A relational theory of effects and coeffects

Ugo Dal Lago, Francesco Gavazzo

A Relational Theory of Effects and Coeffects

UGO DAL LAGO, University of Bologna, Italy and INRIA, France
FRANCESCO GAVAZZO, University of Bologna, Italy and INRIA, France

Graded modal types systems and coeffects are becoming a standard formalism to deal with context-dependent, usage-sensitive computations, especially when combined with computational effects. From a semantic perspective, effectful and coeffectful languages have been studied mostly by means of denotational semantics and almost nothing has been done from the point of view of relational reasoning. This gap in the literature is quite surprising, since many cornerstone results — such as *non-interference*, *metric preservation*, and *proof irrelevance* — on concrete coeffects are inherently relational. In this paper, we fill this gap by developing a general theory and calculus of program relations for higher-order languages with combined effects and coeffects. The relational calculus builds upon the novel notion of a *corelator* (or *comonadic lax extension*) to handle coeffects relationally. Inside such a calculus, we define three notions of effectful and coeffectful program refinements: *contextual approximation*, *logical preorder*, and *applicative similarity*. These are the first operationally-based notions of program refinement (and, consequently, equivalence) for languages with combined effects and coeffects appearing in the literature. We show that the axiomatics of a corelator (together with the one of a relator) is precisely what is needed to prove all the aforementioned program refinements to be precongruences, this way obtaining compositional relational techniques for reasoning about combined effects and coeffects.

CCS Concepts: • **Theory of computation** → **Program semantics**; • **Software and its engineering** → **General programming languages**.

Additional Key Words and Phrases: Coeffects, Graded Modal Types, (Algebraic) Effects, Relational Reasoning, Program Equivalence and Refinement, Logical Relations, Applicative Bisimulation, Lax Extension, (Co)Relator

ACM Reference Format:
Ugo Dal Lago and Francesco Gavazzo. 2022. A Relational Theory of Effects and Coeffects. *Proc. ACM Program. Lang.* 6, POPL, Article 31 (January 2022), 28 pages. https://doi.org/10.1145/3498692

1 INTRODUCTION

In 1961, McCarthy [1961, 1962, 1963] recognised the definition of a theory of equivalence of programs and computational processes as a crucial step towards the development of a mathematical theory of computation. Nowadays, we generically refer to such theories as *program equivalences* and *refinements*, and still consider them some of the most important concepts in the theory of programming languages. In fact, program equivalence is not only of paramount importance for the semantic understanding of programming languages, but it is also a fundamental tool in fields such as program verification, compiler optimisation, program refactoring, and language-based security. Obviously, the more features of a programming language a notion of program equivalence and refinement can account for, the more useful such a notion is. But what exactly does it mean to *have* a theory of program equivalence and refinement?

Authors' addresses: Ugo Dal Lago, University of Bologna, Italy and INRIA, Inria Sophia Méditerranée, Sophia Antipolis, France, ugo.dallago@unibo.it; Francesco Gavazzo, University of Bologna, Italy and INRIA, Inria Sophia Méditerranée, Sophia Antipolis, France, francesco.gavazzo2@unibo.it.

2475-1421/2022/1-ART31
https://doi.org/10.1145/3498692

Relational Reasoning. If the programming language at hand is endowed with a *denotational semantics*, a notion of program equivalence (and possibly of refinement) is given by mathematical equality (preorder) in the semantic model. However, interesting notions of program equivalence and refinement can be given even *without* a denotational semantics, but relying on the static and dynamic semantics of programs, only. Examples of such equivalences and refinements include contextual-like equivalences and approximations, [Mason and Talcott 1991; Morris 1969], logical relations [Plotkin 1973; Reynolds 1983], and (bi)simulation-based relations [Abramsky 1990; Lassen 1999; Sangiorgi 1994]. Despite their differences, all these notions can be placed in a unique and general framework by developing suitable theories of *program relations* wherein performing relational reasoning about programs. Even if operationally-based, such theories usually take the form of relational calculi supporting abstract, modular, and algebraic reasoning about relational properties of programs.

Theories and calculi of program relations have been extensively studied for *pure* higher-order languages [Gordon 1994; Howe 1996; Pitts 1997], a prime example of the power and elegance of the results achieved being Lassen's relational calculus [Lassen 1998a,b], wherein all the aforementioned notions of equivalence and refinements can be defined and studied in a modular fashion. But what happens if one breaks purity, injecting into the calculus features such as computational effects or resource-constraints?

Computational Effects. Since their very beginning, theories and calculi of program relations have been progressively extended to languages featuring computational effects such as pure nondeterminism [Lassen 1998b; Ong 1993], I/O [Gordon 1992], and, more recently, probabilistic nondeterminism [Bizjak and Birkedal 2015; Culpepper and Cobb 2017; Dal Lago and Gavazzo 2019b; Dal Lago et al. 2014]. Such extensions, however, are rather *ad hoc* and rely on specific properties of the concrete effect considered, this way giving credit to the hypothesis that relational techniques are not robust for effectful language extensions. In recent years, such an hypothesis has been proved false by a number of works [Dal Lago and Gavazzo 2019a; Dal Lago et al. 2017a,b, 2020; Goubault-Larrecq et al. 2008; Johann et al. 2010; Simpson and Voorneveld 2018, 2020] developing relational theories of (higher-order) languages featuring general forms of computational effects modelled by monads [Moggi 1989, 1991] and algebraic theories [Plotkin and Power 2001a, 2003].

The injection of computational effects in a programming language drastically impacts the dynamic semantics of programs, making, for instance, program evaluation probabilistic and, more generally, *monadic*. Since theories and calculi of program relations build upon both the static and dynamic semantics of programs, such theories and calculi must become monadic too. This "monadic turn" ultimately reduces to having suitable ways of extending program relations to *monadic* program relations, i.e. relations relating programs encapsulated in a monad, such as distributions of programs. The right axiomatics for these relational extensions have been identified by Dal Lago et al. [2017a] in the notion of a *relator* [Backhouse et al. 1991; Kawahara 1973] or *lax extension* [Barr 1970] of a monad. Relying on such a notion, relational calculi can be uniformly extended to effectful languages: moreover, the defining axioms of a relator are precisely what one needs to prove congruence and precongruence properties of many program equivalences and refinements, including all the aforementioned ones. All of this results in a number of elegant and powerful relational calculi for a large number of languages with computational effects.

Coeffects. Even if many problems have not been solved yet, we may say that relational reasoning about effectful programs is by now pretty well-understood. There is another programming language idiom related to computational effects that is capturing the attention of researchers in the last years: *coeffects* [Brunel et al. 2014; Petricek et al. 2014]. The expression *coeffects* generically refers to the family of program behaviours related to code usage and context-dependency, so that coeffectful languages have explicit forms of control on how code can be used. Typical forms of code usage are

linearity and duplicability [Benton and Wadler 1996; Girard 1987]; *resource consumption* [Ghica and Smith 2014; Girard et al. 1992]; *program sensitivity* [Reed and Pierce 2010]; and *code confidentiality* and *information-flow* [Abadi et al. 1999; Volpano et al. 1996]. On a concrete level, coeffectful behaviours are formalised in the static semantics of a language by means of *graded modal type systems* [Bernardy et al. 2018; Orchard et al. 2019] where now expressions are assigned types specifying their allowed usage.

Due to their many applications, there is a growing interest in the programming language community for modal and coeffectful languages, especially when combined with computational effects [Gaboardi et al. 2016; Orchard et al. 2019]. From a semantic perspective, however, coeffectful languages have been mostly studied by means of denotational semantics [Breuvart and Pagani 2015; Brunel et al. 2014; Gaboardi et al. 2016] and almost nothing has been done from the point of view of relational reasoning, with the exception of the recent work by Abel and Bernardy [2020] on logical relations which, however, deals with coeffects only and does not support computational effects. This gap in the literature is quite surprising, since many cornerstone results — such as *non-interference* [Abadi et al. 1999], *metric preservation* [Reed and Pierce 2010], and *proof irrelevance* [Pfenning 2001] — on concrete coeffects are inherently relational. The main contribution of the present work is precisely to fill such a gap by developing a general relational theory of higher-order languages with both *effects* and *coeffects*.

1.1 The Contribution

How does a general theory (and calculus) of program relations for an effectful and coeffectful language look like? Since the language is effectful, the theory will be monadic and, without much of a surprise, based on the notion of a relator. But the language is also coeffectful and has a *(graded) modal* static semantics. How (and how much) such a semantics affects program relations and their theory?

Modal Relations. To do relational reasoning about coeffectful programs we move from ordinary, binary relations to ternary relations which we generically call *modal relations*. Modal relations relate expressions in possible worlds reflecting the way an expression can be used. For instance, a possible world may encode the security level of a user, so that classified expressions may be related — even if different — in possible worlds with public permission only, but being unrelated in worlds having secret permission, this way formalising the intuition that classified programs cannot be inspected without the right permission. Similarly, possible worlds may represent resource bounds (this way obtaining resource-sensitive relations), or even degrees of relatedness (this way obtaining Fuzzy-like relations akin to metrics). Modal relations are thus the basic semantic elements upon which we build our relational calculus.

Corelators. Having understood that relational reasoning about coeffectul programs is based on modal relations, we face the main question of this work: how coeffects (modal types) *act on* such relations? Coeffects act on the static semantics of programs by means of (graded) modal types. Generically speaking, a modal type changes how an expression can be used: for instance, it may make an expression duplicable to some extent or change its security level. These actions greatly impact the static semantics of the language — making, for instance, term substitution nontrival — but they do not affect the dynamic semantics of programs, this way creating a scenario dual to the one of effects which, instead, have a strong impact on the dynamic semantics of programs but leave almost unchanged their static semantics. What is the relational counterpart of (graded) modal types then? To answer this question, we introduce one of the main novelty and contribution of this work, namely the notion of a lax extension of a *comonad*, and model modalities as lax extensions of the *identity* comonad. We call such lax extensions *corelators*. Corelators allow us to inspect the

relational behaviour that two expressions would have in a given family of possible worlds without leaving the actual one, so that we are allowed to relate expressions in a given world taking into account whether *the same expressions* are related in other worlds specified by the corelator itself. The restriction of corelators to the identity comonad is what licenses us to inspect the relational behaviour of the very same expressions but in different possible worlds. This reflects the fact that coeffects act on the static semantics of programs only, leaving their dynamic semantics essentially unchanged. A corelator thus allows a possible world to have a non-local, yet controlled view on how a relation behaves at other possible worlds. This is crucial to extend static semantic notions of coeffectful languages (such as the one of (modal) substitution) to a relational setting: without such a non-local view, it would be just not possible to define crucial notions such as compatibility and substitutivity (and, ultimately, compositionality).

A Calculus of Effectful and Coeffectful Program Relations. Relying on the notion of a corelator — together with the notion of a relator (properly extended to the setting of modal relations) — we develop a general calculus of (modal) program relations for effectful and coeffectful languages. Such a calculus extends the relational calculus by Lassen [1998b] in a highly nontrivial way: it relies on corelators to define static semantic notions such as compatibility and substitutivity — which become nontrival in a coeffectful setting — and on relators to extend program relations to computational effects. On top of such a calculus, we define the new notions of effectful and coeffectful *contextual approximation*, *logical preorder*, and *applicative similarity*. All these notions — whose definitions take advantage of both the notions of a relator and of a corelator — are novel[1] and constitute the first examples of operationally-based notions of program refinement for effectful and coeffectful languages in the literature. To witness the strength and robustness of our axiomatics, we prove that all these relations are *precongruences*: this way, we obtain the first *relational compositional framework* in a combined effectful and coeffectful scenario. Our proofs of precongruence of applicative similarity and logical preorder are highly nontrivial and require major improvements to the proof techniques available in the literature. Nonetheless, it is remarkable that both these results ultimately rely on the axiomatics of the notions of a relator and of a corelator only. Summing up, our main contributions are:

- The introduction of the novel notion of a corelator. Corelators provide an axiomatics to extend the action of a comonad from functions to modal relations.
- The development of a powerful theory and calculus of program relations for effectful and coeffectful languages. The calculus relies both on the notion of a corelator (to handle coeffects) and of a relator (to handle effects).
- The definition of three notions of effectful and coeffectful program refinement. These notions are the first operationally-based notions of refinement (and thus of equivalence) for effectful and coeffectful languages appearing in the literature.
- The proof of precongruence theorems for the aforementioned program refinements. Such theorems provide the first example of relational *compositional* reasoning in an effectful and coeffectful scenario.

Due to space constraints, the aforementioned contributions focus on program refinement only: however, all our results extend *mutatis mutandis* to equivalences.

2 AN EFFECTFUL CALCULUS WITH GRADED COEFFECTS

In this section, we define the vehicle calculus of this work, namely a call-by-value λ-calculus with graded modal (necessity) types [Gaboardi et al. 2016; Orchard et al. 2019] and generic effects

[1]The closest proposal given in the literature is the already mentioned logical relation by Abel and Bernardy [2020] for a language *without* computational effects, which is nonetheless subsumed by our logical relations.

[Plotkin and Power 2001a, 2003]. Before defining the calculus, we shortly recall the notion of a monad and of an algebraic operation.

2.1 Preliminaries: Monads and Generic Effects

A monad [MacLane 1971] (tacitly assumed to be on the category of sets and functions) is a triple $(T, \eta, \gg\!=)$ where T associates to each set A a set $T(A)$ of T-computations, η (called *unit* of T) is a set-indexed family of functions $\eta_A : A \to T(A)$, and for any function $f : A \to T(B)$ we have a function $\gg\!= f : T(A) \to T(B)$ (the operator $\gg\!=$ is called *bind*). Moreover, these data have to satisfy some suitable equational laws. As it is customary, we often write $t \gg\!= f$ in place of $\gg\!= f(t)$. In this paper we will use the following monads as running examples [Moggi 1989]: (i) the Maybe monad $(\mathcal{M}, \eta, \gg\!=)$, where $\mathcal{M}(A) \triangleq \{just\ a \mid a \in A\} \cup \{\bot\}$, $\eta(a) \triangleq just\ a$, and $t \gg\!= f \triangleq f(a)$ if $t = just\ a$ and $\bot$ otherwise; (ii) the Writer (or output) monad $(\mathcal{O}, \eta, \gg\!=)$ over an output monoid $(O, \cdot, \varepsilon)$, where $\mathcal{O}(X) \triangleq O \times X$, $\eta(x) \triangleq (\varepsilon, x)$, and $(w, x) \gg\!= f \triangleq (w \cdot u, y)$, with $(u, y) = f(x)$; (iii) the (non-empty) powerset monad $(\mathcal{P}, \eta, \gg\!=)$, where $\eta(x) \triangleq \{x\}$, and $g \gg\!= f \triangleq \bigcup_{a \in g} f(a)$; (iv) the discrete distribution monad $(\mathcal{D}, \eta, \gg\!=)$, where $\eta(x) \triangleq \delta_x$ (Dirac distribution on x), and $\mu \gg\!= f \triangleq \sum_i p_i \cdot f(x_i)$, where $\sum_i p_i \cdot \delta_{x_i}$ is a representation of μ as convex sums of Dirac distributions. The subdistribution monad is obtained by postcomposing $\mathcal{D}$ with $\mathcal{M}$.

To actually produce effects, we we rely on *generic effects* [Plotkin and Power 2003]. Given a monad T, a generic effect is an element $op \in T(A)$, for a set A. Examples of generic effects are $arb \in \mathcal{P}(2)$ which corresponds to the binary nondeterministic choice operation; $coin \in \mathcal{D}(2)$ which is the uniform distribution over 2 and corresponds to fair binary probabilistic choice; $out_o \in \mathcal{O}(1)$, which corresponds to the operation printing $o \in O$. In general, we recall that generic effects and algebraic operations are in one-to-one correspondence.

Finally, we require T to be *continuous* (also known as ω-*cppo enriched*) [Dal Lago et al. 2017a; Goguen et al. 1977], so that for any set X, the set $T(X)$ carries a ω-cppo structure [Abramsky and Jung 1994] $(T(X), \sqsubseteq, \bot)$ making unit and bind monotone and continuous. Examples of continuous monads are the maybe, powerset, and subdistribution monads. The writer monad gives a continuous monad when properly combined with the maybe monad [Dal Lago et al. 2017a]. Other examples of continuous monads not mentioned in this work are given by the exception monad, the cost monad, and the tensor of the global state and maybe monads [Hyland et al. 2006].

