathcal{F}} \psi_V$ implies $V' \models_{\mathcal{F}} \psi$, for all V'). Such a formula ψ_V is easily constructed as a countable conjunction (cf. Lemma 17). One then proves, by induction on types, that the $\mathcal{F}$-semantics of $\widehat{\phi}$ (resp. $\widehat{\Phi}$) coincides with the $\mathcal{V}$-semantics of ϕ (resp. Φ). In the case for $\widehat{(V \mapsto \Phi)}$, the induction hypothesis is used to establish that any V' satisfying $V' \models_{\mathcal{F}} \psi_V$ enjoys the property that $V' \equiv_{\mathcal{V}} V$. It then follows from the compatibility of $\equiv_{\mathcal{V}}$ that $WV' \equiv_{\mathcal{V}} WV$, for any W of appropriate type, whence $WV' \equiv_{\mathcal{F}} WV$. The rest of the proof can easily be erected around these observations. □

Combining the above proposition with Theorem 1 we obtain the following.

Corollary 28. *Suppose $\mathcal{O}$ is a decomposable family of Scott-open modalities. Then $\equiv_{\mathcal{F}}$ coincides with $\equiv_{\mathcal{V}}$, and $\sqsubseteq_{\mathcal{F}^+}$ coincides with $\sqsubseteq_{\mathcal{V}^+}$. Hence the open extensions of $\equiv_{\mathcal{F}}$ and $\sqsubseteq_{\mathcal{F}^+}$ are compatible.*

We do not know any proof of the compatibility of the $\equiv_{\mathcal{F}}$ and $\sqsubseteq_{\mathcal{F}^+}$ relations that does not go via the logic $\mathcal{V}$. In particular, the compatibility property of the **fix** operator seems difficult to establish directly for $\equiv_{\mathcal{F}}$ and $\sqsubseteq_{\mathcal{F}^+}$.

8 Discussion and Related Work

The behavioural logics considered in this paper are designed for the purpose of clarifying the notion of 'behavioural property', and for defining behavioural equivalence. As infinitary propositional logics, they are not directly suited to practical applications such as specification and verification. Nevertheless, they serve as low-level logics into which more practical finitary logics can be translated. For this, the closure of the logics under infinitary propositional logic is important. For example, there are standard translations of quantifiers and least and greatest fixed points into infinitary propositional logic. Also, in the case of global store, Hoare triples translate into logical combinations of modal formulas.

Our approach, of basing logics for effects on behavioural modalities, may potentially inform the design of practical logics for specifying and reasoning about effects. For example, Pitts' *evaluation logic* was an early logic for general computational effects [18]. In the light of the general theory of modalities in the present paper, it seems natural to replace the built-in $\Box$ and $\Diamond$ modalities of evaluation logic, with effect-specific modalities, as in Sect. 3.

The *logic for algebraic effects*, of Plotkin and Pretnar [23], axiomatises effectful behaviour by means of an equational theory over the signature of effect operations, following the algebraic approach to effects advocated by Plotkin and Power [22]. Such equational axiomatisations are typically sound with respect to more than one notion of program equivalence. The logic of [23] can thus be used to soundly reason about program equivalence, but does not in itself determine a notion of program equivalence. Instead, our logic is specifically designed as a vehicle for defining program equivalence. In doing so, our modalities can be viewed as a chosen family of 'observations' that are compatible with the effects present in the language. It is the choice of modalities that determines the equational properties that the effect operations satisfy.

The logic of [23] itself makes use of modalities, called *operation modalities*, each associated with a single effect operations in Σ. It would be natural to replace these modalities, which are syntactic in nature, with behavioural modalities of the form we consider. Similarly, our behavioural modalities appear to offer a promising basis for developing a modality-based refinement-type system for algebraic effects. In general, an important advantage we see in the use of behavioural modalities is that our notion of *strong decomposability* appears related to the availability of compositional proof principles for modal properties. This is a promising avenue for future exploration.

A rather different approach to logics for effects has been proposed by Goncharov, Mossakowski and Schröder [3,16]. They assume a semantic setting in which the programming language is rich enough to contain a *pure fragment* that itself acts as a program logic. This approach is very powerful for certain effects. For example, Hoare logic can be derived in the case of global store. However, it appears not as widely adaptable across the range of effects as our approach.

Our logics exhibit certain similarities in form with the endogenous logic developed in Abramsky's *domain theory in logical form* [2]. Our motivation and approach are, however, quite different. Whereas Abramsky shows the usefulness of an axiomatic approach to a finitary logic as a way of characterising denotational equality, the present paper shows that there is a similar utility in considering an infinitary logic from a semantic perspective (based on operational semantics) as a method of defining behavioural equivalence.

The work in this paper has been carried out for fine-grained call-by-value [13], which is equivalent to call-by-value. The definitions can, however, be adapted to work for call-by-name, and even call-by-push-value [11]. Adding type constructors such as sum and product is also straightforward. We have not checked the generalisation to arbitrary recursive types, but we do not foresee any problem.

