Quantitative Logics for Equivalence of Effectful Programs

Niels Voorneveld[1,2]

Faculty of Mathematics and Physics
University of Ljubljana
Ljubljana, Slovenia

Abstract

In order to reason about effects, we can define quantitative formulas to describe behavioural aspects of effectful programs. These formulas can for example express probabilities that (or sets of correct starting states for which) a program satisfies a property. Fundamental to this approach is the notion of quantitative modality, which is used to lift a property on values to a property on computations. Taking all formulas together, we say that two terms are equivalent if they satisfy all formulas to the same quantitative degree. Under sufficient conditions on the quantitative modalities, this equivalence is equal to a notion of Abramsky's applicative bisimilarity, and is moreover a congruence. We investigate these results in the context of Levy's call-by-push-value with general recursion and algebraic effects. For example, the results apply to (combinations of) nondeterministic choice, probabilistic choice, global store, and error.

Keywords: Quantitative Logic, Program Equivalence, Algebraic Effects, Behavioural Equivalence, Applicative Bisimilarity, Modalities, Call-by-push-value, Probability, Nondeterminism, Global Store, Error, Effect Combinations.

1 Introduction

There are many notions of program equivalence for languages with effects. In this paper, we explore the notion of *behavioural equivalence*, which states that programs may be considered *behaviourally equivalent* if they satisfy the same behavioural properties. This can be made rigorous by defining a logic, where each formula ϕ denotes a certain behavioural property. We write $(\underset{\sim}{P} \models \phi)$ to express the satisfaction of formula ϕ by term $\underset{\sim}{P}$, which is usually given by a Boolean truth value (true or false). Two terms $\underset{\sim}{P}$ and $\underset{\sim}{R}$ are said to be behaviourally equivalent if they satisfy the same formulas. Such an approach is taken in for example [9].

In particular, we use this method to define equivalence for a language with *algebraic effects* in the sense of Plotkin and Power [27]. Effects can be seen as aspects of computation which involves interaction with the world 'outside' the environment in which the program runs. They include: exceptions, nondeterminism, probabilistic choice, global store, input/output, cost, etc. The examples given have common ground in the work of Moggi [22], and can moreover be expressed by specific effect triggering operations making them 'algebraic' in nature. In the presence of such algebraic effects, computation terms need not simply reduce to a single *terminal term* (that is a *value*), they may also invoke effects on the way. Following [27,13], we consider a computation term to evaluate to an *effect tree*, whose nodes are effect operators and leaves are terminal terms. The paper [29] introduced modalities that lift boolean properties of values to boolean properties of the trees modelling their computations. See [24,23,28] for alternative ways in which logics can be used to describe properties of effects.

The use of a Boolean logic does however not readily adapt to several examples of effects, for example the combination of probability and nondeterminism. The literature on compositional program verification shows the usefulness of quantitative (e.g. real-number valued) program logics for verifying programs with probabilistic

[1] This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

[2] Email: Niels.Voorneveld@fmf.uni-lj.si

behaviour, possibly in combination with nondeterminism [14,21]. The paper [29] develops a general Boolean-valued framework which, although featuring many examples, does not apply to this combination of probability and nondeterminism.

This paper provides a general framework for *quantitative* logics for expressing behavioural properties of programs with effects, generalising the Boolean-valued framework from [29]. We consider a quantitative (quantity-valued) satisfaction relation '$\models$', where $(\underline{\underline{P}} \models \phi)$ is given by an element from a quantitative truth space $\mathbb{A}$ (a *degree* of satisfaction). This allows us to ask open questions about programs, like "*What is the probability that ...*" or "*What are the correct global starting states for ...*". We define equivalence by stating that programs $\underline{\underline{P}}$ and $\underline{\underline{R}}$ are equivalent, if for any formula ϕ we have $(\underline{\underline{P}} \models \phi) = (\underline{\underline{R}} \models \phi)$ ($\underline{\underline{P}}$ satisfies ϕ precisely as much as $\underline{\underline{R}}$ does). A key feature of the logic is the use of *quantitative modalities* to lift quantitative properties on value types to quantitative properties on computation types.

As in [29], we are able to establish that the behavioural equivalence as above is *compatible*, therefore a *congruence*, as long as suitable properties on the quantitative modalities are satisfied. These properties require notions of *monotonicity*, *continuity*, and a notion of preservation over sequencing called *decomposability*. As in [29], compatibility is established by proving that given one of the properties (leaf-monotonicity), our behavioural equivalence is equal to an effect-sensitive notion of Abramsky's *applicative bisimilarity* [1,3]. Given further properties on the modalities, this relation can be proven to be compatible using Howe's method [11].

The main contribution of this paper is the generalisation of [29], and the corresponding generalised results. This goes through smoothly, though there are some subtleties like what to take as primitive in a quantitative setting. In particular, we will see the necessity of a threshold operation. The other main contributions are the examples illustrating the quantitative approach. Some examples such as the combination of nondeterminism with probabilistic choice, or with global store, do not fit into the Boolean-valued framework of [29], but do work here [3]. But there are also examples, such as probability, global store, and cost, whose treatment is more natural in our quantitative setting, even though they also fit in the framework of [29].

As a vehicle of our investigation we use Levy's call-by-push-value (CBPV) [17,16], together with general recursion and the aforementioned algebraic effects. As such, it generalises [29] in a second way by using call-by-push-value to incorporate both call-by-name (CBN) and call-by-value (CBV) evaluation strategies. This is significant, since once either divergence or effects are present, the distinction between the reduction strategies becomes vital. For example, if we take some probabilistic choice por signifying a fair coin flip, we have that '$\mathsf{por}(\lambda x\,.\,\overline{0}, \lambda x\,.\,\overline{1}) \equiv \lambda x\,.\,\mathsf{por}(\overline{0}, \overline{1})$' holds in CBN, but not in CBV. So it is interesting to consider CBPV, as it expresses both these behaviours. The distinction is expressed in the difference between *producer*-types FA where one explicitly observes effects, and types like $\mathsf{A} \to \underline{\mathsf{C}}$ where the observation of effects is postponed to a later moment. As such, this language is an ideal backdrop for studying effects.

In Section 2 we give the operational semantics of the language, starting with the effect-free version and working towards our treatment of algebraic effects. In Section 3 we present our quantitative logic, introducing quantitative modalities to deal with the observation of effects. In Section 4 we look at the resulting behavioural equivalence and the properties that establish the congruence property (or compatibility in its technical form). In Section 5 we relate this equivalence to applicative (bi)similarities by defining a *relator* using our modalities. This then allows us to adapt a Howe's method proof of compatibility from [3,29] for this equivalence. We finish in Section 6 with some discussions.

2 Operational semantics

We use a simply-typed call-by-push-value functional language as in [16,17], together with general recursion and a ground type for natural numbers, making it a call-by-push-value variant of PCF [25]. To this, we add algebraic-effect-triggering operators in the sense of Plotkin and Power [27]. We first focus on the effect-free part of the language, as we want to consider effects independently of the underlying language.

2.1 The language

We give a brief overview of the language and its semantics. The types are divided into two flavours, *Value types* and *Computation types*. Value types contain value terms that are *passive*, and do not compute anything on their own. Computation types contain computation terms which are *active*, which means they either return something to or ask something of the environment.

Value types A, B and computation types $\underline{\mathsf{C}}, \underline{\mathsf{D}}$ are given by:

$$\mathsf{A}, \mathsf{B} ::= \mathsf{U}\,\underline{\mathsf{C}} \mid 1 \mid \mathsf{N} \mid \Sigma_{i\in I}\mathsf{A}_i \mid \mathsf{A} \times \mathsf{A} \qquad\qquad \underline{\mathsf{C}}, \underline{\mathsf{D}} ::= \mathsf{FA} \mid \mathsf{A} \to \underline{\mathsf{C}} \mid \Pi_{i\in I}\underline{\mathsf{C}}_i$$

[3] The combination of global store and nondeterminism is possible in the framework of [29], only if one considers *angelic* (helpful) nondeterminism. The problem is with general (neutral) or demonic (antagonistic) nondeterminism, combined with global store.

$$\frac{}{\Gamma \vdash^v * : \mathsf{1}} \qquad \frac{}{\Gamma \vdash^v \mathsf{Z} : \mathsf{N}} \qquad \frac{\Gamma \vdash^v V : \mathsf{N}}{\Gamma \vdash^v \mathsf{S}(V) : \mathsf{N}} \qquad \frac{\Gamma \vdash^v V : \mathsf{N} \qquad \Gamma \vdash^c \underline{M} : \underline{\mathsf{C}} \qquad \Gamma, x : \mathsf{N} \vdash^c \underline{N} : \underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{case}\ V\ \mathsf{in}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} : \underline{\mathsf{C}}}$$

$$\frac{}{\Gamma, x : \mathsf{A}, \Gamma' \vdash^v x : \mathsf{A}} \qquad \frac{\Gamma \vdash^v V : \mathsf{A} \qquad \Gamma, x : \mathsf{A} \vdash^c \underline{M} : \underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{let}\ x\ \mathsf{be}\ V\,.\,\underline{M} : \underline{\mathsf{C}}} \qquad \frac{\Gamma \vdash^v V : \mathsf{A}}{\Gamma \vdash^c \mathsf{return}(V) : \mathsf{FA}} \qquad \frac{\Gamma \vdash^c \underline{M} : \mathsf{FA} \qquad \Gamma, x : \mathsf{A} \vdash^c \underline{N} : \underline{\mathsf{C}}}{\Gamma \vdash^c \underline{M}\ \mathsf{to}\ x\,.\,\underline{N} : \underline{\mathsf{C}}}$$

$$\frac{\Gamma \vdash^c \underline{M} : \underline{\mathsf{C}}}{\Gamma \vdash^v \mathsf{thunk}(\underline{M}) : \mathsf{U}\,\underline{\mathsf{C}}} \qquad \frac{\Gamma \vdash^v V : \mathsf{U}\,\underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{force}(V) : \underline{\mathsf{C}}} \qquad \frac{\Gamma, x : \mathsf{A} \vdash^c \underline{M} : \underline{\mathsf{C}}}{\Gamma \vdash^c \lambda x : \mathsf{A}\,.\,\underline{M} : \mathsf{A} \to \underline{\mathsf{C}}} \qquad \frac{\Gamma \vdash^v V : \mathsf{A} \qquad \Gamma \vdash^c \underline{M} : \mathsf{A} \to \underline{\mathsf{C}}}{\Gamma \vdash^c \underline{M} \cdot V : \underline{\mathsf{C}}}$$

$$\frac{\Gamma \vdash^v V : \mathsf{A}_j}{\Gamma \vdash^v (j, V) : \Sigma_{i \in I} \mathsf{A}_i}\, j \in I \qquad \frac{\Gamma \vdash^v V : \Sigma_{i \in I} \mathsf{A}_i \qquad \Gamma, x : \mathsf{A}_i \vdash^c \underline{M}_i : \underline{\mathsf{C}} \ \text{ for each } \ i \in I}{\Gamma \vdash^c \mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x)\,.\,\underline{M}_i, \ldots\} : \underline{\mathsf{C}}} \qquad \frac{\Gamma \vdash^v V : \mathsf{A} \qquad \Gamma \vdash^v V' : \mathsf{B}}{\Gamma \vdash^v (V, V') : \mathsf{A} \times \mathsf{B}}$$

$$\frac{\Gamma \vdash^v V : \mathsf{A} \times \mathsf{B} \qquad \Gamma, x : \mathsf{A}, y : \mathsf{B} \vdash^c \underline{M} : \underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{pm}\ V\ \mathsf{as}\ (x, y)\,.\,\underline{M} : \underline{\mathsf{C}}} \qquad \frac{\Gamma \vdash^c \underline{M}_i : \underline{\mathsf{C}}_i \ \text{ for each } \ i \in I}{\Gamma \vdash^c \langle \underline{M}_i \mid i \in I \rangle : \Pi_{i \in I} \underline{\mathsf{C}}_i} \qquad \frac{\Gamma \vdash^c \underline{M} : \Pi_{i \in I} \underline{\mathsf{C}}_i}{\Gamma \vdash^c \underline{M} \cdot j : \underline{\mathsf{C}}_j} \qquad \frac{\Gamma \vdash^c \underline{M} : \mathsf{U}\,\underline{\mathsf{C}} \to \underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{fix}(\underline{M}) : \underline{\mathsf{C}}}$$

Fig. 1. Typing rules

where I is any *finite* indexing set. By asserting finiteness of I in the case of product types, the number of program terms is kept countable (a property which will have benefits later on in the formulation of the logic). The type $\mathsf{U}\,\underline{\mathsf{C}}$ is a thunk type, which consists of terms which are *frozen*. These terms were initially computation terms but are made inactive by packaging them into a *thunk*. The type N is the type of natural numbers, containing the non-negative integers. With this type, we can program any computable function on the natural numbers as in PCF [25]. The type FA is a *producer* type, which actively evaluates and *returns* values of type A to the current environment. As was stated, this is the type at which we can observe effects. The type $\mathsf{A} \to \underline{\mathsf{C}}$ is a type of functions, which is a computation type since its terms are actively awaiting input.