2.2 Effects and Coeffects: Syntax

The vehicle calculus of this work is a call-by-value *affine* λ-calculus [Maraist et al. 1999] with *generic effects* [Plotkin and Power 2001a,b] and *graded modal (necessity) types* [Gaboardi et al. 2016; Orchard et al. 2019]. Before defining the calculus, we spend a few words on these three features. Our calculus is parametric with respect to two structures: a signature Σ of generic effect symbols, i.e. pairs of the form $(\mathbf{op}, \sigma_{\mathbf{op}})$, and a resource (or grade) algebra $\mathcal{S}$. We use effect symbols $(\mathbf{op}, \sigma_{\mathbf{op}}) \in \Sigma$ as effect-triggering operations, this way following the algebraic perspective on computational effects. Intuitively, for a generic effect $(\mathbf{op}, \sigma_{\mathbf{op}}) \in \Sigma$, once the expression $\mathbf{op}$ is evaluated, a computational effect is produced and a value of type $\sigma_{\mathbf{op}}$ is returned. Generic effects are *the* way computational effects are produced, and thus constitute the actual effectful component of our calculus: such a component mostly impacts the dynamic semantics of the calculus which, as we will see, is *monadic*. The coeffectful component, instead, acts on the static semantics of the calculus. Coeffects and code-usage constraints are described by means of modal types $\Box_s \sigma$ whose inhabitants are pieces of code with capability s (we think about such capabilities as prescribing how code can be manipulated). Elements s are called grades, resources, or capabilities and belong to a resource algebra $\mathcal{S}$, which is usually a semiring-like structure whose operations reflect the way in which capabilities can be

$$
\begin{array}{lll}
\text{Types } \sigma, \tau ::= t & \text{Values } v, w ::= x & \text{Terms } e, f ::= \mathbf{return}\ v \\
\quad \mid \mathbf{unit} & \quad \mid \langle\rangle & \quad \mid \mathbf{proj}_l\ v \\
\quad \mid \sigma \times \tau & \quad \mid \langle v, w\rangle & \quad \mid \mathbf{proj}_r\ v \\
\quad \mid \sigma \otimes \tau & \quad \mid v \otimes w & \quad \mid \mathbf{let}\ x \otimes y = v\ \mathbf{in}\ e \\
\quad \mid \mathbf{void} & \quad \mid \mathbf{in}_l\ v & \quad \mid \mathbf{abort}\ v \\
\quad \mid \sigma + \tau & \quad \mid \mathbf{in}_r\ v & \quad \mid \mathbf{case}\ v\ \mathbf{of}\ (\mathbf{in}_l\ x.e \mid \mathbf{in}_r\ x.f) \\
\quad \mid \sigma \multimap \tau & \quad \mid \lambda x.e & \quad \mid vw \\
\quad \mid \mu t.\sigma & \quad \mid \mathbf{fold}\ v & \quad \mid \mathbf{unfold}\ v \\
\quad \mid \Box_s \sigma & \quad \mid [v] & \quad \mid \mathbf{let}\ [x] = v\ \mathbf{in}\ e \\
\quad \mid \sigma_{\mathbf{op}} & & \quad \mid \mathbf{let}\ x = e\ \mathbf{in}\ f \\
 & & \quad \mid \mathbf{op}
\end{array}
$$

Fig. 1. Types, values, and terms/computations of $\Lambda_{\Sigma,\mathcal{S}}$.

combined. Examples of capabilities include program sensitivity [Reed and Pierce 2010], resource consumption [Orchard et al. 2019], and security levels [Abadi et al. 1999; Volpano et al. 1996].

Finally, the affine base of the calculus gives us tighter control on code manipulation. Accordingly, data can be discarded but not copied: copyable code must be declared such using suitable modal types. The choice of working with an affine base is just a design choice, and our results can be extended to calculi with other bases (even if not substructural) with a minimal effort. Let us now introduce the syntax and semantics of our calculus, which we call $\Lambda_{\Sigma,\mathcal{S}}$.

Definition 1. *Given a signature Σ and a grade algebra $\mathcal{S}$ (which we are going to define), the raw syntax of $\Lambda_{\Sigma,\mathcal{S}}$ is given in Figure 1, where s ranges over elements in $\mathcal{S}$ and* **op** *is a generic effect in Σ.*

Following the methodology of fine-grain call-by-value [Levy et al. 2003], we divide expressions of $\Lambda_{\Sigma,\mathcal{S}}$ in two disjoint classes: *values* and *terms* (also called *computations*). Intuitively, a value is the result of a computation, whereas a computation is a value producer, i.e. an expression that once evaluated may produce a value (the evaluation process might not terminate) as well as side effects. Accordingly, computations must be explicitly sequenced by means of the sequencing constructor $\mathbf{let}\ x = -\ \mathbf{in}\ -$. We adopt standard syntactic conventions [Harper 2016] (notably the so-called *variable convention* [Barendregt 1984]). The notion of a free (resp. bound) variable is defined as usual. As it is customary, we identify terms up to renaming of bound variables and say that a term is closed if it has no free variables. Oftentimes we refer to closed terms as *programs*. Moreover, for finite lists of syntactic expressions (such as variables, computations, or values), we use $\boldsymbol{\phi}$ for a finite list $\phi_1, \ldots, \phi_n$ of ϕs. As a notational convention, we agree that whenever we have a vector $\boldsymbol{\phi}$, the symbol ϕ_i stands for the i-th element of $\boldsymbol{\phi}$.

We write $e[v/x]$ (resp. $w[v/x]$) for the capture-avoiding substitution of the value v for all free occurrences of x in e (resp. w). The notion of a substitution is extended to the one of *simultaneous substitution* in the natural way. Given vectors $\boldsymbol{v}$, $\boldsymbol{x}$ of values and variables (which we tacitly assume to have the same length), respectively, we write $e[\boldsymbol{v}/\boldsymbol{x}]$ for the simultaneous substitution of all values v_i for variables x_i in e. We define $w[\boldsymbol{v}/\boldsymbol{x}]$ similarly. Finally, we extend our syntactic conventions to type. In particular, we work with types modulo renaming and write $\sigma[\tau/t]$ for the capture-avoiding substitution of τ for all the free occurrences of the variable t in σ.

Example 1. Instantiating Σ to the specific generic effects seen in Section 2.1, we obtain several effectful calculi (we will see examples of coeffects in the next section). (i) Taking $\Sigma \triangleq \{\mathbf{arb} : \mathbf{bool}\}$ we obtain a nondeterministic calculus: the intended meaning of **arb** is to nondeterministically evaluate either to true or false. (ii) Similarly, $\Sigma \triangleq \{\mathbf{coin} : \mathbf{bool}\}$ gives a probabilistic calculus, the intended meaning of **coin** being to return true or false both with probability $\frac{1}{2}$. (iii) Given an output monoid O, the signature $\Sigma \triangleq \{\mathbf{out}_o : \mathbf{unit} \mid o \in O\}$ gives a calculus with output, the intended meaning of $\mathbf{out}_o$ being to output the string o.

2.3 Static Semantics

We now endow $\Lambda_{\Sigma,\mathcal{S}}$ with a static semantics accounting for the coeffectful nature of $\Lambda_{\Sigma,\mathcal{S}}$. Let us begin with a precise definition of a grade algebra.

Definition 2. *A* grade algebra $(\mathcal{S}, \leq, +, *, 0, 1, \infty)$ *is a* preordered semiring with top element*, i.e a semiring* $(\mathcal{S}, +, *, 0, 1)$ *together with a preorder* $\leq$ *for which both* $+$ *and* $*$ *are monotone, and* ∞ *is the top element.*

When unambiguous, we denote a grade algebra $(\mathcal{S}, \leq, +, *, 0, 1, \infty)$ simply as $\mathcal{S}$. As already said, elements of $\mathcal{S}$ (to which we refer to as grades, resources, or capabilities) constrain the way in which code can be manipulated. In particular, 0 represents the null capability (usually associated to unused code); 1 represents linearly used code; $+$ is used to combine grades/capabilities in parallel (as in the case of two programs using the same code independently); and $*$ is used to compose grades sequentially, as in the case of an expression using another expression in place of a variable with a given capability. The following examples will clarify the concept.

Example 2.
1. The one element semiring $\{\infty\}$ is used to model an exponential modality in the spirit of linear logic [Benton and Wadler 1996; Girard 1987]. An expression of type $\Box_\infty \sigma$ represents a piece of code that can be freely duplicated and discharged. This property comes from idempotency of the semiring addition, which gives $\infty + \infty = \infty$. Notice that here the zero, unit, and top elements of the semiring coincide.
2. The semiring of natural numbers extended with infinity $(\mathbb{N}^\infty, =, +, \cdot, 0, 1, \infty)$ is used to model the *exact usage* modality of bounded linear logic [Girard et al. 1992]. Accordingly, a term of type $\Box_n \sigma$ represents a piece of code that has to be used exactly n times. Notice that here the semiring addition is not idempotent.
3. The semiring of extended non-negative real numbers[2] $([0, \infty], \leq, +, \cdot, 0, 1, \infty)$ is used to model type systems for program sensitivity [Reed and Pierce 2010]. Accordingly, we think about expressions of type $\Box_s \sigma \multimap \tau$ as representing functions that are s-Lipschitz continuous.
4. Distributive lattices $(\mathcal{L}, \geq, \wedge, \vee, \top, \bot)$ are used to model information flow modalities, such as modalities describing security levels [Abadi et al. 1999; Denning 1976; Volpano et al. 1996]. As a running example, we consider the two-element lattice $\{\mathsf{low} \leq \mathsf{high}\}$. A term of type $\Box_{\mathsf{high}} \sigma$ represents a piece of code accessible only by users with high confidentiality level. Dually, a term of type $\Box_{\mathsf{low}} \sigma$ can be used by users with at least low confidentiality level, and thus by any user.

Remark 1. Example 1 shows that $\Lambda_{\Sigma,\mathcal{S}}$ subsumes several *effectful* calculi, whereas Example 2 shows that $\Lambda_{\Sigma,\mathcal{S}}$ subsumes several *coeffectful* calculi. By taking combinations of signatures in Example 1 and grade algebras in Example 2, one immediately sees that $\Lambda_{\Sigma,\mathcal{S}}$ provides a general model for calculi exhibiting *both* effects and coeffects at the same time. For instance, by taking the probabilistic

[2] Recall that $\infty + r = r + \infty = \infty$ and that $\infty \cdot r = r \cdot \infty = \infty$, if $r \neq 0$, and $\infty \cdot 0 = 0 \cdot \infty = 0$. The latter equality captures our intuitive understanding that unused programs should be always regarded as equivalent.

signature $\{\mathbf{coin} : \mathbf{bool}\}$ and the grade algebra $([0,\infty], \leq, +, \cdot, 0, 1, \infty)$, we recover probabilistic Fuzz [de Amorim et al. 2019; Gavazzo 2018]. Similarly, by taking the signature $\{\mathbf{arb} : \mathbf{bool}\}$ and a distributive lattice of security levels $(\mathcal{L}, \geq, \wedge, \vee, \top, \bot)$, we obtain a nondeterministic calculus for information-flow.

Notice that grade algebras are not required to satisfy $0 \neq 1$. However, it is convenient to require 0 to be the bottom element of the algebra (i.e. $0 \leq s$, for any s). Moreover, we also require the join of two elements to exist, so that for all elements $s, r \in \mathcal{S}$, the element $s \vee r$ exists and belongs to $\mathcal{S}$.[3] As we will see, such a condition is necessary to guarantee soundness of our type system [Gavazzo 2018].

Fixed a grade algebra $(\mathcal{S}, \leq, +, *, 0, 1, \infty)$, we give $\Lambda_{\Sigma,\mathcal{S}}$ static semantics by means of judgments of the form $x_1 :_{s_1} \sigma_1, \ldots, x_n :_{s_n} \sigma_n \vdash e : \sigma$, where $s_1, \ldots, s_n \in \mathcal{S}$. The informal meaning of such a judgment is that e is a term using variable x_i as prescribed by s_i, so that to pass an input v to e for the variable x_i, we need v to have not only type σ_i, but also capability s_i. To define such typing judgments formally, we need to first define the notion of a *typing environment*.

Definition 3. *A typing environment is a a partial function Γ from variables to types and grades such that the domain $dom(\Gamma) \triangleq \{x \mid \Gamma(x) \neq \bot\}$ of Γ is finite.*

Given a typing environment Γ, we use the notation $x_1 :_{s_1} \sigma_1, \ldots, x_n :_{s_n} \sigma_n$ to denote it, where $\{x_1, \ldots, x_n\} = dom(\Gamma)$ and $\Gamma(x_i) = just\ (\sigma_i, s_i)$. Moreover, we write $(x :_s \sigma) \in \Gamma$ to mean that $\Gamma(x) = just\ (\sigma, s)$. We denote by $\cdot$ the totally undefined environment. As for non-linear calculi, we can join typing environments together, provided that their domains are disjoint. We write Γ, Θ for the union of the disjoint typing environments Γ and Θ. Whenever we write Γ, Θ, we assume Γ and Θ to be disjoint. The substructural nature of $\Lambda_{\Sigma,\mathcal{S}}$ makes the disjoint union of typing environments too weak to define an interesting static semantics. To overcome this problem, we extend the operations of $\mathcal{S}$ to typing environments.

Definition 4.
1. *We say that two typing environments Γ, Θ are consistent (notation $\Gamma \sim \Theta$) if $(x :_s \sigma) \in \Gamma$ and $x \in dom(\Theta)$ imply $(x :_r \sigma) \in \Theta$, for some $r \in \mathcal{S}$.*
2. *The sum operation $+$ is defined between* consistent *typing environments as follows:*

$$(\Gamma + \Theta)(x) \triangleq \begin{cases} just\ (\sigma, s + r) & \text{if } (x :_s \sigma) \in dom(\Gamma)\ \&\ (x :_r \sigma) \in dom(\Theta) \\ \Gamma(x) & \text{if } x \notin dom(\Theta) \\ \Theta(x) & \text{if } x \notin dom(\Gamma) \end{cases}$$

3. *The multiplication (or scaling) operation scale grades in a typing environment by a given grade:*

$$(s * \Gamma)(x) \triangleq \begin{cases} just\ (\sigma, s * r) & \text{if } \Gamma(x) = just\ (\sigma, r) \\ \bot & \text{otherwise.} \end{cases}$$

We can now endow $\Lambda_{\Sigma,\mathcal{S}}$ with a static semantics. The latter is based on two kinds of judgments (exploiting the fine-grained style of the calculus): judgments of the form $\Gamma \vdash^{\mathrm{V}} v : \sigma$ for values and judgments of the form $\Gamma \vdash^{\Lambda} e : \sigma$ for terms, where in both such judgments σ is a closed type. The system is defined in Figure 2.

Let us now comment on the rules in Figure 2, starting with the role played by semiring operations on environments. Following the intuition that terms of type $\Box_s \sigma$ are pieces of code of type σ that can be manipulated as prescribed by s, we see that whenever we have a value v with free variables in Γ and we want to use v in place of a variable used by another program according to s, then we need variables in Γ to be themselves usable according to s: this is formalised by the environment

[3] Actually, we do not need $s \vee r$ to exist for any r, but just for $r = 1$.

$$\frac{}{\Gamma, x :_s \sigma \vdash^{\mathcal{V}} x : \sigma}\ (s \geq 1) \qquad \frac{\Gamma, x :_1 \sigma \vdash^{\Lambda} e : \tau}{\Gamma \vdash^{\mathcal{V}} \lambda x.e : \sigma \multimap \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \multimap \tau \quad \Theta \vdash^{\mathcal{V}} w : \sigma}{\Gamma + \Theta \vdash^{\Lambda} vw : \tau}$$

$$\frac{}{\Gamma \vdash^{\mathcal{V}} \langle\rangle : \mathbf{unit}} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \quad \Gamma \vdash^{\mathcal{V}} w : \tau}{\Gamma \vdash^{\mathcal{V}} \langle v, w\rangle : \sigma \times \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \times \tau}{\Gamma \vdash^{\Lambda} \mathbf{proj}_l\ v : \sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \times \tau}{\Gamma \vdash^{\Lambda} \mathbf{proj}_r\ v : \tau}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \quad \Theta \vdash^{\mathcal{V}} w : \tau}{\Gamma + \Theta \vdash^{\mathcal{V}} v \otimes w : \sigma \otimes \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma \otimes \tau \quad \Theta, x :_s \sigma, y :_s \tau \vdash^{\Lambda} e : \rho}{s * \Gamma + \Theta \vdash^{\Lambda} \mathbf{let}\ x \otimes y = v\ \mathbf{in}\ e : \rho} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \mathbf{void}}{\Gamma \vdash^{\Lambda} \mathbf{abort}\ v : \sigma}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} v : \sigma}{\Gamma \vdash^{\mathcal{V}} \mathbf{in}_l\ v : \sigma + \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \tau}{\Gamma \vdash^{\mathcal{V}} \mathbf{in}_r\ v : \sigma + \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma + \tau \quad \Theta, x :_s \sigma \vdash^{\Lambda} e : \rho \quad \Theta, y :_s \tau \vdash^{\Lambda} f : \rho}{s * \Gamma + \Theta \vdash^{\Lambda} \mathbf{case}\ v\ \mathbf{of}\ (\mathbf{in}_l\ x.e \mid \mathbf{in}_r\ x.f) : \rho}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} v : \sigma[\mu t.\sigma/t]}{\Gamma \vdash^{\mathcal{V}} \mathbf{fold}\ v : \mu t.\sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \mu t.\tau}{\Gamma \vdash^{\Lambda} \mathbf{unfold}\ v : \sigma[\mu t.\sigma/t]} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \Box_s \sigma \quad \Theta, x :_{r*s} \sigma \vdash^{\Lambda} e : \tau}{r * \Gamma + \Theta \vdash^{\Lambda} \mathbf{let}\ [x] = v\ \mathbf{in}\ e : \tau}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} v : \sigma}{s * \Gamma \vdash^{\mathcal{V}} [v] : \Box_s \sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} v : \sigma}{\Gamma \vdash^{\Lambda} \mathbf{return}\ v : \sigma} \qquad \frac{\Gamma \vdash^{\Lambda} e : \tau \quad \Theta, x :_s \tau \vdash^{\Lambda} f : \sigma}{(s \vee 1) * \Gamma + \Theta \vdash^{\Lambda} \mathbf{let}\ x = e\ \mathbf{in}\ f : \sigma} \qquad \frac{}{\Gamma \vdash^{\Lambda} \mathbf{op} : \sigma_{\mathbf{op}}}$$

Fig. 2. Static semantics of $\Lambda_{\Sigma,\mathcal{S}}$.