An omission from the present paper is that we have not said anything about *contextual equivalence*, which is often taken to be the default equivalence for applicative languages. In addition to determining the logically defined preorders/equivalences, the choice of the set $\mathcal{O}$ of modalities gives rise to a natural definition of *contextual preorder*, namely the largest compatible preorder that, on computations of unit type $\mathbf{1}$, is contained in the $\preceq$ relation from Sect. 4. The compatibility of $\sqsubseteq_{\mathcal{V}^+}$ established in the present paper means that we have the expected relation inclusions $\equiv_{\mathcal{V}} \subseteq \sqsubseteq_{\mathcal{V}^+} \subseteq \sqsubseteq_{\mathrm{ctxt}}$. It is an interesting question whether the logic can be restricted to characterise contextual equivalence/preorder. A more comprehensive investigation of contextual equivalence is being undertaken, in ongoing work, by Aliame Lopez and the first author.

The crucial notion of modality, in the present paper, was adapted from the notion of *observation* in [8]. The change from a set of trees of type $\mathbf{N}$ (an observation) to a set of unit-type trees (a modality) allows value formulas to be lifted to computation formulas, analogously to *predicate lifting* in coalgebra [7], which is a key characteristic of our modalities. Properties of *Scott-openness* and *decomposability* play a similar role the present paper to the role they play in [8]. However, the notion of decomposability for modalities (Definition 11) is more subtle than the corresponding notion for observations in [8].

There are certain limitations to the theory of modalities in the present paper. For example, for the combination of probability and nondeterminism, one might naturally consider modalities $\Diamond \mathsf{P}_r$ and $\Box \mathsf{P}_r$ asserting the possibility and necessity of the termination probability exceeding r. However, the decomposability property fails. It appears that this situation can be rescued by changing to a quantitative logic, with a corresponding notion of quantitative modality. This is a topic of ongoing research.

Acknowledgements. We thank Francesco Gavazzo, Aliaume Lopez and the anonymous referees for helpful discussions and comments.

References

1. Abramsky, S.: The lazy λ-calculus. In: Research Topics in Functional Programming, pp. 65–117 (1990)
2. Abramsky, S.: Domain theory in logical form. Ann. Pure Appl. Log. **51**(1–2), 1–77 (1991)
3. Goncharov, S., Schröder, L.: A relatively complete generic Hoare logic for order-enriched effects. In: Proceedings of the 28th Annual Symposium on Logic in Computer Science (LICS 2013), pp. 273–282. IEEE (2013)
4. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM (JACM) **32**(1), 137–161 (1985)
5. Howe, D.J.: Equality in lazy computation systems. In: Proceedings of the 4th IEEE Symposium on Logic in Computer Science, pp. 198–203 (1989)
6. Howe, D.J.: Proving congruence of bisimulation in functional programming languages. Inf. Comput. **124**(2), 103–112 (1996)
7. Jacobs, B.: Introduction to Coalgebra: Towards Mathematics of States and Observation. Cambridge University Press, Cambridge (2016)
8. Johann, P., Simpson, A., Voigtländer, J.: A generic operational metatheory for algebraic effects. In: Logic in Computer Science, pp. 209–218 (2010)
9. Ugo, D.L., Gavazzo, F., Levy, P.B.: Effectful applicative bisimilarity: Monads, relators, and the Howe's method. In: Logic in Computer Science, pp. 1–12 (2017)
10. Lassen, S.B.: Relational Reasoning about Functions and Nondeterminism. Ph.D. thesis, BRICS (1998)
11. Levy, P.B.: Call-by-push-value: decomposing call-by-value and call-by-name. Higher-Order Symbol. Comput. **19**(4), 377–414 (2006)
12. Levy, P.B.: Similarity quotients as final coalgebras. In: Hofmann, M. (ed.) FoSSaCS 2011. LNCS, vol. 6604, pp. 27–41. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19805-2_3
13. Levy, P.B., Power, J., Thielecke, H.: Modelling environments in call-by-value programming languages. Inf. Comput. **185**(2), 182–210 (2003)
14. Milner, R.: A Calculus of Communicating Systems. Springer, Heidelberg (1982). https://doi.org/10.1007/3-540-10235-3
15. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991)
16. Mossakowski, T., Schröder, L., Goncharov, S.: A generic complete dynamic logic for reasoning about purity and effects. Formal Aspects Comput. **22**(3–4), 363–384 (2010)
17. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981). https://doi.org/10.1007/BFb0017309
18. Pitts, A.: Evaluation logic. In: Birtwistle, G. (ed.) 4th Higher Order Workshop. Workshops in Computing, pp. 162–189. Springer, London (1990). https://doi.org/10.1007/978-1-4471-3182-3_11
19. Pitts, A.: Parametric polymorphism and operational equivalence. Math. Struct. Comput. Sci. **10**, 321–359 (2000)
20. Plotkin, G.: LCF considered as a programming language. Theor. Comput. Sci. **5**(3), 223–255 (1977)

21. Plotkin, G., Power, J.: Adequacy for algebraic effects. In: Honsell, F., Miculan, M. (eds.) FoSSaCS 2001. LNCS, vol. 2030, pp. 1–24. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45315-6_1
22. Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45931-6_24
23. Plotkin, G., Pretnar, M.: A logic for algebraic effects. In: Proceedings of the Logic in Computer Science, pp. 118–129 (2008)
24. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on the Foundations of Computer Science, pp. 46–57 (1977)
25. Thijs, A.M.: Simulation and fixpoint semantics. Ph.D. thesis (1996)