We have a countably-infinite collection of term variables x, and term *contexts*: $\Gamma ::= \emptyset \mid \Gamma, x : \mathsf{A}$.
A term is *closed* if it has context $\emptyset$, else it's *open*. Note that contexts only contain Value types, meaning that like in call-by-value, we can only ever substitute value terms. This is no loss of generality, as we can simulate substituting computation terms by packaging them into a thunk. The terms of the language are as follows:

Value terms: $V, W ::= * \mid \mathsf{Z} \mid \mathsf{S}(V) \mid x \mid \mathsf{thunk}(\underline{M}) \mid (i, V) \mid (V, W)$

Computation terms: $\underline{M}, \underline{N} ::= \mathsf{case}\ V\ \mathsf{in}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \mid \mathsf{let}\ x\ \mathsf{be}\ V\,.\,\underline{M} \mid \mathsf{return}(V) \mid \underline{M}\ \mathsf{to}\ x\,.\,\underline{N} \mid \mathsf{force}(V)$
$\mid \lambda x : \mathsf{A}\,.\,\underline{M} \mid \underline{M} \cdot V \mid \mathsf{pm}\ V\ \mathsf{as}\ \{\ldots, (i.x)\,.\,\underline{M}_i, \ldots\} \mid \mathsf{pm}\ V\ \mathsf{as}\ (x, y)\,.\,\underline{M} \mid \langle \underline{M}_i \mid i \in I \rangle \mid \underline{M} \cdot i \mid \mathsf{fix}(\underline{M})$

We underline terms $\underline{M}$ and types $\underline{\mathsf{C}}$ when they are *computation* terms and *computation* types respectively. We will also use $\underline{\mathsf{E}}, \underline{\mathsf{F}}$ and $\underline{P}, \underline{R}$ to denote *general* types and their terms, e.g. they could be either value or computation types/terms. Following [16], their typing rules are given in Fig. 1, where we distinguish two typing judgements, $\vdash^v$ and $\vdash^c$, for value and computation terms respectively. We write $\mathit{Terms}(\underline{\mathsf{E}})$ for the set of closed terms of type $\underline{\mathsf{E}}$. Note the addition of the fixpoint operator $\mathsf{fix}(-)$ which has been added to allow for general recursion and hence divergence. We write $\overline{n} : \mathsf{N}$ for the numeral representing the n-th natural number.

2.2 Semantics

We give the semantics of this language by specifying a reduction strategy for computation terms in the style of a CK-machine [5]. We distinguish a special class of computation terms, called *terminal terms*, which will not reduce further. They consist of: $\mathsf{return}(V) : \mathsf{FA}$, $\lambda x : \mathsf{A}\,.\,\underline{M} : \mathsf{A} \to \underline{\mathsf{C}}$, and $\langle \underline{M}_i \mid i \in I \rangle : \Pi_{i \in I} \underline{M}_i$.

We first give the rules for terms we can directly reduce. We denote these using relation symbol $\rightsquigarrow$:

(i) $\mathsf{case}\ \mathsf{Z}\ \mathsf{in}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \rightsquigarrow \underline{M}$.
(ii) $\mathsf{case}\ \mathsf{S}(V)\ \mathsf{in}\ \{\underline{M}, \mathsf{S}(x) \Rightarrow \underline{N}\} \rightsquigarrow \underline{N}[V/x]$.
(iii) $\mathsf{let}\ x\ \mathsf{be}\ V\,.\,\underline{M} \rightsquigarrow \underline{M}[V/x]$.
(iv) $\mathsf{force}(\mathsf{thunk}(\underline{M})) \rightsquigarrow \underline{M}$.
(v) $\mathsf{pm}\ (j, V)\ \mathsf{as}\ \{\ldots, (i.x)\,.\,\underline{M}_i, \ldots\} \rightsquigarrow \underline{M}_j[V/x]$.
(vi) $\mathsf{pm}\ (V, W)\ \mathsf{as}\ (x, y)\,.\,\underline{M} \rightsquigarrow \underline{M}[V/x, W/y]$.
(vii) $\mathsf{fix}(\underline{M}) \rightsquigarrow \underline{M} \cdot \mathsf{thunk}(\mathsf{fix}(\underline{M}))$.

The behaviour of the other non-terminal computation terms; $\underline{M}\ \mathsf{to}\ x\,.\,\underline{N}$, $\underline{M} \cdot V$ and $\underline{M} \cdot i$, is implemented using a system of *stacks* defined recursively: $S, Z ::= \varepsilon \mid S \circ (-)\ \mathsf{to}\ x\,.\,\underline{M} \mid S \circ V \mid S \circ j$.
We write $S\{\underline{M}\}$ for the computation resulting from applying S to $\underline{M}$, which can be seen as evaluating the program $\underline{M}$ within the environment S.

$\varepsilon\{\underline{M}\} := \underline{M}$ $\qquad$ $(S \circ (-)\ \mathsf{to}\ x\,.\,\underline{N})\{\underline{M}\} := S\{\underline{M}\ \mathsf{to}\ x\,.\,\underline{N}\}$
$(S \circ V)\{\underline{M}\} := S\{\underline{M} \cdot V\}$ $\qquad$ $(S \circ i)\{\underline{M}\} := S\{\underline{M} \cdot i\}$

Whenever one encounters a computation of which one needs to first evaluate a subterm, one unfolds the continuation into the Stack and focusses on evaluating that subterm. This method is given by the stack reduction relation $\rightarrowtail$ in the following way:

(i) If $\underline{M} \rightsquigarrow \underline{N}$, then $(S, \underline{M}) \rightarrowtail (S, \underline{N})$.
(ii) $(S, \underline{M}\ \mathsf{to}\ x\,.\,\underline{N}) \rightarrowtail (S \circ (-)\ \mathsf{to}\ x\,.\,\underline{N}, \underline{M})$.
(iii) $(S \circ (-)\ \mathsf{to}\ x\,.\,\underline{N}, \mathsf{return}(V)) \rightarrowtail (S, \underline{N}[V/x])$.
(iv) $(S, \underline{M} \cdot V) \rightarrowtail (S \circ V, \underline{M})$.
(v) $(S \circ V, \lambda x : \mathsf{A}\,.\,\underline{M}) \rightarrowtail (S, \underline{M}[V/x])$.
(vi) $(S, \underline{M} \cdot j) \rightarrowtail (S \circ j, \underline{M})$.
(vii) $(S \circ j, \langle \underline{M}_i \mid i \in I\rangle) \rightarrowtail (S, \underline{M}_j)$.

2.3 Adding algebraic effect operators

We add algebraic effects in the style of [13], given by specific effect operators. We use a type variable α for computation types. Effects are given by operators of the following arities (like in [13,26]):

$\alpha^n \to \alpha \mid \mathsf{N} \times \alpha^n \to \alpha \mid \alpha^{\mathsf{N}} \to \alpha$.

For each effect under consideration, we bundle together effect operators pertaining to that effect in a set called an *effect signature* Σ. Given such a signature, new computation terms can be constructed for each $\mathsf{op} \in \Sigma$, according to the typing rules in Fig. 2.

$$\frac{\forall 1 \le i \le n.(\Gamma \vdash^c \underline{M}_i : \underline{\mathsf{C}})}{\Gamma \vdash^c \mathsf{op}(\underline{M}_1, \ldots, \underline{M}_n) : \underline{\mathsf{C}}}\ \mathsf{op} : \alpha^n \to \alpha \qquad \frac{\Gamma, x : \mathsf{N} \vdash^c \underline{M} : \underline{\mathsf{C}}}{\Gamma \vdash^c \mathsf{op}(x\,.\,\underline{M}) : \underline{\mathsf{C}}}\ \mathsf{op} : \alpha^{\mathsf{N}} \to \alpha$$

$$\frac{\Gamma \vdash^v V : \mathsf{N} \qquad \forall 1 \le i \le n.(\Gamma \vdash^c \underline{M}_i : \underline{\mathsf{C}})}{\Gamma \vdash^c \mathsf{op}(V, \underline{M}_1, \ldots, \underline{M}_n) : \underline{\mathsf{C}}}\ \mathsf{op} : \mathsf{N} \times \alpha^n \to \alpha$$

Fig. 2. Effect typing rules

We look at the running examples of this paper.

Example 1 (Probabilistic choice) *The effect of probability is implemented by the signature* $\Sigma_p := \{\mathsf{por} : \alpha^2 \to \alpha\}$*, where we have a single binary operator for fair probabilistic choice. The computation* $\mathsf{por}(\underline{M}, \underline{N})$ *will have a* $\frac{1}{2}$ *probability of evaluating* $\underline{M}$*, and* $\frac{1}{2}$ *probability of evaluating* $\underline{N}$*.*

Example 2 (Global store) *In the case of global variables for natural numbers, we define a signature* $\Sigma_g := \bigcup_{l \in L} \{\mathit{lookup}_l : \alpha^{\mathsf{N}} \to \alpha, \mathit{update}_l : \mathsf{N} \times \alpha \to \alpha\}$*, where we have a set of locations* L *for storing natural numbers. The computation* $\mathit{lookup}_l(x\,.\,M)$ *looks up the number at location* l *and substitutes it for* x *in* M*. The computation* $\mathit{update}_l(\overline{n}, M)$ *stores* n *at* l *and then continues with the computation* M*.*

Example 3 (Probabilistic choice and global store) *We will also consider the combination of the previous two examples, probabilistic choice with global store, given by effect signature* $\Sigma_{pg} := \Sigma_p \cup \Sigma_g$*.*

Example 4 (Cost) *If we want to keep track of costs of an evaluation, we take the signature* $\Sigma_c := \{\mathsf{cost}_c : \alpha \to \alpha \mid c \in C\}$*, where we have a countable set of real-valued costs* C*. The computation* $\mathsf{cost}_c(M)$ *assigns a cost of* c *to the evaluation of* M*. This cost can represent a time delay or some other resource.*

Example 5 (Combinations with nondeterminism) *We consider a binary operator* $\mathit{nor} : \alpha^2 \to \alpha$ *for nondeterministic choice, which contrary to probabilistic choice is entirely unpredictable. One interpretation is to consider it under the control of some external agent or scheduler (e.g. a compiler), which one may wish to model as being cooperative (angelic), antagonistic (demonic), or neutral. We will consider binary nondeterminism and its operator in combination with any one of the previous three examples. The resulting signatures are named* Σ_{pn}*,* Σ_{gn}*,* Σ_{gpn}*, and* Σ_{cn} *respectively.*

Example 6 (Combinations with error) *Lastly, given some set of error messages* E*, we consider adding error raising effect operations* $\{\mathsf{raise}_e : \alpha^0 \to \alpha \mid e \in E\}$ *to the language, where* $\mathsf{raise}_e()$ *stops the evaluation of a term, and displays message* e*. There is no continuation possible afterwards.*

In the presence of such effects, the evaluation of a computation term might halt when encountering an algebraic effect operator. We broaden the semantics, where a computation term now evaluates to an *effect tree*, a coinductively generated term using operations from our effect signature Σ together with terminal terms and a symbol for divergence $\bot$. This idea appears in [27], but here we adapt the formulation from [13] to call-by-push-value.