$s * \Gamma$, where we use semiring multiplication to give conditions on code to be used inside other code. For instance, if a variable x is used by v twice and we want to replace v for a variable y used 3 times in a term e, then x will be used 6 times in $e[v/y]$.

We can now look at some technical features of the rules in Figure 2. Most of such rules are standard in the context of graded calculi, although there are minor differences with other presentations. For instance, in the first rule in Figure 2 it is often required that $s = 1$ and that the environment $0 * \Gamma$ is used in place of Γ, this way staying closer to linear calculi. Our choice of allowing s to be greater than or equal to 1 is in line with examples coming from differential privacy and information flow [Abadi et al. 1999; de Amorim et al. 2017; Reed and Pierce 2010]. Nonetheless, all our results hold for calculi where one requires $s = 1$. Another important difference is given by the typing rule for sequencing, which comes from type systems for abstract program metrics [Gavazzo 2018] and that has been used in the context of modal types [Abel and Bernardy 2020] more recently. The rationale behind such a rule can be easily understood in terms of resource usage. Suppose to have a term f using a variable x zero times, and a term e using a variable y two times. How many times does $\mathbf{let}\ x = e\ \mathbf{in}\ f$ use y? One may be tempted to say that y is not used in $\mathbf{let}\ x = e\ \mathbf{in}\ f$ at all, as e is simply thrown away in f. However, in a call-by-value scenario the term $\mathbf{let}\ x = e\ \mathbf{in}\ f$ first evaluates e, and then it throws it away. As a consequence, the variable y is still used twice in $\mathbf{let}\ x = e\ \mathbf{in}\ f$. If we were to replace $s \vee 1$ with s in the conclusion of the sequencing rule, then we would obtain that y is not used in $\mathbf{let}\ x = e\ \mathbf{in}\ f$, which is just unsound. Our choice of using $(s \vee 1) * \Gamma$ in place $s * \Gamma$ models the fact that e is evaluated in $\mathbf{let}\ x = e\ \mathbf{in}\ f$, and thus it is used at least once.

Our static semantics enjoys all the standard properties of coeffectful calculi. Among those, the so-called substitution lemma is arguably the most interesting one.

Lemma 1 (Substitution (static semantics)). *The following rules are admissible.*

$$\frac{\Gamma, \boldsymbol{x} :_{\boldsymbol{s}} \boldsymbol{\sigma} \vdash^{\Lambda} e : \sigma \quad \boldsymbol{\Theta} \vdash^{\mathcal{V}} \boldsymbol{v} : \boldsymbol{\sigma} \quad \Gamma \sim \Theta_i}{\Gamma + \sum \boldsymbol{s} \cdot \boldsymbol{\Theta} \vdash^{\Lambda} e[\boldsymbol{v}/\boldsymbol{x}] : \sigma} \qquad \frac{\Gamma, \boldsymbol{x} :_{\boldsymbol{s}} \boldsymbol{\sigma} \vdash^{\mathcal{V}} v : \sigma \quad \boldsymbol{\Theta} \vdash^{\mathcal{V}} \boldsymbol{v} : \boldsymbol{\sigma} \quad \Gamma \sim \Theta_i}{\Gamma + \sum \boldsymbol{s} \cdot \boldsymbol{\Theta} \vdash^{\mathcal{V}} v[\boldsymbol{v}/\boldsymbol{x}] : \sigma}$$

Before moving to the dynamic semantics of $\Lambda_{\Sigma,\mathcal{S}}$, we introduce some useful notation.

$$\Lambda_\sigma \triangleq \{e \mid \exists \Gamma.\, \Gamma \vdash^{\Lambda} e : \sigma\} \qquad \Lambda_{\Gamma\vdash\sigma} \triangleq \{e \mid \Gamma \vdash^{\Lambda} e : \sigma\} \qquad \Lambda^{\bullet}_\sigma \triangleq \{e \mid \cdot \vdash^{\Lambda} e : \sigma\}$$
$$\mathcal{V}_\sigma \triangleq \{v \mid \exists \Gamma.\, \Gamma \vdash^{\mathcal{V}} v : \sigma\} \qquad \mathcal{V}_{\Gamma\vdash\sigma} \triangleq \{v \mid \Gamma \vdash^{\mathcal{V}} v : \sigma\} \qquad \mathcal{V}^{\bullet}_\sigma \triangleq \{v \mid \cdot \vdash^{\mathcal{V}} v : \sigma\}$$

2.4 Dynamic Semantics

Finally, we endow $\Lambda_{\Sigma,\mathcal{S}}$ with a dynamic semantics. Such a semantics is given by a *monadic evaluation* function [Dal Lago et al. 2017a]. Let us fix a continuous monad $(T, \eta, \gg\!=)$ and a generic effect $[\![\mathbf{op}]\!] \in T(\mathcal{V}_{\sigma_{\mathbf{op}}})$ for every operation symbol $(\mathbf{op} : \sigma_{\mathbf{op}}) \in \Sigma$.

Definition 5. *The $\mathbb{N}$-indexed family of functions $[\![-]\!]^n_\varepsilon : \prod_\sigma \Lambda^{\bullet}_\sigma \to T(\mathcal{V}^{\bullet}_\sigma)$ is inductively defined as follows:*[4] $[\![e]\!]^0_\varepsilon \triangleq \bot$ *and*

$$[\![\mathbf{return}\ v]\!]^{n+1}_\varepsilon \triangleq \eta(v) \qquad [\![\langle\rangle]\!]^{n+1}_\varepsilon \triangleq \eta(\langle\rangle)$$
$$[\![\mathbf{let}\ x \otimes y = v \otimes w\ \mathbf{in}\ e]\!]^{n+1}_\varepsilon \triangleq [\![e[v/x, w/y]]\!]^n_\varepsilon \qquad [\![\mathbf{unfold}\ (\mathbf{fold}\ v)]\!]^{n+1}_\varepsilon \triangleq \eta(v)$$
$$[\![\mathbf{case}\ (\mathbf{in}_i\ v)\ \mathbf{of}\ (\mathbf{in}_l\ x.e_1 \mid \mathbf{in}_r\ x.e_i)]\!]^{n+1}_\varepsilon \triangleq [\![e_i[v/x]]\!]^n_\varepsilon \qquad [\![\langle v_1, v_2\rangle.i]\!]^{n+1}_\varepsilon \triangleq \eta(v_i)$$
$$[\![(\lambda x.e)v]\!]^{n+1}_\varepsilon \triangleq [\![e[v/x]]\!]^n_\varepsilon \qquad [\![\mathbf{let}\ [x] = [v]\ \mathbf{in}\ e]\!]^{n+1}_\varepsilon \triangleq [\![e[v/x]]\!]^n_\varepsilon$$
$$[\![\mathbf{let}\ x = e\ \mathbf{in}\ f]\!]^{n+1}_\varepsilon \triangleq [\![e]\!]^n_\varepsilon \gg\!= [\![f[-/x]]\!]^n_\varepsilon \qquad [\![\mathbf{op}]\!]^{n+1}_\varepsilon \triangleq [\![\mathbf{op}]\!].$$

Lemma 2. *For any type σ, the sequence of maps $([\![-]\!]^n_\varepsilon)_{n\geq 0}$ forms an ω-chain in $\Lambda^{\bullet}_\sigma \to T(\mathcal{V}^{\bullet}_\sigma)$.*

Since T is continuous, by Lemma 2 we define the evaluation map $[\![-]\!]_\varepsilon : \prod_\sigma \Lambda^{\bullet}_\sigma \to T(\mathcal{V}^{\bullet}_\sigma)$ as $\bigsqcup_{n\geq 0} [\![-]\!]^n_\varepsilon$. Notice that modalities play essentially no role in the definition of $[\![-]\!]_\varepsilon$.

3 MODAL RELATIONS, RELATORS, AND CORELATORS

Having defined the syntax, static, and dynamic semantics of $\Lambda_{\Sigma,\mathcal{S}}$, in this section we introduce the semantic foundation we will use to develop our calculus of (effectful and coeffectful) program relations, namely *modal relations*, *relators*, and *corelators*. To motivate such a foundation, let us observe that modal types *do not* influence the operational behaviour of programs, as the presence of modalities *de facto* does not affect program execution. According to the reading given in the Introduction, what modalities do is to act on the *external observer* by modifying the way she can use and test (and thus ultimately *discriminate* between) programs. We formalise this intuition by working with *ternary relations*, which we call *modal relations*. Modal relations relate pairs of expressions in possible worlds reflecting the way such expressions can be used. Due to the substructural nature of $\Lambda_{\Sigma,\mathcal{S}}$, we let possible worlds live in a *symmetric monoidal preorder* (or preordered monoid) in the spirit of ternary semantics of substructural logics [Lambek 1968; Routley and Meyer 1973; Urquhart 1972]. Remarkably, the monoidal preorder structure of possible worlds allows us to extend most of the usual algebra of binary relations to a ternary, modal setting.

Having understood that relational reasoning about coeffectful programs is conveniently supported by the shift from binary to ternary relations, two main questions to be answered remain: *how do we model effects relationally? What about coeffects?* We answer the first question relying on the work by Dal Lago et al. [2017a]; Gavazzo [2019] and extending the notion of a *relator* [Backhouse et al. 1991; Kawahara 1973] (also known as *lax extension* [Barr 1970]) to ternary relations. To answer the second question, we introduce the novel construction of a *corelator* (or *lax extension of a comonad*) which we will use to formalise the intuition that modal types do not act on programs, but on possible worlds.

[4] We omit type subscripts.

3.1 The Algebra of Modal Relations

We begin introducing *modal relations* and their algebra.

Definition 6. *A (monoidal Kripke)* frame *is a symmetric monoidal preorder (also known as preordered commutative monoid) $\mathcal{W} = (\mathsf{W}, \leq, \odot, \varepsilon)$ such that ε — the unit of $\odot$ — is the bottom element. A $\mathcal{W}$-*relation *R over $X \times Y$ is a ternary relation $R \subseteq X \times \mathsf{W} \times Y$ monotone in its second argument. That is: $R(x, a, y)$ & $a \leq b \implies R(x, b, y)$.*

Elements $a, b, \ldots$ of a frame are called *possible worlds*. We refer to $\mathcal{W}$-relations for a generic frame $\mathcal{W}$ as modal relations. To improve readability, we oftentimes employ the forcing notation $a \Vdash x\ R\ y$ in place of $R(x, a, y)$.

Example 3. 1. Let $\mathcal{W}$ be $([0, \infty], \leq, +, 0)$. A $\mathcal{W}$-relation R relates objects with respect to non-negative extended real numbers with the intended meaning that if $a \Vdash x\ R\ y$ holds, then the R-distance between x and y is at most a (or, equivalently, that x and y are related up to an error a). Obviously, if $a \leq b$ and $a \Vdash x\ R\ y$, we also have $b \Vdash x\ R\ y$ (if x and y are at most a far, then they also are at most b far). Reed and Pierce [2010] use $\mathcal{W}$-relations to define a logical relation to characterise program distance.
2. Let $\mathcal{W}$ be $(\mathcal{L}, \geq, \wedge, \bot)$, for $\mathcal{L}$ a security lattice. A $\mathcal{W}$-relation relates objects at specific confidentiality levels: we thus read $\ell \Vdash x\, R\, y$ as stating that a user with confidentiality level ℓ regards the objects x and y as R-related. Monotonicity states that if x and y are related at confidentiality level ℓ (i.e. $\ell \Vdash x\ R\ y$), then they are also related at lower confidentiality levels (as a user with a lower confidentiality level has less ways to access x and y, and thus to discriminate between them).

We denote by $\mathcal{W}\text{-Rel}(X, Y)$ the collection of $\mathcal{W}$-relations over sets X and Y, and write $R : X \nrightarrow Y$ in place of $R \in \mathcal{W}\text{-Rel}(X, Y)$, provided that $\mathcal{W}$ is clear from the context. Once endowed with subset inclusion, $\mathcal{W}\text{-Rel}(X, Y)$ has a complete lattice structure, so that we can define $\mathcal{W}$-relations both inductively and coinductively. The identity $\mathcal{W}$-relation $\Delta : X \nrightarrow X$, the composition $R; S : X \nrightarrow Y$ of $\mathcal{W}$-relations $R : X \nrightarrow Y$ and $S : Y \nrightarrow Z$, and the converse (or transpose) $R^\top : Y \nrightarrow X$ of $R : X \nrightarrow Y$ are defined thus:

$$\frac{}{a \Vdash x\ \Delta\ x}\ (a \in \mathsf{W}) \qquad \frac{b \Vdash x\, R\, y \quad c \Vdash y\, S\, z \quad a \geq b \odot c}{a \Vdash x\ (R; S)\ z} \qquad \frac{a \Vdash x\, R\, y}{a \Vdash y\, R^\top\, x}$$

Notice that Δ relates identical elements at any possible worlds, whereas the composition of two $\mathcal{W}$-relations involves the monoidal product of possible worlds. Notice that Δ and $R; S$ are monotone (provided that R and S are), and thus $\mathcal{W}$-relations. Straightforward calculations show that (modal) relation composition is associative with unit element given by Δ. As a consequence, we have a category $\mathcal{W}$-Rel, whose objects are sets and whose arrows are $\mathcal{W}$-relations. Notice that both converse and composition are monotone and that converse behaves as an involution: $(R^\top)^\top = R$, $\Delta^\top = \Delta$, and $(R; S)^\top = S^\top; R^\top$.

Definition 7. *A modal relation $R : X \nrightarrow X$ is reflexive if $\Delta \subseteq R$, symmetric if $R^\top \subseteq R$, and transitive if $R; R \subseteq R$. Altogether, we obtain the notions of a modal preorder and modal equivalence.*

Any function $f : X \to Y$ can be regarded as an arrow in $\mathcal{W}$-Rel via the $\mathcal{W}$-relation $\{(x, a, f(x)) \mid x \in X, a \in \mathsf{W}\}$. For simplicity, we use the notation $f : X \to Y$ even when regarding f as an arrow in $\mathcal{W}$-Rel. Oftentimes we will deal with functions preserving modal relations. Such a property can be succinctly expressed in pointfree style as $R \subseteq f; S; g^\top$, where $R : X \nrightarrow Y$, $S : A \nrightarrow B$, $f : X \to A$, and $g : Y \to B$. In fact, we have $a \Vdash x\ (f; S; g^\top)\ y \iff a \Vdash f(x)\ S\ g(y)$. Calculating with such

$$a \Vdash f\ [R,S]\ g \overset{\triangle}{\iff} \forall b,c,x,y.\ (b \Vdash x\ R\ y\ \&\ c \geq a \odot b \implies c \Vdash f(x)\ S\ g(y))$$

$$a \Vdash (x,x')\ (R \otimes S)\ (y,y') \overset{\triangle}{\iff} \exists b,c.\ (b \Vdash x\ R\ y\ \&\ c \Vdash x'\ S\ y'\ \&\ a \geq b \odot c)$$

$$a \Vdash (x,x')\ (R \times S)\ (y,y') \overset{\triangle}{\iff} a \Vdash x\ R\ y\ \&\ a \Vdash x'\ S\ y.$$

$$a \Vdash t\ (R+S)\ s \overset{\triangle}{\iff} \exists x \in X, y \in Y. \begin{pmatrix} t = in_l\ x \\ s = in_l\ y \\ a \Vdash x\ R\ y \end{pmatrix} \text{ or } \exists x \in A, y \in B. \begin{pmatrix} t = in_r\ x \\ s = in_r\ y \\ a \Vdash x\ S\ y \end{pmatrix}$$

$$\begin{array}{llll}
\Delta_{A\times B} = \Delta_A \times \Delta_B & (R\times S);(Q\times P) = (R;Q)\times(S;P) & R \subseteq Q\ \&\ S \subseteq P \implies R\otimes S \subseteq Q\otimes P \\
\Delta_{A\times B} = \Delta_A \otimes \Delta_B & (R\otimes S);(Q\otimes P) = (R;Q)\otimes(S;P) & R \subseteq Q\ \&\ S \subseteq P \implies R\times S \subseteq Q\times P \\
\Delta_{A\to B} = [\Delta_A, \Delta_B] & [R,S];[Q,P] \subseteq [R;Q,S;P] & Q \subseteq R\ \&\ S \subseteq P \implies [R,S] \subseteq [Q,P] \\
\Delta_{A+B} = \Delta_A + \Delta_B & (R+S);(Q+P) \subseteq (R;Q)+(S;P) & R \subseteq Q\ \&\ S \subseteq P \implies R+S \subseteq Q+P
\end{array}$$

Fig. 3. Modal Relation Constructors and their Algebraic Laws

relational expressions, it is useful to recall the following *adjunction rules*: $R;g \subseteq Q \iff R \subseteq Q;g^\top$ and $f^\top;Q \subseteq S \iff Q \subseteq f;S$.

Finally, we introduce some constructions on modal relations that allow us to extend such relations to structured sets (such as product and function spaces).