We define the notion of an *effect tree* over any set X, where X can be thought of as a set of terminal terms.

Definition 2.1 An *effect tree* (henceforth *tree*) over a set X, determined by a signature Σ of effect operations, is a labelled and possibly non-well-founded tree whose nodes have the possible forms given below:

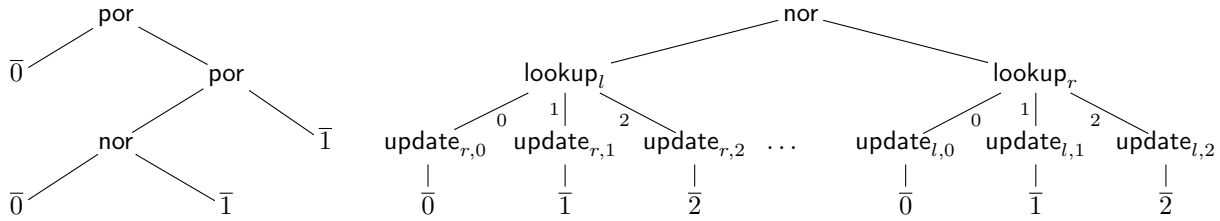

Fig. 3. Tree examples: The unpredictable coin and the nondeterministic copier.

(i) A leaf node labelled $\bot$ (the symbol for nontermination/divergence).

(ii) A leaf node labelled x where $x \in X$.

(iii) A node labelled 'op' with children $t_1, \ldots, t_n$, when $\mathsf{op} \in \Sigma$ has arity $\alpha^n \to \alpha$.

(iv) A node labelled 'op' with children $t_0, t_1, \ldots$, when $\mathsf{op} \in \Sigma$ has arity $\alpha^{\mathsf{N}} \to \alpha$.

(v) A node labelled 'op_m' where $m \in \mathbb{N}$ with children $t_1, \ldots, t_n$, when $\mathsf{op} \in \Sigma$ has arity $\mathsf{N} \times \alpha^n \to \alpha$.

The set of trees over set X and signature Σ is denoted $T_\Sigma(X)$. We can equip this set with a partial order $\leq$, where $t \leq r$ if t can be constructed from r by pruning (possibly infinitely many) subtrees and labelling the pruning points with $\bot$. Moreover, the preorder is ω-*complete*, so each ascending chain of trees $t_0 \leq t_1 \leq \ldots$ has a least upper bound $\sqcup_n t_n$.

For any $x \in X$, we write $\eta(x) \in T_\Sigma(X)$ for the tree which only consists of one leaf labelled x. We have a map $\mu : T_\Sigma(T_\Sigma(X)) \to T_\Sigma(X)$ which flattens a tree of trees into one tree, by transforming the leaves (which are trees) into subtrees. We write $\mathsf{op}\{t_0, \ldots, t_1\}$ or $\mathsf{op}\{m \mapsto t_m\}$ for the tree with node op and children $t_0, t_1, \ldots$.

For each computation type $\underline{\mathsf{C}}$ we define the evaluation map $|-| : \mathit{Terms}(\underline{\mathsf{C}}) \to T_\Sigma(\mathit{Terms}(\underline{\mathsf{C}}))$, which returns a tree, whose leaves are either labelled with $\bot$ or labelled with a terminal term of type $\underline{\mathsf{C}}$. We define this inductively by constructing for each $n \in \mathbb{N}$ the n-th approximation of the tree.

(i) $|S, \underline{M}|_0 := \bot$. (ii) $|\varepsilon, \underline{M}|_{n+1} := \eta(\underline{M})$ if $\underline{M}$ is a terminal computation term.

(iii) $|S, \underline{M}|_{n+1} := |S', \underline{M'}|_n$ if $(S, \underline{M}) \rightarrowtail (S', \underline{M'})$.

(iv) $|S, \mathsf{op}(\underline{M}_1, \ldots, \underline{M}_m)|_{n+1} := \mathsf{op}\{|S, \underline{M}_1|_n, \ldots, |S, \underline{M}_m|_n\}$ if $\mathsf{op} : \alpha^m \to \alpha$.

(v) $|S, \mathsf{op}(x \,.\, \underline{M})|_{n+1} := \mathsf{op}\{m \mapsto |S, \underline{M}[\overline{m}/x]|\}$ if $\mathsf{op} : \alpha^{\mathsf{N}} \to \alpha$.

(vi) $|S, \mathsf{op}(\overline{k}, \underline{M}_1, \ldots, \underline{M}_m)|_{n+1} := \mathsf{op}_k\{|S, \underline{M}_1|_n, \ldots, |S, \underline{M}_m|_n\}$ if $\mathsf{op} : \mathsf{N} \times \alpha^m \to \alpha$.

Using this, we define $|\underline{M}| := \bigsqcup_n |\varepsilon, \underline{M}|_n$. We view $|\underline{M}|$ as an operational semantics of $\underline{M}$ in which $\underline{M}$ is reduced to its (possibly) observable computational behaviours, namely the tree of effect operations potentially performed in the evaluation of M. See Figure 3 for two examples of effect trees.

These trees are still quite syntactic, and may contain lots of unobservable information irrelevant to the real-world behaviour of programs. In the next section, we will set up the quantitative logic which will extract from such trees only the relevant information, using quantitative modalities.

3 Quantitative Logic

We define a quantitative logic expressing *behavioural properties* of terms. Each type has a set of formulas, which can be satisfied by terms of that type to varying *degrees of satisfaction*. These degrees of satisfaction are given by truth values from a countably complete lattice.

A *countably complete lattice* is a set $\mathbb{A}$ with a partial order $\trianglelefteq$, where for each subset $X \subseteq \mathbb{A}$ there is a least upper bound $\sup(X)$ and a greatest lower bound $\inf(X)$. In particular, we define $\boldsymbol{T} := \sup(\mathbb{A}) = \inf(\emptyset)$ as the completely true value, and $\boldsymbol{F} := \inf(\mathbb{A}) = \sup(\emptyset)$ as the completely false value.

We also equip this space with a notion of *negation* or involution, which is a bijective map $\neg : \mathbb{A} \to \mathbb{A}$ such that $\forall a \in \mathbb{A}, \neg(\neg a) = a$ and $\forall a, b \in \mathbb{A}, (a \trianglelefteq b) \Leftrightarrow (\neg b \trianglelefteq \neg a)$. We will use the words involution and negation interchangeably. Given the conditions of an involution, it holds that $\neg \boldsymbol{T} = \boldsymbol{F}$ and $\neg \boldsymbol{F} = \boldsymbol{T}$.[4] Examples of complete lattices with involution/negation used in this paper are:

(i) The *Booleans* $\mathbb{B} := \{\boldsymbol{T}, \boldsymbol{F}\}$, where $\boldsymbol{F} \trianglelefteq \boldsymbol{T}$. Negation swaps the elements.

(ii) The *real number interval* $[0, 1]$, with the usual order. Here $\boldsymbol{T} = 1$, $\boldsymbol{F} = 0$, and $\neg x := 1 - x$.

[4] Results about the positive behavioural equivalence, and the positive logic, do not need negation. So a subset of results in this paper are still valid without it. Moreover, $\mathbb{A}$ need not necessarily be distributive, though in each example it is.

(iii) The powerset $\mathcal{P}(X)$ over some set X, whose order is given by inclusion $\subseteq$, so $\boldsymbol{T} = X$ and $\boldsymbol{F} = \emptyset$. Negation is given by the complement, where $\neg A := X - A = \{x \in X \mid x \notin A\}$.

(iv) For A a complete lattice and X a set, the function space A^X with point-wise order is a complete lattice.

(v) The *infinite interval* $[0, \infty]$ with reversed order. Here, $\boldsymbol{T} = 0$, $\boldsymbol{F} = \infty$, and $\neg x := 1/x$.

We construct a logic for our language in order to define a behavioural preorder. For each type $\underline{\underline{\mathsf{E}}}$, value or computation, we have a set of formulas $\mathit{Form}(\underline{\underline{\mathsf{E}}})$. Greek letters $\phi, \psi, \ldots$ are used for formulas over value types, underlined Greek letters $\underline{\phi}, \underline{\psi}, \ldots$ for formulas over computation types, and underdotted Greek letters $\underline{\underline{\phi}}, \underline{\underline{\psi}}, \ldots$ for formulas over any type. We are aiming to define a quantitative relation $(\underline{\underline{P}} \models \underline{\underline{\phi}})$ to denote the element of $\mathbb{A}$ which describes the degree to which the term $\underline{\underline{P}}$ satisfies the formula $\underline{\underline{\phi}}$ (e.g. this may describe the probability of satisfaction or the amount of time needed for satisfaction). We choose the formulas according to the following two design criteria, as in [29].

Firstly, we design our logic to only contain *behaviourally meaningful* formulas. This means we only want to test properties that are in some sense observable by users or other programs. For the natural numbers type N we have a formula $\{n\}$ which checks whether a term is equal to the numeral $\overline{n}$. For function types we have formulas $V \to \underline{\underline{\phi}}$ which tests a program on an input V, and checks how much the resulting term satisfies $\underline{\underline{\phi}}$.

Secondly, we desire our logic to be as *expressive* as possible. To this end, we add countable disjunction (suprema) $\bigvee$ and conjunction (infima) $\bigwedge$ over formulas, together with negation $\neg$. Moreover, we add two natural quantity-specific primitives: a threshold operation and constants. Both such operations are used frequently (albeit implicitly) in practical examples of quantitative verification, e.g. in [21].

3.1 Quantitative modalities

Fundamental to the design of the logic is how we interpret algebraic effects. In CBPV, effects are observed in producer types FA. In order to formulate *observable* properties of FA-terms in our logic, we include a set of *quantitative modalities* which lift formulas on A to formulas on FA. We bundle our a selection of quantitative modalities together in a set $\mathcal{Q}$.

Each modality $q \in \mathcal{Q}$ denotes a function $[\![q]\!] : T_\Sigma(\mathbb{A}) \to \mathbb{A}$, which is used to translate a *tree of truths* into a singular truth value. Given a quantitative predicate $\Theta : X \to \mathbb{A}$ on a set X, we can use a modality q to lift it to a quantitative predicate $q(\Theta) : T_\Sigma(X) \to \mathbb{A}$ as follows. For $t \in T_\Sigma(X)$, we write $t[\Theta]$ for the tree in $T_\Sigma(\mathbb{A})$ where each non-$\bot$ leaf $x \in X$ is replaced by the value $\Theta(x)$. Then $q(\Theta)(t) := [\![q]\!](t[\Theta])$. [5]

In the examples, we will define the denotation of a modality q by giving for each $n \in \mathbb{N}$ an approximation $[\![q]\!]_n$. These will follow the rules: $[\![q]\!]_0(t) = \boldsymbol{F}$, $[\![q]\!]_n(\bot) = \boldsymbol{F}$, and $[\![q]\!]_{n+1}(\eta(a)) = a$, and effect specific rules given in the examples below. Given these approximations, the denotation $[\![q]\!](t)$ is given by $\sup\{[\![q]\!]_n(t) \mid n \in \mathbb{N}\}$.

Example 1 (Probabilistic choice) *We use as quantitative truth space the real number interval* $\mathbb{A} := [0, 1]$ *with* $\trianglelefteq := \leq$*, which denote probabilities.* [6] *We take a single modality* E *for expectation, where* $(\underline{\underline{M}} \models \mathsf{E}(\Theta))$ *gives the expected value of* $(V \models \Theta)$ *given the probabilistic distribution* $\underline{\underline{M}}$ *induces on its return values* V*. This is achieved by giving* E *the denotation* $[\![\mathsf{E}]\!] : \mathit{Trees}_{\Sigma_p}(\mathbb{A}) \to \mathbb{A}$ *which sends a tree of real numbers to the expected real number, where the approximation of the denotation is given by:* $[\![\mathsf{E}]\!]_{n+1}(\mathsf{por}(t, r)) = ([\![\mathsf{E}]\!]_n(t) + [\![\mathsf{E}]\!]_n(r))/2$.

Example 2 (Global store) *Given a set of locations* L*, we have a set of* states $S := L \to \mathbb{N}$*. Our set of truth values is given by the powerset* $\mathbb{A} := \mathcal{P}(S)$ *with* $\trianglelefteq := \subseteq$*. We have a single modality* G*, where* $(\underline{\underline{M}} \models \mathsf{G}(\Theta))$ *gives the set of starting states for which* $\underline{\underline{M}}$ *terminates with a value* V *such that the end state is contained in* $(V \models \Theta)$*. We define this formally with the following rules:* $[\![\mathsf{G}]\!]_{n+1}(\mathsf{lookup}_l(t_0, t_1, \ldots)) = \{s \in S \mid s \in [\![\mathsf{G}]\!]_{\max(0, n-s(l))}(t_{s(l)})\}$ *and* $[\![\mathsf{G}]\!]_{n+1}(\mathsf{update}_{l,m}(t)) = \{s \mid s[l := m] \in [\![\mathsf{G}]\!]_n(t)\}$*. To ensure that the modality is* continuous*, as required later on in this paper, the definition of* $[\![\mathsf{G}]\!]_{n+1}$ *at* $\mathsf{lookup}(t_0, t_1, \ldots)$ *has been constructed in such a way that it only depends on a finite amount of subtrees, in particular* $t_0, t_1, \ldots, t_n$*.*

Example 3 (Probabilistic choice and global store) *For this combination of effects, we take as truth space the functions* $\mathbb{A} := [0, 1]^S$ *with point-wise order, where* S *is the set of global states and* $[0, 1]$ *the lattice of probabilities with standard order. Intuitively, this space assigns to each starting state a probability that a property is satisfied. We define a single modality* EG *which, for each state* $s \in S$*, is given by the following rules:* $[\![EG]\!]_{n+1}(\mathsf{por}(t, r))(s) := ([\![EG]\!]_n(t)(s) + [\![EG]\!]_n(r)(s))/2$*,* $[\![\mathsf{EG}]\!]_{n+1}(\mathsf{lookup}_l(t_0, t_1, \ldots))(s) = [\![\mathsf{EG}]\!]_{\max(0, n-s(l))}(t_{s(l)})(s)$*, and* $[\![\mathsf{EG}]\!]_{n+1}(\mathsf{update}_{l,m}(t))(s) = [\![\mathsf{EG}]\!]_n(t)(s[l := m])$*.*

Example 4 (Cost) *We use the infinite real number interval* $\mathbb{A} := [0, \infty]$ *with* $\trianglelefteq := \geq$ *denoting an abstract notion of* cost *(e.g. time). Trees are just branches in this example. We have a single modality* C*, where* $(\underline{\underline{M}} \models \mathsf{C}(\Theta))$ *is the cost it takes for* $\underline{\underline{M}}$ *to evaluate plus the cost given by* $(V \models \Theta)$*, where* V *is the value* $\underline{\underline{M}}$

[5] For the examples, each $[\![q]\!]$ will be an Eilenberg-Moore algebra, though this needs not be the case in general.

[6] One could alternatively consider $[0, \infty]$ as the value set, with the standard order.

$$\frac{n \in \mathbb{N}}{\{n\} \in \mathit{Form}(\mathsf{N})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathsf{C}})}{\langle \underline{\phi} \rangle \in \mathit{Form}(\mathsf{U}\,\underline{\mathsf{C}})} \qquad \frac{\phi \in \mathit{Form}(\mathsf{A}_j)}{(j, \phi) \in \mathit{Form}(\Sigma_{i \in I} \mathsf{A}_i)} \qquad \frac{\phi \in \mathit{Form}(\mathsf{A})}{\pi_1 \phi \in \mathit{Form}(\mathsf{A} \times \mathsf{B})}$$

$$\frac{\phi \in \mathit{Form}(\mathsf{B})}{\pi_2 \phi \in \mathit{Form}(\mathsf{A} \times \mathsf{B})} \qquad \frac{V \in \mathit{Terms}(\mathsf{A}) \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathsf{C}})}{(V \mapsto \underline{\phi}) \in \mathit{Form}(\mathsf{A} \to \underline{\mathsf{C}})} \qquad \frac{j \in I \qquad \underline{\phi} \in \mathit{Form}(\underline{\mathsf{C}}_j)}{(j \mapsto \underline{\phi}) \in \mathit{Form}(\Pi_{i \in I} \underline{\mathsf{C}}_i)}$$

$$\frac{q \in \mathcal{Q} \qquad \phi \in \mathit{Form}(\mathsf{A})}{q(\phi) \in \mathit{Form}(\mathsf{FA})} \qquad \frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\mathsf{E}})}{\bigvee X \in \mathit{Form}(\underline{\mathsf{E}})} \qquad \frac{X \subseteq_{\text{countable}} \mathit{Form}(\underline{\mathsf{E}})}{\bigwedge X \in \mathit{Form}(\underline{\mathsf{E}})}$$

$$\frac{\underline{\phi} \in \mathit{Form}(\underline{\mathsf{E}}) \qquad a \in \mathbb{A}}{\underline{\phi}_{\trianglerighteq a} \in \mathit{Form}(\underline{\mathsf{E}})} \qquad \frac{a \in \mathbb{A}}{\kappa_a \in \mathit{Form}(\underline{\mathsf{E}})} \qquad \frac{\underline{\phi} \in \mathit{Form}(\underline{\mathsf{E}})}{\neg \underline{\phi} \in \mathit{Form}(\underline{\mathsf{E}})}$$

Fig. 4. Formula constructors

evaluates to. The definition of $[\![\mathsf{C}]\!] : \mathit{Trees}_{\Sigma_c}(\mathbb{A}) \to \mathbb{A}$ *is such that* $[\![\mathsf{C}]\!]_{n+1}(\mathsf{cost}_c(t)) = c + [\![\mathsf{C}]\!]_n(t)$*. Note that for any tree* t *either infinite or with leaf* $\bot$*, we have* $[\![\mathsf{C}]\!](t) = \infty$*. This reflects the idea that a diverging computation will exceed any possible finite cost.*

Example 5 (Combinations with nondeterminism) *To add nondeterminism to any of the previous examples, we keep their truth space* $\mathbb{A} \in \{[0,1], \mathcal{P}(S), [0,1]^X, [0,\infty]\}$*, and extend the definition of their modality* $q \in \{\mathsf{E}, \mathsf{G}, \mathsf{EG}, \mathsf{C}\}$ *in two ways, creating an optimistic modality* $q_\Diamond$ *and a pessimistic modality* $q_\Box$ *such that:*

$$[\![q_\Diamond]\!]_{n+1}(\mathsf{nor}(t,r)) = [\![q_\Diamond]\!]_n(t) \vee [\![q_\Diamond]\!]_n(r) \qquad [\![q_\Box]\!]_{n+1}(\mathsf{nor}(t,r)) = [\![q_\Box]\!]_n(t) \wedge [\![q_\Box]\!]_n(r).$$

We add both for neutral nondeterminism. We can see the nondeterministic choice as being controlled by an external agent, which chooses a strategy *for resolving the nondeterministic choices, like in a Markov decision process. E.g.,* $\mathsf{E}_\Diamond$ *and* $\mathsf{E}_\Box$ *respectively maximize and minimize the expected score (see [19] for more descriptions of nondeterminism with probability). Similarly,* $\mathsf{C}_\Diamond$ *and* $\mathsf{C}_\Box$ *respectively minimize and maximize cost.*

For instance, if the denotation $|\underline{M}|$ *of a term* $\underline{M}$ *of type* FN *is given by the first tree in Fig. 3, then* $(\underline{M} \models \mathsf{E}_\Diamond(\{1\})) = 1/2$ *and* $(\underline{M} \models \mathsf{E}_\Box(\{1\})) = 1/4$*. If* $|\underline{N}|$ *is given by the second tree from Fig. 3, then* $(\underline{N} \models \mathsf{G}_\Diamond(\{0\})) = \{s \in S \mid s(l) = 0 \vee s(r) = 0\}$ *and* $(\underline{N} \models \mathsf{G}_\Box(\{0\})) = \{s \in S \mid s(l) = 0 = s(r)\}$*.*

Example 6 (Combinations with error) *There are two ways of defining combinations with error messages, akin to the sum and tensor approach of combining effects from e.g. [12]. Let* Σ*,* $\mathbb{A}$ *and* $\mathcal{Q}$ *be the signature, truth space, and quantitative modalities of the effects to which we want to add error messages from a set* E*. Given a modality* $q \in \mathcal{Q}$ *and some function* $f : E \to \mathbb{A}$*, assigning to each message a value, we define a new modality* q_f *which, besides inheriting the rules from* q*, follows the rule* $[\![q_f]\!]_{n+1}(\mathsf{raise}_e) = f(e)$*. We define two new sets of modalities for this combination,* $\mathcal{Q}^+ := \{q_f \mid q \in \mathcal{Q}, f : E \to \{\boldsymbol{F}, \boldsymbol{T}\}\}$ *and* $\mathcal{Q}^\times := \{q_f \mid q \in \mathcal{Q}, f : E \to \mathbb{A}\}$*, each giving a different interpretation of error.*

For instance, in the presence of global store (Example 2), the modalities from $\mathcal{Q}^+$ *are not able to observe the final global state when an error message has been raised, whereas some modalities from* $\mathcal{Q}^\times$ *can. For instance, for* $e \in E$ *and* $f : E \to \mathbb{A}$ *such that* $f(e) := \{s[l := 1] \mid s \in S\}$*, it holds that* G_f *is in* $\mathcal{Q}^\times$ *but not in* $\mathcal{Q}^+$*. Moreover,* $(\mathsf{update}_l(\overline{1}, \mathsf{raise}_e()) \models (\mathsf{G}_f(\top))) = \boldsymbol{T}$ *whereas* $(\mathsf{update}_l(\overline{0}, \mathsf{raise}_e()) \models \mathsf{G}_f(\top)) = \boldsymbol{F}$*. Those two terms are however not distinguishable by any modality from* $\mathcal{Q}^+$*.*

All the Boolean-valued examples of modalities for effects in [29], can also be accommodated in our quantitative setting by taking $\mathbb{A} := \{\boldsymbol{T}, \boldsymbol{F}\}$. These include modalities for the Input/Output effect.

3.2 Formulation of the logic

We write $\mathit{Form}(\underline{\mathsf{E}})$ for the set of formulas over type $\underline{\mathsf{E}}$, which is defined by induction on the structure of $\underline{\mathsf{E}}$. Fig. 4 gives the inductive rules for generating these formulas. We have modality formulas $q(\phi)$, constant formulas κ_a, and step formulas $\underline{\phi}_{\trianglerighteq a}$. Note that conjunctions and disjunctions (i.e., meets and joins) are taken over countable sets of formulas only.