Definition 8. *Given modal relations $R : X \nrightarrow Y$ and $S : A \nrightarrow B$, define the Reynolds arrow $[R,S] : A^X \nrightarrow B^Y$, the tensor $R \otimes S : X \times A \nrightarrow Y \times B$, the (cartesian) product $R \times S : X \times A \nrightarrow Y \times B$, and the coproduct $R + S : X + A \nrightarrow Y + B$ by the clauses in Figure 3.*

Notice that the tensor of modal relations requires to split worlds, whereas the cartesian product uses the same world to relate all elements inside pairs. The Reynolds arrow relates two functions at a world a if whenever we have input related at b, outputs are related at any world bigger than $a \odot b$. Figure 3 lists some useful algebraic laws for such constructions of Definition 8. Notice that Reynolds arrow is monotone in the second argument and antitone in the first one.

3.2 Relators and Corelators

Constructions in Definition 8 reflect most of the type constructors of $\Lambda_{\Sigma,\mathcal{S}}$ and thus the reader should be able to guess their role (see Section 6). If, for instance, we have defined program equivalence $\sim_\sigma$, $\sim_\tau$ at types σ, τ, respectively, then we essentially use $\sim_\sigma \otimes \sim_\tau$ to define equivalence at type $\sigma \otimes \tau$. The relational constructions of Definition 8, however, do not account neither for coeffects/modal types nor for computational effects. For the latter, we extend the notion of a *relator* to modal relations. For the former, instead, we introduce the novel notion of a *corelator* — i.e. of a *lax extension of a graded (monoidal) comonad* — which is to comonads what a relator is to a monad.

Definition 9. *Let F be a functor.*[5] *A* lax extension *of F is a (family of) map(s) $\Gamma : \mathcal{W}\text{-Rel}(X,Y) \to \mathcal{W}\text{-Rel}(F(X),F(Y))$ satisfying the laws in the first column of Figure 4.*

Definition 9 is nothing but the usual definition of a lax extension [Barr 1970] properly modified to the setting of modal relations. The map Γ extends the action of F on functions to modal relations, and it does so (lax) functorially.

[5]Recall that when referring to functors, monads, and comonads we tacitly assume them to be on Set.

$$\begin{array}{lll}
\Delta_{F(A)} \subseteq \Gamma\Delta_A & !_1 R \subseteq R & \\
\Gamma R; \Gamma S \subseteq \Gamma(R;S) & !_{s*r} R \subseteq !_s !_r R & R \subseteq \eta; \Gamma R; \eta^\top \\
F(f) \subseteq \Gamma(f) & !_s R \otimes !_s S \subseteq !_s (R \otimes S) & \Gamma\Gamma R \subseteq \mu; \Gamma R; \mu^\top \\
F(f)^\top \subseteq \Gamma(f^\top) & !_{s+r} R \subseteq c; (!_s R \otimes !_r R); c^\top & \\
R \subseteq S \implies \Gamma R \subseteq \Gamma S & s \leq r \implies !_r R \subseteq !_s R &
\end{array}$$

Fig. 4. F-relator (first column); Corelator (second column); T-relator (third column)

Example 4. The maps $- \circ S$, $R \circ -$, for $\circ \in \{\times, \otimes, +\}$, induced by Definition 8 are all lax extensions (of the corresponding functors on Set). The map $[R, -]$ is lax extension too, although $[-, S]$ is not, due the failure of monotonicity.

Lax extensions of functors are at the heart of many relational constructions. Dealing with computational effects, however, functors are not enough: we need to extend *monads* to modal relations.

Definition 10. *Let (T, η, μ) be a monad on* Set. *A* lax extension *of (T, η, μ) is a lax extension of T that additionally satisfies the laws in the third column of Figure 4.*

Due to their importance, we simply refer to lax extensions of a monad T as *T-relators* (and just as *relators* if the actual monad is not relevant or clear from the context). If Γ is a T-relator, then it properly interacts with the unit and multiplication of T. From an operational perspective, that means that relators nicely interact with the computational behaviour of the sequencing and return constructs. In particular, the condition $\Gamma\Gamma R \subseteq \mu; \Gamma R; \mu^\top$ can be replaced with the inclusion $R \subseteq f; \Gamma S; g^\top \implies \Gamma R \subseteq \gg\!= f; \Gamma S; (\gg\!= g)^\top$, which makes monadic binding explicit. Pointwise, this inclusion gives the following law.

$$\frac{\forall x, y, b.\ b \Vdash x\ R\ y \implies b \Vdash f(x)\ \Gamma S\ g(y) \quad a \Vdash \alpha\ \Gamma R\ \beta}{a \Vdash \alpha \gg\!= f\ \Gamma S\ \beta \gg\!= g}$$

Actually, since we work with continuous monads, we require relators to be continuous too, meaning that (i) $a \Vdash \bot\ \Gamma R\ \alpha$ holds for any possible world a and $\alpha \in T(X)$ (recall that $\bot$ denotes the bottom element of $T(X)$); (ii) and that we have the following induction principle, where $(\alpha_n)_n$ is an ω-chain in $T(X)$: $(\forall n)\ a \Vdash \alpha_n\ \Gamma R\ \beta \implies a \Vdash \bigsqcup_n \alpha_n\ \Gamma R\ \beta$.

Before showing examples of relators, we introduce the last (but definitely not least) construction on modal relations, which is a novel contribution of this work: *lax extensions of comonads*. Actually, since our fundamental intuition is that the action of modal types do not quite act on programs but on the possible world where programs are compared, we are concerned not with extending arbitrary comonads, but only a specific one: the *identity* comonad. For pedagogical reasons, we give the simplified version of a lax extension of the identity comonad only (but see Remark 2), this way avoiding the introduction of the notion of graded monoidal comonad [Gaboardi et al. 2016; Katsumata 2018]. We call the resulting notion a *corelator*.

Definition 11. *Given a grade algebra $(\mathcal{S}, \leq, +, *, 0, 1, \infty)$, a* corelator *associates to any modal relation $R : X \nrightarrow Y$ a $\mathcal{S}$-indexed family of modal relations $!_s R : X \nrightarrow Y$ in such a way that each $!_s$ is a lax extension of the identity functor and that additionally the laws in the second column of Figure 4 holds, where $c : X \to X \times X$ denotes the contraction map sending x to (x, x).*

Definition 11 reflects our reading of elements in $\mathcal{S}$ as usage prescriptions. Accordingly, we read a statement $a \Vdash x\ !_s R\ y$ as stating that a world a the objects x and y are R-related for a tester

which can use them according to s (for instance, s may tells how many times x and y can be used). The last law in the second column of Figure 4 then states that the more one can use expressions, the more she can discriminate between them (for instance, if x and y are equivalent when used n times, then they must be so when used $n-k$ times). Similarly, the inclusion $!_1 R \subseteq R$ states that, by default, modal relations are linear (notice also that we have $!_s R \subseteq R$, for any $s \geq 1$). Intuitively, the action of the map $!_s$ at a world a is to compare programs at different worlds b which are essentially determined by s. However, instead of giving an explicit definition of what it means to "change possible world", we have given a set of structural axioms (namely the ones in Definition 11) that any such a notion should satisfy.

Remark 2. Definition 11 is a specific case of the more general notion of a lax extension of a graded monoidal comonad. Such a level of generality is not needed in this work: nonetheless, the reader familiar with graded comonads can easily generalise Definition 11 to arbitrary comonads. For instance, the first two inclusions in the second column of Figure 4 should be replaced with $!_1 R \subseteq \varepsilon; R; \varepsilon^{\top}$ and $!_{s*r} R \subseteq \delta_{s,r}; !_s !_r R; \delta^{\top}_{s,r}$, respectively, where $\varepsilon : C_1(A) \to A$ and $\delta_{s,r} : C_{s*r}(A) \to C_s(C_r(A))$ are the (graded) counit and comultiplication of the (graded) comonad C, respectively.[6]

Finally, we need effects and coeffects to interact well with one another. Formally, we model such a well-behaved interaction by saying that a corelator distributes over a relator, laxly.

Definition 12. *Given a relator* Γ *and a corelator* $!$*, we say that* $!$ distributes laxly *over* Γ *if* $!_s(\Gamma R) \subseteq \Gamma(!_s R)$ *holds for any* $s \geq 1$.

As usual, the requirement $s \geq 1$ indicates that we look at those cases where objects can be used nontrivially.

3.3 Examples

In this section, we give some examples of relators and corelators that apply to the (co)effectful calculi seen so far.

Example 5 (Maybe monad)**.** Let $R : X \nrightarrow Y$ be a modal relation. Define $\hat{\mathcal{M}} R : \mathcal{M}(X) \nrightarrow \mathcal{M}(Y)$ as follows:

$$a \Vdash \alpha \; \hat{\mathcal{M}} R \; \beta \overset{\triangle}{\iff} [\alpha = \mathit{just}\ x \implies (\beta = \mathit{just}\ y \;\&\; a \Vdash x \; R \; y)].$$

Then, $\hat{\mathcal{M}}$ is a $\mathcal{M}$-relator. $\hat{\mathcal{M}} R$ generalises the usual clause used to define operational preorders between programs. Accordingly, a program e approximates the behaviour of a term f at world a if either e diverges or both e and f converge and the resulting values are related at a. If we take the frame $([0,\infty], \leq, +, 0)$ and read $a \Vdash e \; R \; f$ as stating that the R-distance between e and f is at most a, then $a \Vdash e \; \hat{\mathcal{M}} R \; f$ tells us that if e diverges, then the $\hat{\mathcal{M}} R$-distance between e and f is bounded by any a — and thus it is bounded by 0. Otherwise, e converges to value v, and thus f converges to a value w such that the $\hat{\mathcal{M}} R$-distance between e and f is the R-distance between v and w.

Example 6 (Powerset Monad)**.** Let $R : X \nrightarrow Y$ be a modal relation. Define $\hat{\mathcal{P}} R : \mathcal{P}(X) \to \mathcal{P}(Y)$ as follows:

$$a \Vdash \alpha \; \hat{\mathcal{P}} R \; \beta \overset{\triangle}{\iff} \forall x \in \alpha.\ \exists y \in \beta.\ \exists b.\ a \geq b \;\&\; b \Vdash x \; R \; y.$$

Then $\hat{\mathcal{P}}$ is a $\mathcal{P}$-relator. $\hat{\mathcal{P}}$ generalises the usual extension used to define, e.g. simulation relations on transition systems, to modal relations. Notice that such an extension is defined for any frame. For instance, taking the frame $([0,\infty], \leq, +, 0)$ and defining the distance $\rho(x,y) \triangleq \inf\{a \mid a \Vdash x \, R \, y\}$, we

[6] In the case of the identity comonad, counit and comultiplication are trivially given by the identity function.

see that $a \Vdash \alpha \hat{\mathcal{P}} R \beta$ holds if and only if $a \geq \sup_{x \in \alpha} \inf_{y \in \beta} \rho(x, y)$, meaning that $\inf\{a \mid a \Vdash \alpha \hat{\mathcal{P}} R \beta\}$ is nothing but the asymmetric Hausdorff extension of ρ [Searcóid 2006].

Example 7 (Distribution Monad)**.** In this example, we consider modal relations on the frame $([0, \infty], \leq, +, 0)$ only. We show that there is a natural lax extension of the distribution monad which is nothing but the relational version of the well-known Wasserstein-Kantorovich distance. Recall that a *coupling* between two distributions μ, ν over sets X, Y, respectively, is a distribution ω over $X \times Y$ such that: $\sum_y \omega(x, y) = \mu(x)$ and $\sum_x \omega(x, y) = \nu(y)$. We denote the collection of couplings of μ and ν by $\Omega(\mu, \nu)$. Given a distance $\rho : X \times Y \to [0, \infty]$, the Wasserstein-Kantorovich distance induced by ρ is the distance $\hat{\mathcal{D}}\rho : \mathcal{D}(X) \times \mathcal{D}(Y) \to [0, \infty]$ defined by $\hat{\mathcal{D}}\rho(\mu, \nu) \triangleq \inf_{\omega \in \Omega(\mu,\nu)} \mathbb{E}(\mathcal{D}(\rho)(\omega))$, where $\mathbb{E} : \mathcal{D}[0, \infty] \to [0, \infty]$ is the expectation map. We can rephrase the definition of the Wasserstein-Kantorovich distance in terms of modal relations using ternary couplings. Given a modal relation $R : X \nrightarrow Y$, we define $\hat{\mathcal{D}}R : \mathcal{D}(X) \nrightarrow \mathcal{D}(Y)$ by:

$$a \Vdash \mu \, \hat{\mathcal{D}}R \, \nu \overset{\triangle}{\iff} \exists \omega \in \mathcal{D}(X \times [0, \infty] \times Y). \begin{cases} \mathcal{D}(\langle \pi_1, \pi_3 \rangle) \in \Omega(\mu, \nu) \\ \mathbb{E}[\mathcal{D}(\pi_2)(\omega)] \leq a \\ \omega(x, a, y) > 0 \implies a \Vdash x \, R \, y. \end{cases}$$

Indeed, defining $\rho(x, y) \triangleq \inf\{a \mid a \Vdash x \, R \, y\}$, then we have $\hat{\mathcal{D}}(\rho)(\mu, \nu) = \inf\{a \mid a \Vdash \mu \, \hat{\mathcal{D}}R \, \nu\}$. Using this correspondence, we can show that $\hat{\mathcal{D}}$ is a $\mathcal{D}$-relator.[7]

Let us now move to examples of corelators.

Example 8 (Action Extension)**.** Our first example is an abstract extension which we call *action extension*. Given a frame $\mathcal{W} = (\mathsf{W}, \leq, \odot, \varepsilon)$, a *lax action* is a monotone map $\circ : \mathcal{S} \times \mathsf{W} \to \mathsf{W}$ satisfying the following laws, where we write $s \circ a$ for the action of $\circ$ on (s, a):

$$s \circ \varepsilon \leq \varepsilon \qquad s \circ (a \odot b) \leq (s \circ a) \odot (s \circ b) \qquad s \circ (r \circ a) \leq (s * r) \circ a$$
$$1 \circ a \leq a \qquad (s \circ a) \odot (r \circ a) \leq (s + r) \circ a$$

The *action extension* $[-]$ is then defined by

$$a \Vdash x \, [s]R \, y \overset{\triangle}{\iff} \exists b. \, a \geq s \circ b \;\&\; b \Vdash x \, R \, y.$$

The defining laws of a lax action ensures $[-]$ to be a corelator. The action extension has been extensively used in program metrics [de Amorim et al. 2017, 2019; Reed and Pierce 2010], where one takes the frame $(\mathcal{S}, \leq, +, 0)$ induced by the grade algebra with lax action given by grade multiplication: $s \circ r \triangleq s * r$. In those cases, we have $r \Vdash x \, [s]R \, y$ iff $\exists p. \, r \geq s * p \;\&\; p \Vdash x \, R \, y$. Notice that any modal calculus comes with this "canonical" corelator, to which we refer to as the canonical corelator. Moreover, the canonical corelator (properly instantiated) distributes over all the examples of relators given so far. Let us see some concrete instances of the canonical corelator.

1. When instantiated on the grade algebra $([0, \infty], \leq, +, \cdot, 0, 1, \infty)$, the canonical corelator gives

$$r \Vdash x \, [s]R \, y \iff \exists p. \, r \geq s \cdot p \text{ and } p \Vdash x \, R \, y.$$

In particular, if $r \Vdash x \, R \, y$, then $s \cdot r \Vdash x \, [s]R \, y$. This is essentially the modal type clause used by Reed and Pierce [2010] to define metric logical relations .

[7] There is a rich theory of relators acting on (general) distance-like functions rather than relations [Hofmann et al. 2014]. The Wasserstein-Kantorovich lifting is a relator in that sense [Gavazzo 2018], and from such a result it follows that our candidate relator is indeed a $\mathcal{D}$-relator. This is a specific instance of a more general correspondence between modal reasoning and (abstract) metric reasoning (see Section 8).

2. Given a frame $\mathcal{W}$, the set $\mathsf{End}(\mathsf{W})$ of W-endomorphisms forms a grade algebra with semiring multiplication given by function composition, unit element given by the identity function, and all other operations defined pointwise. This way, we obtain an action $\circ : \mathsf{End}(\mathsf{W}) \times \mathsf{W} \to \mathsf{W}$ given by function application: $h \circ a = h(a)$. In this case, the canonical corelator gives

$$a \Vdash x\ [h]R\ y \iff \exists b.\ a \geq h(b) \text{ and } b \Vdash x\ R\ y.$$

In particular, by taking W as $([0,\infty], \leq, +, \cdot, 0, 1, \infty)$ and looking at $\mathcal{W}$-relations as defining distance functions, we obtain something close to the so-called *f-sensitivity* [Barthe et al. 2018]. Similar instances of the canonical corelator have been studied in the context of quantale-based program distances [Gavazzo 2018].

Example 9. Our second example of a corelator comes from intuition modal logic [Simpson 1994]. Let $\mathcal{S}$ be the one-element grade algebra and consider the (cartesian) frame $(\mathsf{W}, \leq, \vee, \top)$. Frames of this form are used, for instance, to give possible world semantics to (propositional) intuitionistic logic and to define Kripke-style logical relations [Mitchell 1996]. Let us now endow the frame $(\mathsf{W}, \leq, \vee, \top)$ with a reflexive and transitive relation $Q \subseteq \mathsf{W} \times \mathsf{W}$ such that $\leq; Q \subseteq Q$, and define the Kripke extension (we do not write the unique grade subscript) $\Box$ by:

$$a \Vdash x\ \Box R\ y \overset{\triangle}{\iff} \forall b.\ a\ Q\ b \implies b \Vdash x\ R\ y.$$

Then, $\Box$ gives a corelator. The map $\Box$ is nothing but the relational counterpart of the propositional construction used in the possible-worlds semantics of the box modality in intuitionistic modal logics.