We define a quantitative satisfaction relation $\models$: $\mathit{Terms}(\underline{\mathsf{E}}) \times \mathit{Form}(\underline{\mathsf{E}}) \to \mathbb{A}$, thus for $\underline{P} \in \mathit{Terms}(\underline{\mathsf{E}})$ and $\phi \in \mathit{Form}(\underline{\mathsf{E}})$, the *satisfaction* statement $(\underline{P} \models \phi)$ denotes an element of $\mathbb{A}$. Satisfaction of the formulas is defined inductively by the following rules:

$$(V \models \{n\}) := \begin{cases} \boldsymbol{T} & \text{if } V = \overline{n} \\ \boldsymbol{F} & \text{otherwise.} \end{cases} \qquad ((i,V) \models (j,\phi)) := \begin{cases} (V \models \phi) & \text{if } i = j. \\ \boldsymbol{F} & \text{otherwise.} \end{cases}$$

$$(V \models \langle \underline{\phi} \rangle) := (\mathsf{force}(V) \models \underline{\phi}) \qquad ((V_1, V_2) \models \pi_1 \phi) := (V_1 \models \phi).$$

$$((V_1, V_2) \models \pi_2 \phi) := (V_2 \models \phi). \qquad (\underline{M} \models (V \mapsto \underline{\phi})) := (\underline{M} \cdot V \models \underline{\phi}).$$

$$(\underline{M} \models (j \mapsto \underline{\phi})) := (\underline{M} \cdot j \models \underline{\phi}). \qquad (\underline{M} \models q(\phi)) := [\![q]\!](|\underline{M}|[\phi])$$

The modality formula $q(\phi)$ is particularly important, as it expresses how the quantitative modalities are used to observe effects. The last couple of satisfaction rules are for formula constructors occurring at each type.

$$(\underline{P} \models \bigvee X) := \sup\{(\underline{P} \models \underline{\phi}) \mid \underline{\phi} \in X\}. \qquad (\underline{P} \models \bigwedge X) := \inf\{(\underline{P} \models \underline{\phi}) \mid \underline{\phi} \in X\}.$$

$$(\underline{P} \models \underline{\phi}_{\trianglerighteq a}) := \begin{cases} \boldsymbol{T} & \text{if } (\underline{P} \models \underline{\phi}) \trianglerighteq a. \\ \boldsymbol{F} & \text{otherwise.} \end{cases} \qquad \begin{array}{c} (\underline{P} \models \kappa_a) := a. \\ (\underline{P} \models \neg(\underline{\phi})) := \neg(\underline{P} \models \underline{\phi}). \end{array}$$

All formulas together form the *general logic* $\mathcal{V}$. We distinguish a specific fragment of $\mathcal{V}$, the *positive logic* $\mathcal{V}^+$ excluding all formulas which use $\neg()$. The logic $\mathcal{V}^+$ can be interpreted without defining an involution for $\mathbb{A}$.

We end this section by looking at some interesting properties we can construct using the logic, illustrating the expressibility of the logic. In case of global store (Example 2), we can construct formulas in the style of Hoare logic. For instance, taking two subsets $P, Q \in \mathbb{A} = \mathcal{P}(S)$ of global states, the statement $\underline{M} \models \mathsf{G}(\kappa_Q)_{\trianglerighteq P}$ will give $\boldsymbol{T}$, precisely if, when starting the execution of $\underline{M}$ with a state from P, the execution will terminate with a state from Q. As another example, in case of global store with probability (Example 3), where $\mathbb{A} := [0,1]^S$, we can construct, given a formula $\underline{\phi}$ and a distribution of states $\mu \in [0,1]^S$, a formula $\Sigma_\mu(\underline{\phi})$ such that $(\underline{M} \models \Sigma_\mu(\underline{\phi}))(s) = \min(1, \sum_{s \in S} \mu(s) \cdot (\underline{M} \models \underline{\phi})(s))$. Then $(\underline{M} \models \Sigma_\mu(\mathsf{EG}(\kappa_{\boldsymbol{T}})))$ expresses the probability of termination of $\underline{M}$, given that the starting state is sampled from μ. In the same vein, we can look at the combination of probability and nondeterminism (Example 1 and 5), where $(\underline{M} \models \bigvee_{a,b \in [0,1]} (\mathsf{E}_\Diamond(\kappa_{\boldsymbol{T}})_{\trianglerighteq a} \wedge \mathsf{E}_\Box(\kappa_{\boldsymbol{T}})_{\trianglerighteq b} \wedge \kappa_{(a+b)/2}))$ expresses the probability that $\underline{M}$ terminates, given that the agent/scheduler in control of nondeterministic choice is sampled from a distribution of which 50% is helpful and 50% is antagonistic.

4 Behavioural equivalence

We can define a logical preorder for any sub-collection of formulas $\mathcal{L}$.

Definition 4.1 For any fragment of the logic $\mathcal{L} \subseteq \mathcal{V}$, the *logical preorder* $\sqsubseteq_{\mathcal{L}}$ is given by:
$\forall \underline{P}, \underline{R} \in \mathit{Terms}(\underline{\mathsf{E}}),\ \underline{P} \sqsubseteq_{\mathcal{L}} \underline{R} \iff (\forall \underline{\phi} \in \mathit{Form}(\underline{\mathsf{E}}), (\underline{P} \models \underline{\phi}) \trianglelefteq (\underline{R} \models \underline{\phi}))$
The *logical equivalence* $\equiv_{\mathcal{L}}$ is given by $\sqsubseteq_{\mathcal{L}} \cap \sqsupseteq_{\mathcal{L}}$.

The *general behavioural preorder* $\sqsubseteq$ is the logical preorder $\sqsubseteq_{\mathcal{V}}$, whereas the *positive behavioural preorder* $\sqsubseteq^+$ is the logical preorder $\sqsubseteq_{\mathcal{V}^+}$. We denote $\equiv$ and $\equiv^+$ for the logical equivalences $\equiv_{\mathcal{V}}$ and $\equiv_{\mathcal{V}^+}$ respectively (the behavioural equivalences). These closed relations can be extended to relations on open terms by using the *open extension* (where two open terms are related if they are related for any substitution of context variables).

A *basic* formula is a non-constant formula (not necessarily atomic) which on the top level does not have conjunction $\bigwedge$, disjunction $\bigvee$, negation $\neg$, a constant formula κ_a or step-construction $(-)_{\trianglerighteq a}$. It is not difficult to see that both $\sqsubseteq$ and $\sqsubseteq^+$ are completely determined by basic formulas. Note that since $\mathcal{V}^+ \subseteq \mathcal{V}$, it holds that $(\sqsubseteq) \subseteq (\sqsubseteq^+)$ and $(\equiv) \subseteq (\equiv^+)$.

Lemma 4.2 *The general behavioural preorder $\sqsubseteq$ is symmetric, so $(\sqsubseteq) = (\equiv)$.*

Proof. Assume $\underline{P} \sqsubseteq \underline{R}$, then for any formula $\underline{\phi}$ we have $\neg(\underline{P} \models \underline{\phi}) = (\underline{P} \models \neg(\underline{\phi})) \trianglelefteq (\underline{R} \models \neg(\underline{\phi})) = \neg(\underline{R} \models \underline{\phi})$. Hence $(\underline{R} \models \underline{\phi}) \trianglelefteq (\underline{P} \models \underline{\phi})$ for any $\underline{\phi}$, so $\underline{R} \sqsubseteq \underline{P}$. □

Lemma 4.3 *For any fragment $\mathcal{L}$ of the logic $\mathcal{V}$ closed under countable conjunctions, it holds that for any term $\underline{P} : \underline{\mathsf{E}}$ there is a formulas $\chi^{\underline{P}}$ s.t.: If $\underline{P} \sqsubseteq_{\mathcal{L}} \underline{R}$ then $(\underline{R} \models \chi^{\underline{P}}) = \boldsymbol{T}$, and if $\underline{P} \not\sqsubseteq_{\mathcal{L}} \underline{R}$ then $(\underline{R} \models \chi^{\underline{P}}) = \boldsymbol{F}$.*

Proof. For any $\underline{R} : \underline{\mathsf{E}}$ such that $\underline{P} \not\sqsubseteq_{\mathcal{L}} \underline{R}$ we can find a formula $\psi^{\underline{R}}$ such that $(\underline{P} \models \psi^{\underline{R}}) \not\trianglelefteq (\underline{R} \models \psi^{\underline{R}})$. We choose such a formula for each $\underline{R}$ as above, and define $X := \{\psi^{\underline{R}}_{\trianglerighteq (\underline{P} \models \psi^{\underline{R}})} \mid \underline{R} : \underline{\mathsf{E}}, \underline{P} \not\sqsubseteq_{\mathcal{L}} \underline{R}\}$, which is countable since there are countably many terms. Then $\chi^{\underline{P}} := \bigwedge X$ has the desired properties. □

Lemma 4.4 *For $\mathcal{L} \in \{\mathcal{V}, \mathcal{V}^+\}$, we have the following characterisations of the logical preorder $\mathcal{R} := \sqsubseteq_{\mathcal{L}}$:*

(i) $V \mathcal{R}_{\mathsf{N}} W \iff V = W$.
(ii) $\underline{M} \mathcal{R}_{\mathsf{U}\underline{\mathsf{C}}} \underline{N} \iff \mathsf{force}(\underline{M})\, \mathcal{R}_{\underline{\mathsf{C}}}\, \mathsf{force}(\underline{N})$.
(iii) $(j, V) \mathcal{R}_{\Sigma_{i \in I} \mathsf{A}_i} (k, W) \iff (j = k) \wedge V \mathcal{R}_{\mathsf{A}_j} W$.
(iv) $(V, V') \mathcal{R}_{\mathsf{A} \times \mathsf{B}} (W, W') \Leftrightarrow V \mathcal{R}_{\mathsf{A}} W \wedge V' \mathcal{R}_{\mathsf{B}} W'$.
(v) $\underline{M} \mathcal{R}_{\mathsf{A} \to \underline{\mathsf{C}}} \underline{N} \iff \forall V : \mathsf{A},\ \underline{M} \cdot V\, \mathcal{R}_{\underline{\mathsf{C}}}\, \underline{N} \cdot V$.
(vi) $\underline{M} \mathcal{R}_{\Pi_{i \in I} \underline{\mathsf{C}}_i} \underline{N} \iff \forall j \in I,\ \underline{M} \cdot j\, \mathcal{R}_{\underline{M}_j}\, \underline{N} \cdot j$.

Proof. Note that at each type level, the preorder is completely determined by basic formulas. All other formulas depend solely on the satisfaction of basic formulas, by a simple induction. As such, the above characterisations are a simple consequence of unfolding the satisfaction relation of basic formulas. □

4.1 Congruence properties

A relation on terms is *compatible*, if it is preserved over the typing rules from Fig. 1. We introduce the three properties that we will require in order to establish that (the open extensions of) the behavioural preorders are compatible, hence precongruences. The space $T_\Sigma(\mathbb{A})$, which forms the basis of the technical definition of the modalities, plays a fundamental role in this.

The first property considers the leaf order $T_\Sigma(\trianglelefteq)$ on $T_\Sigma(\mathbb{A})$, where $t\ T_\Sigma(\trianglelefteq)\ r$ if r can be created by replacing leaves $a \in \mathbb{A}$ of t by leaves $b \in \mathbb{A}$ of higher value $a \trianglelefteq b$. The $\bot$ leaves can however not be replaced.

Definition 4.5 A modality $q \in \mathcal{Q}$ is *leaf-monotone* if $\forall t, r \in T_\Sigma(\mathbb{A}),\ t\ T_\Sigma(\trianglelefteq)\ r \implies [\![q]\!](t) \trianglelefteq [\![q]\!](r)$.

This property is useful for establishing a variety of different results, but mainly just shows that modalities preserve the implicit (point-wise) order $\underset{\sim}{\phi} \Rightarrow \underset{\sim}{\psi}$ on formulas ($\underset{\sim}{\phi} \Rightarrow \underset{\sim}{\psi}$ iff $\forall \underset{\sim}{P}, (\underset{\sim}{P} \models \underset{\sim}{\phi}) \trianglelefteq (\underset{\sim}{R} \models \underset{\sim}{\psi})$).

The second property considers the ω-complete tree order $\leq$ on $T_\Sigma(\mathbb{A})$, defined just after Definition 2.1.

Definition 4.6 A modality $q \in \mathcal{Q}$ is *Scott tree continuous* if for any ascending chain $t_0 \leq t_1 \leq t_2 \leq \dots$ it holds that $[\![q]\!](\bigsqcup_{n\in\mathbb{N}} t_n) = \sup\{[\![q]\!](t_n) \mid n \in \mathbb{N}\}$.

This is property is necessary in the congruence proof for inductively approximating the satisfaction value of infinite trees generated by the fixpoint operator and infinite arity effect operators. Note that this continuity assumption rules out countable nondeterminism as a possible example in this framework.

The third and final property is the most technical one, and considers the preservation of the behavioural preorder over sequence operations such as $(-)\ \mathsf{to}\ x\,.\,(-)$. It considers the monad multiplication map $\mu : T_\Sigma(T_\Sigma(\mathbb{A})) \to T_\Sigma(\mathbb{A})$, and requires that the abstract generalisation of the behavioural preorder on $T_\Sigma(T_\Sigma(\mathbb{A}))$ and $T_\Sigma(\mathbb{A})$ is preserved by the μ-map. To formulate this, we need first define these abstract relations.

We write $h : \mathbb{A} \to_\trianglelefteq \mathbb{A}$ to say that h is a *monotone* function from $\mathbb{A}$ to $\mathbb{A}$, so $a \trianglelefteq b \Rightarrow h(a) \trianglelefteq h(b)$. For a function $h : X \to Y$ we write $h^* : T_\Sigma(X) \to T_\Sigma(Y)$ for its (functorial) *lifting* defined by $h^*(t) := t[x \to h(x)]$. For a function $h : X \to \mathbb{A}$ (a *valuation* on X) and a modality $q \in \mathcal{Q}$, we write $t \in q(h)$ for $[\![q]\!](h^*(t))$.

Definition 4.7 For any two trees $t, t' \in T_\Sigma(\mathbb{A})$, $t \preceq t' \iff \forall q \in \mathcal{Q}, \forall h : \mathbb{A} \to_\trianglelefteq \mathbb{A},\ (t \in q(h)) \trianglelefteq (t' \in q(h))$.

For any relation $\mathcal{R} \subseteq X \times Y$, and valuation $h : X \to \mathbb{A}$, we define $(\mathcal{R}\uparrow(h)) : Y \to \mathbb{A}$ to be the function such that $\mathcal{R}\uparrow(h)(b) := \sup_{a\in X, a\mathcal{R}b}(h(a))$. We classify abstract quantitative behavioural properties on $T_\Sigma(\mathbb{A})$. A function $H : T_\Sigma(\mathbb{A}) \to \mathbb{A}$ is called *quantitative behaviourally saturated* if for any two trees $t, t' \in T_\Sigma(\mathbb{A})$ such that $t \preceq t'$, it holds that $H(t) \trianglelefteq H(t')$. We write QBS for the set of quantitative behaviourally saturated functions. Note that $H \in \mathsf{QBS}$ if and only if there is a function $F : T_\Sigma(\mathbb{A}) \to \mathbb{A}$ such that $H = \preceq\uparrow(F)$. Moreover, for any $q \in \mathcal{Q}$, it is easy to see that $[\![q]\!] \in \mathsf{QBS}$. We define a relation on *quantitative double trees* $T_\Sigma(T_\Sigma(\mathbb{A}))$.

Definition 4.8 We define the preorder $\preccurlyeq$ on $T_\Sigma(T_\Sigma(\mathbb{A}))$ by: for any two quantitative double trees $r, r' \in T_\Sigma(T_\Sigma(\mathbb{A}))$, $r \preccurlyeq r' \iff \forall q \in \mathcal{Q}, \forall H \in \mathsf{QBS},\ r \in q(H) \trianglelefteq r' \in q(H)$.

Lemma 4.9 *If all modalities $q \in \mathcal{Q}$ are leaf-monotone, then for any two $r, r' \in T_\Sigma(T_\Sigma(\mathbb{A}))$,*
$r \preccurlyeq r' \iff \forall F : T_\Sigma(\mathbb{A}) \to \mathbb{A}, \forall q \in \mathcal{Q},\ r \in q(F) \trianglelefteq r' \in q(\preceq\uparrow(F))$.

Proof. For '$\Rightarrow$', note that for any $t \in T_\Sigma(\mathbb{A})$, $F(t) \trianglelefteq \preceq\uparrow(F)(t)$ so the result follows from leaf-monotonicity and the fact that $\preceq\uparrow(F) \in \mathsf{QBS}$. For '$\Leftarrow$', use that for $H \in \mathsf{QBS}$, $\preceq\uparrow(H) = H$. □

We can define the third property, decomposability, together with its stronger counterpart, sequentiality. [7]

Definition 4.10 $\mathcal{Q}$ is *decomposable* if for all $t, r \in T_\Sigma(T_\Sigma(\mathbb{A}))$, if $t \preccurlyeq r$ then $\mu t \preceq \mu r$. A modality $q \in \mathcal{Q}$ is *sequential* if for all $t \in T_\Sigma(T_\Sigma(\mathbb{A})), [\![q]\!](\mu t) = [\![q]\!]([\![q]\!]^*(t))$.

Lemma 4.11 *If all modalities $q \in \mathcal{Q}$ are leaf-monotone and sequential, then $\mathcal{Q}$ is decomposable.*

Proof. Assume $r \preccurlyeq r'$, and let $q \in \mathcal{Q}$ and $h : \mathbb{A} \to_\trianglelefteq \mathbb{A}$. It is easy to see that $[\![q]\!] \in \mathsf{QBS}$, so $([\![q]\!] \circ h^*) \in \mathsf{QBS}$ too. Since q is sequential, $[\![q]\!](h^*(\mu r)) = [\![q]\!](\mu(h^{**}(r))) = [\![q]\!]([\![q]\!]^* \circ h^{**}(r)) = (r \in q([\![q]\!] \circ h^*)) \trianglelefteq (r' \in q([\![q]\!] \circ h^*))$ which is by the same steps equal to $[\![q]\!](h^*(\mu r'))$. Hence $\mu r \preceq \mu r'$, and as such, $\mathcal{Q}$ is decomposable. □

The three properties defined above allow us to establish compatibility, as argued in the next section.

Theorem 4.12 *If $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott tree continuous modalities, then $\sqsubseteq$ and $\sqsubseteq^+$ are compatible, hence precongruences.*

[7] Sequentiality is one of two properties for $[\![q]\!]$ to be an Eilenberg-Moore algebra for the monad $T_\Sigma(-)$.

All our examples satisfy these three properties. Both leaf-monotonicity and Scott tree continuity are consequences of the inductive and hence continuous definitions of the modalities, while decomposability holds by observing that any modality from the examples is sequential. We illustrate this in the following lemma.

Lemma 4.13 *In Example 5 of combined probability and nondeterminism,* $\mathsf{E}_\Box$ *is sequential.*

Proof. Take $r, r' \in T_\Sigma(T_\Sigma(\mathbb{A}))$ as above and assume $[\![\mathsf{E}_\Box]\!](\mu r) > a \in [0,1]$, then since $[\![\mathsf{E}_\Box]\!](\mu r) = \sup_n([\![\mathsf{E}]\!]_n(\mu r))$ there must be an $n \in \mathbb{N}$ such that $[\![\mathsf{E}]\!]_n(\mu r) > a$. By the recursive definition of $[\![\mathsf{E}_\Box]\!]_{(-)}$ we can see that $[\![\mathsf{E}]\!]_n(r[t \mapsto [\![\mathsf{E}]\!]_n(t)]) \geq [\![\mathsf{E}]\!]_n(\mu r)$, and hence by leaf- and Scott tree continuity $[\![\mathsf{E}_\Box]\!]([\![\mathsf{E}_\Box]\!]^*(r)) > a$.

Now assume $[\![\mathsf{E}_\Box]\!]([\![\mathsf{E}_\Box]\!]^*(r)) > a$, then there must be an m such that $[\![\mathsf{E}]\!]_m([\![\mathsf{E}_\Box]\!]^*(r)) > a$. Now, $[\![\mathsf{E}]\!]_m$ only looks at a finite amount of leaves, and hence there must be a k such that $[\![\mathsf{E}]\!]_m([\![\mathsf{E}]\!]^*_k(r)) > a$. Again, studying the recursive definition of $[\![\mathsf{E}_\Box]\!]_{(-)}$ we observe that $[\![\mathsf{E}_\Box]\!]_{m+k}(\mu r) \geq [\![\mathsf{E}]\!]_m([\![\mathsf{E}]\!]^*_k(r'))$, so we conclude that $[\![\mathsf{E}_\Box]\!](\mu r) > a$. This is for all such $a \in \mathbb{A}$, so $[\![\mathsf{E}_\Box]\!](\mu r) = [\![\mathsf{E}_\Box]\!]([\![\mathsf{E}_\Box]\!]^*(r))$. □

We end this section with an example of an equivalence and an in-equivalence. It has to be said that the purpose of this paper is to give a widely applicable approach to *defining* equivalence, not to prove equivalence of terms. Moreover, for practical purposes, establishing an in-equivalence is easier than establishing an equivalence, since you only have to find one formula which distinguishes the two.

As an example, we look at the combination of cost and nondeterminism, given by signature Σ_{cn}, truth space $[0,\infty]$ with reverse order ($\trianglelefteq := \geq$), and modalities $\{\mathsf{C}_\Diamond, \mathsf{C}_\Box\}$. Consider the two terms $\underline{M} := \mathsf{cost}_1(\mathsf{return}(\overline{7}))$ and $\underline{N} := \mathsf{nor}(\mathsf{return}(\overline{7}), \mathsf{cost}_3(\mathsf{return}(\overline{7})))$, both of which always return 7, though $\underline{N}$ may return it at a lower or higher cost. Then $\underline{M} \not\sqsubseteq^+ \underline{N}$, since $(\underline{M} \models \mathsf{C}_\Box(\{7\})) = 1 \not\trianglelefteq 3 = (\underline{N} \models \mathsf{C}_\Box(\{7\}))$, and $\underline{N} \not\sqsubseteq^+ \underline{M}$, since $(\underline{N} \models \mathsf{C}_\Diamond(\{7\})) = 0 \not\trianglelefteq 1 = (\underline{M} \models \mathsf{C}_\Diamond(\{7\}))$. However, taking $\underline{K} := \mathsf{nor}(\underline{M}, \underline{N})$, it can be shown that $\underline{K} \equiv \underline{N}$, since for any formula $\phi \in \mathit{Form}(\mathsf{N})$, $(\underline{N} \models \mathsf{C}_\Diamond(\phi)) = (\overline{7} \models \phi) = (\underline{K} \models \mathsf{C}_\Diamond(\phi))$ and $(\underline{N} \models \mathsf{C}_\Box(\phi)) = (\overline{7} \models \phi) + 3 = (\underline{K} \models \mathsf{C}_\Box(\phi))$.

5 Applicative Bisimilarity

We investigate how our quantitative modalities can be used to define a notion of Abramsky's applicative bisimilarity [1], related to the behavioural equivalence (Theorem 5.7), starting off by defining a relator [18,30].