Example 10. Our last example of a corelator deals with information flow. Let us fix a (op-)lattice of security levels $(\mathcal{L}, \geq, \wedge, \vee, \bot, \top)$. Define the *masking extension* $\uparrow$ thus:

$$b \Vdash x \uparrow_a R\ y \overset{\triangle}{\iff} a \not\leq b \text{ or } b \Vdash x\ R\ y.$$

Then, $\uparrow$ is a corelator. The action of $\uparrow_a$ is to make code invisible to users with permission below a. Recall that a judgment of the form $b \Vdash x\ R\ y$ has the intended meaning that x and y are R-indistinguishable to users with security permission b. For instance, two *classified* elements x, y are indistinguishable to a user with *low* confidentiality permission, even if the two terms are actually different.

4 RELATIONAL REASONING

In the previous section, we have introduced modal relations, relators, and corelators, in the abstract. However, a relational account of $\Lambda_{\Sigma,\mathcal{S}}$ requires to manipulate not just modal relations, but relations between $\Lambda_{\Sigma,\mathcal{S}}$ terms, to which we refer as *term relations*. In this section, we develop a calculus of such relations in the style of the relational calculus by Lassen [1998b]. The relational machinery we are going to define allows us to abstract from the syntactic bureaucracy intrinsic to $\Lambda_{\Sigma,\mathcal{S}}$ avoiding involved definitions, such as those of syntactic occurrences and term contexts. Additionally, such a level of abstraction reduces some long, syntactic proofs to simple algebraic calculations in pointfree style. From now on, let us assume to have fixed a frame $\mathcal{W}$, a continuous monad (T, η, μ), a continuous T-relator Γ, and a corelator !.

Definition 13. *1. A* closed *term relation is a pair $R = (R^{\wedge}, R^{\vee})$ of maps associating to each type σ modal relations $R^{\wedge}_{\sigma} : \Lambda^{\bullet}_{\sigma} \nrightarrow \Lambda^{\bullet}_{\sigma}$ and $R^{\vee}_{\sigma} : \mathcal{V}^{\bullet}_{\sigma} \nrightarrow \mathcal{V}^{\bullet}_{\sigma}$, respectively.*

*2. An (*open*) term relation R associates to each sequent $\Gamma \vdash \sigma$ a modal relation $\Gamma \vdash^{\wedge} - \Vdash - R - : \sigma$ on $\Lambda_{\Gamma \vdash^{\wedge} \sigma}$ and a modal relation $\Gamma \vdash^{\mathcal{V}} - \Vdash - R - : \sigma$ on $\mathcal{V}_{\Gamma \vdash^{\mathcal{V}} \sigma}$. We require term relations to be*

closed under weakening:

$$\frac{\Gamma \vdash^{\Lambda} a \Vdash e\ R\ f : \sigma}{\Gamma + \Theta \vdash^{\Lambda} a \Vdash e\ R\ f : \sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v\ R\ w : \sigma}{\Gamma + \Theta \vdash^{\mathcal{V}} a \Vdash v\ R\ w : \sigma}$$

Example 11. Both the discrete term relation Δ and the indiscrete relation ∇ defined below are (open) term relations. The empty relation is a term relation too.

$$\frac{}{\Gamma \vdash^{\Lambda} a \Vdash e\ \Delta\ e : \sigma} \qquad \frac{}{\Gamma \vdash^{\mathcal{V}} a \Vdash v\ \Delta\ v : \sigma} \qquad \frac{}{\Gamma \vdash^{\Lambda} a \Vdash e\ \nabla\ f : \sigma} \qquad \frac{}{\Gamma \vdash^{\mathcal{V}} a \Vdash v\ \nabla\ w : \sigma}$$

We write Rel and $\mathsf{Rel}^{\mathrm{c}}$ for the collections of open and closed term relations, respectively. Notice that Rel carries a complete lattice structure given by set-theoretic inclusion: $R \subseteq S$ holds if

$$\Gamma \vdash^{\Lambda} a \Vdash e\ R\ f : \sigma \implies \Gamma \vdash^{\Lambda} a \Vdash e\ S\ f : \sigma, \qquad \Gamma \vdash^{\mathcal{V}} a \Vdash v\ R\ w : \sigma \implies \Gamma \vdash^{\mathcal{V}} a \Vdash v\ S\ w : \sigma.$$

As a consequence, we can define term relations both inductively and coinductively. Term relation composition and converse are defined in the obvious way: for instance, we define (the computation part) of term relation composition by:

$$\frac{\Gamma \vdash^{\Lambda} a \Vdash e\ R\ f : \sigma \quad \Gamma \vdash^{\Lambda} b \Vdash f\ S\ g : \sigma \quad c \geq a \odot b}{\Gamma \vdash^{\Lambda} c \Vdash e\ (R;S)\ g : \sigma}$$

Notice that $R;S$ is indeed a term relation (viz. it is monotone). We can thus extend the notion of a reflexive, transitive, and symmetric (modal) relation to term relations. Composition and transpose satisfies all the laws seen for abstract modal relations. Moreover, term relation composition is monotone and continuous in both arguments. Finally, we observe that all the general constructions introduced in previous sections extend to term relations *mutatis mutandis*. We now introduce some constructions specific to term relations, which are at the heart of our relational calculus. We begin with ways to move and back forth between open and closed term relations.

Definition 14. 1. *Given a term relation $R \in \mathsf{Rel}$, the* closed restriction R^{c} *of R associates to each type σ the modal relations $\cdot \vdash^{\Lambda} - \Vdash - R - : \sigma$ and $\cdot \vdash^{\mathcal{V}} - \Vdash - R - : \sigma$.*

2. *Given a closed term relation R, the* open extension *of R as the (open) term relation R^{o} defined thus:*

$$\frac{a \Vdash e[\boldsymbol{v}/\boldsymbol{x}]\ R^{\Lambda}_{\sigma}\ f[\boldsymbol{v}/\boldsymbol{x}]}{\boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\Lambda} a \Vdash e\ R^{\mathrm{o}}\ f : \sigma} \qquad \frac{a \Vdash v[\boldsymbol{v}/\boldsymbol{x}]\ R^{\mathcal{V}}_{\sigma}\ w[\boldsymbol{v}/\boldsymbol{x}]}{\boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\mathcal{V}} a \Vdash v\ R^{\mathrm{o}}\ w : \sigma}$$

3. *Given a closed term relation R, we define the* substitutive extension *of R as the (open) term relation R^{s} defined thus:*[8]

$$\frac{\boldsymbol{b} \Vdash \boldsymbol{v}\ !_s R^{\mathcal{V}}_{\boldsymbol{\sigma}}\ \boldsymbol{w}\ \&\ c \geq a \odot \bigodot \boldsymbol{b} \implies c \Vdash e[\boldsymbol{v}/\boldsymbol{x}]\ R^{\Lambda}_{\sigma}\ f[\boldsymbol{w}/\boldsymbol{x}]}{\boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\Lambda} a \Vdash e\ R^{\mathrm{s}}\ f : \sigma}$$

$$\frac{\boldsymbol{b} \Vdash \boldsymbol{v}\ !_s R^{\mathcal{V}}_{\boldsymbol{\sigma}}\ \boldsymbol{w}\ \&\ c \geq a \odot \bigodot \boldsymbol{b} \implies c \Vdash v[\boldsymbol{v}/\boldsymbol{x}]\ R^{\mathcal{V}}_{\sigma}\ w[\boldsymbol{w}/\boldsymbol{x}]}{\boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\mathcal{V}} a \Vdash v\ R^{\mathrm{s}}\ w : \sigma}$$

We now introduce two fundamental constructions on term relations: *relation substitution* and *compatible refinement*. The rich type system of $\Lambda_{\Sigma,\mathcal{S}}$ makes these definition a bit involved, as the former mimics the substitution lemma for the static semantics (Lemma 1), whereas the latter follows the defining rules of the static semantics itself. However, once we have in our arsenal such constructions, then we can derive many other important relational notions (as well as prove their properties) algebraically. This results in a major improvement in concision (as well as in precision) of our analysis.

[8] Notice that we employ the vector notation and write $\boldsymbol{a} \Vdash \boldsymbol{e}\ !_s R\ \boldsymbol{f}$ for $a_1 \Vdash e_1\ !_{s_1} R\ f_1, \ldots, a_n \Vdash e_n\ !_{s_n} R\ f_n$. For $a = a_1, \ldots, a_n$, we also write $\bigodot a$ in place of $a_1 \odot \cdots \odot a_n$.

Definition 15. *Given term relations R, S, define the* substitution of S into R *as the term relation $R[S]$ defined thus (where we assume $\Gamma \sim \Theta_i$):*

$$\frac{\Gamma, \boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\Lambda} a \Vdash e\ R\ f : \sigma \quad \boldsymbol{\Theta} \vdash^{\mathcal{V}} \boldsymbol{b} \Vdash \boldsymbol{v}\ !_s S\ \boldsymbol{w} \quad c \geq a \odot \bigodot \boldsymbol{b}}{\Gamma + \sum \boldsymbol{s} * \boldsymbol{\Theta} \vdash^{\Lambda} c \Vdash e[\boldsymbol{v}/\boldsymbol{x}]\ R[S]\ f[\boldsymbol{w}/\boldsymbol{x}] : \sigma}$$

$$\frac{\Gamma, \boldsymbol{x} :_s \boldsymbol{\sigma} \vdash^{\mathcal{V}} a \Vdash v\ R\ w : \sigma \quad \boldsymbol{\Theta} \vdash^{\mathcal{V}} \boldsymbol{b} \Vdash \boldsymbol{v}\ !_s S\ \boldsymbol{w} \quad c \geq a \odot \bigodot \boldsymbol{b}}{\Gamma + \sum \boldsymbol{s} * \boldsymbol{\Theta} \vdash^{\mathcal{V}} c \Vdash v[\boldsymbol{v}/\boldsymbol{x}]\ R[S]\ w[\boldsymbol{w}/\boldsymbol{x}] : \sigma}$$

Notice that Definition 15 actively uses the corelator ! to cope with the *non-local* behaviour of modal substitution: if we substitute two expressions v, w related at a world a — so that $a \Vdash v\,R\,w : \sigma$ — in an expression $x :_s \sigma \vdash^{\Lambda} e : \tau$, then the resulting expressions $e[v/x]$ and $e[w/x]$ are in general not related at a, as e is licensed to use v (resp. w) as prescribed by s. For that reason, we perform relation substitution only when R is under the scope of $!_s$. Let us continue with the development of our relational calculus.

Definition 16. *A term relation R is* value-substitutive *if $R[\Delta] \subseteq R$ and it is* substitutive *if $R[R] \subseteq R$. A closed term relation is (value) substitutive if its open extension is.*

Lemma 3. *Let R be a closed term relation. Then, the open extension of R is value substitutive ($R^{\mathrm{o}}[\Delta] \subseteq R^{\mathrm{o}}$) and that the substitutive extension of R is substitutive ($R^{\mathrm{s}}[R^{\mathrm{s}}] \subseteq R^{\mathrm{s}}$).*

We now move to the definition of the compatible refinement of a term relation. Intuitively, the compatible refinement $\widehat{R}$ of R relates expressions with identical outermost constructor and immediate subterms pairwise related by R.

Definition 17. *The* compatible refinement $\widehat{R}$ *of an open term relation R is defined by the rules in Figure 5. We say that R is compatible if $\widehat{R} \subseteq R$, and that a closed term relation is compatible if its open extension is. If R is compatible, reflexive, and transitive, we say that it is a* precongruence*; if, additionally, it is symmetric, we say that R is a* congruence.

Figure 5 defines a map $R \mapsto \widehat{R}$ on term relations which is monotone and distributes over term relation composition ($\widehat{R;S} = \widehat{R};\widehat{S}$) and converse ($\widehat{R^{\top}} = (\widehat{R})^{\top}$). Notice that R is compatible if it is a pre-fixed point of $R \mapsto \widehat{R}$. The identity term relation Δ is such a pre-fixed point (i.e. $\widehat{\Delta} \subseteq \Delta$), and actually it is the least one: $\Delta = \mu X.\widehat{X}$. As a consequence, we have that any compatible relation is reflexive. Finally, we notice that compatible term relations form a complete lattice.

Lemma 4. *The collection of compatible term relations forms a complete lattice ordered by $\subseteq$.*

Proof Sketch. Given a set ρ of compatible relations, we define the meet and join of ρ as $\bigcap \rho$ and $\bigcap\{S \mid \widehat{S} \subseteq S, \bigcup \rho \subseteq S\}$, respectively. Finally, we observe that the top and bottom element of the lattice are given by the indiscrete ∇ and discrete Δ term relations, respectively. □

Summing up, in this section we have defined a core relational calculus extending the calculus by Lassen [Lassen 1998b] to an effectful and coeffectful scenario. The rest of this paper shows how three main notions of program refinement (and consequently of program equivalence) can be defined and studied inside our calculus. We begin with *contextual approximation* [Morris 1969].

5 CONTEXTUAL APPROXIMATION

Morris' style *contextual approximation* relates two programs e, f if whenever we plug such programs inside an arbitrary term context C, there is no observable behaviour of $C[e]$ that is not an observable behaviour of $C[f]$. Although appealing, making this intuition into a rigorous formal definition is

$$\frac{}{\Gamma, x :_s \sigma \vdash^{\mathcal{V}} a \Vdash x \,\widehat{R}\, x : \sigma} \qquad \frac{}{\Gamma \vdash^{\Lambda} a \Vdash \mathbf{op}\,\widehat{R}\,\mathbf{op} : \sigma_{\mathbf{op}}}$$

$$\frac{\Gamma, x :_1 \sigma \vdash^{\Lambda} a \Vdash e \,R\, f : \tau}{\Gamma \vdash^{\mathcal{V}} a \Vdash \lambda x.e \,\widehat{R}\, \lambda x.f : \sigma \multimap \tau} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, u : \sigma \multimap \tau \quad \Theta \vdash^{\mathcal{V}} b \Vdash w \,R\, z : \sigma \quad a \geq a \odot b}{\Gamma + \Theta \vdash^{\Lambda} c \Vdash v w \,\widehat{R}\, u z : \tau}$$

$$\frac{}{\Gamma \vdash^{\mathcal{V}} a \Vdash \langle\rangle \,\widehat{R}\, \langle\rangle : \mathbf{unit}} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, u : \sigma \quad \Gamma \vdash^{\mathcal{V}} a \Vdash w \,R\, z : \tau}{\Gamma \vdash^{\mathcal{V}} a \Vdash \langle v, w\rangle \,\widehat{R}\, \langle u, z\rangle : \sigma \times \tau}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v_1 \,R\, v_2 : \sigma_1 \times \sigma_2}{\Gamma \vdash^{\Lambda} a \Vdash v_1.i \,\widehat{R}\, v_2.i : \sigma_i} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, u : \sigma \quad \Theta \vdash^{\mathcal{V}} b \Vdash w \,R\, z : \tau \quad c \geq a \odot b}{\Gamma + \Theta \vdash^{\mathcal{V}} c \Vdash v \otimes w \,\widehat{R}\, u \otimes z : \sigma \otimes \tau}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,!_s R\, w : \sigma \otimes \tau \quad \Theta, x :_s \sigma, y :_s \tau \vdash^{\Lambda} b \Vdash e \,R\, f : \rho \quad c \geq a \odot b}{s * \Gamma + \Theta \vdash^{\Lambda} c \Vdash \mathbf{let}\, x \otimes y = v \,\mathbf{in}\, e \,\widehat{R}\, \mathbf{let}\, x \otimes y = w \,\mathbf{in}\, f : \rho}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, w : \mathbf{void}}{\Gamma \vdash^{\Lambda} a \Vdash \mathbf{abort}\, v \,\widehat{R}\, \mathbf{abort}\, w : \sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v_i \,R\, v_i : \sigma_i}{\Gamma \vdash^{\mathcal{V}} a \Vdash \mathbf{in}_i\, v_1 \,\widehat{R}\, \mathbf{in}_i\, v_2 : \sigma_1 + \sigma_2}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,!_s R\, w : \sigma + \tau \quad \Theta, x :_s \sigma \vdash^{\Lambda} b \Vdash e \,R\, g : \rho \quad \Theta, y :_s \tau \vdash^{\Lambda} b \Vdash f \,R\, h : \rho \quad c \geq a \odot b}{s * \Gamma + \Theta \vdash^{\Lambda} c \Vdash \mathbf{case}\, v \,\mathbf{of}\, (\mathbf{in}_l\, x.e \mid \mathbf{in}_r\, x.f) \,\widehat{R}\, \mathbf{case}\, w \,\mathbf{of}\, (\mathbf{in}_l\, x.g \mid \mathbf{in}_r\, x.h) : \rho}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, w : \sigma[\mu t.\sigma/t]}{\Gamma \vdash^{\mathcal{V}} a \Vdash \mathbf{fold}\, v \,\widehat{R}\, \mathbf{fold}\, w : \mu t.\sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, w : \mu t.\tau}{\Gamma \vdash^{\Lambda} a \Vdash \mathbf{unfold}\, v \,\widehat{R}\, \mathbf{unfold}\, w : \sigma[\mu t.\sigma/t]}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,!_s R\, w : \sigma}{s * \Gamma \vdash^{\mathcal{V}} a \Vdash !v \,\widehat{R}\, !w : !_s \sigma} \qquad \frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,!_r R\, w : !_s \sigma \quad \Theta, x :_{r*s} \sigma \vdash^{\Lambda} b \Vdash e \,R\, f : \tau \quad c \geq a \odot b}{r * \Gamma + \Theta \vdash^{\Lambda} c \Vdash \mathbf{case}\, v \,\mathbf{of}\, \{!x \to e\} \,\widehat{R}\, \mathbf{case}\, w \,\mathbf{of}\, \{!x \to f\} : \tau}$$

$$\frac{\Gamma \vdash^{\mathcal{V}} a \Vdash v \,R\, w : \sigma}{\Gamma \vdash^{\Lambda} a \Vdash \mathbf{return}\, v \,\widehat{R}\, \mathbf{return}\, w : \sigma} \qquad \frac{\Gamma \vdash^{\Lambda} a \Vdash e \,!_{s \vee 1} R\, g : \tau \quad \Gamma, x :_s \tau \vdash^{\Lambda} b \Vdash f \,R\, h : \sigma \quad c \geq a \odot b}{(s \vee 1) * \Gamma + \Theta \vdash^{\Lambda} c \Vdash \mathbf{let}\, x = e \,\mathbf{in}\, f \,\widehat{R}\, \mathbf{let}\, x = g \,\mathbf{in}\, h : \sigma}$$