Definition 5.1 Given $\mathcal{Q}$, we have a *relator* $\mathcal{Q}(-) : \mathcal{P}(X \times Y) \to \mathcal{P}(TX \times TY)$ given by:

$$t\, \mathcal{Q}(\mathcal{R})\, r \iff \forall q \in \mathcal{Q}, \forall h : X \to \mathbb{A}, t \in q(h) \trianglelefteq r \in q(\mathcal{R} \uparrow (h))$$

We write $x\mathcal{R}y$ for $(x,y) \in \mathcal{R}$. Remember from the previous section that $(t \in q(h)) = [\![q]\!](h^*(t)) = [\![q]\!](t[x \mapsto h(x)])$ and $(\mathcal{R} \uparrow (h))(b) := \sup\{h(a) \mid a \in X, a\mathcal{R}b\}$. Note that $\mathcal{Q}(\trianglelefteq) = \preceq$ and $\mathcal{Q}(\preceq) = \preccurlyeq$ (see Lemma 4.9). The following characterisation of the relator is immediate:

Lemma 5.2 *For any* $t \in TX$, $r \in TY$, *and* $\mathcal{R} \subseteq X \times Y$, $t\, \mathcal{Q}(\mathcal{R})\, r \iff \forall h : X \to \mathbb{A}, h^*(t) \preceq (\mathcal{R} \uparrow (h))^*(r)$.

The following lemma shows that this satisfies the usual properties of monotone relators from [18,30]. The proof is technical yet straightforward, and is left out to preserve space.

Lemma 5.3 *If all quantitative modalities from* $\mathcal{Q}$ *are leaf-monotone, then* $\mathcal{Q}(-)$ *has the following properties:*

(i) *If* $\mathcal{R}$ *is reflexive, then so is* $\mathcal{Q}(\mathcal{R})$.

(ii) *If* $\mathcal{R} \subseteq \mathcal{S}$, *then* $\mathcal{Q}(\mathcal{R}) \subseteq \mathcal{Q}(\mathcal{S})$.

(iii) *For* $\mathcal{R} \subseteq X \times Y$ *and* $\mathcal{S} \subseteq Y \times Z$, $\mathcal{Q}(\mathcal{R})\mathcal{Q}(\mathcal{S}) \subseteq \mathcal{Q}(\mathcal{R}\mathcal{S})$. *Here* $\mathcal{R}\mathcal{S}$ *is relational concatenation.*

(iv) $\forall f : X \to Z, g : Y \to W, \mathcal{R} \subseteq Z \times W$ *it holds that* $\mathcal{Q}((f \times g)^{-1}\mathcal{R}) = (f^* \times g^*)^{-1}\mathcal{Q}(\mathcal{R})$
where $(f \times g)^{-1}(\mathcal{R}) = \{(x,y) \in X \times Y \mid f(x)\, \mathcal{R}\, g(y)\}$.

Fundamental to the definition of the relator is the notion of the *right-predicate* $\mathcal{R} \uparrow (h)$. When the relation in question is our behavioural preorder, these right-predicates can be expressed in the logic.

Lemma 5.4 *Take* $\mathcal{L} \in \{\mathcal{V}, \mathcal{V}^+\}$. *For any predicate* $D : \mathit{Terms}(\mathsf{A}) \to \mathbb{A}$, *there is a formula* $\phi^D \in \mathcal{L}$ *such that* $(V \models \phi^D) = (\sqsubseteq_{\mathcal{L}} \uparrow (D))(V)$ *for any term* $V \in \mathit{Terms}(\mathsf{A})$.

Proof. We use Lemma 4.3 to define $\phi^D := \bigvee\{\bigwedge\{\kappa_{D(V)}, \chi^V\} \mid V \in T_\Sigma(\mathsf{A})\}$. □

In the case that $\mathcal{R}$ is a relation on terms of some value type A, we write $\mathcal{Q}(\mathcal{R})$ for the relation on terms of type FA given by $\mathcal{Q}(\{(\mathsf{return}(V), \mathsf{return}(W)) \mid V \mathcal{R}_{\mathsf{A}} W\})$. A relation $\mathcal{R}$ on terms is *well-typed*, if it only relates terms of the same type and context, and $\mathcal{R}$ is *closed* if it only relates closed terms.

Definition 5.5 A well-typed closed relation $\mathcal{R}$ is an *applicative* $\mathcal{Q}$*-simulation* if:

(i) $V \mathcal{R}_{\mathsf{N}} W \implies V = W$.
(ii) $\underline{M} \mathcal{R}_{\mathsf{U}\underline{\mathsf{C}}} \underline{N} \implies \mathsf{force}(\underline{M})\, \mathcal{R}_{\underline{\mathsf{C}}}\, \mathsf{force}(\underline{N})$.
(iii) $(j, V) \mathcal{R}_{\Sigma_{i\in I}\mathsf{A}_i} (k, W) \implies (j = k) \wedge V \mathcal{R}_{\mathsf{A}_j} W$.
(iv) $(V, V') \mathcal{R}_{\mathsf{A}\times\mathsf{B}} (W, W') \implies V \mathcal{R}_{\mathsf{A}} W \wedge V' \mathcal{R}_{\mathsf{B}} W'$.
(v) $\underline{M} \mathcal{R}_{\mathsf{A}\to\underline{\mathsf{C}}} \underline{N} \implies \forall V : \mathsf{A},\ \underline{M} \cdot V\, \mathcal{R}_{\underline{\mathsf{C}}}\, \underline{N} \cdot V$.
(vi) $\underline{M} \mathcal{R}_{\Pi_{i\in I}\underline{\mathsf{C}}_i} \underline{N} \implies \forall j \in I,\ \underline{M} \cdot j\, \mathcal{R}_{\underline{M}_j}\, \underline{N} \cdot j$.
(vii) $\underline{M} \mathcal{R}_{\mathsf{FA}} \underline{N} \implies |\underline{M}|\, \mathcal{Q}(\mathcal{R}_{\mathsf{A}})\, |\underline{N}|$.

The *applicative $\mathcal{Q}$-similarity* is the largest applicative $\mathcal{Q}$-simulation, whereas the *applicative $\mathcal{Q}$-bisimilarity* is the largest *symmetric* applicative $\mathcal{Q}$-simulation.

Theorem 5.6 *If all quantitative modalities from $\mathcal{Q}$ are leaf-monotone, then the positive behavioural preorder $\sqsubseteq^+$ is the applicative $\mathcal{Q}$-similarity.*

Proof. Note that $\sqsubseteq^+$ satisfies the first 6 properties for being a $\mathcal{Q}$-simulation as a consequence of Lemma 4.4. We prove the seventh property:

Assume $\underline{M} \sqsubseteq^+ \underline{N}$, $q \in \mathcal{Q}$ and $D : \mathit{Terms}(\mathsf{A}) \to \mathbb{A}$. We use Lemma 5.4 to find a formula ϕ^D such that $(V \models \phi^D) = (\sqsubseteq^+ \uparrow (D))(V)$. By reflexivity of $\sqsubseteq^+$, we have $D(V) \trianglelefteq (\sqsubseteq^+ \uparrow (D))(V)$, so by leaf-monotonicity and $\underline{M} \sqsubseteq^+ \underline{N}$ it holds that: $[\![q]\!](D^*(|\underline{M}|)) \trianglelefteq (\underline{M} \models q(\phi^D)) \trianglelefteq (\underline{N} \models q(\phi^D)) = [\![q]\!]((\sqsubseteq^+ \uparrow (D))^*(|\underline{N}|))$. We can conclude that $|\underline{M}| \mathcal{Q}(\sqsubseteq^+_{\mathsf{A}}) |\underline{N}|$. So we proved that $\sqsubseteq^+$ is a $\mathcal{Q}$-simulation.

We now need to prove that $\sqsubseteq^+$ contains any other $\mathcal{Q}$-simulation $\mathcal{R}$. To do that, we show that $\mathcal{R}$ preserves any formula $\underline{\phi}$ in the following sense: If $\underline{P}\, \mathcal{R}\, \underline{R}$, then $(\underline{P} \models \underline{\phi}) \trianglelefteq (\underline{R} \models \underline{\phi})$. We do this by induction on formulas, using the fact that any formula is well-founded. Assume $\underline{P}\, \mathcal{R}\, \underline{R}$. Suppose $\mathcal{R}$ preserves any formula from $X \subseteq \mathit{Form}(\underline{P})$. Then $(\underline{P} \models \bigvee X) = \sup\{(\underline{P} \models \underline{\phi}) \mid \underline{\phi} \in X\} \trianglelefteq \sup\{(\underline{R} \models \underline{\phi}) \mid \underline{\phi} \in X\} = (\underline{R} \models \bigvee Y)$ and $(\underline{P} \models \bigwedge X) = \inf\{(\underline{P} \models \underline{\phi}) \mid \underline{\phi} \in X\} \trianglelefteq \inf\{(\underline{R} \models \underline{\phi}) \mid \underline{\phi} \in X\} = (\underline{R} \models \bigwedge Y)$. Assume $\mathcal{R}$ preserves formula $\underline{\phi}$, and $(\underline{P} \models \underline{\phi}_{\trianglerighteq a}) \neq \boldsymbol{F}$, then $(\underline{P} \models \underline{\phi}) \trianglerighteq a$, so $(\underline{R} \models \underline{\phi}) \trianglerighteq a$, hence $(\underline{R} \models \underline{\phi}_{\trianglerighteq a}) = \boldsymbol{T}$. $\mathcal{R}$ obviously preserves constant formulas, since $(\underline{P} \models \kappa_a) = a \trianglelefteq a = (\underline{R} \models \kappa_a)$.

It is not difficult to prove that $\mathcal{R}$ preserves most basic formulas. The only difficult formula to consider is $q(\phi) \in \mathit{Form}(\mathsf{FA})$. Assume $\underline{M}\mathcal{R}\underline{N}$, so by simulation property $|\underline{M}|\, \mathcal{Q}(\mathcal{R})\, |\underline{N}|$. By induction hypothesis and relator property 2 in Lemma 5.3, it holds that $|\underline{M}| \mathcal{Q}(\sqsubseteq^+_{\mathsf{A}}) |\underline{N}|$. Hence $(\underline{M} \models q(\phi)) \trianglelefteq [\![q]\!]((\sqsubseteq^+ \uparrow (V \mapsto (V \models \phi)))^*(|\underline{N}|))$, which since $(\sqsubseteq^+ \uparrow (V \mapsto (V \models \phi)))(W) \trianglelefteq (W \models \phi)$ means with leaf-monotonicity of q that $(\underline{M} \models q(\phi)) = [\![q]\!](|\underline{M}|[\phi]) \trianglelefteq [\![q]\!](|\underline{N}|[\phi]) = (\underline{N} \models q(\phi))$. We conclude that $\underline{M} \sqsubseteq^+ \underline{N}$.

We can conclude that $\sqsubseteq^+$ is the largest $\mathcal{Q}$-simulation, hence it is equal to $\mathcal{Q}$-similarity. □

Note the crucial use of Lemma 5.4 in the proof, which explains the need for the step-formulas in the logic.

Theorem 5.7 *If all quantitative modalities from $\mathcal{Q}$ are leaf-monotone, then the general behavioural preorder $\sqsubseteq$ is the largest symmetric applicative $\mathcal{Q}$-simulation, and hence equal to applicative $\mathcal{Q}$-bisimilarity.*

Proof. Firstly, it holds by Lemma 4.2 that $\sqsubseteq$ is symmetric. Secondly, $\sqsubseteq$ is a $\mathcal{Q}$-simulation by the same proof as above. Lastly, any symmetric $\mathcal{Q}$-simulation $\mathcal{R}$ is included in $\sqsubseteq$, using a similar proof as above, proving with induction on formulas $\underline{\phi}$ that $\underline{P}\, \mathcal{R}\, \underline{R}$ implies $(\underline{R} \models \underline{\phi}) \trianglelefteq (\underline{P} \models \underline{\phi})$ and $(\underline{P} \models \underline{\phi}) \trianglelefteq (\underline{R} \models \underline{\phi})$. □

5.1 Howe's method

In this subsection, we briefly outline how Howe's method [10,11] can be used to establish compatibility for the open extension of applicative $\mathcal{Q}$-similarity and $\mathcal{Q}$-bisimilarity as in [3,29]. Firstly, we need some properties of the relators in addition to Lemma 5.3. The proofs are technical and are left out to preserve space.