Fig. 5. Compatible refinement $\Lambda_{\Sigma,\mathcal{S}}$.

quite challenging, as the syntactic notion of a context — which is notoriously difficult, especially in presence of graded modal types — is involved. We overcome the problem by taking advantages of our relational calculus and defining contextual approximation as the *largest preadequate and compatible term relation*. In this setting, preadequacy is a property of term relations which is meant to capture some minimal desiderata on program approximations. Standard examples of notions of preadequacy usually involve both effectful and coffectful notions. For instance, in a nondeterministic language for information-flow, (asymmetric) may and must converge at a given security level is an example of a preadequacy predicate; whereas in a language with program sensitivity and probabilistic nondeterminism, a preadequacy predicate is given by convergence (possibly to some specific set of values) below or above a certain threshold. Here, we do not commit ourselves to any specific notion of preadequacy, and just consider a minimal axiomatics for it. Notice, however, that the informal examples just sketched show that such an axiomatics cannot requires preadequacy predicates to be closed under term relation inclusion (in fact, most of the preadequacy predicates proposed in the literature do not satisfy such a condition), so that we cannot appeal to the Knaster-Tarski Theorem to infer the existence of the largest preadequate

and compatible term relation as the greatest fixed point of a monotone map. We thus prove the existence of such a desired relation explicitly.

Lemma 5. *Let* Adeq $\subseteq$ Rel *be a predicate on term relations closed under non-empty union and relation composition (i.e.* $\bigcup_{i\in I} R_i \in$ Adeq*, whenever* $I \neq \emptyset$ *and* $R_i \in$ Adeq *for any* i*; and* $R; S \in$ Adeq *whenever* $R \in$ Adeq *and* $S \in$ Adeq*). If* $\Delta \in$ Adeq*, then there exists a largest compatible term relation* $S \in$ Adeq*. Additionally,* S *is transitive.*

Proof Sketch. Define the term relation S as $S \triangleq \bigcup\{R \in \text{Adeq} \mid \widehat{R} \subseteq R\}$. By hypothesis $\Delta \in$ Adeq, so that $\{R \in \text{Adeq} \mid \widehat{R} \subseteq R\}$ is non-empty. As a consequence, since Adeq is closed under non-empty union, $S \in$ Adeq. To conclude the first part of the thesis it remains to prove $\widehat{S} \subseteq S$. This is proved by induction on the definition of $\widehat{S}$ using the defining properties of corelators. Since S is the largest compatible relation in Adeq, to prove that it is transitive, it is enough to show that $S; S$ is compatible and belongs to Adeq. The latter follows since Adeq is closed under composition, whereas the former follows by compatibility of S ($\widehat{S;S} = \widehat{S};\widehat{S} \subseteq S; S$.) □

Definition 18. *Let* Adeq $\subseteq$ Rel *be as in the statement of Lemma 5. We define the term relation* $\leq^{\text{ctx}}$*, called* contextual approximation*, as the largest compatible term relation contained in* Adeq.

Notice that $\leq^{\text{ctx}}$ is compatible, reflexive, and transitive (and thus a precongruence). Additionally, $\leq^{\text{ctx}}$ comes with proof techniques resembling a coinduction proof principle: to prove that two expressions are in contextual approximation relation, it is enough to exhibit a compatible term relation in Adeq containing these expressions. Symbolically: $\widehat{R} \subseteq R \;\&\; R \in \text{Adeq} \implies R \subseteq \leq^{\text{ctx}}$.

From a structural perspective, contextual approximation have all the desirable properties a notion of program refinement should have: what it lacks is practical usability. Most of the times, proofs by contextual approximations are just to difficult to be practically feasible. For that reason, any theory of program relations should support also other, more usable notions of program equivalence.

6 LOGICAL RELATIONS

In this section, we define effectful and coeffectful logical relations. Logical relations are closed term relations reflecting the logical structure and complexity of types. Historically, logical relations have been defined as a *relational semantics*, whereby each type is inductively interpreted as a set endowed with a logical relation on it and (open) terms are interpreted as relational homomorphisms. The latter result directly follows from the so-called fundamental lemma of logical relations, which states that logical relations are reflexive. From such a result it also follows that logical relations are compatible. Logical relations can also be understood in purely operational terms simply by taking the semantics of a type as a suitable relation over *expressions* of that type. This approach is the main one followed in the more recent literature on program semantics [Ahmed 2006], and the one we follow too. For ease of exposition, in this section we restrict to the fragment of $\Lambda_{\Sigma,\mathcal{S}}$ without recursive types. That makes the calculus strongly normalising, and allows us to work with a cleaner notion of logical relation avoiding complicacies such as step-indexing [Appel and McAllester 2001].

Definition 19. *The logical relation* $\preceq$*, called* logical preorder*, is the* closed *term relation defined inductively on types in Figure 6. We extend* $\preceq$ *to an open term relation by taking its* substitutive extension.

Definition 19 resembles the logical relation defined by Abel and Bernardy [2020] but with a crucial difference: our logical relation deals with both effects and coeffects (whereas the one by Abel and Bernardy cannot account for computational effects): this is made evident in the defining clause of $\preceq^{\Lambda}_{\sigma}$ — which relies on the relator Γ — and in the defining clause of $\preceq^{\mathcal{V}}_{\Box_s\sigma}$ — which relies on the corelator $!_s$.

$$
\begin{array}{rl}
a \Vdash e \preceq^{\Lambda} f : \sigma & \stackrel{\triangle}{\Longleftrightarrow} a \Vdash [\![e]\!]_{\mathcal{E}}\ \Gamma \preceq^{\mathcal{V}} [\![f]\!]_{\mathcal{E}} : \sigma \\
a \Vdash \langle\rangle \preceq^{\mathcal{V}} \langle\rangle : \mathbf{unit} & \stackrel{\triangle}{\Longleftrightarrow} \text{always} \\
a \Vdash v \otimes w \preceq^{\mathcal{V}} u \otimes z : \sigma \otimes \tau & \stackrel{\triangle}{\Longleftrightarrow} \exists b, c.\ (b \Vdash v \preceq^{\mathcal{V}} u : \sigma)\ \&\ (b \Vdash w \preceq^{\mathcal{V}} z : \tau)\ \&\ a \geq b \odot c \\
a \Vdash (v, w) \preceq^{\mathcal{V}} (u, z) : \sigma \times \tau & \stackrel{\triangle}{\Longleftrightarrow} (a \Vdash v \preceq^{\mathcal{V}} u : \sigma)\ \&\ (a \Vdash w \preceq^{\mathcal{V}} z : \tau) \\
a \Vdash \mathbf{in}_i\ v \preceq^{\mathcal{V}} \mathbf{in}_j\ w : \sigma_1 + \sigma_2 & \stackrel{\triangle}{\Longleftrightarrow} i = j\ \&\ a \Vdash v \preceq^{\mathcal{V}} w : \sigma_i \\
a \Vdash \lambda x.e \preceq^{\mathcal{V}} \lambda x.f : \sigma \multimap \tau & \stackrel{\triangle}{\Longleftrightarrow} \forall v, w, b, c. \left(\begin{array}{c} b \Vdash v \preceq^{\mathcal{V}} w : \sigma \\ c \geq a \odot b \end{array} \right) \Longrightarrow c \Vdash e[v/x] \preceq^{\Lambda} f[v/x] : \tau \\
a \Vdash [v] \preceq^{\mathcal{V}} [w] : \Box_s \sigma & \stackrel{\triangle}{\Longleftrightarrow} a \Vdash v\ !_s(\preceq)^{\mathcal{V}}\ w : \sigma
\end{array}
$$

Fig. 6. Logical preorder

$$
\begin{array}{lll}
\mathcal{L}(\mathcal{V}^{\bullet}_{\mathbf{unit}}) \triangleq \mathcal{V}_{\mathbf{unit}} & [\![\langle\rangle]\!]_{\mathcal{L}} \triangleq \langle\rangle & \preceq^{\mathcal{V}}_{\mathbf{unit}} \triangleq \Delta^{\mathcal{V}}_{\mathbf{unit}} \\
\mathcal{L}(\mathcal{V}^{\bullet}_{\mathbf{void}}) \triangleq \emptyset & [\![\langle v, w\rangle]\!]_{\mathcal{L}} \triangleq (v, w) & \preceq^{\mathcal{V}}_{\mathbf{void}} \triangleq \emptyset \\
\mathcal{L}(\mathcal{V}^{\bullet}_{\sigma\times\tau}) \triangleq \mathcal{V}^{\bullet}_{\sigma} \times \mathcal{V}^{\bullet}_{\tau} & [\![v \otimes w]\!]_{\mathcal{L}} \triangleq (v, w) & \preceq^{\mathcal{V}}_{\sigma\times\tau} \triangleq [\![-]\!]_{\mathcal{L}}; (\preceq^{\mathcal{V}}_{\sigma} \times \preceq^{\mathcal{V}}_{\sigma}); [\![-]\!]^{\top}_{\mathcal{L}} \\
\mathcal{L}(\mathcal{V}^{\bullet}_{\sigma\otimes\tau}) \triangleq \mathcal{V}^{\bullet}_{\sigma} \times \mathcal{V}^{\bullet}_{\tau} & [\![\mathbf{in}_l\ v]\!]_{\mathcal{L}} \triangleq in_l\ v & \preceq^{\mathcal{V}}_{\sigma+\tau} \triangleq [\![-]\!]_{\mathcal{L}}; (\preceq^{\mathcal{V}}_{\sigma} + \preceq^{\mathcal{V}}_{\sigma}); [\![-]\!]^{\top}_{\mathcal{L}} \\
\mathcal{L}(\mathcal{V}^{\bullet}_{\sigma+\tau}) \triangleq \mathcal{V}^{\bullet}_{\sigma} + \mathcal{V}^{\bullet}_{\tau} & [\![\mathbf{in}_r\ v]\!]_{\mathcal{L}} \triangleq in_r\ v & \preceq^{\mathcal{V}}_{!_s\sigma} \triangleq [\![-]\!]_{\mathcal{L}}; !_s \preceq^{\mathcal{V}}_{\sigma}; [\![-]\!]^{\top}_{\mathcal{L}} \\
\mathcal{L}(\mathcal{V}^{\bullet}_{!_s\sigma}) \triangleq \mathcal{V}^{\bullet}_{\sigma} & [\![[v]]\!]_{\mathcal{L}} \triangleq v & \preceq^{\mathcal{V}}_{\sigma\multimap\tau} \triangleq [\![-]\!]_{\mathcal{L}}; [\preceq^{\mathcal{V}}_{\sigma}, \preceq^{\Lambda}_{\tau}]; [\![-]\!]^{\top}_{\mathcal{L}} \\
\mathcal{L}(\mathcal{V}^{\bullet}_{\sigma\multimap\tau}) \triangleq \mathcal{V}^{\bullet}_{\sigma} \to \Lambda^{\bullet}_{\tau} & [\![\lambda x.f]\!]_{\mathcal{L}} \triangleq v \mapsto f[v/x] & \preceq^{\mathcal{V}}_{\sigma\otimes\tau} \triangleq [\![-]\!]_{\mathcal{E}}; (\preceq^{\mathcal{V}}_{\sigma} \otimes \preceq^{\mathcal{V}}_{\tau}); [\![-]\!]^{\top}_{\mathcal{E}} \\
 & & \preceq^{\Lambda}_{\sigma} \triangleq [\![-]\!]_{\mathcal{E}}; \Gamma \preceq^{\mathcal{V}}_{\sigma}; [\![-]\!]^{\top}_{\mathcal{E}}
\end{array}
$$

Fig. 7. Logical Semantics

It is useful and advantageous to take a more relational (and pointfree) perspective and rephrase Definition 19 using the relational constructions on modal relations of Definition 8. To do so, we unpack the logical meaning of types. For instance, the logical meaning of a closed value $\lambda x.f$ of type $\sigma \multimap \tau$ is to behave as a (linear) function mapping values v of type σ to computations $f[v/x]$ of type τ. We can thus say that the logical meaning of $\lambda x.f$ is the function $(v \mapsto f[v/x]) \in \mathcal{V}^{\bullet}_{\sigma} \to \Lambda^{\bullet}_{\tau}$. The crucial point that defines our logical preorder is then the following: assuming to have defined $\preceq^{\mathcal{V}}_{\sigma}$, $\preceq^{\Lambda}_{\tau}$, we then define $\preceq^{\mathcal{V}}_{\sigma\multimap\tau}$ relying on the logical meaning of values in $\mathcal{V}^{\bullet}_{\sigma\multimap\tau}$ and the Reynolds arrow constructor. In fact, since such logical meanings belong to $\mathcal{V}^{\bullet}_{\sigma} \to \Lambda^{\bullet}_{\tau}$ it is enough to extend $\preceq^{\mathcal{V}}_{\sigma}$ and $\preceq^{\Lambda}_{\tau}$ to the function space $\mathcal{V}^{\bullet}_{\sigma} \to \Lambda^{\bullet}_{\tau}$, which is precisely what $[\preceq^{\mathcal{V}}_{\sigma}, \preceq^{\Lambda}_{\tau}]$ does. We formalise this intuition in Figure 7, where the term relation $\preceq$ is defined relying on a (type-indexed) logical semantics function $[\![-]\!]_{\mathcal{L}} : \mathcal{V}^{\bullet}_{\sigma} \to \mathcal{L}(\mathcal{V}^{\bullet}_{\sigma})$ associating to each closed value its logical meaning.

Remark 3. Notice that the Figure 7 defines $\preceq$ explicitly and not via some universal property, as we did with contextual approximation and as we will do with applicative similarity. To follow such a path, we may consider the defining equality of $\preceq$ as a system of (relational) equations. Obviously, we know that such a system of equations has a solution, viz. $\preceq$. Additionally, solutions are not unique, Δ^{c} being a solution as well. We may then want to find solutions to such a system in terms of least/greatest fixed points. If we try to do so, however, we immediately realise that the functional

associated to those equations is not monotone, due to antitonicity of Reynolds arrow in the first argument.

Let us now move to the meta-theoretical properties of $\preceq^s$. Obviously, $\preceq^s$ is substitutive. We prove that it is also reflexive, transitive, and compatible. Remarkably, compatibility and transitivity both follow from reflexivity. For that reason the result stating that a logical relation is reflexive is usually called the fundamental lemma of logical relations.

Proposition 1 (Fundamental Lemma). $\Delta \subseteq \preceq^s$.

Proof Sketch. The proof is highly nontrivial. We first observe that $\Delta \subseteq \preceq^s$ if and only if for all $\boldsymbol{\sigma}$, $\boldsymbol{s}$, and σ we have $\bigotimes !_{\boldsymbol{s}}(\preceq) \otimes \Delta \subseteq \mathsf{subst}; \preceq; \mathsf{subst}^\top$, where $\mathsf{subst} : \prod_i \mathcal{V}^\bullet_{\sigma_i} \times \Lambda_{\boldsymbol{x}:\boldsymbol{s}\boldsymbol{\sigma}\vdash\sigma} \to \Lambda^\bullet_\sigma$ is defined by $\mathsf{subst}(v_1, \ldots, v_n, e) \triangleq e[v_1, \ldots, v_n/x_1, \ldots, x_n]$. Since $\Delta = \widehat{\Delta}$ (recall that $\Delta = \mu X.\widehat{X}$), it is enough to prove $\bigotimes !_{\boldsymbol{s}}(\preceq) \otimes \widehat{\Delta} \subseteq \mathsf{subst}; \preceq; \mathsf{subst}^\top$, and we do so by induction on the definition of compatible refinement. All cases are handled relying on the axiomatics of a corelator. The most difficult case is the one corresponding to sequencing, where we also need to use the axiomatics of a relator. To do so, we have to "permute" $!_{s\vee 1}$ with Γ. But that is precisely what the lax distributive law in Definition 12 does: $!_{s\vee 1}\Gamma R \subseteq \Gamma !_{s\vee 1} R$ (notice that $s \vee 1 \geq 1$). □

Lemma 6. *$\preceq^s$ is transitive.*

Proof. Since transitivity of a closed term relation implies transitivity of its substitutive extension, it is enough to prove $\preceq; \preceq \subseteq \preceq$. We proceed by induction on types relying on the structural properties of lax extensions. The only interesting case is the one of arrow types, where we calculate:

$$\preceq^{\mathcal{V}}_{\sigma\to\tau}; \preceq^{\mathcal{V}}_{\sigma\to\tau} = [\![-]\!]_{\mathcal{L}}; [\preceq^{\mathcal{V}}_{\sigma}, \preceq^{\Lambda}_{\tau}]; [\![-]\!]^{\top}_{\mathcal{L}}; [\![-]\!]_{\mathcal{L}}; [\preceq^{\mathcal{V}}_{\sigma}, \preceq^{\Lambda}_{\tau}]; [\![-]\!]^{\top}_{\mathcal{L}} \subseteq [\![-]\!]_{\mathcal{L}}; [\preceq^{\mathcal{V}}_{\sigma}; \preceq^{\mathcal{V}}_{\sigma}, \preceq^{\Lambda}_{\tau}; \preceq^{\Lambda}_{\tau}]; [\![-]\!]^{\top}_{\mathcal{L}}.$$

At this point, we cannot rely on the induction hypothesis, since $[-,-]$ is antitone in the first argument. However, by Lemma 1, $\Delta^{\mathcal{V}}_\sigma \subseteq \preceq^{\mathcal{V}}_\sigma$, and thus $\preceq^{\mathcal{V}}_\sigma \subseteq \preceq^{\mathcal{V}}_\sigma; \Delta^{\mathcal{V}}_\sigma \subseteq \preceq^{\mathcal{V}}_\sigma; \preceq^{\mathcal{V}}_\sigma$. We conclude $[\preceq^{\mathcal{V}}_\sigma; \preceq^{\mathcal{V}}_\sigma, \preceq^{\Lambda}_\tau] \subseteq [\preceq^{\mathcal{V}}_\sigma, \preceq^{\Lambda}_\tau]$ by antitonocity, and thus $[\preceq^{\mathcal{V}}_\sigma; \preceq^{\mathcal{V}}_\sigma, \preceq^{\Lambda}_\tau; \preceq^{\Lambda}_\tau] \subseteq [\preceq^{\mathcal{V}}_\sigma, \preceq^{\Lambda}_\tau]$ by induction hypothesis. □

Theorem 1. *$\preceq^s$ is a precongruence.*

Proof. By Lemma 6 and Proposition 1 it is enough to prove compatibility, i.e. $\widehat{\preceq^s} \subseteq \preceq^s$. Notice that for any closed term relation R, we have $R^s[R^s] \subseteq R^s$ and $\widehat{R^s} \subseteq \Delta[R^s]$. We can thus calculate as follows: $\widehat{\preceq^s} \subseteq \Delta[\preceq^s] \subseteq \preceq^s[\preceq^s] \subseteq \preceq^s$ where the ante-penultimate passage uses Lemma 1. □

7 APPLICATIVE SIMILARITY

Our last program refinement is the extension of Abramsky's *applicative similarity* [Abramsky 1990] to $\Lambda_{\Sigma,\mathcal{S}}$. Applicative bisimilarity is a coinductively-defined notion of refinement that compares functions according to the function extensionlity principle. Using our relational apparatus, we see that a (close term) relation R is functionally extensional if $R_{\sigma\multimap\tau} = [\Delta_\sigma, R_\tau]$. Notice the difference between being (functionally) extensional and being logical: in the first case, the antecedent of the Reynolds arrow is the identity relation; in the second case, it is the relation itself. The main advantage of function extensionality is that the operator $[\Delta, -]$ is monotone, and thus one is licensed to define term relations as fixed points of monotone functions.

Definition 20. *Define the mapping $R \mapsto [R]$ on* closed *term relations as follows, where the logical semantics of Figure 6 is extended with $\mathcal{L}(\mathcal{V}^{\bullet}_{\mu t.\sigma}) \triangleq \mathcal{V}^{\bullet}_{\sigma[\mu t.\sigma/t]}$ and $[\![\mathbf{fold}\ v]\!]_{\mathcal{L}} \triangleq v$.*

$$[R]^{\mathcal{V}}_{\mathbf{unit}} \triangleq \Delta^{\mathcal{V}}_{\mathbf{unit}} \qquad [R]^{\mathcal{V}}_{\sigma\times\tau} \triangleq [\![-]\!]_{\mathcal{L}}; (R^{\mathcal{V}}_{\sigma} \times R^{\mathcal{V}}_{\sigma}); [\![-]\!]^{\top}_{\mathcal{L}}$$
$$[R]^{\mathcal{V}}_{\mathbf{void}} \triangleq \emptyset \qquad [R]^{\mathcal{V}}_{\mu t.\sigma} \triangleq [\![-]\!]_{\mathcal{L}}; R^{\mathcal{V}}_{\sigma[t/\mu t.\sigma]}; [\![-]\!]^{\top}_{\mathcal{L}}$$
$$[R]^{\mathcal{V}}_{\sigma+\tau} \triangleq [\![-]\!]_{\mathcal{L}}; (R^{\mathcal{V}}_{\sigma} + R^{\mathcal{V}}_{\sigma}); [\![-]\!]^{\top}_{\mathcal{L}} \qquad [R]^{\mathcal{V}}_{\sigma\multimap\tau} \triangleq [\![-]\!]_{\mathcal{L}}; [\Delta^{\mathcal{V}}_{\sigma}, R^{\wedge}_{\tau}]; [\![-]\!]^{\top}_{\mathcal{L}}$$
$$[R]^{\wedge}_{\sigma} \triangleq [\![-]\!]_{\mathcal{E}}; \Gamma R^{\mathcal{V}}_{\sigma}; [\![-]\!]^{\top}_{\mathcal{E}} \qquad [R]^{\mathcal{V}}_{!_s\sigma} \triangleq [\![-]\!]_{\mathcal{L}}; !_s R^{\mathcal{V}}_{\sigma}; [\![-]\!]^{\top}_{\mathcal{L}}$$

We say that a closed term relation R is an applicative simulation *if $R \subseteq [R]$.*

Since Γ and $!_s$ are monotone by definition, and, thanks to the presence of $[\Delta, -]$. all the constructions involved in the definition of $[R]$ are monotone (see laws in Figure 3), the map $X \mapsto [X]$ is monotone too, and thus it has a greatest fixed point, which we call *applicative similarity*.

Definition 21. *Define* applicative similarly $\lesssim$ *as the term relation $\nu X.[X]$. We extend $\lesssim$ to open terms by taking its* open *extension $\lesssim^{\circ}$.*

Notice that applicative similarity $\lesssim$ being defined coinductively it comes with an associated coinduction proof principle: $R \subseteq [R] \implies R \subseteq \lesssim$. For instance, we easily prove that $\lesssim$ (and thus $\lesssim^{\circ}$) is reflexive and transitive by showing $\Delta \subseteq [\Delta]$ and $\lesssim;\lesssim \subseteq [\lesssim;\lesssim]$.

Coeffectful Howe's Method. We have seen that (the open extension of) applicative similarity is a preorder. We now show that it is also compatible and substitutive (and thus a precongruence, in particular). The proof of such a result is highly nontrivial, and requires to extend Howe's method [Howe 1996; Pitts 2011] (and its effectful extension [Dal Lago et al. 2017a]) to combined effectful and coeffectful setting.

Howe's method works by constructing a term relation (called the precongruence candidate) $\lesssim^{H}$ which is substitutive and compatible by construction, and that extends $\lesssim^{\circ}$. The so-called Key Lemma then states that (the closed restriction of) $\lesssim^{H}$ is an applicative simulation, so that one concludes $\lesssim$ and $\lesssim^{H}$ to coincide. In non-coeffectful calculi, the proof of substitutivity of $\lesssim^{H}$ consists of a routine induction, whereas proving the Key Lemma is more challenging, as it requires a mixed induction-coinduction argument built upon the axiomatic of a relator. Modal and coeffectful calculi present an additional difficulty: in those calculi also proving substitutivity of $\lesssim^{H}$ is nontrivial. And in fact, the axiomatics of a corelator turns out to be precisely what is needed to ensure such a property. Finally, the proof of the Key Lemma relies to the lax distributive law in Definition 12 to handle the interact between the relator Γ (effects) and the corelator $!$ (coeffects). This is exactly the same patter one has in the proof of the Fundamental Lemma (Lemma 1).

Definition 22. *The* Howe extension R^{H} *of a closed term relation R is defined as $\mu X.\widehat{X}; R^{\circ}$.*

The Howe extension of a term relation enjoys several nice properties.

Lemma 7. *Let R be a reflexive and transitive closed term relation. Then R^{H} is a compatible term relation such that $R^{\circ} \subseteq R^{H}$ and $R^{H}; R^{\circ} \subseteq R^{H}$. In particular, $\lesssim^{H}$ is a compatible and reflexive term relation that extends $\lesssim$.*

Lemma 8 (Substitutivity). *If R is a reflexive and transitive term relation, then R^{H} is substitutive.*

Proof sketch. Define the substitution map $\mathrm{subst} : \Lambda_{\Gamma,\boldsymbol{x}:_{\boldsymbol{s}}\sigma\vdash\tau} \times \prod_i \mathcal{V}_{\Theta_i\vdash\sigma_i} \to \Lambda_{\Gamma+\sum \boldsymbol{s}*\Theta\vdash\tau}$ by $\mathrm{subst}(e, v_1, \ldots, v_n) \triangleq e[v_1, \ldots, v_n/x_1, \ldots, x_n]$. Substitutivity amounts to show $R^{H} \otimes \bigotimes_i !_{s_i} R^{H} \subseteq \mathrm{subst}; R^{H}; \mathrm{subst}^{\top}$. Denoting by R^{H}_n the n-th approximation of R^{H}, we have

$$R^{H} \otimes \bigotimes_i !_{s_i} R^{H} = (\bigcup_{n\geq 0} R^{H}_n) \otimes \bigotimes_i !_{s_i} R^{H} \subseteq \bigcup_{n\geq 0} (R^{H}_n \otimes \bigotimes_i !_{s_i} R^{H}).$$

It is thus sufficient to show $\forall n \geq 0.\ R_n^H \otimes \bigotimes_i !_{s_i} R^H \subseteq \mathsf{subst}; R^H; \mathsf{subst}^\top$ The latter is proved by induction on n relying on the axiomatics of a corelator. □

Finally, we have the so-called Key Lemma.

Lemma 9 (Key Lemma). *For any reflexive and transitive applicative simulation R, R^H (restricted to closed terms) is an applicative simulation.*

Proof Sketch. The proof follows the strategy of effectful Howe's method [Dal Lago et al. 2017a]. The crux of the argument is showing that $(R^H)^\wedge_\sigma \subseteq [\![-]\!]_\mathcal{E}; \Gamma R^H; [\![-]\!]^\top_\mathcal{E}$. Assuming $\cdot \vdash^\wedge a \Vdash e\ R^H\ f : \sigma$, we proceed by induction on the evaluation of e relying on the fact that Γ is continuous. The base case of the induction is trivial, whereas for the inductive step we proceed by cases analysis on the shape of e. The difficult case is given by sequencing, which is handled by the axiomatic of a relator. To do so, however, we have to "permute" $!_{s\vee 1}$ with Γ, which is precisely what the lax distributive law (Definition 12) does: $!_{s\vee 1}\Gamma R \subseteq \Gamma !_{s\vee 1} R$ (notice that $s \vee 1 \geq 1$). □

Theorem 2. *Applicative similarly is a precongruence.*

Proof. It is sufficient to show that $(\lesssim)^{\mathrm{o}} = (\lesssim)^H$. Since $(\lesssim)^{\mathrm{o}} \subseteq (\lesssim)^H$, is enough to prove the converse inclusion. First, notice that since $((\lesssim)^H)^{\mathrm{c}}$ is an applicative simulation, $((\lesssim)^H)^{\mathrm{c}}$ is contained in $\lesssim$, and thus $(((\lesssim)^H)^{\mathrm{c}})^{\mathrm{o}}$ is contained in $(\lesssim)^{\mathrm{o}}$. We are done since $(\lesssim)^H \subseteq (((\lesssim)^H)^{\mathrm{c}})^{\mathrm{o}}$. □

8 CONCLUSION

In this work, we have developed a theory and calculus of program relations for a higher-order language with effects and coeffects. Such calculus has its semantic foundations in the notions of modal relations, relators, and the novel notion of a corelator. We have witnessed the strength and robustness of our calculus by defining and studying three notions of (combined effectful and coeffectful) program refinement: contextual approximation, logical preorder, and applicative similarity. To the best of the authors' knowledge, these are the first operationally-based notions of program refinement (and *de facto* equivalence) accounting for combined effects and coeffects appearing in the literature. Using the axiomatics of a relator and of a corelator, we prove general precongruence theorems for such program refinements, this way achieving the first form of *compositional relational reasoning* for combined effects and coeffects in the field. Even if our proofs are highly nontrivial and require major improvements on the existing proof techniques, it is remarkable that the very same abstract, relational notions of a relator and corelator ensure precongruence properties of both logical and applicative refinements.

Our analysis is foundational and, due to space constraints, several examples have been omitted. Nonetheless, the notions of effects and coeffects we deal with are mainstream, and several examples to which our results directly apply can be easily found in the literature. For instance, it is easy to see [Dal Lago and Gavazzo 2021b] that our precongruence results generalise cornerstone results on coeffectful languages, such as metric preservation [Reed and Pierce 2010] and non-interference [Abadi et al. 1999].

Modalities as Metrics, Metrics as Modalities. An interesting observation that we have not treated in this paper is the relationship between our relational techniques and the abstract, quantale-based [Lawvere 1973] theory of program distances [Gavazzo 2018]. In a nutshell, the two approaches (properly generalised) are two faces of the same coin: under some mild conditions, modal relations (on a frame $\mathcal{W}$) correspond to distances taking values in the quantale 2^{W} of modal predicates. Vice versa, any quantale-valued distance defines a modal relation on the frame given by the quantale itself. This correspondence is fruitful in both directions: on the one hand, the theory of relators/lax extensions has been extensively studied in the setting of abstract metrics [Clementino et al. 2004;

Hoffman 2015; Hofmann et al. 2014]; on the other hand, one can use modal relations to define new metrics (such as Böhm tree-like metrics) taking advantages of the notion of a corelator.

Future Work. Concerning future work, the authors plan to use the relational calculus introduced in this work to develop other notions of equivalence and refinement, such as normal-forms (bi)simulations [Dal Lago and Gavazzo 2019a; Lassen 2005]. Another interesting, more challenging research direction is to establish relationship between the notions of refinement (and equivalence) developed: our precongruence theorems imply that, as long as logical relations and applicative similarity are adequate, then they are contained in contextual approximation. Are there some conditions ensuring the opposite inclusion to hold as well? This is challenging question to answer, as the combined presence of effects drastically improve the discriminating power of contextual approximation/equivalence [Dal Lago and Gavazzo 2021c]. Finally, it is yet unclear what is the relationship between coeffectful relational reasoning and other forms of contextual reasoning, such as resource-sensitive and differential ones [Dal Lago and Gavazzo 2020, 2021a,c, 2022; Dal Lago et al. 2019].

Related Work. In recent years, there has been a growing interest for typing disciplines regulating how code can be manipulated. Specific examples of such disciplines date back at least to the 90s, originating from (bounded) linear logic [Benton and Wadler 1996; Girard 1987; Girard et al. 1992], programming languages-based approaches to information flow [Abadi et al. 1999; Volpano et al. 1996], and investigations into the Curry-Howard correspondence for modal logic(s) [Bierman and de Paiva 2000; Pfenning and Davies 2001; Pfenning and Wong 1995]. More recently, researchers started to design calculi with types governing more general notions of resource consumptions [Brunel et al. 2014; Ghica and Smith 2014], quantitative aspects of code usage [Atkey 2018; Orchard et al. 2019; Reed and Pierce 2010], and environmental requirements [Orchard 2014; Petricek et al. 2014], this way obtaining general modal-like type systems [Abel and Bernardy 2020; Gaboardi et al. 2016; Orchard et al. 2019]. From a semantic perspective, such systems have been investigated by means of (comonadic) denotational semantics [Breuvart and Pagani 2015; Gaboardi et al. 2016; Ghica and Smith 2014] and (mostly heap-based) resource sensitive operational semantics [Abel and Bernardy 2020; Bernardy et al. 2018; Brunel et al. 2014; Choudhury et al. 2021; Orchard et al. 2019]. From a denotational perspective, we can think about our program refinements (and equivalences) as defining a denotational model in a category of modal relational spaces and relational homomorphism. Relators and corelators then define ways to extend monads and (the identity) comonads on such a category. From that perspective, our work can be place in the general categorical framework based on monads, comonads, and distributive laws between them by Gaboardi et al. [2016].

Concerning (operationally-based) program equivalence, the work closest to ours is the one by Abel and Bernardy [2020], where logical relations for a (call-by-name) λ-calculus with modal and polymorphic types are introduced. The language of Abel and Bernardy includes polymorphism (which we do not have) but it does *not* support computational effects.