Lemma 5.8 *If all $q \in \mathcal{Q}$ are leaf-monotone and Scott tree continuous, then the following four properties hold:*

(i) $\forall \mathcal{R} \subseteq X \times Y, t \in T_\Sigma(X), r \in T_\Sigma(Y),\ t\mathcal{Q}(\mathcal{R})r \wedge t' \leq t \wedge r \leq r' \implies t'\mathcal{Q}(\mathcal{R})r'$.
(ii) *For any two chains of trees $t_0 \leq t_1 \leq \ldots$ and $r_0 \leq r_1 \leq \ldots$,* $\forall n \in \mathbb{N}.(t_n \mathcal{Q}(\mathcal{R}) r_n) \Rightarrow (\sqcup_n t_n)\mathcal{Q}(\mathcal{R})(\sqcup_n r_n)$.
(iii) $\forall \mathcal{R} \subseteq X \times Y, \mathcal{S} \subseteq A \times B$ *and* $(f, g) : X \times Y \to A \times B$, $(x \mathcal{R} y \Rightarrow f(x) \mathcal{S} g(y)) \Rightarrow (t \mathcal{Q}(\mathcal{R}) r \Rightarrow f^*(t) \mathcal{Q}(\mathcal{S}) g^*(r))$.
(iv) $\forall \mathcal{R} \subseteq X \times Y, x \in X, y \in Y,\ x \mathcal{R} y \implies \eta(x) \mathcal{Q}(\mathcal{R}) \eta(y)$

The following lemma necessarily uses the property of decomposability.

Lemma 5.9 *Given a decomposable set $\mathcal{Q}$ of leaf-monotone modalities, then: $a \mathcal{Q}(\mathcal{Q}(\mathcal{R}))b \Rightarrow \mu a \mathcal{Q}(\mathcal{R}) \mu b$.*

Proof. Assume $a\, \mathcal{Q}(\mathcal{Q}(\mathcal{R}))\, b$ and take some $h : X \to \mathbb{A}$. Let $t := h^{**}(a)$ and $r := (\mathcal{R} \uparrow (h))^{**}(b)$, then $\mu t = h^*(\mu a)$ and $\mu r = \mathcal{R} \uparrow (h)^*(\mu b)$. In the next paragraph, we are going to prove that $t \preccurlyeq r$ using Lemma 4.9, so we can use decomposability to derive $\mu t \preceq \mu r$, from which we can conclude that $h^*(\mu a) \preceq (\mathcal{R} \uparrow (h))^*(\mu b)$, which by Lemma 5.2 implies $\mu a \mathcal{Q}(\mathcal{R}) \mu b$.

We prove that $t \preccurlyeq r$. Let $q \in \mathcal{Q}$ and $H \in \mathsf{QBS}$, then $(t \in q(H)) = [\![q]\!](H^*(t)) = [\![q]\!]((H \circ h^*)^*(a)) = (a \in q(H \circ h^*)) \trianglelefteq (b \in q(\mathcal{Q}(\mathcal{R}) \uparrow (H \circ h^*)))$. Now $(\mathcal{Q}(\mathcal{R}) \uparrow (H \circ h^*))(k) = \sup\{H(h^*(l)) \mid l\mathcal{Q}(\mathcal{R})k\} \trianglelefteq \sup\{H(h^*(l)) \mid h^*(l) \preceq (\mathcal{R} \uparrow (h))^*(k)\}$ by Lemma 5.2. Since $H \in \mathsf{QBS}$, the statement is smaller than $H((\mathcal{R} \uparrow (h))^*(k))$. Hence using leaf-monotonicity, $(b \in q(\mathcal{Q}(\mathcal{R}) \uparrow ((H \circ h^*)))) \trianglelefteq (b \in q(H \circ (\mathcal{R} \uparrow (h))^*) = (r \in q(H))$. So we can conclude that $(t \in q(H)) \trianglelefteq (r \in q(H))$, and we are finished. □

Corollary 5.10 *Given that $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott tree continuous modalities:*

(i) $(\forall x, y, x\mathcal{R}y \Rightarrow f(x)\mathcal{Q}(\mathcal{S})g(y)) \wedge t\mathcal{Q}(\mathcal{R})r \Rightarrow \mu(f^*(t))\mathcal{Q}(\mathcal{S})\mu(g^*(r))$

(ii) $(\forall k, u_k\mathcal{Q}(\mathcal{S})v_k) \Rightarrow \mathsf{op}(u_0, u_1, ...)\mathcal{Q}(\mathcal{S})\mathsf{op}(v_0, v_1, ...)$

One of the contributions of this paper is identifying the properties on quantitative modalities for which the above relator properties are satisfied, such that we can apply Howe's method. The application of Howe's method itself is however not novel, and is simply an alteration of the proof used for the call by value case in [3,29] (untyped and simply-typed respectively), using results from [15]. As such, details of the proof have been omitted. In short, Howe's method allows us to establish the following theorem.

Theorem 5.11 *If $\mathcal{Q}$ is a decomposable set of leaf-monotone and Scott tree continuous quantitative modalities, then $\mathcal{Q}$-similarity and $\mathcal{Q}$-bisimilarity are compatible.*

Combining theorems 5.6, 5.7, and 5.11 we can derive Theorem 4.12, that the general and positive behavioural equivalence (and preorder) are compatible, and therefore (pre)congruences.

6 Discussions

We have generalised the logic from [29] to a *quantitative* logic for terms of a call-by-push-value language with general recursion and several (combinations of) algebraic effects. The quantitative logic is expressive, contains only meaningful behavioural properties, and induces a compatible program equivalence on terms.

In this paper, we consider program properties (or observations) as the primary way of describing program behaviour. According to this philosophy, the generalisation to quantitative properties is natural. Alternatively, one could consider relations (or comparisons) as primary, and instead generalise to quantitative relations. The resulting theory is that of metrics, along the lines of [2,4,20]. Relating the logic from this paper, or a variation thereof, to metrics (e.g. like the ones in [7]) is a topic for future research.

The quantitative logic does not however naturally induce a metric on the terms. This is mainly because of the inclusion of step-formulas $\phi_{\trianglerighteq a}$, which takes the quantitative information from ϕ and collapses it to a binary value. These step-formulas are necessary for relating the behavioural equivalence with applicative bisimilarity. Their necessity can be seen as a natural consequence of the non-linearity of the language. E.g., in the case of probability with $\mathbb{A} := [0, \infty]$, the step-formula can be constructed using products of formulas.

The quantitative logic is very expressive, allowing one to deal with some awkward combinations of effects that are not amenable to a boolean treatment. Despite the many examples of combination of effects, there is no general theory for quantitative modalities of combined effects. Such a theory is a potential subject for further research. It would also be interesting to look at other examples of effects which the quantitative logic could describe, like a the algebraic jump effect described in [6], or some form of concurrency.

The logic and examples from [29] can be considered as further examples for this paper, where one considers $\mathbb{A} := \{\boldsymbol{T}, \boldsymbol{F}\}$. The property of Scott openness is the Boolean version of a combination of Scott tree continuity and leaf-monotonicity, and the notion of decomposability is a quantitative generalisation of the notion from [29] with the same name. It should be noted, however, that most modalities from [29] are not sequential.

Along the lines of [29], it is possible to define a *pure variation* of the logic. This is a logic independent of the term syntax, using function formulas of the form $(\psi \mapsto \underline{\phi})$ instead of $(V \mapsto \underline{\phi})$, where $(\underline{M} \models (\psi \to \underline{\phi})) := \bigwedge\{(\underline{M} \cdot V \models \underline{\phi}) \mid (V \models \psi) = \boldsymbol{T}\}$. The logical equivalence of this pure logic will be equal to the behavioural equivalence, if (the open extension of) the behavioural equivalence is compatible.

The denotation $[\![q]\!] : T\mathbb{A} \to \mathbb{A}$ of quantitative modalities are, in the case of the running examples, Eilenberg-Moore algebras. These are algebras $a : TX \to X$ such that $a \circ \eta_X = id_X$ and $a \circ Ta = a \circ \mu_X$, the second statement coincides with the property of sequentiality in this paper. As such, our example modalities potentially fit into the framework of Hasuo [8]. Connections between the two approaches may be explored in the future.

Since the theory has been formulated for call-by-push-value, one can extract logics for specific reduction strategies including; call-by-name, call-by-value and lazy PCF [16,17]. The language can also be extended with universal (but not existential) polymorphic and recursive types. These extensions of the language are worked out in the author's forthcoming thesis. Further extensions could be considered in the future.

Acknowledgements: I would like to thank Alex Simpson, Francesco Gavazzo, Aliaume Lopez, and the anonymous reviewers for all the helpful discussions and comments.

References

[1] Samson Abramsky. The lazy λ-calculus. *Research Topics in Functional Programming*, pages 65–117, 1990.

[2] André Arnold and Maurice Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11(2):181 – 205, 1980.

[3] Ugo Dal Lago, Francesco Gavazzo, and Paul Blain Levy. Effectful applicative bisimilarity: Monads, relators, and Howe's method. *Logic in Computer Science*, pages 1–12, 2017.

[4] Martín Hötzel Escardó. A metric model of PCF, 1999. In: Workshop on Realizability Semantics and Applications.

[5] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the lambda-calculus. *Formal Description of Programming Concepts*, pages 193–217, 1986.

[6] Marcelo Fiore and Sam Staton. Substitution, jumps, and algebraic effects. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 41:1–41:10, New York, NY, USA, 2014. ACM.

[7] Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 452–461, 2018.

[8] Ichiro Hasuo. Generic weakest precondition semantics from monads enriched with order. *Theor. Comput. Sci.*, 604(C):2–29, November 2015.

[9] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM (JACM)*, 32(1):137–161, 1985.

[10] Douglas J. Howe. Equality in lazy computation systems. *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 198–203, 1989.

[11] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[12] Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1):70–99, July 2006.

[13] Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. *Logic in Computer Science*, pages 209–218, 2010.

[14] Dexter Kozen. Probabilistic PDL. *Journal of Computer and System Sciences*, 30:162–178, 1985.

[15] Søren Bøgh Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, BRICS, 1998.

[16] Paul Blain Levy. *Call-By-Push-Value.* PhD thesis, University of London, 2001.

[17] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006.

[18] Paul Blain Levy. Similarity quotients as final coalgebras. *Lecture Note in Computer Science*, 6604:27–41, 2011.

[19] Aliaume Lopez and Alex Simpson. Basic operational preorders for algebraic effects in general, and for combined probability and nondeterminism in particular. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 29:1–29:17, 2018.

[20] Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Quantitative algebraic reasoning. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 700–709, 2016.

[21] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science).* SpringerVerlag, 2004.

[22] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[23] Eugenio Moggi. A general semantics for evaluation logic. *9th LICS*, page 353–362, 1994.

[24] Andrew M. Pitts. Evaluation logic. In *Proceedings of 4th Higher Order Workshop*, pages 162–189, 1991.

[25] Gordon Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977.

[26] Gordon Plotkin. Adequacy for infinitary algebraic effects. In *Proceedings of the 3rd International Conference on Algebra and Coalgebra in Computer Science*, CALCO'09, pages 1–2, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] Gordon Plotkin and John Power. Adequacy for algebraic effects. *Foundations of Software Science and Computation Structures*, pages 1–24, 2001.

[28] Gordon Plotkin and Matija Pretnar. A logic for algebraic effects. *23rd Symposium on Logic in Computer Science*, pages 118–129, 07 2008.

[29] Alex Simpson and Niels Voorneveld. Behavioural equivalence via modalities for algebraic effects. In *Programming Languages and Systems - 27th European Symposium on Programming*, pages 300–326, 2018.

[30] Albert Marchienus Thijs. *Simulation and fixpoint semantics.* PhD thesis, University of Groningen, 1996.