ACKNOWLEDGMENTS

This work is supported by the ERC CoG "DIAPASoN" under Grant No. GA 818616 and MIUR PRIN "ASPRA" under Grant No. 201784YSZ5

REFERENCES

Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. 1999. A Core Calculus of Dependency. In *Proc. of POPL 1999*. 147–160. https://doi.org/10.1145/292540.292555

Andreas Abel and Jean-Philippe Bernardy. 2020. A unified view of modalities in type systems. *Proc. ACM Program. Lang.* 4, ICFP (2020), 90:1–90:28. https://doi.org/10.1145/3408972

S. Abramsky. 1990. The Lazy Lambda Calculus. In *Research Topics in Functional Programming*, D. Turner (Ed.). Addison Wesley, 65–117.

S. Abramsky and A. Jung. 1994. Domain Theory. In *Handbook of Logic in Computer Science*. Clarendon Press, 1–168.

Amal J. Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *Proc. of ESOP 2006*. 69–83. https://doi.org/10.1007/11693024_6

A.W. Appel and D.A. McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* 23, 5 (2001), 657–683. https://doi.org/10.1145/504709.504712

Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *Proc. of LICS 2018*. 56–65. https://doi.org/10.1145/3209108.3209189

Roland Carl Backhouse, Peter J. de Bruin, Paul F. Hoogendijk, Grant Malcolm, Ed Voermans, and Jaap van der Woude. 1991. Polynomial Relators (Extended Abstract). In *Proc. of (AMAST '91 (Workshops in Computing)*. Springer, 303–326.

H.P. Barendregt. 1984. *The lambda calculus: its syntax and semantics*. North-Holland.

M. Barr. 1970. Relational algebras. *Lect. Notes Math.* 137 (1970), 39–55.

Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. Proving expected sensitivity of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 57:1–57:29. https://doi.org/10.1145/3158145

P. N. Benton and Philip Wadler. 1996. Linear Logic, Monads and the Lambda Calculus. In *Proc. of LICS 1996*. 420–431. https://doi.org/10.1109/LICS.1996.561458

Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2018. Linear Haskell: practical linearity in a higher-order polymorphic language. *PACMPL* 2, POPL (2018), 5:1–5:29. https://doi.org/10.1145/3158093

Gavin M. Bierman and Valeria de Paiva. 2000. On an Intuitionistic Modal Logic. *Stud Logica* 65, 3 (2000), 383–416. https://doi.org/10.1023/A:1005291931660

A. Bizjak and L. Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Proc. of FOSSACS 2015*. 279–294. https://doi.org/10.1007/978-3-662-46678-0_18

Flavien Breuvart and Michele Pagani. 2015. Modelling Coeffects in the Relational Semantics of Linear Logic. In *Proc. of CSL 2015*. 567–581. https://doi.org/10.4230/LIPIcs.CSL.2015.567

Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. 2014. A Core Quantitative Coeffect Calculus. In *Proc. of ESOP 2014*. 351–370. https://doi.org/10.1007/978-3-642-54833-8_19

Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–32. https://doi.org/10.1145/3434331

Maria Manuel Clementino, Dirk Hofmann, and Walter Tholen. 2004. One Setting for All: Metric, Topology, Uniformity, Approach Structure. *Appl. Categorical Struct.* 12, 2 (2004), 127–154. https://doi.org/10.1023/B:APCS.0000018144.87456.10

Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *Proc. of ESOP 2017*. 368–392. https://doi.org/10.1007/978-3-662-54434-1_14

Ugo Dal Lago and Francesco Gavazzo. 2019a. Effectful Normal Form Bisimulation. In *Proc. of ESOP 2019*. 263–292. https://doi.org/10.1007/978-3-030-17184-1_10

Ugo Dal Lago and Francesco Gavazzo. 2019b. On Bisimilarity in Lambda Calculi with Continuous Probabilistic Choice. In *Proc. of MFPS 2019*. 121–141. https://doi.org/10.1016/j.entcs.2019.09.007

Ugo Dal Lago and Francesco Gavazzo. 2020. Differential Logical Relations Part II: Increments and Derivatives. In *Proc. of ICTCS 2020*. 101–114.

Ugo Dal Lago and Francesco Gavazzo. 2021a. Differential logical relations, part II increments and derivatives. *Theoretical Computer Science* (2021). https://doi.org/10.1016/j.tcs.2021.09.027

Ugo Dal Lago and Francesco Gavazzo. 2021b. Modal Reasoning = Metric Reasoning, via Lawvere. *CoRR* abs/2103.03871 (2021). https://arxiv.org/abs/2103.03871

Ugo Dal Lago and Francesco Gavazzo. 2021c. Resource Transition Systems and Full Abstraction for Linear Higher-Order Effectful Programs. In *Proc. of FSCD 2021 (LIPIcs, Vol. 195)*. 23:1–23:19. https://doi.org/10.4230/LIPIcs.FSCD.2021.23

Ugo Dal Lago and Francesco Gavazzo. 2022. Effectful Program Distancing. *Proc. ACM Program. Lang.* 6, POPL (2022). https://doi.org/10.1145/3498680

Ugo Dal Lago, Francesco Gavazzo, and Paul Blain Levy. 2017a. Effectful applicative bisimilarity: Monads, relators, and Howe's method. In *Proc. of LICS 2017*. 1–12. https://doi.org/10.1109/LICS.2017.8005117

Ugo Dal Lago, Francesco Gavazzo, and Ryo Tanaka. 2017b. Effectful Applicative Similarity for Call-by-Name Lambda Calculi. In *Proc. of ICTCS 2017*. 87–98.

Ugo Dal Lago, Francesco Gavazzo, and Ryo Tanaka. 2020. Effectful applicative similarity for call-by-name lambda calculi. *Theor. Comput. Sci.* 813 (2020), 234–247. https://doi.org/10.1016/j.tcs.2019.12.025

Ugo Dal Lago, Francesco Gavazzo, and Akira Yoshimizu. 2019. Differential Logical Relations, Part I: The Simply-Typed Case. In *Proc. of ICALP 2019*. 111:1–111:14. https://doi.org/10.4230/LIPIcs.ICALP.2019.111

Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. 2014. On coinductive equivalences for higher-order probabilistic functional programs. In *Proc. of POPL 2014*. 297–308. https://doi.org/10.1145/2535838.2535872

A.A. de Amorim, M. Gaboardi, J. Hsu, S. Katsumata, and I. Cherigui. 2017. A semantic account of metric preservation. In *Proc. of POPL 2017*. 545–556. https://doi.org/10.1145/3009837.3009890

Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Probabilistic Relational Reasoning via Metrics. In *Proc. of LICS 2019*. 1–19. https://doi.org/10.1109/LICS.2019.8785715

Dorothy E. Denning. 1976. A Lattice Model of Secure Information Flow. *Commun. ACM* 19, 5 (1976), 236–243. https://doi.org/10.1145/360051.360056

Marco Gaboardi, Shin-ya Katsumata, Dominic A. Orchard, Flavien Breuvart, and Tarmo Uustalu. 2016. Combining effects and coeffects via grading. In *Proc. of ICFP 2016*. 476–489. https://doi.org/10.1145/2951913.2951939

Francesco Gavazzo. 2018. Quantitative Behavioural Reasoning for Higher-order Effectful Programs: Applicative Distances. In *Proc. of LICS 2018*. 452–461. https://doi.org/10.1145/3209108.3209149

Francesco Gavazzo. 2019. *Coinductive Equivalences and Metrics for Higher-order Languages with Algebraic Effects*. Ph. D. Dissertation. University of Bologna, Italy. http://amsdottorato.unibo.it/9075/

Dan R. Ghica and Alex I. Smith. 2014. Bounded Linear Types in a Resource Semiring. In *Proc. of ESOP 2014*. 331–350. https://doi.org/10.1007/978-3-642-54833-8_18

Jean-Yves Girard. 1987. Linear Logic. *Theor. Comput. Sci.* 50 (1987), 1–102. https://doi.org/10.1006/inco.1998.2700

J-Y. Girard, A. Scedrov, and P.J. Scott. 1992. Bounded Linear Logic: A Modular Approach to Polynomial-Time Computability. *Theor. Comput. Sci.* 97 (1992), 1–66. https://doi.org/10.1016/0304-3975(92)90386-T

Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. 1977. Initial Algebra Semantics and Continuous Algebras. *J. ACM* 24, 1 (1977), 68–95. https://doi.org/10.1145/321992.321997

A.D. Gordon. 1994. A Tutorial on Co-induction and Functional Programming. In *Workshops in Computing*. Springer London, 78–95. https://doi.org/10.1007/978-1-4471-3573-9_6

Andrew Donald Gordon. 1992. *Functional programming and input/output*. Ph. D. Dissertation. University of Cambridge, UK.

Jean Goubault-Larrecq, Slawomir Lasota, and David Nowak. 2008. Logical relations for monadic types. *Math. Struc. Comput. Sci.* 18, 6 (2008), 1169–1217. https://doi.org/10.1017/S0960129508007172

Robert Harper. 2016. *Practical Foundations for Programming Languages (2nd. Ed.)*. Cambridge University Press.

D. Hoffman. 2015. A cottage industry of lax extensions. *Categories and General Algebraic Structures with Applications* 3, 1 (2015), 113–151.

D. Hofmann, G.J. Seal, and W. Tholen (Eds.). 2014. *Monoidal Topology. A Categorical Approach to Order, Metric, and Topology*. Number 153 in Encyclopedia of Mathematics and its Applications. Cambridge University Press.

D.J. Howe. 1996. Proving Congruence of Bisimulation in Functional Programming Languages. *Inf. Comput.* 124, 2 (1996), 103–112. https://doi.org/10.1006/inco.1996.0008

Martin Hyland, Gordon D. Plotkin, and John Power. 2006. Combining effects: Sum and tensor. *Theor. Comput. Sci.* 357, 1-3 (2006), 70–99. https://doi.org/10.1016/j.tcs.2006.03.013

Patricia Johann, Alex Simpson, and Janis Voigtländer. 2010. A Generic Operational Metatheory for Algebraic Effects. In *Proc. of LICS 2010*. IEEE Computer Society, 209–218. https://doi.org/10.1109/LICS.2010.29

Shin-ya Katsumata. 2018. A Double Category Theoretic Analysis of Graded Linear Exponential Comonads. In *Proc. of FOSSACS 2018*. 110–127. https://doi.org/10.1007/978-3-319-89366-2_6

Yasuo Kawahara. 1973. Notes on the universality of relational functors. *Memoirs of the Faculty of Science, Kyushu University. Series A, Mathematics* 27, 2 (1973), 275–289.

Joachim Lambek. 1968. Deductive Systems and Categories I. Syntactic Calculus and Residuated Categories. *Math. Syst. Theory* 2, 4 (1968), 287–318. https://doi.org/10.1007/BF01703261

S.B. Lassen. 1998a. Relational Reasoning About Contexts. In *Higher Order Operational Techniques in Semantics*, Andrew D. Gordon and Andrew M. Pitts (Eds.). 91–136.

S.B. Lassen. 1998b. *Relational Reasoning about Functions and Nondeterminism*. Ph. D. Dissertation. Dept. of Computer Science, University of Aarhus.

Søren B. Lassen. 1999. Bisimulation in Untyped Lambda Calculus: Böhm Trees and Bisimulation up to Context. *Electr. Notes Theor. Comput. Sci.* 20 (1999), 346–374. https://doi.org/10.1016/S1571-0661(04)80083-5

Søren B. Lassen. 2005. Eager Normal Form Bisimulation. In *Proc. of LICS 2005*. 345–354. https://doi.org/10.1109/LICS.2005.15

F.W. Lawvere. 1973. Metric spaces, generalized logic, and closed categories. *Rend. Sem. Mat. Fis. Milano* 43 (1973), 135–166.

P.B. Levy, J. Power, and H. Thielecke. 2003. Modelling Environments in Call-by-Value Programming Languages. *Inf. Comput.* 185, 2 (2003), 182–210. https://doi.org/10.1016/S0890-5401(03)00088-9

S. MacLane. 1971. *Categories for the Working Mathematician*. Springer-Verlag.

J. Maraist, M. Odersky, D.N. Turner, and P. Wadler. 1999. Call-by-name, Call-by-value, Call-by-need and the Linear lambda Calculus. *Theor. Comput. Sci.* 228, 1-2 (1999), 175–210. https://doi.org/10.1016/S0304-3975(98)00358-2

Ian A. Mason and Carolyn L. Talcott. 1991. Equivalence in Functional Languages with Effects. *J. Funct. Program.* 1, 3 (1991), 287–327. https://doi.org/10.1017/S0956796800000125

John McCarthy. 1961. A basis for a mathematical theory of computation, preliminary report. In *Papers presented at the 1961 western joint IRE-AIEE-ACM computer conference, IRE-AIEE-ACM 1961 (Western), Los Angeles, California, USA, May 9-11, 1961*. 225–238. https://doi.org/10.1145/1460690.1460715

John McCarthy. 1962. Towards a Mathematical Science of Computation. In *Proc. of IFIP 1962*. 21–28.

John McCarthy. 1963. A Basis for a Mathematical Theory of Computation). In *Computer Programming and Formal Systems*, P. Braffort and D. Hirschberg (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 35. Elsevier, 33 – 70.

John C. Mitchell. 1996. *Foundations for programming languages*. MIT Press.

Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proc. of LICS 1989*. IEEE Computer Society, 14–23. https://doi.org/10.1109/LICS.1989.39155

Eugenio Moggi. 1991. Notions of Computation and Monads. *Inf. Comput.* 93, 1 (1991), 55–92. https://doi.org/10.1016/0890-5401(91)90052-4

J. Morris. 1969. *Lambda Calculus Models of Programming Languages*. Ph. D. Dissertation. MIT.

C.-H. Luke Ong. 1993. Non-Determinism in a Functional Setting. In *Proc. of LICS 1993*. IEEE Computer Society, 275–286. https://doi.org/10.1109/LICS.1993.287580

Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative Program Reasoning with Graded Modal Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 110 (2019), 110:1–110:30 pages. https://doi.org/10.1145/3341714

Dominic A. Orchard. 2014. *Programming contextual computations*. Ph. D. Dissertation. University of Cambridge, UK.

Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *Proc. of ICFP 2014*. 123–135. https://doi.org/10.1145/2628136.2628160

Frank Pfenning. 2001. Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory. In *Proc. of LICS 2001*. 221–230. https://doi.org/10.1109/LICS.2001.932499

Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Math. Struct. Comput. Sci.* 11, 4 (2001), 511–540. https://doi.org/10.1017/S0960129501003322

Frank Pfenning and Hao-Chi Wong. 1995. On a modal lambda calculus for S4. In *Proc. of MFPS 1995*. 515–534. https://doi.org/10.1016/S1571-0661(04)00028-3

A.M. Pitts. 2011. Howe's Method for Higher-Order Languages. In *Advanced Topics in Bisimulation and Coinduction*, D. Sangiorgi and J. Rutten (Eds.). Cambridge Tracts in Theoretical Computer Science, Vol. 52. Cambridge University Press, 197–232.

A. M. Pitts. 1997. Operationally-Based Theories of Program Equivalence. In *Semantics and Logics of Computation*, P. Dybjer and A. M. Pitts (Eds.). Cambridge University Press, 241–298.

Gordon Plotkin. 1973. Lambda-definability and logical relations. (1973). Technical Report SAI-RM-4, School of A.I., University of Edinburgh.

Gordon D. Plotkin and John Power. 2001a. Adequacy for Algebraic Effects. In *Proc. of FOSSACS 2001*. 1–24. https://doi.org/10.1007/3-540-45315-6_1

Gordon D. Plotkin and John Power. 2001b. Semantics for Algebraic Operations. *Electr. Notes Theor. Comput. Sci.* 45 (2001), 332–345. https://doi.org/10.1016/S1571-0661(04)80970-8

Gordon D. Plotkin and John Power. 2003. Algebraic Operations and Generic Effects. *Applied Categorical Structures* 11, 1 (2003), 69–94. https://doi.org/10.1023/A:1023064908962

J. Reed and B.C. Pierce. 2010. Distance makes the types grow stronger: a calculus for differential privacy. In *Proc. of ICFP 2010*. 157–168. https://doi.org/10.1145/1863543.1863568

J.C. Reynolds. 1983. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*. 513–523.

Richard Routley and Robert K. Meyer. 1973. The Semantics of Entailment. In *Truth, Syntax and Modality*, Hugues Leblanc (Ed.). Studies in Logic and the Foundations of Mathematics, Vol. 68. Elsevier, 199 – 243.

Davide Sangiorgi. 1994. The Lazy Lambda Calculus in a Concurrency Scenario. *Inf. Comput.* 111, 1 (1994), 120–153. https://doi.org/10.1006/inco.1994.1042

Mícheál Ó Searcóid. 2006. *Metric Spaces*. Springer London.

A. Simpson and N. Voorneveld. 2018. Behavioural equivalence via modalities for algebraic effects. In *Proc. of ESOP 2018*. 300–326. https://doi.org/10.1007/978-3-319-89884-1_11

Alex Simpson and Niels F. W. Voorneveld. 2020. Behavioural Equivalence via Modalities for Algebraic Effects. *ACM Trans. Program. Lang. Syst.* 42, 1 (2020), 4:1–4:45. https://doi.org/10.1145/3363518

Alex K. Simpson. 1994. *The proof theory and semantics of intuitionistic modal logic*. Ph. D. Dissertation. University of Edinburgh, UK.

Alasdair Urquhart. 1972. Semantics for Relevant Logics. *J. Symb. Log.* 37, 1 (1972), 159–169. https://doi.org/10.2307/2272559

Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. 1996. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security* 4, 2/3 (1996), 167–188. https://doi.org/10.3233/JCS-1996-42-304